Near-Optimal Topology-adaptive Parameter Synchronization in Distributed DNN Training

Zhe Zhang*, Chuan Wu*, Zongpeng Li†
*The University of Hong Kong, Email: {zzhang2, cwu}@cs.hku.hk †Wuhan University, Email: zongpeng@whu.edu.cn

***Abstract*—Distributed machine learning with multiple concurrent workers has been widely adopted to train large deep neural networks (DNNs). Parameter synchronization is a key component in each iteration of distributed training, where workers exchange locally computed gradients through an AllReduce operation or parameter servers, for global parameter updates. Parameter synchronization often constitutes a significant portion of the training time; minimizing the communication time contributes substantially to DNN training speed-up. Standard ring-based AllReduce or PS architecture work efficiently mostly with homogeneous inter-worker connectivity. However, available bandwidth among workers in real-world clusters is often heterogeneous, due to different hardware configurations, switching topologies, and contention with concurrent jobs. This work investigates the best parameter synchronization topology and schedule among workers for most expedited communication in distributed DNN training. We show that the optimal parameter synchronization topology should be comprised of trees with different workers as roots, each for aggregating or broadcasting a partition of gradients/parameters. We identify near-optimal forest packing to maximally utilize available bandwidth and overlap aggregation and broadcast stages to minimize communication time. We provide theoretical analysis of the performance bound, and show that our scheme outperforms state-of-the-art parameter synchronization schemes by up to 18.3 times with extensive evaluation under various settings.**

I. Introduction

Deep neural networks (DNNs) trained on large datasets have been widely used to drive various applications in domains including computer vision [1], natural language processing [2], and robotics [3]. To train a large DNN model (e.g., Bert [2]), the workload is typically distributed to multiple concurrent workers, to expedite model convergence with parallel training. Data parallelism is the most widely adopted paradigm to train DNNs for production applications [4]. With data-parallel training, each worker has a copy of the DNN model, computes parameter updates (i.e., gradients) based on the local dataset iteratively, and exchanges gradients with other workers for global model updates in each training iteration.

Allreduce algorithm [5] and parameter server (PS) architecture [6] are commonly adopted for exchanging gradients and updating global parameters among workers. This process is referred to as *communication* or *parameter synchronization* in distributed machine learning (ML). Such communication may occupy 50-90% of per-iteration training time [7] [8], which can be especially high in clusters with fast GPUs but relatively low inter-GPU connection bandwidth. Therefore, minimizing the communication time can contribute substantially to distributed DNN training speed-up.

Standard ring-based Allreduce [5] or PS architecture works most efficiently in homogeneous environments (i.e., with the same worker configurations and inter-worker bandwidth) [9]. In a production ML cluster, workers in a training job are commonly configured with the same type of GPUs. However, inter-GPU connectivity is often heterogeneous, depending on whether the workers reside on the same physical server or not, the inter-GPU connection type in the servers and across servers, what the available bandwidth is on inter-server links (due to contention with concurrent jobs in the cluster), etc.

A few recent studies have addressed heterogeneous inter-worker connectivity [8]–[11].Hierarchical parameter synchronization plans have been proposed [10]–[13] to maximally exploit available bandwidth in inter-server switching topologies, such as fat-tree [14] and BCube [15], but they still rely on the symmetry of the topologies. Blink [8] advocates that workers communicate through spanning trees inside a machine for gradient aggregation and then among machines using ring Allreduce for parameter synchronization, before broadcasting updated parameters inside each machine. However, it is less efficient when the number of workers participating in the current round of communication varies across machines, as intra-machine gradient aggregation may finish at different time.

This paper seeks to answer the following question: given any inter-GPU connectivity and bandwidth, what is the best parameter synchronization scheme among a specified set of workers in a distributed DNN training job, to fully utilize the available bandwidth and minimize the communication time? The parameter synchronization scheme specifies how the gradient/parameter chunks should be exchanged among workers, in terms of the interchange topologies and the amount of gradient/parameter data to exchange on the topologies.

To identify the best communication scheme, we first reveal that parameter synchronization in general can be modeled into two stages, gradient/parameter reduce and updated parameter broadcast, and the best reduce or broadcast topology for any indivisible gradient/parameter chunk is a tree. We then aim at optimal packing of directed Steiner trees in the GPU inter-connection network, while these trees are rooted at different workers and responsible for reduce or broadcast of different gradient/parameter chunks. We carefully quantify the communication time on each reduce/broadcast tree, derive the best chunk assignment among trees, and formulate an optimal reduce and broadcast tree packing problem to minimize overall parameter synchronization time. Adopting an efficient primal-

dual algorithm framework, we identify the set of reduce and broadcast trees through an efficient ellipsoid algorithm that exploits the minimum Steiner tree problem as an oracle, and then compute the proportion of chunks to allocate to each tree accordingly. Beyond the basic case of no reduce and broadcast overlap, we further explore the potential of overlapping the reduce and broadcast stages for communication time reduction.

Our algorithm design is supported by solid theoretical analysis. We also carry out extensive evaluations under various realistic inter-GPU connectivity settings, comparing with state-of-the-art parameter synchronization schemes. Our evaluation results reveal that our proposed scheme can reduce up to 2.9 times of the communication time when all workers participate in parameter synchronization, and up to 18.3 times in cases of partial active workers, as compared with state-of-the-art parameter synchronization schemes, including Blink [8], Ring Allreduce [5] and more.

II. Background and Related Work

A. Distributed DNN Training

Synchronous data-parallel training has been most commonly adopted for learning DNN models in production ML clusters [4]. The training dataset is partitioned among multiple workers. At each worker, the local dataset is further divided into mini-batches of samples. In each training iteration, each worker processes one mini-batch to obtain a loss and then computes gradients based on the loss; then, it exchanges locally-computed gradients with others to obtain global parameter updates, apply updated parameters to the local model, and proceed to the next training iteration.

B. Communication Paradigms

The following two parameter synchronization paradigms are dominant in today's production ML workloads [16] [17].

Parameter Server (PS) architecture [6] (Fig. 1(a)). PS nodes are included in the training, which maintain a global copy of parameters. The workers send their locally-computed gradients to the PSs that maintain the respective parameters. Then PSs update their maintained global parameters with a particular updating formula, e.g., using averaged gradients to update parameters with stochastic gradient descent (SGD) method. The workers then pull updated parameters from the PSs.

AllReduce. The workers synchronize parameter updates directly with each other using an AllReduce algorithm. Ring AllReduce [5] (Fig. 1(b)) is commonly adopted [17]: N workers are inter-connected as a ring; the parameters at each worker are split into N chunks. In the i-th round, the worker w concurrently sends the $(w-i+1)\%N$-th chunk to its successor worker along the ring and receives the $(w-i)\%N$-th chunk from its predecessor worker. Then, it averages the received chunk with corresponding parameters. After $N-1$ rounds of communication, each worker obtains one fully averaged chunk. Then, each worker sends its fully averaged chunk to the successor, who will pass it on along the ring; after another $N-1$ rounds, all workers receive all globally updated parameters. We refer to the first $N-1$ rounds as the **reduce stage** and the latter $N-1$ rounds as the **broadcast stage**. Ring AllReduce is proven to be bandwidth optimal [18], i.e., each worker achieves parameter synchronization with the minimum amount of data communication, in a homogeneous environment with the same bandwidth between each pair of workers. However, it is less efficient when the bandwidth of inter-work connections differs.

To reduce parameter synchronization latency, hierarchical ring AllReduce [10], 2D-torus [13] and 2-D mesh [12] based AllReduce algorithms have been proposed, which first reduce parameters inside worker groups, then conduct ring AllReduce across groups and finally broadcast updated global parameters inside each group. A double binary tree algorithm [19] is used in NCCL 2.4 [20], which uses two concurrent reduce/broadcast trees for synchronizing two partitions of parameters. Nonetheless, these schemes still assume homogeneous bandwidth between workers.

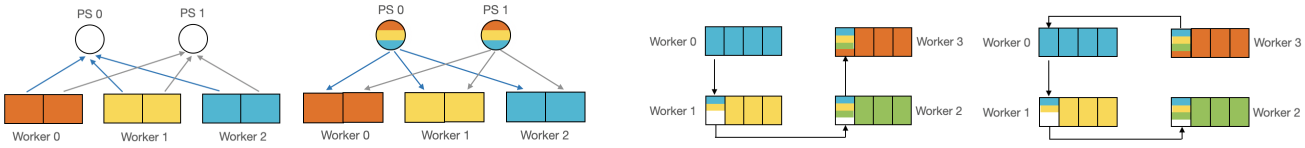
C. Physical Topologies

1) Inter-worker Connections: PCIe is a common type of connections between GPUs. The PCIe link between two devices can vary from 1 to 32 lanes and each lane is full-duplex, e.g., with 4GB/s per direction with PCIe 5.0 [21]. NVLink is another inter-GPU link type, supporting 25GB/s communication for each direction in one link [22]. Across machines, InfiniBand links are commonly used, with throughput up to 50GB/s in each direction [23], e.g., achieved with InfiniBand GPUDirect-RDMA [24] [25].

2) Topologies: Inside a machine, GPUs are typically mounted through PCIe switches and communicate through the CPUs with QuickPath Interconnect (QPI) [25], or directly inter-connected through NVLink (e.g., as in NVIDIA DGX-1 [26] and DGX-2 [27]). For cross-machine communication, the typical network topology in a data center running ML workloads includes a two-layer or three-layer fat-tree, [14] [28], BCube [15], and DCell [29], while the inter-connection network among workers in a single ML job is part of the data center topology, which varies according to the placement of the workers in the data center.

D. Existing Parameter Synchronization Schemes on Different Physical Topologies

Blueconnect [11] is designed on the fat-tree structure [14] and leverages network capacity through concurrent reduce-scatter and all-gather. The reduce-scatter operation iteratively reduces parameters in three levels of rings, making each worker obtain one totally reduced chunk. Then each worker broadcasts the reduced chunk on the three levels of rings through an all-gather operation. Wang *et al.* [30] argue that fat-tree does not match well the traffic pattern in DNN training and propose BML on the BCube [15] topology, utilizing two-level switches. The workers first obtain the partially reduced parameters through level-0 switches in groups and then communicate through level-1 switches to obtain the totally reduced parameters. Both schemes require a symmetric network topology to achieve optimal bandwidth utilization through different levels of switches.



(a) PS: reduce and broadcast

(b) Ring Allreduce: N-1 reduce steps and N-1 broadcast steps

Figure 1: PS and Ring All-reduce

Considering an arbitrary inter-worker connection topology, Matcha [31] decomposes the topology into matchings and carries out concurrent communication among different sampled worker pairs in each round. It requires carefully designed sampling methods to ensure model convergence; the bandwidth on connections between not sampled nodes is unused and thus wasted in the current round of communication. Plink [32] uses one reduce/broadcast tree inside each machine, and further uses tree Allreduce for inter-machine parameter synchronization. However, the inter-machine connections are idle when conducting intra-machine reduce/broadcast, while the intra-machine connections are idle when conducting inter-machine communication. Blink [8] is the most similar work to ours: it packs spanning trees in the intra-machine topology for parameter reduce, synchronizes locally-reduced parameters through ring Allreduce among machines, and then broadcasts updated parameters within each machine. However, it fails when only part of workers participate in the communication and does not utilize the bandwidth of inter-machine bandwidth effectively. Instead, we pack Steiner trees to connect active workers in the whole topology.

III. Problem Model

A. System Abstraction

We consider a data-parallel DNN training job with V workers. The workers are inter-connected using an arbitrary physical network. Without loss of generality, each worker runs on one device (e.g., GPU). The worker inter-connection topology can be described by a graph $\mathcal{G} = (\mathcal{V}, E)$, where $\mathcal{V}$ is the set of devices (workers), and $e \in E$ indicates a connection between two devices. Considering that the links between devices are mostly full-duplex in today's data centers, $\mathcal{G}$ is a directed graph: for workers u and v, we have two edges $e = \overrightarrow{uv}$ and $e' = \overrightarrow{vu}$ indicating two directions of the connection, with the same bandwidth $b_e = b_{e'}$. $b_e = 0$ indicates that there is no connection from u to v.

Let $\tilde{\mathcal{V}}$ denote the given set of N workers to synchronize parameters in the current round of communication. We have $\tilde{\mathcal{V}} \subseteq \mathcal{V}$ (i.e., $N \leq V$), to allow flexibility in all cases other than ideal synchronous training. For example, when worker computation speeds differ, parameters can be synchronized among a subset of workers of similar speeds, instead of all (similar to asynchronous training [33] [9]). We refer to the workers in $\tilde{\mathcal{V}}$ as *active workers* in the current round of communication.

Let C be the set of parameters in the DNN model. We consider that the global parameter update is calculated by averaging parameter updates from all active workers.

B. Reduce-Broadcast Communication

We model the parameter synchronization procedure among active workers in two stages: (i) *reduce*, where locally-computed gradients or parameters are passed around the workers for averaging; (ii) *broadcast*, where the worker with a completely reduced parameter chunk[1] sends the updated parameter chunk to all other workers. This two-stage abstraction can represent common parameter synchronization schemes, e.g., the PS architecture and AllReduce algorithms.

With the PS architecture (Fig. 1(a)), active workers send local gradients to PS nodes (reduce stage), and PSs send updated parameters back to the workers (broadcast stage). The communication topology can be viewed as S trees rooted at S PSs. Each tree is employed to reduce and broadcast a subset of model parameters that the respective PS maintains.

For Ring AllReduce (Fig. 1(b)), the first $|\tilde{\mathcal{V}}| - 1$ communication steps correspond to the reduce stage and the next $|\tilde{\mathcal{V}}| - 1$ communication steps are the broadcast stage. Two chain-shape trees (with $N - 1$ edges each) are rooted at each worker: one responsible for reducing a chunk of $|C|/|\tilde{\mathcal{V}}|$ parameters, and the other for broadcasting the completely reduced chunk.

Then, we show that for any indivisible parameter chunk c_i, the optimal broadcast topology can only be a tree, and so does its optimal reduce topology. Note that the data communication direction in a reduce tree is different from that in the broadcast tree of the same root: gradients/parameters are transmitted towards the root in the reduce tree, while updated parameters are sent outwards from the root in the broadcast tree.

Property 1. *The best reduce or broadcast topology for an indivisible parameter chunk c_i is a tree.*

The property holds as each worker only needs to send (receive) the chunk to (from) one other worker for reduce (broadcast) in the optimal case, as the chunk is indivisible. We aim to construct the optimal collections of reduce and broadcast trees, and then schedule parameter chunks to each tree to maximally utilize available bandwidth in the worker graph $\mathcal{G}$ and thus minimize parameter synchronization time among active workers.

C. Communication Time

In a reduce (broadcast) tree, the reduce (broadcast) time is the maximum chunk delivery time between the root and any leaf node. We show that the reduce (broadcast) communication

[1]The worker has a complete average of these parameters of all workers, or it has the averaged gradients of the parameters from all workers and applies it to obtain the updated parameters.

time along a tree is decided by the size of the parameter chunk and minimum bandwidth of links in the tree.

Property 2. *Omitting gradient/parameter processing time at each worker (i.e., averaging for reduce and forwarding for broadcast) and assuming the parameter chunk c_i can be divided into infinitely small sub-chunks for pipelined transmission, the communication time for reducing (broadcasting) parameter chunk c_i in a reduce (broadcast) tree $\mathcal{R}_i$ is asymptotically $\mathcal{T}_{tree} = \frac{|c_i|}{\min_{e\in\mathcal{R}_i} b_e}$.*

Proof. When splitting the parameter chunk c_i to s equal-sized sub-chunks, the time taken for transmitting one sub-chunk from worker v directly to worker v' is $\frac{|c_i|/s}{b_e}$, where b_e is the bandwidth of edge $e = \overrightarrow{vv'}$. Denote this as one unit of transmission time. Consider the case of a reduce tree. We need at most $N-1$ units of transmission time for the first sub-chunk to reach the root and another $s-1$ units for the rest $s-1$ sub-chunks to be received by the root. Therefore, we need at most $s+N-2$ units of transmission time for the root to receive the chunk from worker v. Let $\mathcal{R}_i$ denote the set of edges on the path from v to the root in the reduce tree. The total communication time from v to the root is no larger than $(s+N-2)\times\frac{|c_i|/s}{\min_{e\in\mathcal{R}_i} b_e} = \frac{s+N-2}{s}\times\frac{|c_i|}{\min_{e\in\mathcal{R}_i} b_e}$. When $s \gg N$, we have $\frac{s+N-2}{s}\to 1$. Consequently, the communication time for reducing the chunks from all workers in the reduce tree is at most $\mathcal{T}_{tree} = \frac{|c_i|}{\min_{e\in\mathcal{R}_i} b_e}$. We can prove the case of the broadbast tree in the same way. □

Based on Property 2, the *effective bandwidth* of reduce (broadcast) along the tree $\mathcal{R}_i$ is $w(\mathcal{R}_i) = \min_{e\in\mathcal{R}_i} b_e$.

To maximally utilize the bandwidth of all inter-worker connections, we identify a collection of reduce (broadcast) trees, and then partition parameters C to chunks $c_1, c_2, \ldots$ for reduce (broadcast) in different trees. Along each tree, the parameter chunk is further divided into small sub-chunks, which are transmitted one after another in the pipelined manner, to maximize concurrent link usage and approach the ideal communication time given in Property 2. An illustration is given in Fig. 2.

The following property further gives the best parameter assignment to the reduce (broadcast) trees for minimizing communication time.

Property 3. *Given the set of reduce (or broadcast) trees, $\{\mathcal{R}_1, \ldots, \mathcal{R}_r\}$, the minimum reduce (or broadcast) communication time is achieved when the model parameters C are allocated to the reduce (or broadcast) trees in proportion to the trees' effective bandwidth, i.e., with $|c_i| \propto w(\mathcal{R}_i)$, at $\mathcal{T}_{stage} = \frac{|C|}{\sum_{i\in\{1,\ldots,r\}} w(\mathcal{R}_i)} = \frac{|c_i|}{w(\mathcal{R}_i)}, \forall i \in \{1,\ldots,r\}$.*

Proof. Let $\frac{|c_i|}{w(\mathcal{R}_i)} = \gamma,\ \forall i \in \{1,\ldots,r\}$. We have $\frac{\sum_{i\in\{1,\ldots,r\}}|c_i|}{\sum_{i\in\{1,\ldots,r\}} w(\mathcal{R}_i)} = \frac{|C|}{\sum_{i\in\{1,\ldots,r\}} w(\mathcal{R}_i)} = \gamma$ as well. According to Property 2, the communication time in each tree $\mathcal{R}_i$ is γ. Since all reduce (broadcast) trees can transmit chunks in parallel, the minimum communication time in the reduce (broadcast) stage is achieved at $\mathcal{T}_{stage} = \gamma$. □

Table I: Notation

Notation	Description
$\mathcal{G}$	$\mathcal{G} = (\mathcal{V}, E)$: inter-connection topology among workers in $\mathcal{V}$ with edge set E
$\tilde{\mathcal{V}}$	set of active workers
N	# of active workers
C	parameters in the DNN model
$\mathcal{I}^{(v)}$	reduce trees rooted at worker v
$\mathcal{J}^{(v)}$	broadcast trees rooted at v (in pure broadcast stage)
$\mathcal{K}^{(v)}$	broadcast trees rooted at v (in overlapped broadcast stage)
x_e^i (x_e^j)	whether e is in tree $i \in \mathcal{I}^{(v)}$ ($j \in \mathcal{J}^{(v)}$)
b_e	available bandwidth of edge e
$\alpha, \mathcal{A}$	α-approximate minimum Steiner tree algorithm $\mathcal{A}$
w_i	weight (effective bandwidth) of tree i
$\chi_{(v)}$	aggregate weight of reduce/broadcast trees rooted at v

D. Optimal Reduce and Broadcast Tree Packing Problem

Let $\mathcal{I}^{(v)}$ be the set of possible reduce trees and $\mathcal{J}^{(v)}$ be the set of possible broadcast trees inter-connecting active workers $\tilde{\mathcal{V}}$ in graph $\mathcal{G}$, rooted at worker v, $\forall v \in \tilde{\mathcal{V}}$. Each reduce (broadcast) tree is a Steiner tree with directed edges pointing towards (outward from) the root, spanning all active workers (possibly including some other workers as relays). We use vector $x^i \in \{0,1\}^{|E|}$ ($x^j \in \{0,1\}^{|E|}$) to describe a reduce (broadcast) tree $i \in \mathcal{I}^{(v)}$ ($j \in \mathcal{J}^{(v)}$): $x_e^i = 1$ ($x_e^j = 1$) if directed edge e is in the tree and $x_e^i = 0$ ($x_e^j = 0$), otherwise, $\forall e \in E$. Let w_i (w_j) be the weight of reduce tree i (broadcast tree j), denoting the bandwidth that each of its edges occupies in the respective physical link. A reduce (broadcast) tree in set $\mathcal{I}^{(v)}$ ($\mathcal{J}^{(v)}$) of weight 0 is not selected for reduce (broadcast).

Let C_v be the set of parameter chunks allocated for reduce and broadcast by trees rooted at worker v. That is, these chunks are reduced by the reduce trees $\mathcal{I}^{(v)}$ rooted at worker v; worker v obtains updated parameters and then broadcasts these parameters along the broadcast trees $\mathcal{J}^{(v)}$. Note that the set of reduce trees $\mathcal{I}^{(v)}$ can differ from the set of broadcast trees $\mathcal{J}^{(v)}$. On the other hand, the total effective bandwidth of trees in $\mathcal{I}^{(v)}$ and $\mathcal{J}^{(v)}$ equals, as the same set of parameters C_v are reduced by trees in $\mathcal{I}^{(v)}$ and broadcast by trees in $\mathcal{J}^{(v)}$, i.e., $\sum_{i\in\mathcal{I}^{(v)}} w_i = \sum_{j\in\mathcal{J}^{(v)}} w_j$.

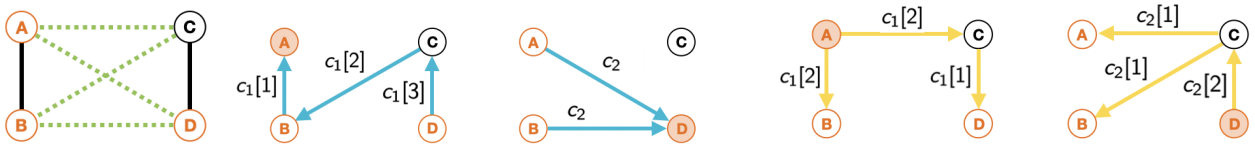
Base on Property 3, the parameters are allocated to the trees in proportion to the corresponding effective bandwidth. Therefore, the reduce time for C_v over all reduce trees rooted at v is $\mathcal{T}_{reduce}^{(v)} = \frac{|C_v|}{\sum_{i\in\mathcal{I}^{(v)}} w_i}$, $\forall v \in \tilde{\mathcal{V}}$, and the reduce time of all chunks is:

$$\mathcal{T}_{reduce} = \frac{\sum_{v\in\tilde{\mathcal{V}}}|C_v|}{\sum_{v\in\tilde{\mathcal{V}}}\sum_{i\in\mathcal{I}^{(v)}} w_i} = \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}}\sum_{i\in\mathcal{I}^{(v)}} w_i} \quad (1)$$

Similarly, the broadcast time of all updated parameters is $\mathcal{T}_{broadcast} = \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}}\sum_{j\in\mathcal{J}^{(v)}} w_j}$. Then the total communication time among N active workers is:

$$\mathcal{T}_{comm} = \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}}\sum_{i\in\mathcal{I}^{(v)}} w_i} + \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}}\sum_{j\in\mathcal{J}^{(v)}} w_j}. \quad (2)$$

Our optimization problem to identify the optimal sets of reduce and broadcast trees and allocate bandwidth to the trees is formulated as follows, which minimizes the parameter synchronization time in Eqn. (2) (the size of parameters $|C|$ is fixed according to the DNN model and is hence omitted from the objective function). Notations are summarized in Table I.



(a) Active: worker A, B, D (b) Reduce trees rooted at A and D (c) Broadcast trees rooted at A and D

Figure 2: Reduce and broadcast trees: an example with 4 GPUs and 3 active workers

$$\text{minimize} \quad \frac{1}{\sum_{v\in\tilde{\mathcal{V}}}\sum_{i\in\mathcal{I}^{(v)}} w_i} + \frac{1}{\sum_{v\in\tilde{\mathcal{V}}}\sum_{j\in\mathcal{J}^{(v)}} w_j} \tag{3}$$

$$\text{subject to} \quad \forall v \in \tilde{\mathcal{V}}: \sum_{i\in\mathcal{I}^{(v)}} w_i = \sum_{j\in\mathcal{J}^{(v)}} w_j \tag{3a}$$

$$\forall e \in E: \sum_{v\in\tilde{\mathcal{V}}}\sum_{i\in\mathcal{I}^{(v)}} w_i x_e^i \le b_e \tag{3b}$$

$$\forall e \in E: \sum_{v\in\tilde{\mathcal{V}}}\sum_{j\in\mathcal{J}^{(v)}} w_j x_e^j \le b_e \tag{3c}$$

$$\forall v \in \tilde{\mathcal{V}}, i \in \mathcal{I}^{(v)}, j \in \mathcal{J}^{(v)}: w_i \ge 0, w_j \ge 0. \tag{3d}$$

Constraints in (3a) indicate that the same set of chunks are reduced by trees rooted at v and then broadcast by trees rooted at v. Constraints in (3b) bound the aggregate bandwidth consumption of reduce trees by the capacity of each edge, and (3c) are the bandwidth constraints for broadcast trees. In this formulation, reduce trees and broadcast trees do not share link bandwidth, assuming that the broadcast stage starts after parameter reduce procedures on all reduce trees finish. We will design efficient algorithms to approximately solve this problem in the following section. In Sec. V, we further investigate the case where reduce stage and broadcast stage may overlap as much as the network bandwidth and chunk reduce-broadcast dependencies allow.

Let $\chi_{(v)} = \sum_{i\in\mathcal{I}^{(v)}} w_i = \sum_{j\in\mathcal{J}^{(v)}} w_j$, denoting the aggregate weight of reduce or broadcast trees rooted at v. Optimization problem (3) can then be converted to the following:

$$\text{maximize} \quad \sum_{v\in\tilde{\mathcal{V}}} \chi_{(v)} \tag{4}$$

$$\text{subject to} \quad \forall v \in \tilde{\mathcal{V}}: \sum_{i\in\mathcal{I}^{(v)}} w_i = \chi_{(v)} \tag{4a}$$

$$\forall v \in \tilde{\mathcal{V}}: \sum_{j\in\mathcal{J}^{(v)}} w_j = \chi_{(v)} \tag{4b}$$

$$\forall e \in E: \sum_{v\in\tilde{\mathcal{V}}}\sum_{i\in\mathcal{I}^{(v)}} w_i x_e^i \le b_e \tag{4c}$$

$$\forall e \in E: \sum_{v\in\tilde{\mathcal{V}}}\sum_{j\in\mathcal{J}^{(v)}} w_j x_e^j \le b_e \tag{4d}$$

$$\forall v \in \tilde{\mathcal{V}}, i \in \mathcal{I}^{(v)}, j \in \mathcal{J}^{(v)}: \tag{4e}$$

$$w_i \ge 0, w_j \ge 0, \chi_{(v)} \ge 0. \tag{4f}$$

Given x_e^i's and x_e^j's (aka given the reduce and broadcast trees), optimization problem (4) is a linear program (LP) with $|\tilde{V}|$ variables, $\chi_{(v)}$'s, and no more than $|\tilde{V}|(k_r + k_b)$ variables, w_i's (k_r and k_b are the maximal number of reduce or broadcast trees at each root, respectively). However, the choices of $x \in \{0,1\}^{|E|}$ is exponential at $2^{|E|}$, leading to exponentially-many potential trees and corresponding variables in the LP. Instead of finding all trees, we will use an efficient algorithm to find a polynomial number of Steiner trees $\tilde{\mathcal{I}}^{(v)}$ and $\tilde{\mathcal{J}}^{(v)}$ for each root $v \in \tilde{\mathcal{V}}$, and use these trees for reduce and broadcast, with weights computed by solving the LP efficiently.

IV. Parameter Synchronization Topology Construction

In this section, we design an efficient algorithm to find the reduce and broadcast trees for parameter synchronization.

When the broadcast stage happens after the entire reduce stage is finished, problem (3) is separable into two sub-problems, for finding the set of reduce trees and the set of broadcast trees, respectively. Finding reduce (broadcast) trees connecting active workers in the given graph is essentially a directed Steiner forest packing problem [34], with directed Steiner trees rooted at different active workers. When all the workers in the graph are active, the Steiner trees become spanning trees, while optimal directed spanning forest packing can be computed in polynomial time [35]. In the general case, the directed Steiner forest packing problem is NP-hard [34].

We follow the primal-dual framework proposed by Jain *et al.* [36] in our algorithm design to identify a set of reduce (broadcast) trees, achieving a proven approximate ratio to the optimum. We will present our algorithm using the case of finding broadcast trees (i.e., broadcast forest packing), while reduce forest packing can be computed similarly except for reversing the direction of edges in the algorithm.

The sub-problem of packing broadcast trees in problem (3) is equivalent to the following (w_j's are variables):

$$\text{maximize} \quad \sum_{v\in\tilde{\mathcal{V}}}\sum_{j\in\mathcal{J}^{(v)}} w_j \tag{5}$$

$$\text{subject to} \quad \forall e \in E: \sum_{v\in\tilde{\mathcal{V}}}\sum_{j\in\mathcal{J}^{(v)}} w_j x_e^j \le b_e \tag{5a}$$

$$\forall v \in \tilde{\mathcal{V}}, j \in \mathcal{J}^{(v)}: w_j \ge 0 \tag{5b}$$

The dual problem of (5) is as follows:

$$\text{minimize} \quad \sum_{e\in E} b_e y_e \tag{6}$$

$$\text{subject to} \quad \forall v \in \tilde{\mathcal{V}}, \forall j \in \mathcal{J}^{(v)},: \sum_{e\in E} y_e x_e^j \ge 1 \tag{6a}$$

$$\forall e \in E: y_e \ge 0 \tag{6b}$$

We solve the dual problem (6) following the Ellipsoid algorithm in [36] and identify a set of Steiner trees to use. We add constraint $\sum_{e\in E} b_e y_e \le D$ to the dual problem, and

find the minimum D^* through binary search that makes the following set Y_D non-empty:

$$Y_D = \Big\{ y \in \mathbb{R}_+^{|E|} \mid \sum_{e \in E} b_e y_e \leq D, \sum_{e \in E} y_e x_e^j \geq 1, \forall j \in \mathcal{J} \Big\} \quad (7)$$

Our Ellipsoid algorithm proceeds as follows. In each iteration, we evaluate whether $\sum_{e \in E} b_e y_e \leq D$ is satisfied. If not, we cut down the set of feasible points by the objective cut defined by $\sum_{e \in E} b_e y_e = D$ (removing the $\sum_{e \in E} b_e y_e > D$ part) and update y_e's to the center of the new feasible set. If $\sum_{e \in E} b_e y_e \leq D$, we call an algorithm $\mathcal{A}$ to find the approximate minimum Steiner tree $\tilde{x}$ in the graph with edge weight y_e's. For the identified Steiner tree, if $\sum_{e \in E} y_e \tilde{x}_e \geq 1$, y_e's are feasible, the current set $Y_D \neq \emptyset$ and $D > D^*$; otherwise, we cut the feasible set using the cutting plane $\sum_{e \in E} y_e \tilde{x}_e = 1$ (removing the $\sum_{e \in E} y_e \tilde{x}_e < 1$ portion), and update y_e's to be the center of the new feasible set. The tree packing algorithm to solve the dual problem is given in Alg. 1.

We then use the Steiner trees found by algorithm $\mathcal{A}$ throughout the Ellipsoid algorithm steps as $\tilde{\mathcal{J}}$ in the primal problem, and solve the resulting LP to compute corresponding w_j's (set $w_j = 0, \forall j \in \mathcal{J} - \tilde{\mathcal{J}}$), resulting in a polynomial number of variables to compute. Especially, the number of trees in $\tilde{\mathcal{J}}$ is polynomial at $O(\sqrt{D}K|\tilde{\mathcal{V}}|)$, where K is the maximum number of iterations that the ellipsoid algorithm runs, and $\sqrt{D}$ is due to the binary search for the dual objective in (6).

There exist efficient algorithms to approximately solve the minimum Steiner tree problem [37]. The following theorem shows that if algorithm $\mathcal{A}$ that we use to find the minimum Steiner tree has an approximation ratio of α, then the approximation ratio is also α when we use the set of trees output by Alg. 1 to solve problem (5), as compared to solving it using all possible Steiner trees.

Theorem 1. *Given an α-approximate algorithm $\mathcal{A}$ for the minimum Steiner tree problem, problem (5) can be solved with an α-approximate ratio in polynomial-time, using the primal-dual approach.*

Proof. Let D^* be the minimum value of dual problem (6), computed with all possible broadcast trees in $\mathcal{J}$. Let $\tilde{D}$ be the result produced by Alg. 1, which packs Steiner trees in $\tilde{\mathcal{J}}$ found by algorithm $\mathcal{A}$. We have $\tilde{D} \leq D^*$ as $\tilde{D}$ is computed on a relaxed problem with fewer constraints. The output $\tilde{\mathcal{J}}$ of Alg. 1 ensures that if and only if $\mathbf{y} \in Y_{\tilde{\mathcal{D}}}$, the weight of trees in $\tilde{\mathcal{J}}$ is greater or equal than 1, i.e., $\mathbf{y}(\tilde{x}) = \sum_{e \in E} y_e \tilde{x}_e \geq 1, \forall \tilde{x} \in \tilde{\mathcal{J}}$. Since algorithm $\mathcal{A}$ is α-approximate, the approximate minimum Steiner tree $\tilde{x}^k$ found in iteration k of Alg. 1 with edge weights $\mathbf{y}^k$ has weight $\alpha \mathbf{y}^k(x^{*k}) \geq \mathbf{y}^k(\tilde{x}^k) \geq \mathbf{y}^k(x^{*k})$, where x^{*k} is the exact minimum Steiner tree. Therefore, for the feasible solution $\tilde{y}$ produced by Alg. 1, it satisfies $\alpha \tilde{\mathbf{y}}(x^*) \geq \tilde{\mathbf{y}}(\tilde{x}) \geq 1$, $\forall \tilde{x} \in \tilde{\mathcal{J}}$, which leads to $\tilde{\mathbf{y}}(x^*) \geq \frac{1}{\alpha}$. Given that x^* is the exact minimum Steiner tree under weight $\tilde{y}$, we know $\tilde{\mathbf{y}}(x) \geq \tilde{\mathbf{y}}(x^*) \geq \frac{1}{\alpha}$, $\forall x \in \mathcal{J}$. In other words, $\sum_{e \in E} \tilde{y}_e x_e \geq \frac{1}{\alpha}$, $\forall x \in \mathcal{J}$ and hence $\sum_{e \in E} (\alpha \tilde{y}_e) x_e \geq 1, \forall x \in \mathcal{J}$. Therefore, $(\alpha \tilde{y})$ is feasible solution for original dual problem and $\sum_{e \in E} b_e (\alpha \tilde{y}_e) \geq D^*$. Accordingly, $\tilde{D} = \sum_{e \in E} b_e \tilde{y}_e \geq \frac{1}{\alpha} D^*$.

Algorithm 1: Packing Trees

Input $\mathcal{A}$: α-approximate algorithm for minimum Steiner tree problem; K: number of iterations to run the ellipsoid algorithm
Output $\tilde{\mathcal{J}}$: trees to pack
for $v \in \tilde{\mathcal{V}}$ **do**
  D_{rb} := large enough number // upper bound of D
  D_{lb} := 0 // lower bound of D
  y_e := random(0, 0.1), $\forall e \in E$
  while $D_{rb} > D_{lb}$ **do**
    $D := (D_{rb} + D_{lb})/2$
    $k := 0$
    while $k < K$ **do**
      if $\sum_{e \in E} b_e y_e > D$ **then**
        Cut feasible set of y_e's by $\sum_{e \in E} b_e y_e = D$ and set y_e's to the new center
      else
        $\tilde{x}$:= minimum Steiner tree rooted at v found by $\mathcal{A}$.
        if $\sum_{e \in E} y_e \tilde{x} < 1$ **then**
          $\tilde{\mathcal{J}}^{(v)} := \tilde{\mathcal{J}}^{(v)} \cup \{\tilde{x}\}$
          Cut feasible set of y_e's by $\sum_{e \in E} y_e \tilde{x}_e = 1$ and set y_e's to the new center of the ellipsoid
        else
          $D_{rb} := D$
          break
        end
        $k = k + 1$
      end
    end
    $D_{lb} := D$
  end
end

Let $\tilde{P}$ (P^*) be the optimal value of primal problem (5) solved based on trees in $\tilde{\mathcal{J}}$ ($\mathcal{J}$). By duality of linear program, we have $\tilde{P} = \tilde{D}$ and $P^* = D^*$. Consequently, $\tilde{P}$ satisfies $\frac{1}{\alpha} P^* \leq \tilde{P} \leq P^*$.

Therefore, using the Steiner trees produced by Alg. 1 in problem (5), we can solve the LP (5) to achieve an α-approximate solution. The Ellipsoid algorithm to solve the dual problem in Alg. 1 runs in polynomial time and produces $O(\sqrt{D}K|\tilde{\mathcal{V}}|)$ trees. Given the trees, the primal LP can be solved in polynomial time too. □

We use Alg. 1 to produce the set of broadcast trees and the set of reduce trees, respectively. Then we use these trees in solving problem (4), i.e., solving the linear program to derive weights associated with the reduce and broadcast trees, ensuring that the total bandwidth of reduce trees equals that of the broadcast trees at each root. The following theorem gives the approximation ratio of our approach in solving the original problem (3).

Theorem 2. *Problem (3) can be solved with an α-approximate ratio in polynomial time.*

Proof. Let $\tilde{\chi}_{(v)}$ be the aggregate weight of reduce/broadcast trees rooted at v computed by our approach on the reduce/broadcast trees found by Alg. 1, and $\chi^*_{(v)}$ be the optimum computed on all possible reduce/broadcast trees. Based on Theorem 1, we know that when we use Alg. 1 to reduce the trees to pack from $\mathcal{I}^{(v)}$ ($\mathcal{J}^{(v)}$) to $\tilde{\mathcal{I}}^{(v)}$ ($\tilde{\mathcal{J}}^{(v)}$), computed weights $\tilde{w}_i$'s ($\tilde{w}_j$'s) satisfy $\frac{1}{\alpha}w^*_i \leq \tilde{w}_i \leq w^*_i$ ($\frac{1}{\alpha}w^*_j \leq \tilde{w}_j \leq w^*_j$) and constraints (3b) ((3c)). Note that $\tilde{w}_i$ and $\tilde{w}_j$, and w^*_i and w^*_j may be scaled down from their computed values by solving the respective problem (5) to satisfy constraint (3a). Therefore, the aggregate weight of trees $\tilde{\chi}_{(v)} = \sum_{i\in\mathcal{I}^{(v)}} \tilde{w}_i = \sum_{j\in\mathcal{J}^{(v)}} \tilde{w}_j$ satisfies $\frac{1}{\alpha}\sum_{v\in\tilde{\mathcal{V}}} \chi^*_{(v)} \leq \sum_{v\in\tilde{\mathcal{V}}} \tilde{\chi}_{(v)} \leq \sum_{v\in\tilde{\mathcal{V}}} \chi^*_{(v)}$. Let O^* be the optimum objective value of problem (3). Therefore, the objective value of problem (3) achieved by our approach, $\tilde{O} = \frac{2}{\sum_{v\in\tilde{\mathcal{V}}} \tilde{\chi}_{(v)}}$, satisfies: $O^* \leq \tilde{O} \leq \alpha O^*$. Based on similar time complexity analysis as in proof of Theorem 1, problem (3) can be solved by our approach in polynomial time too. □

V. Partially Overlapped Reduce and Broadcast

In Sec. IV, problem (3) is solved based on the separation of reduce and broadcast stages, i.e., a root v does not broadcast the updated parameter chunks C_v until it has reduced all chunks. It is worth noting that in the reduce stage, every worker sends the chunk(s) towards the root, and there might be bandwidth left unused on links pointing outwards from the root. Therefore, to fully utilize the network bandwidth for expediting parameter synchronization, a root can start broadcasting updated parameters (i.e., sub-chunks) after reducing part of a chunk. Furthermore, we can release allocated bandwidth of some reduce trees for such advanced broadcast, if that helps in expediting overall parameter synchronization completion.

We refer to the advanced parameter broadcast during the reduce stage as **overlapped broadcast**, and the broadcast after the reduce stage has completed as the **pure broadcast** stage. Let $\mathcal{K}^{(v)}$ denote the set of broadcast trees rooted at v for overlapped broadcast, and w_k be the weight (i.e., effective bandwidth) of broadcast tree $k \in \mathcal{K}^{(v)}$. Vector $x^k \in \{0,1\}^{|E|}$ describes broadcast tree $k \in \mathcal{K}^{(v)}$. C_{res} represents the set of updated parameters to broadcast in the pure broadcast stage, which can be obtained by excluding the chunks broadcasted during the overlapped broadcast stage from the whole set of parameters, i.e., $|C_{res}| = |C| - \mathcal{T}_{reduce} \times (\sum_{v\in\tilde{\mathcal{V}}} \sum_{k\in\mathcal{K}^{(v)}} w_k)$. Based on Property 3 and Eqn. (2), the overall communication time for parameter synchronization is now:

$$\mathcal{T}_{comm} = \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{i\in\mathcal{I}^{(v)}} w_i} + \frac{|C_{res}|}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{j\in\mathcal{J}^{(v)}} w_j}$$

$$= \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{i\in\mathcal{I}^{(v)}} w_i} + \frac{|C| - \frac{|C| \sum_{v\in\tilde{\mathcal{V}}} \sum_{k\in\mathcal{K}^{(v)}} w_k}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{i\in\mathcal{I}^{(v)}} w_i}}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{j\in\mathcal{J}^{(v)}} w_j}$$

(according to Eqn. 1)

$$= \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{i\in\mathcal{I}^{(v)}} w_i} + \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{j\in\mathcal{J}^{(v)}} w_j}\left(1 - \frac{\sum_{v\in\tilde{\mathcal{V}}} \sum_{k\in\mathcal{K}^{(v)}} w_k}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{i\in\mathcal{I}^{(v)}} w_i}\right)$$

Let $\chi_{(v)} = \sum_{i\in\mathcal{I}^{(v)}} w_i = \sum_{k\in\mathcal{K}^{(v)}} w_k + \sum_{j\in\mathcal{J}^{(v)}} w_j$ be the aggregate weight of reduce or broadcast trees rooted at v. We further have:

$$\mathcal{T}_{comm} = \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}} \chi_{(v)}} + \frac{|C|}{\sum_{v\in\tilde{\mathcal{V}}} \sum_{j\in\mathcal{J}^{(v)}} w_j} \times \frac{\sum_{v\in\tilde{\mathcal{V}}} (\chi_{(v)} - \sum_{k\in\mathcal{K}^{(v)}} w_k)}{\sum_{v\in\tilde{\mathcal{V}}} \chi_{(v)}}$$

$$= \frac{2|C|}{\sum_{v\in\tilde{\mathcal{V}}} \chi_{(v)}}$$

Allowing overlap of reduce and broadcast stages, the optimization problem to identify the optimal sets of reduce trees and broadcast trees (in both overlapped and pure broadcast stages) with allocated bandwidths is as follows:

$$\text{maximize} \sum_{v\in\tilde{\mathcal{V}}} \chi_{(v)} \tag{8}$$

subject to

$$\forall v \in \tilde{\mathcal{V}} : \chi_{(v)} = \sum_{i\in\mathcal{I}^{(v)}} w_i \tag{8a}$$

$$\forall v \in \tilde{\mathcal{V}} : \chi_{(v)} = \sum_{k\in\mathcal{K}^{(v)}} w_k + \sum_{j\in\mathcal{J}^{(v)}} w_j \tag{8b}$$

$$\forall v \in \tilde{\mathcal{V}} : \sum_{i\in\mathcal{I}^{(v)}} w_i \geq \sum_{k\in\mathcal{K}^{(v)}} w_k \tag{8c}$$

$$\forall e \in \mathcal{E} : \sum_{v\in\tilde{\mathcal{V}}} \sum_{i\in\mathcal{I}^{(v)}} w_i x^i_e + \sum_{v\in\tilde{\mathcal{V}}} \sum_{k\in\mathcal{K}^{(v)}} w_k x^k_e \leq b_e \tag{8d}$$

$$\forall e \in \mathcal{E} : \sum_{v\in\tilde{\mathcal{V}}} \sum_{j\in\mathcal{J}^{(v)}} w_j x^j_e \leq b_e \tag{8e}$$

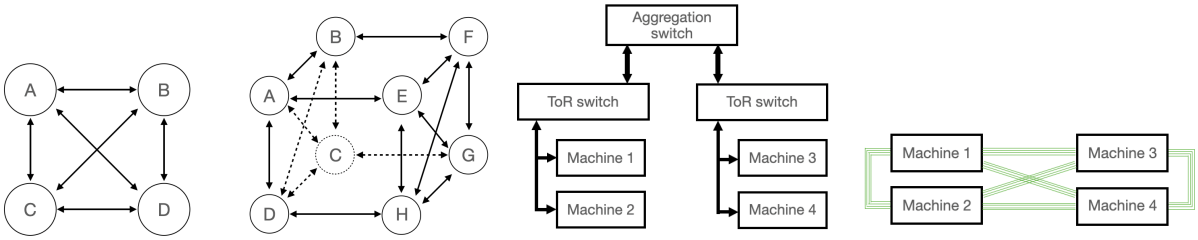
$$\forall v \in \tilde{\mathcal{V}}, i \in \mathcal{I}^{(v)}, j \in \mathcal{J}^{(v)}, k \in \mathcal{K}^{(v)} : w_i, w_j, w_k, \chi_{(v)} \geq 0 \tag{8f}$$

Constraints in (8c) ensure that at each root, the aggregated reduce bandwidth in the reduce stage is no smaller than the aggregated broadcast bandwidth for overlapped broadcast because root can only broadcast an updated chunk after it has received the reduced chunk. The overlapped broadcast trees share bandwidth on links with the reduce trees (Constraints (8d)) as they happen simultaneously.

To solve the problem (8), we generate reduce trees and broadcast trees, respectively, using Alg. 1. We use the obtained broadcast trees in both overlapped and pure broadcast stages (but the allocated bandwidth to the broadcast trees may differ in the overlapped and pure broadcast stages). Given the trees (i.e., x^e_i's, x^e_k's and x^e_j's), we solve the linear program in (8) and obtain the corresponding tree weights for reduce, overlapped broadcast, and pure broadcast.

Theorem 3. *Problem (8) can be solved within an α-approximate ratio in polynomial time.*

Proof. We use Alg. 1 to find the reduce/broadcast trees $\tilde{\mathcal{I}}^{(v)}$, $\tilde{\mathcal{J}}^{(v)}$, $\tilde{\mathcal{K}}^{(v)}$. Therefore, similar to the proof of Theorem 2, $\tilde{w}$'s that satisfy constraints in problem (8) also satisfy $\frac{1}{\alpha}w^* \leq \tilde{w} \leq w^*$. Therefore the aggregate weight $\tilde{\chi}_{(v)} = \sum_{i\in\tilde{\mathcal{I}}^{(v)}} \tilde{w}_i = \sum_{k\in\tilde{\mathcal{K}}^{(v)}} \tilde{w}_k + \sum_{j\in\tilde{\mathcal{J}}^{(v)}} \tilde{w}_j$ satisfies $\frac{1}{\alpha}\chi^*_{(v)} \leq \tilde{\chi}_{(v)} \leq \chi^*_{(v)}$. In other words, we can obtain at least $\frac{1}{\alpha}$ aggregate weight of the original problem by solving the updated problem with trees generated by Alg. 1. □



(a) 4 GPUs inside a machine (b) 8 GPUs inside a machine (c) Topology of 4 machines (d) Inter-connections of 4 machines

Figure 3: Intra-machine and Inter-machine Connection Topologies.

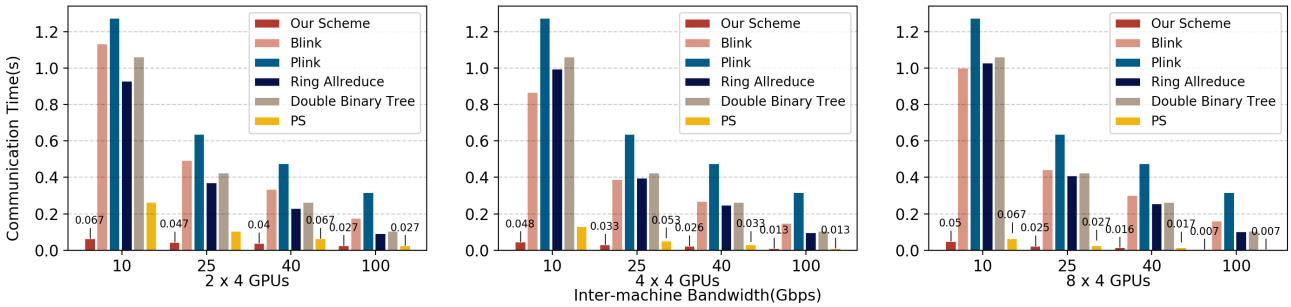


Figure 4: Per-round communication time: all workers are active

VI. Performance Evaluation

A. Methodology

Settings. We simulate training of a DNN model over 2, 4, and 8 machines. Each machine is equipped with 4 or 8 GPUs, with intra-machine GPU connectivity given in Fig. 3(a) and (b). The machines are placed on a rack, connected to a ToR (top of rack) switch, and then to an aggregation switch, as illustrated in Fig. 3(c) (with the example of 4 machines). Fig. 3(d) shows inter-connectivity abstraction among 4 machines; inter-machine bandwidth is decided by available bandwidth along the physical path between the machines. Intra-machine connections have bandwidth around 100Gbps in each direction for machines using PCIe inter-GPU connect [21] [25], and around 300Gbps on machines with NVLink [22]. By default, machines use PCIe inter-connect. We evaluate our scheme in different networks with 10Gbps, 25Gbps, 40Gbps, or 100Gbps inter-machine connection bandwidth (in each direction), respectively. The parameter size of the DNN model is 1.33GB, around the size of a BERT-LARGE model [2].

Baselines. We compare our scheme (allowing reduce and broadcast overlap, unless stated otherwise), with state-of-the-art parameter synchronization paradigms: (i) Blink [8]; (ii) Plink [32]; (iii) Ring Allreduce [5]; (iv) Double Binary Trees as in NCCL 2.4 [20] [19]; (v) PS architecture. Please refer to Sec. II for details of these schemes.

B. All Workers are Active

Fig. 4 presents the parameter synchronization time in each communication round when training the model with different number of GPUs and inter-GPU bandwidths, with all workers participating in parameter synchronization. Here, $A \times B$ means A machines with B GPUs each. We observe that the communication time with our scheme is 2.9, 2.1, 1.2 times smaller than the optimum among baselines, when the inter-machine bandwidth is 10 Gbps, 25 Gbps, and 40 Gbps, respectively. Our scheme achieves the same communication time as the PS architecture when the inter-machine and intra-machine bandwidth is at the same level (i.e., 100 Gbps), as one one-hop tree rooted at each worker constitutes the best topology which utilizes all bandwidth on all links. The excellent performance with our method compared to baselines is due to that: 1) as compared to Blink and Plink, we consider the intra-machine and inter-machine paths together, minimizing the influence of bottleneck connections on the overall communication time; 2) we choose the paths that minimize the communication time with proven performance bound instead of using simple heuristics; 3) we use packing tree methods, and can maximally use all connections without being limited by the specific structures like rings and double trees.

C. Different Active Worker Distributions

We next vary the distribution of active workers when we train the model on 4 machines and each holds 4 GPUs. In Fig. 5, (A, B, C, D) denotes A, B, C, and D active workers in each machine, respectively. The results shows that our scheme achieves superior performance consistently in all cases. Our scheme reduces up to 18.3 times the communication time when compared to optimal baselines. The advantages of our method are more obvious in settings where the inter-machine bandwidth is much less than intra-machine bandwidth and fewer workers are active. In addition to efficient communi-

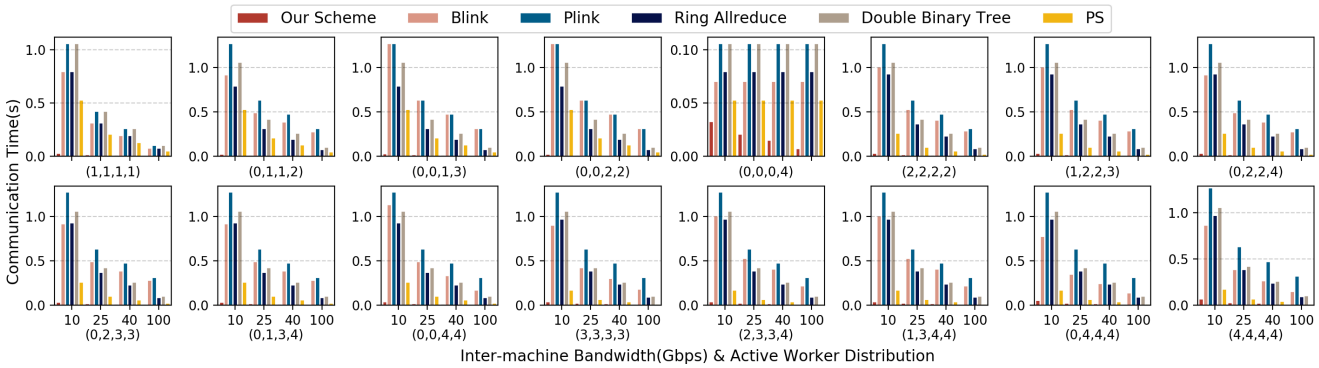


Figure 5: Per-round communication time: different active worker distributions among 4 machines with 4 GPUs each.

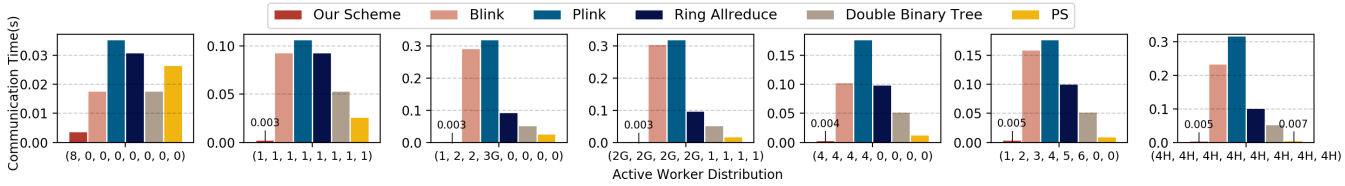


Figure 6: Per-round communication time: different active worker distributions on 8 machines with 8 GPUs each.

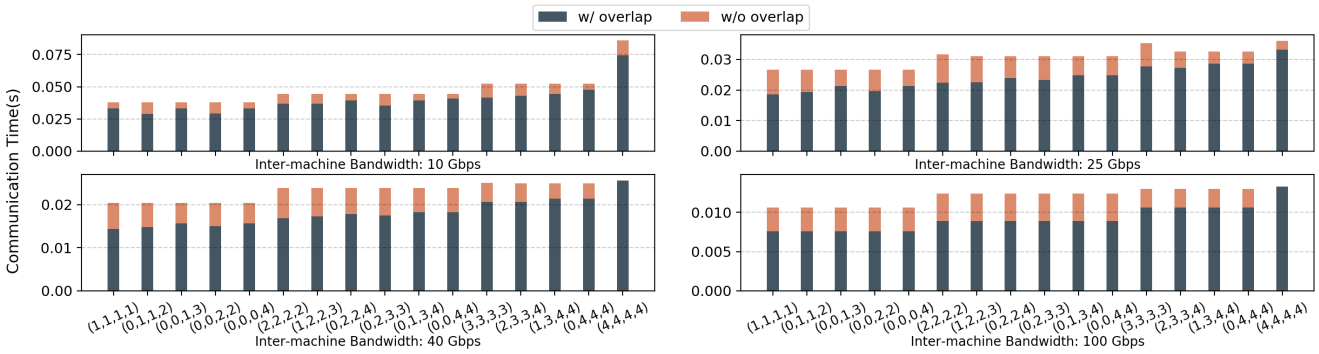


Figure 7: Overlap vs. non-overlap under different active worker distributions on 4 machines with 4 GPUs each.

cation among active workers, our method may use inactive workers as relays in the communication, which improves the overall utilization of available connections (when non-active workers are not synchronizing their parameters).

We further examine different active worker distributions on 8 machines and each holds 8 GPUs, simulating a DGX-1 system (Fig. 3(b)). The inter-machine bandwidth is set to 100 Gbps. For each machine, we use $2G$ to denote choosing active workers A and G, $3G$ to denote choosing workers A, B, and G, and $4H$ to denote choosing workers A, B, G, and H; further, we use a single number to denote choosing workers with consecutive indices, e.g., 6 means choosing workers $A-F$. The 8-element vector in Fig. 6 corresponds to such active worker selection on the 8 machines. Our scheme exhibits persistent excellent performance in all cases and reduces communication time for at least 1.44 times compared to optimal results among the baselines.

D. Reduce/broadcast overlap vs. non-overlap

We further examine the gain of overlapping reduce and broadcast trees with 100Gbps Intra-machine bandwidth. In Fig. 7, we observe that the overlapped broadcast can reduce up to 30% communication time compared to no reduce and broadcast overlap, especially in networks with larger inter-machine bandwidth.

VII. Conclusion

This paper studies optimal parameter synchronization topology and schedule for data-parallel distributed DNN training, under any inter-GPU connectivity and bandwidth. We design algorithms to construct parameter reduce and broadcast trees, optimally decide parameter chunks allocated to the trees, and allow the best overlap of reduce and broadcast, for expedited communication for parameter synchronization. We evaluate the proposed scheme extensively under various realistic settings and demonstrate its superior performance in communication time reduction compared to state-of-the-art parameter synchronization schemes.

Acknowledgment

This work was supported in part by grants from Hong Kong RGC under the contracts HKU 17204619, 17208920.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2016.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019.

[3] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *I. J. Robotics Res.*, 2018.

[4] Y. Zou, X. Jin, Y. Li, Z. Guo, E. Wang, and B. Xiao, "Mariana: Tencent deep learning platform and its applications," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1772–1777, 2014. [Online]. Available: http://www.vldb.org/pvldb/vol7/p1772-tencent.pdf

[5] A. Gibiansky, "Bringing hpc techniques to deep learning," in *https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/*, 2017.

[6] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI*, 2014.

[7] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for DNN training," in *Proceedings of SOSP*, 2019.

[8] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. R. Devanur, and I. Stoica, "Blink: Fast and generic collectives for distributed ML," in *Proceedings of MLSys*, 2020.

[9] Q. Luo, J. He, Y. Zhuo, and X. Qian, "Prague: High-performance heterogeneity-aware asynchronous decentralized training," in *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, J. R. Larus, L. Ceze, and K. Strauss, Eds. ACM, 2020, pp. 401–416. [Online]. Available: https://doi.org/10.1145/3373376.3378499

[10] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *CoRR*, vol. abs/1807.11205, 2018.

[11] M. Cho, U. Finkler, D. S. Kung, and H. C. Hunter, "Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy," in *Proceedings of MLSys*, 2019.

[12] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," *CoRR*, vol. abs/1811.06992, 2018.

[13] 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.

[14] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM*, 2008.

[15] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *Proceedings of the ACM SIGCOMM*, 2009.

[16] 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," in *Proceedings of SOSP 2019*, 2019.

[17] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *CoRR*, vol. abs/1802.05799, 2018.

[18] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distributed Comput.*, 2009.

[19] P. Sanders, J. Speck, and J. L. Träff, "Two-tree algorithms for full bandwidth broadcast, reduction and scan," *Parallel Comput.*, 2009.

[20] "NCCL 2.4," https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/.

[21] "PCIe 5.0," https://techreport.com/news/32064/pcie-4-0-specification-finally-out-with-16-gts-on-tap/.

[22] "NVLink," https://www.nvidia.com/en-us/data-center/nvlink/.

[23] "InfiniBand," https://lenovopress.com/lp1195-mellanox-connectx-6-hdr-adapters.

[24] "GPUDirect RDMA," https://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

[25] A. Li, S. L. Song, J. Chen, J. Li, X. Liu, N. R. Tallent, and K. J. Barker, "Evaluating modern GPU interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect," *IEEE Trans. Parallel Distrib. Syst.*, 2020.

[26] "DGX-1," https://www.nvidia.com/en-us/data-center/dgx-1/.

[27] "DGX-2," https://www.nvidia.com/en-us/data-center/dgx-2/.

[28] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mob. Netw. Appl.*, Apr. 2014.

[29] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *Proceedings of the ACM SIGCOMM*, 2008.

[30] S. Wang, D. Li, Y. Cheng, J. Geng, Y. Wang, S. Wang, S. Xia, and J. Wu, "BML: A high-performance, low-cost gradient synchronization algorithm for DML training," in *NeurIPS*, 2018.

[31] J. Wang, A. K. Sahu, Z. Yang, G. Joshi, and S. Kar, "MATCHA: speeding up decentralized SGD via matching decomposition sampling," *CoRR*, vol. abs/1905.09435, 2019.

[32] L. Luo, P. West, A. Krishnamurthy, L. Ceze, and J. Nelson, "Plink: Discovering and exploiting locality for accelerated distributed training on the public cloud," in *Proceedings of MLSys*, 2020.

[33] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., 2018.

[34] J. Cheriyan and M. R. Salavatipour, "Hardness and approximation results for packing steiner trees," *Algorithmica*, 2006.

[35] H. N. Gabow and K. S. Manu, "Packing algorithms for arborescences (and spanning trees) in capacitated graphs," *Math. Program.*, 1998.

[36] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing steiner trees," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[37] D. Watel and M.-A. Weisser, "A practical greedy approximation for the directed steiner tree problem," *J. Comb. Optim.*, 2016.