Enhancing Collective Communication in MCM Accelerators for Deep Learning Training

Sabuj Laskar[§]
Texas A&M University
sabuj.laskar@tamu.edu

Pranati Majhi[§]
Texas A&M University
pmajhi@tamu.edu

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

Farabi Mahmud
Texas A&M University
farabi@tamu.edu

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

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

***Abstract*—With the widespread adoption of Deep Learning (DL) models, the demand for DL accelerator hardware has risen. On top of that, DL models are becoming massive in size. To accommodate those models, multi-chip-module (MCM) emerges as an effective approach for implementing large-scale DL accelerators. While MCMs have shown promising results for DL inference, its potential for Deep Learning Training remains largely unexplored. Current approaches fail to fully utilize available links in a mesh interconnection network of an MCM accelerator. To address this issue, we propose two novel AllReduce algorithms for mesh-based MCM accelerators - RingBiOdd and Three Tree Overlap (TTO). RingBiOdd is a ring-based algorithm that enhances the bandwidth of AllReduce by creating two unidirectional rings using bidirectional interconnects. On the other hand, TTO is a tree-based algorithm that improves AllReduce performance by overlapping data chunks. TTO constructs three topology-aware disjoint trees and runs different steps of the AllReduce operation in parallel. We present a detailed design and implementation of the proposed approaches. Our experimental results over seven DL models indicate that RingBiOdd achieves 50% and 8% training time reduction over unidirectional Ring AllReduce and MultiTree. Furthermore, TTO demonstrates 33% and 29% training time reduction over state-of-the-art MultiTree and Bidirectional Ring AllReduce, respectively.**

I. Introduction

The widespread adoption of Deep Neural Networks (DNNs) across various domains, such as image recognition [27], [41], [65], [69], language modeling [68], [76], autonomous vehicles [6], [22], and audio synthesis [75], has led to a focus on enhancing their accuracy. A common strategy involves increasing the model size, resulting in state-of-the-art (SOTA) DNN models [8], [20], [27], [45], [57], [71], [76] with trillions of parameters and megabytes of storage requirements. However, this expansion poses a challenge as the model size surpasses a single chip's computation and storage capacity. As a solution, chiplet-based architectures [4], [32], [62] have emerged to effectively scale up the system.

In recent times, the integration of multi-chip-modules (MCMs) has become prevalent in the construction of large-scale CPUs [9], [36], [38] and GPUs [3], [83]. This approach offers reduced costs through the utilization of smaller chiplets and the availability of high-speed, high-bandwidth signaling [81], enabling chiplet-based systems using MCMs to achieve superior performance at a lower cost compared to large monolithic chip designs. Moreover, the flexibility to adjust the number of chiplets within a package allows for the straightforward design of systems of different scales without the need for separate chip designs for each market segment. Consequently, numerous studies have concentrated on applying MCMs to enhance the performance of DNN inference tasks [4], [32], [42], [49], [62]. However, the potential of MCMs for DNN training remains largely unexplored to date.

The growing trend of employing larger models for DNN training has led to widespread use of Data Parallelism [61], [63], [86] in parallel and distributed training [5], [73]. To enhance model accuracy, Stochastic Gradient Descent (SGD) is commonly used as an optimization technique, requiring collective communication between computing nodes. AllReduce, an iterative algorithm, is widely adopted for this purpose during training. However, as the number of nodes increases, the data communicated by each node remains constant [21], [85], resulting in increased overall communication. Consequently, this creates an unfavorable computation and communication ratio, making AllReduce a bottleneck for large-scale distributed training [47].

Numerous AllReduce algorithms have been proposed to address the aforementioned challenge. Baidu Research introduced the Ring AllReduce [18], [51], a straightforward and bandwidth-efficient algorithm, which has been incorporated into NVIDIA Collective Communication Library (NCCL) [48] and Horovod [60]. Other well known AllReduce algorithms are Bidirectional Ring AllReduce [34], Halving-doubling (HDRM) [14] for BiGraphs and topology-aware tree-based algorithm MultiTree [31].

Up until now, the existing algorithms have demonstrated effectiveness in various topologies such as Torus, BiGraph [14] and Hybrid cube-mesh [1]. However, in the context of MCM-based architectures where chiplets are interconnected via on-package links forming a mesh-like topology, the absence of wrap-around links, as seen in Torus, limits the performance of existing AllReduce algorithms. For instance, the Bidirectional

[§]*Both authors contributed equally to this work.*

Ring AllReduce, aimed at improving bandwidth and link utilization in the Ring algorithm, cannot form bidirectional rings in an odd-sized[1] mesh due to the lack of a Hamiltonian cycle [15]. So, the Bidirectional Ring AllReduce algorithm is unavailable for such cases. Another tree-based algorithm called Double Binary Tree (DBTree) [59], also implemented in NCCL, forms two binary trees to maximize bandwidth utilization. However, it lacks topology awareness, resulting in sub optimal tree mapping within the physical topology. The MultiTree algorithm forms a set of topology-aware trees to maximize link utilization in any topology. However, the tree heights increase significantly when the underlying topology is mesh, leading to extra communication latency compared to other topologies like torus. Overall, all existing algorithms utilize only around 50% of the total links(more details in section III-C), highlighting the need for novel algorithms specifically tailored for mesh topology to achieve much higher link utilization.

In this paper, we introduce two novel AllReduce algorithms specifically designed for mesh topology. To address the absence of a Hamiltonian cycle in an odd-sized mesh, the first algorithm, RingBiOdd, forms two unidirectional rings in the opposite directions among $N-1$ chiplets[2] instead of N. Moreover, RingBiOdd optimizes communication for the remaining chiplet such that both the ReduceScatter and AllGather stages are completed in $N-1$ hops, matching the hop count of Bidirectional Ring AllReduce for even-sized mesh. RingBiOdd can be easily integrated with popular DNN collective communication libraries like NCCL and Horovod which already use Bidirectional Ring AllReduce for even-sized mesh [35]. As a result, RingBiOdd makes Bidirectional Ring AllReduce a practical choice for any mesh topology.

The paper also proposes a tree-based algorithm, TTO, which surpasses all existing algorithms, including RingBiOdd, in link utilization, leading to improved performance. TTO utilizes three topology-aware disjoint trees, overlapping multiple data chunks[3] to maximize link usage and achieve high speedup. Building three disjoint trees requires running training in $N-1$ chiplets within an N-chiplet system, increasing training time. However, the significant performance improvement from TTO's AllReduce phase outweighs the extended computation time in end-to-end training, making it an effective solution to existing AllReduce algorithm challenges. Although TTO performs superior to all other AllReduce algorithms, we propose RingBiOdd to ensure Bidirectional Ring AllReduce's applicability for any mesh topology, considering the widespread usage of the Ring AllReduce algorithm.

While our paper primarily analyzes the efficiency of RingBiOdd and TTO within MCM-based systems, these algorithms can be applied to any mesh-based systems requiring AllReduce functionality. AllReduce is an essential primitive used in parallel computing applications, including Scientific Computing [2], [16], [29], [53], [55], [80], Graph Processing [87], and Artificial Intelligence [48], [60]. An extensive five-year profiling of HPC applications [54] showed that more than 17% of operational time was spent in AllReduce communication. Therefore, both proposed algorithms will improve communication latency during AllReduce operations for any system with mesh topology.

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

- We identify inefficiencies in the existing state-of-the-art AllReduce algorithms in MCM-based systems, more specifically mesh-based topologies.
- We propose RingBiOdd, a Bidirectional Ring AllReduce algorithm for odd-sized mesh, which ensures that Bidirectional Ring AllReduce can be applied to any mesh topology, effectively doubling the bandwidth usage and performance compared to widely used Unidirectional Ring AllReduce.
- To further improve the performance by overlapping data chunks using unused links, we propose a Tree-based algorithm, TTO , which constructs three topology-aware disjoint trees by running the training in $N-1$ chiplets in an N-chiplet system to perform ReduceScatter and AllGather in parallel over multiple chunks.
- Our evaluation using synthetic data and SOTA DNN models show that RingBiOdd achieves a $1.9\times$ and $1.1\times$ communication speedup as well as 50% and 8% training time reduction over unidirectional Ring AllReduce and MultiTree, respectively. Furthermore, TTO demonstrates a $1.6\times$ and $1.4\times$ speedup including 33% and 29% training time reduction over MultiTree and Bidirectional Ring AllReduce, respectively.

The rest of the paper is organized as follows: II and III introduce the background and motivation, respectively. IV and V present the two proposed AllReduce algorithms. The methodology is described in VI. Results and further discussions are outlined in VII and VIII, respectively followed by more related work in IX. Finally, we conclude the paper in X.

II. Background

This section introduces the background of Data Parallel Training for DNN, the AllReduce Algorithm and the MCM architecture.

A. Data Parallel Training for Deep Learning Applications

DNN training iteratively adjusts neural network weights and biases using backpropagation, computing gradients of the loss function. It involves forward propagation, loss calculation, backpropagation for gradients, and parameter updates via an optimization algorithm like stochastic gradient descent. This process is repeated for epochs or until convergence.

To scale and speed up training with abundant data, Data Parallel Distributed Training across nodes is common [5]. For that, data is first divided into mini-batches and distributed among nodes. Then gradients are exchanged among the nodes

[1]A mesh of size $m \times n$ is called odd-sized if both m and n are odd, else it is even-sized.

[2]Throughout the paper, we use nodes and chiplets interchangeably.

[3]A chunk represents a portion of gradient for the collective.

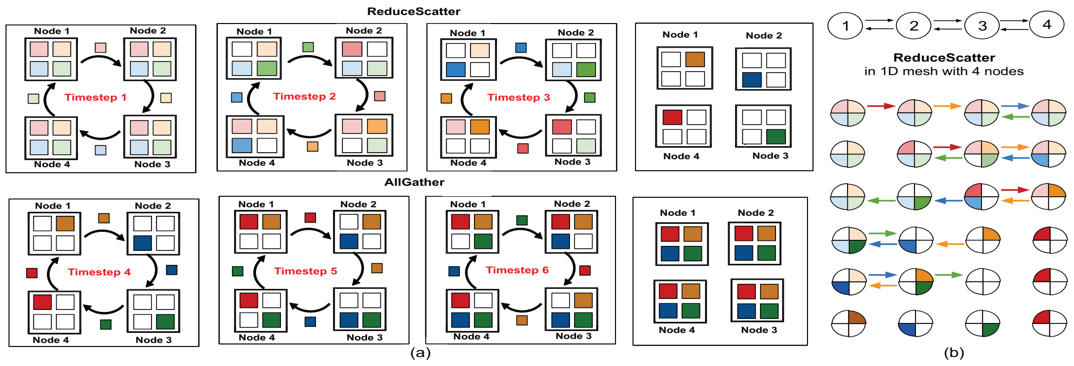


Fig. 1: (a) An example of Ring AllReduce algorithm showcasing different levels of aggregation with varying color intensity. Here top and bottom rows represent ReduceScatter and AllGather stages respectively. With four nodes in total, the total gradient is divided into four parts, each indicated by a different color. (b) Ring ReduceScatter for 4 nodes in 1D mesh. Here, the color of arrow denotes which part is transferred from a node.

to facilitate the effective training process in Data Parallel Training.

B. *AllReduce Algorithm*

The AllReduce algorithm synchronizes gradients among nodes in Data Parallel Distributed Computing. It consists of two phases: ReduceScatter which aggregates gradients, and AllGather which distributes them. There are many algorithms proposed for this purpose such as Direct, Ring-based AllReduce [18], Halving Doubling [14], MultiTree [31] and Double Binary Tree Algorithm [59].

In Ring-based AllReduce (Figure 1(a)), a cyclic path is used to visit each node exactly once and takes $2(N-1)$ timesteps for a topology with N nodes. To leverage bidirectional links in an MCM architecture, the same ring in the opposite direction can be used for another half of data [34]. This approach doubles the performance by utilizing the full bandwidth of ring.

The Ring-based AllReduce method can be extended to an $n \times n$ mesh topology in a hierarchical fashion by splitting the total gradient data into two halves and concurrently applying AllReduce over both halves along the x and y dimensions independently. Following ReduceScatter in the first dimension, the second ReduceScatter begins in the opposite dimension with $1/2n$ of total data for each half. However, due to the absence of wrap-around links in each dimension of the mesh, communication between the first and last nodes requires $n-1$ hops, leading to a significant performance degradation. In Figure 1, we illustrate the steps of the Ring AllReduce algorithm in a 1D mesh with 4 nodes. Throughout this paper, we denote this algorithm as Ring-2D.

In addition, tree-based algorithms like MultiTree [31] and Double Binary Tree Algorithm [59] exist. MultiTree is a topology-aware tree-based AllReduce algorithm that constructs N trees for an N-node system, ensuring no link conflicts during AllReduce. It is a greedy algorithm applicable to many interconnect topologies, generating ReduceScatter schedules through bottom-up traversal of the trees and AllGather schedules through top-down traversal. Another approach is the Double Binary Tree Algorithm [59], which forms two binary trees to efficiently synchronize data.

C. *Multi-Chip-Modules*

MCMs are a promising approach in deep learning, enhancing performance and efficiency [62], [66]. They integrate multiple chips on a single substrate, creating a tightly-coupled and highly interconnected system [3], [36], enabling efficient communication and parallel processing [88]. High-bandwidth data exchange between chips minimizes communication bottlenecks, reducing training times [62]. MCMs also offer benefits like reduced costs and scalability [62], significantly accelerating deep learning tasks.

III. Motivation

In this section, we provide the motivation for our research work by analyzing the feasibility of data parallelism in MCMs and highlighting the limitations of existing AllReduce algorithms for mesh networks.

A. *Feasibility of Data Parallel Training in MCMs*

As MCMs have not been explored for data parallel training of DNN models, here we analyze the feasibility of data parallel training for both small and large DNN models with current MCM technologies. MCMs, primarily used for model parallelism in inference, have seen recent advancements. Simba [62] and Zimmer et al. [89] both present a 36-chiplet system in a 6×6 mesh, while Tangram [17] has a 16×16 chip-engine, where each engine similar to the Eyeriss [11] accelerator. Despite challenges in increasing chiplet numbers, research pushes boundaries: SPRINT [43] uses 64 chiplets with 64 Processing Elements(PEs) each, where each PE resembles Simba, and SPACX [44] uses 32 chiplets with a vector MAC width of 32. This progression suggests the imminent evolution of larger MCMs with more PEs and MAC units.

Despite the new advancements of MCM-based design, the scale of data parallel training for large DNN models in MCMs are limited due to the constrained memory and processing capabilities of individual chiplets. As the size of recent DNN models ranges from megabytes to gigabytes, fitting the entire model into a single chiplet isn't feasible. To reduce the memory footprint of large models, different types of technologies are increasingly adopted. For inference, 2-4 bit precision is common [77], while training often uses 8-bit precision [67], [79], a drop from the usual 16bit or 32bit. Leveraging sparsity also decrease memory in large DNNs [23], and model compression techniques also help shrink model sizes [26]. Network Pruning [25] and Deep Compression [24] are two more compression methods capable of reducing the size of the AlexNet model from its original 240 MB to 27 MB (9x) and 6.9 MB (35x), respectively [33].

Although storing a full copy of a large DNN model into a single chiplet is not feasible, smaller DNN models designed for embedded systems and IoT devices can use MCMs for data parallel training. The recent popularity of small DNN models in embedded systems stems from meeting three critical requirements: fast distributed data parallel training, fitting conveniently within the memory of a single-chip for enhanced power efficiency, and facilitating easy over-the-air updates. SqueezeNet [33] is a compact DNN model that achieves a similar Top-1 accuracy as AlexNet but with a significantly reduced model size of 4.8 MB($50\times$) without any compression and 0.47 MB($510\times$) with Deep Compression compared to original AlexNet model. Similar small-scale models like MobileNet [30], MnasNet [72], and PROXYLESSNAS [10] typically cover around 5 MB in weight size without compression. These compact models find effective application in tasks like character recognition [82] and semantic segmentation [50]. Using SPRINT's model [43] with 32KiB weight buffer per PE and 64 PEs per chiplet, a chiplet can store up to 1MB weights surpassing the compressed model size of SqueezeNet. This suggests that, MCMs can effectively facilitate data parallel training of small DNN models.

Additionally, to address the challenge posed by large DNN models, an alternative approach involves training them layer by layer rather than storing the entire model in memory. Typically, the weight size of the largest layers in DNN models ranges from 576KB to 5MB, considering an 8-bit precision. With each chiplet capable of accommodating up to 1MB of weights, it's viable for the largest layers in models like Transformer, AlphaGoZero, and GoogLeNet to comfortably fit within a single chiplet. Given the trend of increasing PE count in MCMs, storing larger layers in a chiplet will be feasible, highlighting the potential for data parallel training in MCMs in near future.

B. Unavailability of Bidirectional Ring AllReduce for Odd-sized mesh

To leverage bidirectional interconnects, Bidirectional Ring AllReduce uses two unidirectional rings in opposite directions for collective communication. As it effectively doubles the bandwidth usage, Bidirectional Ring AllReduce is always a better choice than Unidirectional Ring AllReduce when bidirectional ring is available. The NVIDIA NCCL [48] library provides a notable example of this approach, generating 4 and 6 unidirectional rings from the 2 and 3 bidirectional rings available in DGX-1 systems [1] equipped with P100 and V100 GPUs, respectively [35]. The DGX-1 system's GPU connectivity is designed to get the benefits of Bidirectional Ring AllReduce, which is a popular and widely used approach due to its simplicity and efficiency.

However, forming a bidirectional ring is not trivial and it depends on the underlying physical topology. For instance, finding a bidirectional ring in topologies like Torus, which possess wrap-around links, is relatively straightforward. However, in topologies such as mesh, where no wrap-around links are available, locating a bidirectional ring is not always feasible. The challenge in finding a bidirectional ring and identifying a Hamiltonian cycle in a topology are similar. As in a odd-sized mesh, there is no Hamiltonian cycle available [15], it is impossible to form a bidirectional ring using all the nodes. Given the widespread use of Ring AllReduce and the increased bandwidth utilization offered by bidirectional rings over unidirectional ones, it becomes imperative to devise a Bidirectional Ring AllReduce solution specifically catering to odd-sized mesh topologies, ensuring the effective application of Bidirectional Ring AllReduce across all types of topologies.

C. Link Underutilization of Existing AllReduce Algorithms

TABLE I: Used Link Percentage for Different AllReduce Algorithms in mesh Topology

Algorithm	Even-sized mesh		Odd-sized mesh	
	Applicability	Used Link Percentage	Applicability	Used Link Percentage
Unidirectional Ring	Easy	29%	Easy	28%
Bidirectional Ring	Easy	57%	Inapplicable	-
Ring-2D	Hard	55%	Hard	53%
DBTree	Hard	-	Hard	-
HDRM	Inapplicable	-	Inapplicable	-
MultiTree	Easy	53%	Easy	51%

Table I presents a comprehensive evaluation of different AllReduce algorithms concerning their applicability to both even and odd-sized mesh topologies. The default topologies used for link usage computation are 8×8 for even-sized mesh and 9×9 for odd-sized mesh, with similar trends observed for larger and smaller topologies. Hierarchical Ring-2D is found to be unsuitable for mesh due to the complexities of forming a ring in each dimension and the latency imposed by the slowest link, typically the connection between two far-end nodes. Similarly, DBTree's topology-oblivious nature and poorly mapped trees in the physical topology hinder its applicability. HDRM, designed solely for Bigraph, is not suitable for mesh topologies. Unidirectional ring AllReduce applies to all mesh configurations but exhibits low link usage. Bidirectional Ring AllReduce offers increased link usage; however, it is limited to even-sized mesh only. MultiTree, while compatible with all topologies, still achieves only around 50% link utilization. This limitation arises from mesh's lack of wrap-around connections like Torus, leading to a higher number of timesteps required to construct all trees, thereby degrading link usage. Consequently, the development of an algorithm specifically tailored for mesh with higher link utilization is of paramount importance.

IV. RingBiOdd : Bidirectional Ring AllReduce for odd-sized mesh

In this section, we first discuss how to find a bidirectional ring in an odd-sized mesh and then proceed to the algorithm that generates the AllReduce schedules.

A. *Forming a Bidirectional Ring for Odd-sized mesh*

For odd-sized meshes, a bidirectional ring that includes all nodes cannot be formed. To address this in an $m \times n$ mesh where both m and n are odd, we exclude one node and utilize $mn - 1$ nodes to create a bidirectional ring. It is always possible to find a Hamiltonian cycle in linear time in an $m \times n$ odd-sized mesh whose one corner node has been excluded [39]. Figure 2 illustrates an example of bidirectional ring formation in 3x3 and 3x5 meshes where a corner is excluded(marked as yellow). We then schedule the AllReduce process in parallel using two unidirectional rings (marked as blue and red arrows). To transfer the data of the excluded node to the nodes in the formed ring, we exploit any two available bidirectional connections of the excluded node (marked in purple in Figure 2) with its neighbors.

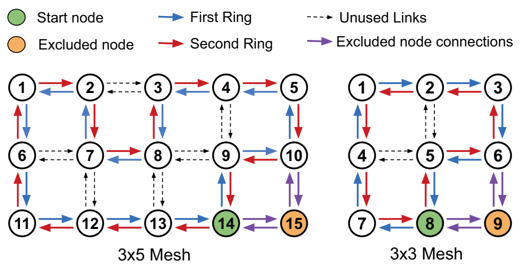


Fig. 2: Bidirectional ring formation for odd-sized mesh by excluding one corner node. To schedule data of excluded node for reduction and broadcast in both rings, purple links are used.

B. *AllReduce Schedule for RingBiOdd*

Since we are creating a ring with $N-1$ nodes in an N-node system, we need to carefully schedule the ReduceScatter and AllGather processes using the given Algorithm 1 to handle data aggregation and broadcast from the excluded node. The algorithm starts by forming a Hamiltonian cycle with all the nodes excluding a corner node (Lines 1-2). Every node within the network divides its gradient data into two halves, corresponding to each direction in the bidirectional ring (Line 3). These halves are then further partitioned into $(m \times n - 1)$ parts, representing the number of nodes in the ring (Line 4).

From timestep 1 to timestep $m \times n - 2$, the excluded node transmits its gradients to the adjacent neighbors using two

Algorithm 1 Finding AllReduce Schedule for Odd-sized mesh

```
Input: mesh topology of size $m \times n$
Output: $reduce\_scatter\_schedule$, $all\_gather\_schedule$
 1: $G$ = {set of nodes in $m \times n$} - {a corner node}
 2: $R$ = FormHamiltonianCycle($G$)
 3: Divide the data in each node into two halves, one for the rings
    in each direction.
 4: Divide each halve further into $(mn-1)$ parts for AllReduce com-
    munication.
                                                        ▷ Timestep 1
 5: Add connections from excluded node to two neighbors to
    $reduce\_scatter\_schedule$ and send one part to each neighbor.
                                       ▷ Timestep 2 to Timestep $(mn-1)$
 6: Add Bidirectional Ring ReduceScatter schedule for $R$ to
    $reduce\_scatter\_schedule$.
 7: for $timestep = 2, ..., mn-1$ do
 8:     Add connections from excluded node to two neighbors to
        $reduce\_scatter\_schedule$ and send rest of the parts.
 9: end for
10: $all\_gather\_schedule$ = Reverse($reduce\_scatter\_schedule$)
```

bidirectional links(Line 5). Subsequently, the two neighbors merge data from the excluded node and execute two simultaneous Ring AllReduce algorithms in opposite directions along the Hamiltonian cycle, spanning from timestep 2 to timestep $m \times n - 1$(Lines 6-9). In this process, the neighbors of the excluded node aggregate data from both the excluded node itself and its own parent within the ring. All the other nodes that are part of the ring follow the standard Bidirectional Ring AllReduce algorithm during timestep 2 to timestep $m \times n - 1$. The entire process is explained in Figure 3 with an example, demonstrating the ReduceScatter process of RingBiOdd in a 3×3 mesh for a single direction. The AllGather schedule can similarly be derived by reversing the order of data transfer during ReduceScatter in the algorithm.

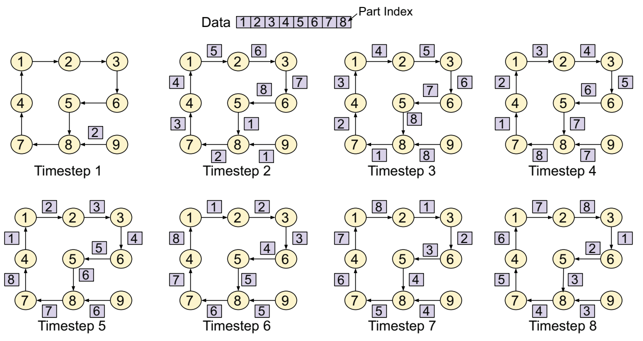


Fig. 3: The ReduceScatter steps using RingBiOdd in a 3×3 mesh. Node indices are represented by numbers in circles, and data part indices are shown in rectangles.

To begin, we establish a bidirectional ring comprising nodes (1, 2, 3, 6, 5, 8, 7, 4). With 8 nodes in the ring, we split the halves of data into 8 parts, i.e., $N-1$ parts, as opposed to the traditional ring-based all reduce, which uses N parts for an N-node system. In this scheme, node 8 serves as the merging node, aggregating data from both its parent (node 5) and the excluded node (node 9) during the process.

The ReduceScatter proceeds as follows: At timestep 1, node 9 sends part 2 to node 8, and node 8 aggregates it. In timestep 2, the ring-based AllReduce starts, with node 8 transmitting the aggregated part 2 from both node 5 and node 9 to node 7. Simultaneously, node 8 receives part 1 from nodes 5 and 9 and aggregates it. This continues for 8 timesteps, after which each node in the ring possesses $1/8$ of fully aggregated data.

For the AllGather stage, a similar ring-based broadcast takes place, utilizing link 8 to 9 to retrieve all the aggregated parts from node 8. This process also requires 8 timesteps. Hence, in a total of 16 timesteps, we can complete AllReduce stages. In comparison to the ring-based AllReduce for even-sized mesh, our proposed algorithm finishes within $2N-2$ timesteps. However, the amount of data transmitted in each timestep now becomes $D/(N-1)$ instead of D/N, where D is the size of data to be aggregated per node. For AllReduce in the opposite direction, we follow the same procedure, but this time node 6 is used as the merging node.

V. TTO: Three Tree AllReduce

In this section, we introduce TTO, a tree-based AllReduce algorithm that divides gradients into multiple chunks and schedules their communication in a way that allows multiple chunks to participate in AllReduce concurrently. To achieve this overlapping, which requires thoughtful construction of multiple trees for collective scheduling, we begin by discussing the rationale behind chunk overlapping in a 1D mesh. Subsequently, we provide a detailed explanation of TTO.

A. Overlapping of Chunks in 1D mesh

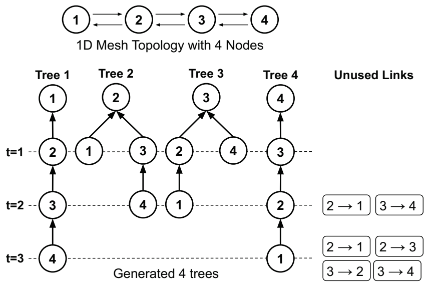


Fig. 4: Tree-based AllReduce in a 1D mesh with 4 nodes, where total 6 links remain unused.

Existing tree-based AllReduce algorithm builds large number of trees and suffers from significant link underutilization. Figure 4 shows how trees are formed in a 1D mesh with 4 nodes using MultiTree algorithm. In this example, 4 trees are created to enable AllReduce communication among the nodes, with each tree aggregating $D/4$ data, where D is the total data size. ReduceScatter(or AllGather) using 4 trees takes 3 timesteps, keeping 6 links unused. To utilize the unused links, we can split the data into multiple chunks and run ReduceScatter operation of different chunks in parallel with unused links. In the paper, by overlapping chunks, we represent parallel reduction or broadcast of multiple chunks.

The issue with building 4 trees is that there is limited opportunity of increasing link utilization by overlapping multiple chunks. Some links, like $(4 \to 3)$ and $(1 \to 2)$, are used in multiple trees across all timesteps, leaving little room for overlap. However, if we build only two trees(tree 1 and 4), we can start overlapping between chunks. For example, when chunk 1 finishes using link $(4 \to 3)$ and $(1 \to 2)$, chunk 2 can immediately start using these links. However, using two trees increases the data sent by each tree to $D/2$, double that of using 4 trees.

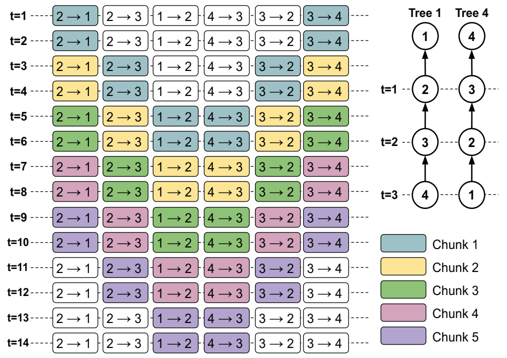


Fig. 5: The overlapping of 5 chunks in a 1D mesh with 4 nodes, where all the chunks use only tree 1 and tree 4 which results in using 2 timesteps for each data transfer. With overlapping, 5 chunks can be finished within 14 timesteps instead of 15 which use all the trees.

Figure 5 illustrates the overlapping of chunks in a 1D mesh with 5 chunks. Using all trees, each chunk needs 3 timesteps to finish the ReduceScatter stage, totaling 15 timesteps for all 5 chunks. However, with only two trees, overlapping becomes possible, and each step of the ReduceScatter stage takes 2 timesteps. As a result, chunk 1 finishes within 6 timesteps, which is 3 more than using all trees. But, due to the availability of overlapping, chunk 2 can start ReduceScatter at timestep 3 and finish within timestep 8, and the same process continues for all chunks. With this approach, ReduceScatter can be finished within 14 timesteps for all 5 chunks, which is less than when using all the trees.

In general, in an N-node system, with N chunks, ReduceScatter using all trees and two trees with corner nodes as roots take the same amount of time. However, after that point, overlapping becomes beneficial, reducing the reduce scatter time for each chunk to $\lceil N/2 \rceil$, instead of $N-1$ when using all trees. As the number of chunks is generally very high, this overlapping approach provides a significant speedup over using all trees. Additionally, all the links remain utilized across the full AllReduce process, resulting in a significantly higher link usage percentage compared to using all trees.

Although overlapping in 1D mesh provides significant performance improvement, MCMs are generally 2D mesh. In a 2D mesh, a hierarchical AllReduce approach can be applied, performing ReduceScatter or AllGather in each dimension with an overlapping approach. However, as overlapping in 1D mesh stores the data in the corner nodes, after the first ReduceScatter of hierarchical AllReduce, all the chunks will be stored in the edge nodes of the mesh. So, during the second ReduceScatter, none of the interior links will be used, leading to significant link underutilization.

To address this, we need an algorithm that can utilize links in a 2D mesh and apply overlapping effectively. The observations from the 1D mesh provide following insights regarding tree formation where chunks can be overlapped:

- To perfectly overlap the chunks, all trees should be disjoint. If there is a common link between trees, it can't be utilized simultaneously, hindering overlapping. For instance, in Figure 4, when attempting to overlap Tree 1 and Tree 2, link $(4 \rightarrow 3)$ is being used by both trees at different timestep which limits overlapping of chunks.
- We need to build the maximum number of disjoint trees possible to minimize data sending in each timestep and maximize link utilization. For 1D mesh, maximum number of possible disjoint tree is always 2. As 2D mesh has more links we should focus on building more disjoint trees.

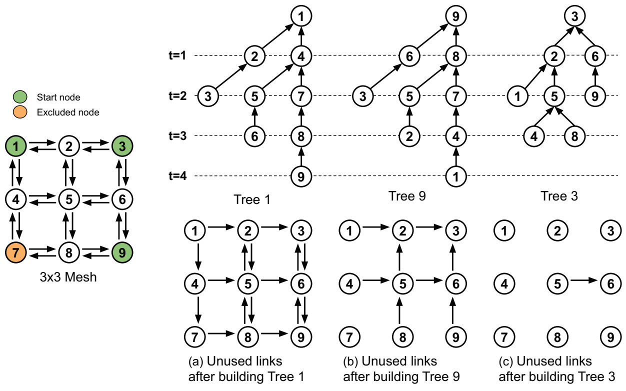
B. Disjoint Tree Formation in 2D mesh

To make better use of the available links and implement an overlapping approach for parallel AllReduce of multiple chunks, we can create multiple disjoint trees using the links in a 2D mesh such that no two trees use a common link and each tree contains all nodes. It is crucial for the trees to be disjoint to avoid contention during broadcast and aggregation of the chunks. However, there are certain considerations to address while forming these disjoint trees for a 2D mesh:

- **Maximum Possible Disjoint Tree is 3:** In a 2D mesh, due to link scarcity, it is not feasible to build more than three disjoint trees. We know that, any node in a mesh topology can have at most 4 outgoing/incoming links. As each node needs to be connected to all of the disjoint trees, we can't build more than 4 disjoint trees by any means. However, the total links available in an $n \times n$ mesh is $4n^2 - 4n$. If we want to form 4 disjoint trees, we need total $4n^2 - 4$ links as each tree requires $n^2 - 1$ links. For any $n > 1$, the required links to form 4 disjoint trees are greater than available links in the topology, which makes forming four disjoint trees in a 2D mesh impossible.
- **Unavailability of 3 Disjoint Trees with All Nodes:** To have three disjoint trees, each node should have at least three outgoing links. However, in a 2D mesh, all corner nodes have at most two outgoing links. One possible workaround can be forming disjoint trees rooted at the corner nodes so that we don't need to connect the corner nodes in their own tree and use available two links to make connections in two other trees. However, even with this approach, when we use three out of the four corner nodes as roots to build disjoint trees, the remaining corner node still needs to send data to three trees, requiring three outgoing links, which is unavailable.

To overcome these challenges, we propose a solution that involves training the DNN with $n^2 - 1$ nodes. Specifically, one corner node will not be used during the training phase, and the other three nodes will form three disjoint trees. By doing this, we can effectively form three trees using all the available links, and since no corner node needs to use three links, the trees can be made disjoint to enable parallel communication of multiple chunks. Although using one less node reduces the mini-batch size during training, leading to increased training time, the gains in AllReduce speedup outweigh this drawback significantly.

C. TTO: Three Tree AllReduce



Start node
Excluded node
3x3 Mesh
t=1
t=2
t=3
t=4
Tree 1
Tree 9
Tree 3
(a) Unused links after building Tree 1
(b) Unused links after building Tree 9
(c) Unused links after building Tree 3

Fig. 6: An example of three disjoint tree formation in a 3×3 mesh. We use node 1, 3 and 9 to form the trees and excluded node 7 for training. Unused links after forming each tree are also shown.

Figure 6 demonstrates the process of forming three disjoint trees in a 2D mesh. Out of the four corner nodes, three nodes(node 1, 3 and 9) are selected as tree roots, while one node is excluded(node 7) from the training. The steps to form the trees are as follows:

1) **Tree rooted at 1:** First, we traverse in y-axis of node 1 to connect all reachable nodes. Here, by traversing in y-axis, we connect 4 with 1 and 7 with 4. Then, we connect nodes that are in the x-axis of nodes 1, 4, and 7, and add them to the tree using the appropriate connections. The unused links after forming the tree rooted at 1 are shown in Figure 6(a). To note here, although we are not using node 7 for training, we are using its available connections to form disjoint trees.
2) **Tree rooted at 9:** The process of forming the tree rooted at 9 is similar to tree rooted at 1. However, here we connect nodes that are in the x-axis first, and then the nodes which are in the y-axis. The unused links after forming this tree are shown in Figure 6(b).
3) **Tree rooted at 3:** For the last tree, we can now consider the topology as an undirected graph and apply Breadth-first search(BFS) starting at node 3. We stop when all

the nodes except the excluded node(node 7) is connected to the graph.

It's important to note that although we illustrate tree formation approach with a 3x3 mesh, this procedure can be applied to meshes of any dimensions. The procedure to form the trees for TTO is outlined in Algorithm 2. In general, for an $n \times m$ mesh, we designate nodes 1, n, and nm as the roots for the three disjoint trees and node $n(m-1)+1$ as the excluded node[4](Line 1). For the tree rooted at node 1, we connect nodes in the $y-axis$ first and then in the $x-axis$ (Lines 5-8). Similarly, for the tree rooted at node nm, connections are made in the $x-axis$ first and then in the $y-axis$ (Lines 5-8). Lastly, for the tree rooted at n, all nodes are connected via BFS traversal starting from n, except for node $n(m-1)+1$ (Line 11). The complexity of forming all the trees is $\mathcal{O}(n)$, and this process ensures that the tree heights are $2n-2$, which is the lowest possible height. Finally, if there are C chunks and tree height is H, TTO requires $H+C-1$ timesteps for the ReduceScatter and AllGather stage.

Algorithm 2 Forming Disjoint Trees for TTO

```
Input: Dimensions of mesh $n$, $m$
Output: Three disjoint trees
 1: $roots = [1, n, nm]$, $trees = []$        ▷ List of root nodes and trees
 2: for each $root \in roots$ do
 3:     $tree = []$                            ▷ List to store tree
 4:     if $root == 1$ or $root == nm$ then
 5:         $t1, t2$ = (y-axis, x-axis) if root is 1, (x-axis, y-axis) otherwise
                                               ▷ Traverse order for node 1 and $nm$
 6:         $tree = [root]$                    ▷ Add $root$ to $tree$
 7:         for $t = t1, t2$ do
 8:             Add all the nodes reachable from $tree$ by adding links that traverse in $t$ direction only
 9:         end for
10:     else
11:         Add all nodes in $tree$ traversed by BFS starting from root.
12:     end if
13:     Add $tree$ in $trees$
14: end for
15: return $trees$
```

To generate the ReduceScatter schedule, we traverse the formed trees bottom-up manner to schedule all nodes for chunk 1. As multiple chunks can overlap, when one node finishes sending data for a chunk, it can immediately start sending the next chunk using the available link. For example, after node 9 completes sending chunk 1 to node 8, it can immediately begin sending chunk 2(Figure 6) and so on. Once the full ReduceScatter schedule for all chunks is obtained, we initiate the AllGather in a similar manner, but the AllGather traversal is done in a top-down fashion of the constructed trees.

Figure 7 shows the improvement achieved by TTO over MultiTree, in a 3×3 mesh with 5 chunks. We can see that, with only 5 chunks, TTO can finish 6 timesteps before MultiTree. This improvement significantly speeds up the AllReduce

[4]Here, we number nodes in a row-major order, starting from 1

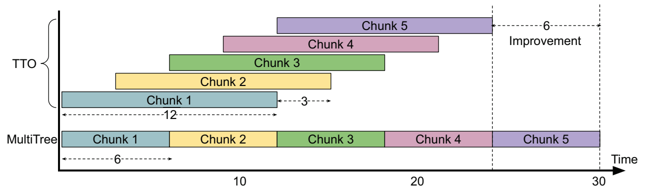


Fig. 7: The improvement of TTO over the MultiTree algorithm in a 3×3 mesh with 5 chunks. MultiTree takes 6 timesteps per chunk, totalling 30 timesteps. In contrast, TTO tree height is 4(Figure 6) and in each step 3 times more data is sent, resulting in 12 timesteps per chunk. But by overlapping, TTO is finished within 24 timesteps with improvement of 6 timesteps.

process, which is particularly advantageous when dealing with a high number of chunks typically found in DNN training.

VI. Methodology

A. Simulation

In our experiments, we utilize SCALE-Sim [58] and BookSim [37] for training time simulation and interconnect modeling, respectively. We extend SCALE-Sim to support both forward and backward propagation of DNN training with output stationary dataflow. Each chiplet in our system consists of 4×4 PEs where each PE has a 256×256 MAC array, unless stated otherwise. These chiplets are used for both training and aggregation during the AllReduce stage. To ensure interaction between the network and chiplets, we implement a Python interface connecting SCALE-Sim and BookSim, which schedules all the benchmarks in same configuration. Double buffering and sufficient memory bandwidth are assumed for each chiplet to achieve the highest compute throughput. The network buffer size is configured to cover the credit round-trip loop, and the network bandwidth is set at 25GBps which is aligned with Simba [62]. Both the chiplet and router clock frequency are configured at 1GHz, and virtual cut-through is employed for flow control within the network. The system configuration parameters are listed in Table II. We run all of our benchmarks with both even and odd-sized mesh, in small scale(16-node and 25-node) and large scale(64-node and 81-node) systems. To show that our proposed schemes are scalable enough, we run a scalability study starting from 9 chiplets to 256 chiplets.

B. Workloads

To show the effectiveness and versatility of our proposed schemes, we conduct a synthetic study of AllReduce bandwidth usage, varying the AllReduce data size from 1MB to 1GB, and evaluate the scalability using a total AllReduce data size of $375KB \times N$, where N is the total number of chiplets in the topology. Additionally, we evaluate the performance of our systems with real-life DNN models from SCALE-Sim, including AlexNet [41], AlphaGoZero [64], Faster-RCNN [19], GoogLeNet [70], NCF-Recommendation(NCF) [28], ResNet152 [27], and Transformer [76]. The evaluation

TABLE II: System Configuration

Parameters		Configurations
Chiplet	Number of PEs	4×4
	Clock Frequency	1GHz
PE	MAC Array	256×256
	Precision	32 bits
	Dataflow	Output Stationary
Network	Bandwidth	25GBps
	Packet Size	8192B
	Flit Size	512B
	Per Flit Latency	21ns
	Router Clock Frequency	1GHz
	Topology	Mesh
	Flow Control	Virtual Cut-Through
	Number of VCs	4
	VC Buffer Depth	318 flits

is performed with a mini-batch size of $16 \times N$ for an N-chiplet system, corresponding to 16 samples per chiplet, and we measure the training time required for one epoch. To ensure a fair comparison, we account for the fact that the TTO algorithm utilizes $N-1$ chiplets for training, while other algorithms use all N chiplets. For this purpose, we consider the full epoch time of a training set. The training set size is set to 1.2 million, similar to ImageNet [41], a leading dataset in computer vision. By calculating the total iterations based on the mini-batch sizes for each scheme, we obtain the training time of one epoch. It is essential to note that our proposed schemes have no impact on the accuracy or convergence of a given neural network, as they do not alter the computation ordering. Additionally, the trade-off between mini-batch size, training time, and model accuracy is beyond the scope of this research. We use default chunk size as $98304B$ for N-chiplet systems in TTO to ensure proper aggregation during ReduceScatter and maintain efficient overlapping.

C. Target Benchmarks

We evaluate both of our schemes with the following state-of-the-art baseline algorithms for mesh topology.

- Ring: Unidirectional Ring AllReduce algorithm [18].
- Ring-2D: Two dimensional Hierarchical Ring AllReduce algorithm [84].
- DBTree: Topology-oblivious Double Binary Tree algorithm [59].
- MultiTree: Tree-based topology-aware MultiTree algorithm [31].
- RingBiEven: Bidirectional Ring AllReduce algorithm for even-sized mesh topology.

Experiments for Ring, Ring-2D, DBTree, MultiTree and TTO are run over all topologies. However, RingBiEven is for even-sized mesh, while RingBiOdd is exclusively applicable to odd-sized mesh topologies.

VII. Results

In this section, we show the performance of our proposed algorithms by evaluating bandwidth utilization, DNN runtimes, link utilization, and scalability across multiple mesh sizes.

A. AllReduce Bandwidth Utilization

To showcase the superiority of our proposed schemes, we establish networks of size 4×4, 5×5, 8×8, and 9×9 in the mesh topology. RingBiOdd, designed for odd-sized meshes, is evaluated on 5×5 and 9×9 networks, while RingBiEven is assessed on 4×4 and 8×8 networks. All other benchmarks are tested on all networks. We vary the data size from 1MB to 1GB, encompassing both small and large AllReduce data scenarios, and record the simulation time. By dividing the AllReduce data size by the simulation time, we obtain the bandwidth usage. While TTO distributes data among N-1 nodes, we concentrate solely on the AllReduce time here, without considering the potential increase in training time due to the usage of 1 less chiplet. In section VII-C, we address this concern by presenting the end-to-end training time for one epoch of DNN training. The results of AllReduce bandwidth utilization are showed in Figure 8.

Among all the benchmarks, TTO consistently outperforms others across all topologies. This advantage arises from the efficient utilization of links achieved by the three disjoint trees, which allows overlapping data chunks and ensures that most links remain occupied during both the ReduceScatter and AllGather stages.

For both even and odd-sized mesh, the Bidirectional Ring AllReduce algorithms, namely RingBiEven and RingBiOdd, consistently outperform all baseline algorithms. RingBiOdd, designed for odd-sized mesh, achieves similar bandwidth usage as RingBiEven, despite sending slightly more data ($D/(N-1)$ in each step) compared to D/N for an N-chiplet-based system. The fact that RingBiOdd maintains the same hop count for the AllReduce algorithm contributes to its comparable performance with RingBiEven. Moreover, both algorithms demonstrate superior performance compared to MultiTree because the bidirectional ring exhibits higher link utilization, giving it an edge over MultiTree.

Among all the topologies, DBTree shows the worst performance due to its topology-oblivious nature and poor mapping of tree nodes to the network, resulting in significant contention. Additionally, Ring AllReduce has limitations due to its unidirectional ring, which prevents it from utilizing bidirectional interconnects. Similarly, Ring-2D performs considerably worse than other benchmarks for two reasons. First, for any hierarchical algorithm, transmitting large data in the ReduceScatter and AllGather stages in the first dimension slows down the AllReduce process. Second, in a 1D mesh, the latency between two nodes of a ring is primarily determined by the slowest pair of nodes, which are typically the farthest nodes in the topology. Overall, DBTree, Ring, and Ring-2D are not suitable AllReduce algorithm for mesh topology.

In general, when compared to Ring, Ring-2D, DBTree, MultiTree, and Bidirectional Ring AllReduce, TTO exhibits an average speedup of $3.2\times$, $2.6\times$, $5.3\times$, $1.6\times$, and $1.4\times$ respectively. On the other hand, RingBiOdd demonstrates an average speedup of $1.9\times$, $1.75\times$, $3.7\times$ and $1.1\times$, compared to Ring, Ring-2D, DBTree and MultiTree.

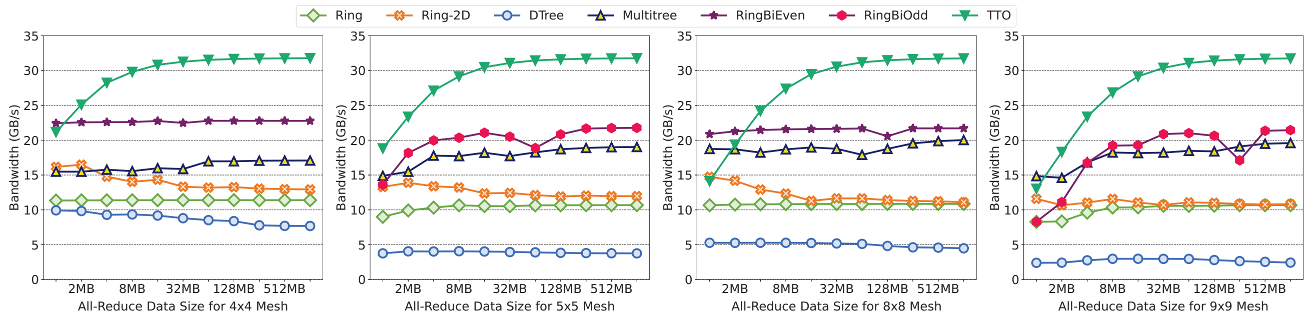


Fig. 8: AllReduce bandwidth with various data size for 4×4, 5×5, 8×8 and 9×9 mesh

B. Scalability Results

In a scalability test with chiplets ranging from 9 to 256, we used AllReduce data sized at $375 \times N$ KB, where N is the chiplet number. The results, shown in Figure 9, use synthetic data, ignoring computation time increase for TTO. The left figure shows even-sized mesh scalability, while the right displays odd-sized mesh. Communication time is normalized to Ring AllReduce's performance in a 4×4 mesh for even-sized and a 3×3 mesh for odd-sized.

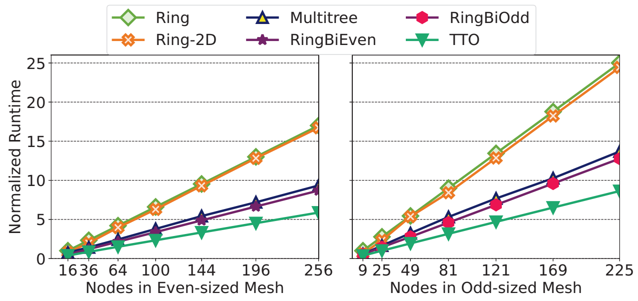


Fig. 9: Scalability with $375 \times N$ KB AllReduce data size normalized to Ring AllReduce of 4×4 mesh and 3×3 mesh for even-sized and odd-sized mesh, respectively, where N is the number of nodes.

Across all topologies, we observe that all algorithms scale linearly with the number of nodes, although they sustain different linear factors, with TTO being the best and Ring AllReduce the worst in terms of performance. The superiority of TTO can be attributed to its efficient network usage and chunk overlapping. The three disjoint trees in TTO utilize the majority of links in any topology, and the parallel movement of chunks fully exploit the network with proper chunk size, thereby preventing performance degradation with increasing chiplets. Additionally, TTO ensures lowest tree heights possible, resulting in the lowest increase in communication hops across all benchmarks as the mesh size increases.

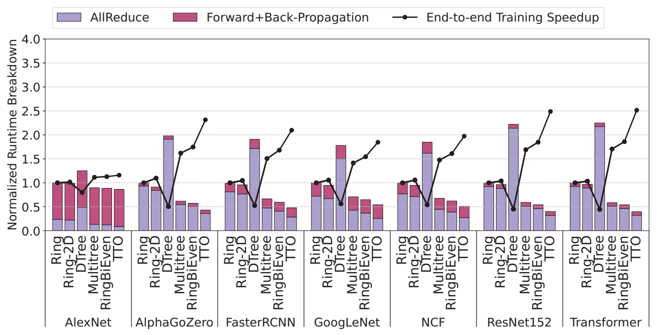
RingBiOdd shows consistent scaling performance across all dimensions, surpassing the Ring, Ring-2D, and MultiTree. Additionally, RingBiOdd follows a similar trend to RingBiEven while scaling out and nearly halves the communication time compared to the Unidirectional Ring. This positions RingBiOdd as an excellent alternative to use when the Bidirectional Ring AllReduce algorithm is needed, and the mesh size is odd.

Ring and Ring-2D scale linearly but perform poorly. Long ring length for unidirectional Ring increases latency and worsens performance with higher dimensions. Ring-2D's performance drops with more chiplets because of the imperfect 1D ring. While MultiTree scales linearly, its longer tree height underperforms compared to TTO and Bidirectional Ring AllReduce. In summary, RingBiOdd achieves around $1.1\times$ average speedup over MultiTree while TTO exceeds all the benchmarks and shows $2.8\times$, $1.6\times$ and $1.4\times$ average speedup over Ring, MultiTree and Bidirectional Ring AllReduce respectively.

C. Model Results

To assess the performance of our schemes under real training workloads, we conduct simulations to measure the training time for one epoch using 8x8 and 9x9 mesh topologies with 7 real-world DNN models. For this experiment, we assume a training data size of 1.2M, similar to ImageNet [41]. To effectively utilize the processing element (PE) performance, we use a mini-batch size of $16 \times N$ for an N-node system. Consequently, the mini-batch sizes for the 8x8 and 9x9 topologies are 1024 and 1296, respectively, for all benchmarks except TTO, which uses 1008 and 1280 as mini-batch sizes due to utilizing one less chiplet for training. To calculate the time for one epoch, we first divide the training data by the mini-batch size to obtain the number of iterations. Next, we multiply the iteration time (training time + AllReduce time for a mini-batch) with the total iterations to finally obtain the total training time for one epoch. As TTO involves one less chiplet for training, it has relatively more iterations than the other benchmarks. The results are presented in Figures 10a and 10b for 8x8 and 9x9 mesh topologies, respectively.

Most of the models in our experiments have a high communication-to-computation ratio. AlexNet is a computation-intensive model but has less parameters which make its training time computation dominant. On the other hand, models like GoogLeNet, ResNet152, which are also computation-intensive but have a higher number of parameters, experience more dominant communication time. Models like


(a) Training time breakdown for 8×8 mesh

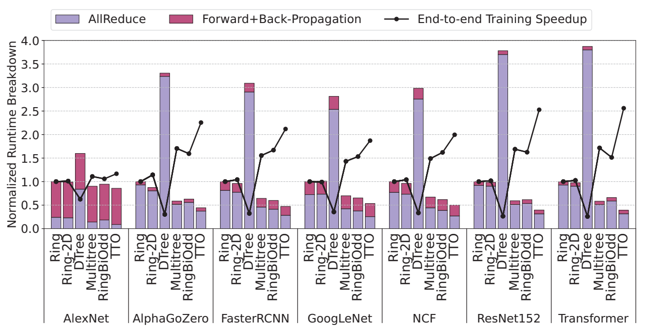

(b) Training time breakdown for 9×9 mesh

Fig. 10: Training time breakdown for one epoch over popular DNN models on 8×8 and 9×9 mesh. AllReduce, Forward+Back-Propagation and End-to-end training speedup normalized to Ring AllReduce results.

NCF_Recommendation and Transformer, with embedding and attention layers, are naturally communication-intensive.

Among the proposed schemes, TTO demonstrates the highest speedup compared to all benchmarks. On average, TTO achieves average speedups of $1.04-1.5\times$ and $1.1-1.69\times$ for all models over MultiTree and Bidirectional Ring AllReduce, respectively. Although TTO requires more iterations to complete a single epoch, its significant AllReduce speedup outweighs the computation time increase, resulting in substantial training time reduction. With larger topologies, the impact of sacrificing one chiplet is reduced, giving TTO more speedup over all the benchmarks.

Finally, RingBiOdd achieves average speedups of $1.29-1.97\times$, $1.24-1.9\times$, $4.1-7.3\times$, $0.88-1.09\times$ over Ring, Ring-2D, DTree, and MultiTree, respectively. With larger topologies, MultiTree creates more balanced trees, giving it a slight advantage over Bidirectional Ring AllReduce in some cases. Overall, our proposed schemes show more prominent results when the communication is dominant, as seen in models like Transformer, ResNet152, NCF, etc. In all cases, DBTree performs poorly among all benchmarks due to its poor mapping of tree nodes with actual physical topology. Unidirectional Ring and Ring-2D also perform poorly due to the long latency of the formed ring and unavailability of perfect ring in 1D mesh.

To examine the effect of overlapping computation and communication on reducing AllReduce overhead, we experiment with a layer-wise AllReduce approach shown in Figure 11. For compute-intensive workloads like AlexNet, FasterRCNN and GoogLeNet, there is substantial potential for overlap between computation and AllReduce, effectively reducing the communication bottleneck. However, in communication-intensive networks like NCF and Transformer with extensive embedding and attention layers, only minimal overlap occurs, making communication a bottleneck. Overall, with overlap, TTO achieves speedups ranging from $0.99-2.5\times$ over Ring, $0.99-2.35\times$ over Ring-2D, $1.03-5.3\times$ over DTree, $0.98-1.54\times$ over MultiTree, and $0.99-1.6\times$ over RingBiEven.

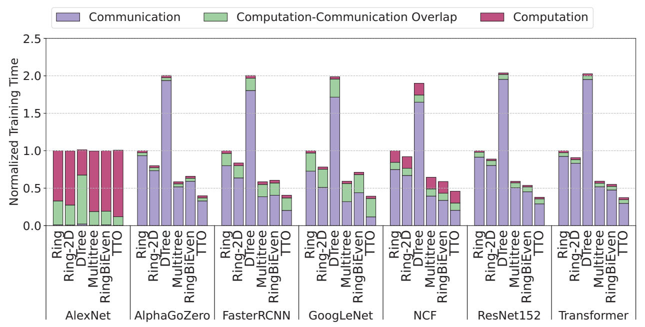


Fig. 11: Training time breakdown normalized to Ring AllReduce using overlapped training approach with layer-wise AllReduce on 8×8 mesh and configurations in II.

D. Link Utilization

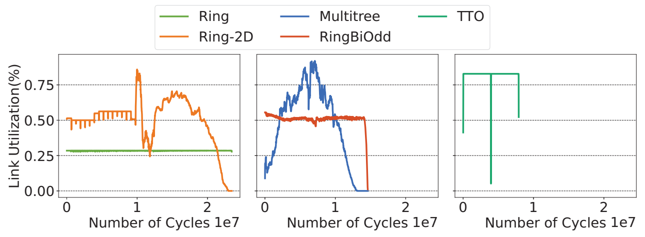


Fig. 12: Link utilization percentage of all benchmarks in a 9×9 mesh with 256MB AllReduce data.

In Figure 12, we present an analysis of the link utilization percentage for various benchmarks in a 9×9 mesh with 256MB of AllReduce data. We can see that, TTO consistently maintains around 83% link utilization throughout the entire AllReduce phase. Due to the overlapping of chunks, all the links used in the trees are always utilized, which results in high link utilization rate.

We can see that, RingBiOdd achieves about 57% link utilization, comparable to Bidirectional Ring AllReduce in even-sized meshes, while Ring AllReduce only reaches 30%.

Although MultiTree shows peak utilization at 85% due to its simultaneous 64-tree usage for some part of the execution, but averages 55-60%. Ring-2D has higher link usage but depicts extended AllReduce time due to more data transmission and lack of a perfect ring in 1D mesh. Overall, TTO achieves 20-50% more link utilization across all the benchmarks.

VIII. Discussions

A. Performance of TTO in an MCM-based Accelerator

In this section, we evaluate TTO using the MCM-based Simba accelerator, featuring 36 chiplets in a 6×6 mesh. Each chiplet contains 16 PEs with an 8×8 MAC array. We present results for 16×16 and 32×32 MAC arrays instead of 8×8, as smaller ones is not a good fit for data parallel training due to higher computation requirements. Recent studies like SPRINT [43] and SPACX [44] further support the potential of larger MAC arrays and more PEs in MCMs.

From Figure 13, we can see that as the MAC array size decreases, computation dominates over communication time, which reduces the end to end speedup. As the model size remains consistent regardless of the MAC array size, TTO achieves similar AllReduce speedup across different MAC array sizes. For both 32×32 and 16×16 MAC array sizes, TTO achieves AllReduce speedups of approximately $2.76\times$, $2.52\times$, $9.7\times$, $1.64\times$, and $1.41\times$ over Ring, Ring-2D, DBTree, MultiTree, and RingBiEven respectively.

However, smaller MAC arrays considerably increase the time needed for both forward and backward propagation, which then takes up a significant portion of the overall training duration. As a result, the end-to-end training speedup for one iteration with 32×32 and 16×16 MAC arrays range between $0.97 - 1.8\times$, $0.97 - 1.73\times$, $0.98 - 5.56\times$, $0.97 - 1.31\times$, and $0.97 - 1.17\times$ over Ring, Ring-2D, DBTree, MultiTree, and RingBiEven respectively, whereas smaller MAC arrays show comparatively less speedup. There's a slight reduction in overall speedup for compute-intensive models like AlexNet, FasterRCNN, GoogleNet, and AlphaGoZero due to the use of one less chiplet in TTO. In general, TTO delivers the most significant performance boost for models with higher communication to computation ratio such as NCF_Recommendation and Transformer.

B. Overhead of Using One Less Chiplet for Training

To assess the impact of using one less chiplet for TTO, we employ the following equations to calculate its performance gain in an N-chiplet system.

$$I\_base, I\_tto = \lceil \frac{D}{(N*d)} \rceil, \lceil \frac{D}{((N-1)*d)} \rceil \tag{1}$$

$$gain = I\_base * (T + C\_b) - I\_tto * (T + C\_t) \tag{2}$$

Let D represent the training dataset size, d be the number of data processed per chiplet (assuming equal distribution of data in each chiplet), and T represent the Forward and Back-Propagation time for a single chiplet. I_base is the number of iterations for one epoch in any baseline AllReduce algorithm using all chiplets, and I_tto is the number of iterations for one epoch in TTO. Additionally, C_b and C_t represent the AllReduce time for the base algorithm and TTO, respectively.

To demonstrate the superiority of TTO, even with one less chiplet, we compare it with base algorithm RingBiEven and explore the overhead of using one less chiplet. Our analysis uses Resnet152 with an 8×8 mesh topology, which has total 64 chiplets($N = 64$). The reference dataset is ImageNet with 1281167 training image($D = 1281167$). The mini-batch sizes($N * d$) for RingBiEven and TTO are 1024 and 1008, respectively, using $16 \times N$ for the mini-batch size.

From equation 1, the total iterations for RingBiEven and TTO are 1252 and 1271, respectively. TTO requires an overhead of 19 additional iterations compared to RingBiEven. From our experiments, we obtain $T = 1832399$, $C_b = 10350425$, and $C\_t = 7076228$. Plugging these values into equation 2, we find a total performance gain of 3930030731, i.e., 44% improvement. The overlapping of chunks in 3 disjoint trees in TTO results in a significant AllReduce speedup and outweighs the overhead due to increased iterations, overall leads to superior end-to-end training performance. Moreover, as the mesh size increases, the overhead of not using one chiplet decreases, making TTO an excellent AllReduce algorithm for communication-intensive and large-sized mesh.

However, TTO might not exhibit significant speedup over baseline algorithms in certain cases:

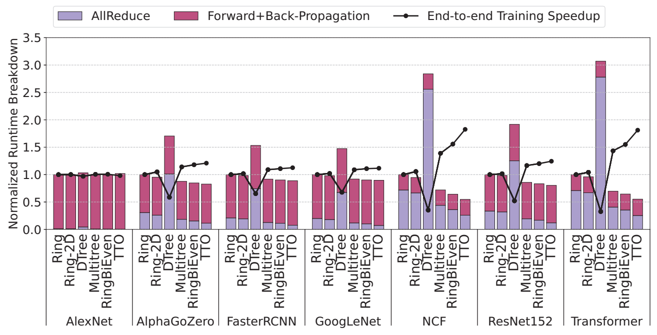
1) **In very small mesh topologies:** The iteration count is inversely proportional to the number of chiplets in the system. In small meshes, excluding one chiplet increases iteration overhead considerably compared to larger meshes, potentially limiting TTO's performance.
2) **For Computation-intensive models:** Usage of one less chiplet also raises training time overhead. For computation-intensive DNN models, it raises training time significantly, offsetting the AllReduce speedup.
3) **Small MAC array in PE:** If the underlying hardware in each PE has a smaller number of MAC arrays, it could increase training time and make training computation-bound, negatively affecting TTO's performance.

C. Optimal Chunk Size for TTO

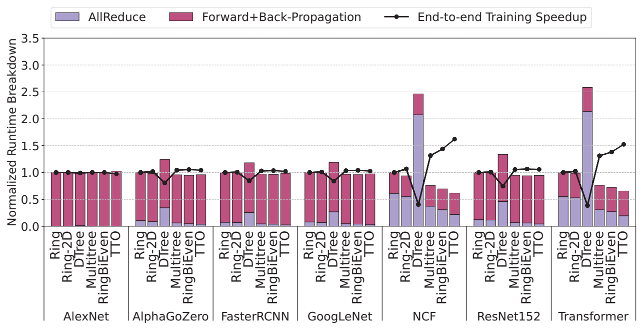
In Figure 14, we demonstrate the impact of different chunk sizes on bandwidth usage in an 8×8 mesh with 128MB of data by varying chunk size from 12KB to 6MB. TTO achieves higher bandwidth usage when the chunk size is relatively small, specifically in the range of 96KB to 192KB. Since the packet size is 8KB, these chunk sizes allow for efficient utilization of links without packet fragmentation and facilitate efficient aggregation in PEs. During AllReduce, the bandwidth usage for TTO remains consistently high due to continuous overlapping of other chunks. When chunk size increases, the total number of chunks decreases, resulting in reduced overlap opportunity, ultimately leading to performance degradation.

IX. Related Works

Numerous studies have been conducted on collective communications in high-performance computing (HPC) with a fo-



(a) Training time breakdown with 32 × 32 MAC array



(b) Training time breakdown with 16 × 16 MAC array

Fig. 13: Training time breakdown for one epoch over popular DNN models on Simba with 6×6 mesh. AllReduce, Forward+Back-Propagation and End-to-end training speedup normalized to Ring AllReduce results.

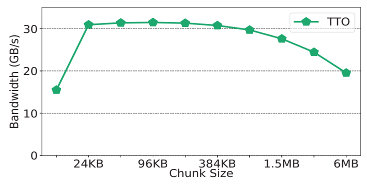


Fig. 14: Impact of different chunk sizes over TTO in an 8×8 mesh with 128MB of data.

cus on the MPI communication interface [16]. Various implementations of collective communication in the MPI interface [7], [52] have been proposed in the HPC domain, optimized for different message sizes [74]. While these algorithms remain applicable to deep-learning workloads, collectives in HPC typically involve small message sizes (few KB-MB), unlike deep learning workloads that deal with larger messages (10s of MB-GB) that lie in the critical path [40].

Numerous AllReduce algorithms, such as ring and DBTree, have been implemented in different communication libraries like NCCL [48] and Horovod [60]. Many works also consider the physical topology hierarchy to perform localized reduction/broadcast in each network hierarchy [12], [46], [56], [78]. For instance, Blink [78] generates efficient collective algorithms based on the underlying network topology using a spanning tree packing approach. Themis [56] proposes an AllReduce algorithm for heterogeneous systems through dynamic chunk scheduling. BlueConnect [12] is a communication library for distributed DL on GPU-based platforms and breaks down a single AllReduce operation into parallelizable ReduceScatter and AllGather operations. PLink [46] is a collective scheme designed to handle heterogeneous networks and variable performance of public cloud platforms.

However, most of these works primarily focus on off-chip interconnects, such as connections between GPUs or target heterogeneous networks. In contrast, our proposed schemes are intended for MCMs, where chiplets are connected by on-chip networks, and the underlying topology is a mesh with restricted connections between chiplets. Notably, TTO is the first method to propose overlapping chunks for maximum bandwidth utilization in a mesh topology.

C-Cube [13], a recently proposed AllReduce algorithm, improves the performance of the DBTree implemented in NCCL by overlapping ReduceScatter and AllGather phases using extra connections in the Nvidia DGX-1 [1] system. However, that approach is tailored for the Nvidia DGX-1, and the extra connections they rely on are not available in a mesh topology like ours.

X. Conclusion

This paper proposes RingBiOdd and TTO, two algorithms addressing inefficiency in existing AllReduce approaches for mesh topology in MCMs. RingBiOdd enables Bidirectional Ring AllReduce for odd-sized meshes, while TTO employs a tree-based approach to overlap data chunks and improve network utilization. Compared to unidirectional Ring AllReduce and MultiTree, RingBiOdd achieves 50% and 8% training time reduction, respectively, while TTO outperforms MultiTree and Bidirectional Ring algorithms, with 33% and 29% training time reduction for seven DNN models, respectively.

References

[1] (2019) Nvidia dgx platform. https://www.nvidia.com/en-us/data-center/dgx-platform/.
[2] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng, “Optimization of mpi collective communication on bluegene/l systems,” in *Proceedings of the 19th Annual International Conference on Supercomputing*, ser. ICS ’05. New York, NY, USA: Association for Computing Machinery, 2005, p. 253–262. [Online]. Available: https://doi.org/10.1145/1088149.1088183
[3] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, “Mcm-gpu: Multi-chip-module gpus for continued performance scalability,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 320–332.
[4] G. Ascia, V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, “Improving inference latency and energy of dnns through wireless enabled multi-chip-module-based architectures and model parameters compression,” in *2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2020, pp. 1–6.
[5] T. Ben-Nun and T. Hoefler, “Demystifying parallel and distributed deep learning: An in-depth concurrency analysis,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[6] M. Bojarski, P. Yeres, A. Choromańska, K. Choromanski, B. Firner, L. D. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *ArXiv*, vol. abs/1704.07911, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:2889646

[7] S. Bokhari and H. Berryman, "Complete exchange on a circuit switched mesh," in *Proceedings Scalable High Performance Computing Conference SHPCC-92.*, 1992, pp. 300–306.

[8] 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," 2020.

[9] T. Burd, N. Beck, S. White, M. Paraschou, N. Kalyanasundharam, G. Donley, A. Smith, L. Hewitt, and S. Naffziger, ""zeppelin": An soc for multichip architectures," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 133–143, 2019.

[10] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," 2019.

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

[12] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter, "Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy," *IBM Journal of Research and Development*, vol. 63, no. 6, pp. 1:1–1:11, 2019.

[13] S. Cho, H. Son, and J. Kim, "Logical/physical topology-aware collective communication in deep learning training," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 56–68.

[14] 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 *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 610–622.

[15] S. Dong Chen, H. Shen, and R. Topor, "An efficient algorithm for constructing hamiltonian paths in meshes," *Parallel Computing*, vol. 28, no. 9, pp. 1293–1305, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819102001357

[16] M. P. I. Forum. (2021) Mpi: A message-passing interface standard. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf.

[17] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–820. [Online]. Available: https://doi.org/10.1145/3297858.3304014

[18] A. Gibiansky and J. Hestness. (2017) baidu-research/tensorflow-allreduce. https://github.com/baidu-research/tensorflow-allreduce.

[19] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[21] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," 2018.

[22] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, apr 2020. [Online]. Available: https://doi.org/10.1002%2Frob.21918

[23] C. Guo, B. Y. Hsueh, J. Leng, Y. Qiu, Y. Guan, Z. Wang, X. Jia, X. Li, M. Guo, and Y. Zhu, "Accelerating sparse dnn models without hardware-support via tile-wise sparsity," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–15.

[24] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1510.00149

[25] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf

[26] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 1135–1143.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[28] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," 2017.

[29] T. Hoefler, T. Schneider, and A. Lumsdaine, "Loggopsim: Simulating large-scale applications in the loggops model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 597–604. [Online]. Available: https://doi.org/10.1145/1851476.1851564

[30] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[31] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum, and E. J. Kim, "Communication algorithm-architecture co-design for distributed deep learning," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 181–194.

[32] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA '20. IEEE Press, 2020, p. 968–981. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00083

[33] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," 2016.

[34] N. Jain and Y. Sabharwal, "Optimal bucket algorithms for large mpi collectives on torus interconnects," in *Proceedings of the 24th ACM International Conference on Supercomputing*, ser. ICS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 27–36. [Online]. Available: https://doi.org/10.1145/1810085.1810093

[35] S. Jeaugey. (2017) Nccl 2.0. https://on-demand.gputechconf.com/gtc/2017/presentation/s7155-jeaugey-nccl.pdf.

[36] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free?" in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 458–470.

[37] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," pp. 86–96, 2013.

[38] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 546–558.

[39] F. Keshavarz-Kohjerdi and A. Bagheri, "A linear-time algorithm for finding hamiltonian (s,t)-paths in even-sized rectangular grid graphs with a rectangular hole," *Theoretical Computer Science*, vol. 690, pp. 26–58, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397517304759

[40] 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)*, 2020, pp. 996–1009.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[42] Y. Li, A. Louri, and A. Karanth, "Scaling deep-learning inference with chiplet-based architecture and photonic interconnects," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 931–936.

[43] Y. Li, A. Louri, and A. Karanth, “Scaling deep-learning inference with chiplet-based architecture and photonic interconnects,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 931–936.

[44] Y. Li, A. Louri, and A. Karanth, “Spacx: Silicon photonics-based scalable chiplet accelerator for dnn inference,” in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 831–845.

[45] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” 2015.

[46] L. Luo, P. West, A. Krishnamurthy, L. Ceze, and J. Nelson, “Plink: Discovering and exploiting datacenter network locality for efficient cloud-based distributed training,” 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218980024

[47] S. A. Mojumder, M. S. Louis, Y. Sun, A. K. Ziabari, J. L. Abellàn, 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.

[48] NVIDIA. Nvidia collective communications library (nccl). https://developer.nvidia.com/nccl.

[49] J. Park, H. Kwon, S. Kim, J. Lee, M. Ha, E. Lim, M. Imani, and Y. Kim, “Quiltnet: Efficient deep learning inference on multi-chip accelerators using model partitioning,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1159–1164. [Online]. Available: https://doi.org/10.1145/3489517.3530589

[50] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, “Enet: A deep neural network architecture for real-time semantic segmentation,” 2016.

[51] P. Patarasuk and X. Yuan, “Bandwidth optimal all-reduce algorithms for clusters of workstations,” *J. Parallel Distrib. Comput.*, vol. 69, no. 2, p. 117–124, feb 2009. [Online]. Available: https://doi.org/10.1016/j.jpdc.2008.09.002

[52] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. Fagg, E. Gabriel, and J. Dongarra, “Performance analysis of mpi collective operations,” in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 8 pp.–.

[53] K. E. Prikopa, W. N. Gansterer, and E. Wimmer, “Parallel iterative refinement linear least squares solvers based on all-reduce operations,” *Parallel Computing*, vol. 57, pp. 167–184, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819116300473

[54] R. Rabenseifner, “Automatic profiling of mpi applications with hardware performance counters,” in *PVM/MPI*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:12865475

[55] R. Rabenseifner, “Optimization of collective reduction operations,” in *International Conference on Conceptual Structures*, 2004. [Online]. Available: https://api.semanticscholar.org/CorpusID:8174425

[56] S. Rashidi, W. Won, S. Srinivasan, S. Sridharan, and T. Krishna, “Themis: A network bandwidth-aware collective scheduling policy for distributed training of dl models,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 581–596. [Online]. Available: https://doi.org/10.1145/3470496.3527382

[57] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” 2016.

[58] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, “Scale-sim: Systolic cnn accelerator simulator,” 2019.

[59] 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, selected papers from the 14th European PVM/MPI Users Group Meeting. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819109000957

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

[61] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, “Measuring the effects of data parallelism on neural network training,” *arXiv preprint arXiv:1811.03600*, 2018.

[62] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, “Simba: Scaling deep-learning inference with multi-chip-module-based architecture,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’52. New York, NY, USA: Association for Computing Machinery, 2019, p. 14–27. [Online]. Available: https://doi.org/10.1145/3352460.3358302

[63] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman, “Mesh-tensorflow: Deep learning for supercomputers,” *CoRR*, vol. abs/1811.02084, 2018. [Online]. Available: http://arxiv.org/abs/1811.02084

[64] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, pp. 354–, Oct. 2017. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[65] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[66] L. T. Su, S. Naffziger, and M. Papermaster, “Multi-chip technologies to unleash computing performance gains over the next decade,” in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 1.1.1–1.1.8.

[67] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, *Hybrid 8-Bit Floating Point (HFP8) Training and Inference for Deep Neural Networks*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[68] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” 2014.

[69] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014.

[70] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[71] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” 2015.

[72] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” 2019.

[73] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, “Communication-efficient distributed deep learning: A comprehensive survey,” *arXiv preprint arXiv:2003.06307*, 2020.

[74] 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. [Online]. Available: https://doi.org/10.1177/1094342005051521

[75] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2016.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.

[77] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, H. Tran, Y. Li, E. Ogawa, K. Ishizaki, H. Inoue, M. Schaal, M. Serrano, J. Choi, X. Sun, N. Wang, C.-Y. Chen, A. Allain, J. Bonano, N. Cao, R. Casatuta, M. Cohen, B. Fleischer, M. Guillorn, H. Haynie, J. Jung, M. Kang, K.-h. Kim, S. Koswatta, S. Lee, M. Lutz, S. Mueller, J. Oh, A. Ranjan, Z. Ren, S. Rider, K. Schelm, M. Scheuermann, J. Silberman, J. Yang, V. Zalani, X. Zhang, C. Zhou, M. Ziegler, V. Shah, M. Ohara, P.-F. Lu, B. Curran, S. Shukla, L. Chang, and K. Gopalakrishnan, “Rapid: Ai accelerator for ultra-low precision training and inference,” in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 153–166.

[78] G. Wang, S. Venkataraman, A. Phanishayee, J. Thelin, N. Devanur, and I. Stoica, “Blink: Fast and generic collectives for distributed ml,” 2019.

[79] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, “Training deep neural networks with 8-bit floating point numbers,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS’18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 7686–7695.

[80] Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang, “Plda: Parallel latent dirichlet allocation for large-scale applications,” in *Algorithmic Aspects in Information and Management*, A. V. Goldberg and Y. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 301–314.

[81] J. M. Wilson, W. J. Turner, J. W. Poulton, B. Zimmer, X. Chen, S. S. Kudva, S. Song, S. G. Tell, N. Nedovic, W. Zhao, S. R. Sudhakaran, C. T. Gray, and W. J. Dally, “A 1.17pj/b 25gb/s/pin

ground-referenced single-ended serial link for off- and on-package communication in 16nm cmos using a process- and temperature-adaptive voltage regulator," *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 276–278, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:3872872
[82] X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, and T. Chang, "Building fast and compact convolutional neural networks for offline handwritten chinese character recognition," 2017.
[83] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh, "Modular routing design for chiplet-based systems," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 726–738.
[84] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," 2018.
[85] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," 2017.
[86] H. Zhang, Y. Li, Z. Deng, X. Liang, L. Carin, and E. Xing, "Autosync: Learning to synchronize for data-parallel distributed deep learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 906–917, 2020.
[87] H. Zhao and J. Canny, "Kylix: A sparse allreduce for commodity clusters," in *2014 43rd International Conference on Parallel Processing*, 2014, pp. 273–282.
[88] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany, "A 0.11 pj/op, 0.32-128 tops, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16nm," in *2019 Symposium on VLSI Circuits*, 2019, pp. C300–C301.
[89] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. S. Emer, C. T. Gray, S. W. Keckler, and B. Khailany, "A 0.32–128 tops, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, 2020.

Appendix

A. Abstract

This section is mainly the guideline to perform artifact evaluation for this paper. Most of the information to reproduce the results are in the readme.md file of the repository. We first present the check-list for the evaluation. Then we discuss the hardware and software dependencies. Finally, installation and experiment workflow shows how to setup the environment and use the scripts to perform detailed validation. It is posisble to reproduce all the results of the paper following the guideline of readme.md file.

B. Artifact check-list (meta-information)

- **Program: Python = 3.7, ScaleSim, BookSim2**
- **Run-time environment: Ubuntu = 22.04**
- **Hardware: 1 machine with Ubuntu = 22.04**
- **Output: Json files containing simulation results**
- **Experiments: Bandwidth utilization, scalability results, performance of DNN benchmarks**
- **How much disk space required (approximately)?: Around 20GB.**
- **How much time is needed to prepare workflow (approximately)?: Around 15 minutes**
- **How much time is needed to complete experiments (approximately)?: Usually 10 minutes to 48 hours per simulation. Several days might take for all experiments. Parallel simulation recommended**
- **Publicly available?: Yes.**
- **Archived (provide DOI)?: https://doi.org/10.5281/zenodo.10109597**

C. Description

1) How to access: The code is available at https://doi.org/10.5281/zenodo.10109597. It contains a zip file. Please unzip the file for the final code.

2) Hardware dependencies: A single machine can handle the individual simulation. However, to concurrently execute multiple simulations and replicate all results efficiently, it is recommended to use a machine with many cores or use a server. We don't need GPU for our simulation.

3) Software dependencies: We ran our experiments on the Ubuntu 22.04 LTS operating system, but other versions of Ubuntu should also work. Python is a prerequisite for our simulation. Further information on all software dependencies can be found in the readme.md file.

D. Installation

Please see the readme.md file of the repository, which contains a detailed step-by-step "setup" guide.

E. Evaluation and expected results

Instructions for running the experiments and generating the results of this paper can be found in the readme.md file. Specifically, we offer all simulation results, along with scripts to create the figures presented in the paper. Results are stored in the **HPCA_2024_final** folder, and the Python scripts for generating graphs are located in the **utils/python_scripts** folder. Additionally, we provide scripts to reproduce all output files of our simulation in the **utils/run_scripts** folder. Since each of these shell scripts initiates multiple simultaneous simulations in the background, it is recommended to use machines with a higher number of cores. The results of new simulation will be saved in the **HPCA_2024** folder.