

# XTree on EquiMesh: Topology and Algorithm Co-Design for Collective Communication

Junwei Cui, Le Qin, Weilin Cai, Jiayi Huang\*

Thrust of Microelectronics, The Hong Kong University of Science and Technology (Guangzhou)

Guangdong-Macao Joint Laboratory for Modular Chip Design and Testing

The Hong Kong University of Science and Technology (Guangzhou)

{jcui382,lqin674,wcai738}@connect.hkust-gz.edu.cn, hjy@hkust-gz.edu.cn \*Corresponding Author

**Abstract**—Mesh topology is widely adopted for both on-chip and chiplet-based interconnects due to its placement-friendly physical layout. However, the low-degree nodes at the edges and corners create bandwidth bottlenecks for common collectives such as AllGather and AllReduce. We address this limitation with EquiMesh, an augmented 2D-Mesh with equivalent-degree nodes without incurring switching complexity. To fully exploit EquiMesh, we propose XTree, a topology-aware algorithm that maximizes utilization of available bandwidth, and MirrorXTree, which constructs ReduceScatter and AllReduce on top of XTree through topology mirroring. Our evaluation shows that EquiMesh with XTree and MirrorXTree achieves  $2\times$  and  $1.2\times$  higher effective bandwidth than state-of-the-art mesh-based topology-algorithm co-designs for AllGather and AllReduce, respectively.

## I. INTRODUCTION

Mesh topology is widely used to scale systems as it maps naturally onto a 2D plane, enabling a scalable physical design. It is adopted not only for on-chip interconnects, ranging from classical many-core processors [1] to modern AI accelerators [2], but also for off-chip links among multiple chiplets [3], across dies and even wafers [4], [5], as well as in large-scale distributed systems [6]. With the rapid growth of Large Language Models (LLMs) and their demand for computing resources [7], mesh-based scaling has become more prevalent.

LLM training commonly combines tensor parallelism (TP) [8], sequence parallelism (SP) [9], pipeline parallelism (PP) [10], and data parallelism (DP) [11] to distribute heavy workloads across multiple devices. Among these, TP introduces AllReduce and SP introduces AllGather, each accounting for roughly half of the total data traffic during training [12]. However, implementing AllGather and AllReduce efficiently on a mesh is challenging due to inherent topology limitations.

From a topology perspective, the inherent low and unequal node degree of a mesh is the primary constraint underlying collective-communication bottlenecks. We count one input and one output port of a node (a bidirectional link) as one degree. In a 2D mesh, corner nodes have a degree of two, edge nodes have three, and interior nodes have four. These low-degree nodes limit the overall bandwidth of collective communication. For example, the state-of-the-art AllReduce algorithm, TidalMesh [29], achieves nearly full bandwidth utilization on square meshes, but incurs idle time on high-degree (interior) nodes during AllGather or when applied to non-square meshes,

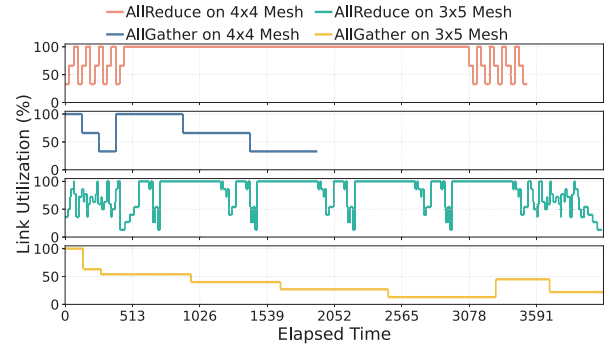


Fig. 1: Bandwidth utilization of TidalMesh for AllReduce and AllGather on  $4\times 4$  and  $3\times 5$  Mesh.

as shown in Fig. 1. The key challenge is to compensate for the mesh’s low degrees while preserving its simple structure and avoiding substantial extra complexity; otherwise, a folded torus [30] would be a more natural alternative.

The key to compensating for the low degree of a mesh is to maintain its primary structure without introducing substantial extra complexity. We propose EquiMesh, which adds supplemental links around the edge nodes of a mesh rather than around interior nodes, thereby preserving the placement-friendly nature of mesh. EquiMesh is a full-degree topology without extra switching complexity, enabling the full utilization of all links during AllReduce communication. We further propose the XTree algorithm to unlock the full potential of EquiMesh. Our main contributions are summarized as follows:

- We introduce EquiMesh, a 2D mesh-based full-degree topology that adds no switching complexity while preserving efficient AllReduce communication.
- We propose XTree, a topology-aware algorithm to fully utilize the available bandwidth of EquiMesh.
- We propose MirrorXTree, which builds ReduceScatter and AllReduce on top of XTree via topology mirroring, matching the performance of AllGather.
- Our evaluation shows that EquiMesh with XTree and MirrorXTree achieves  $2\times$  and  $1.2\times$  higher bandwidth utilization than state-of-the-art mesh-based topology-algorithm co-designs for AllGather and AllReduce, respectively.

## II. BACKGROUND AND MOTIVATION

### A. Mesh Topology

2D mesh is widely used in on-chip and distributed systems, ranging from router-based chips to large-scale, directly connected servers. The core of a router is a crossbar switch that connects multiple input ports to distribute data to multiple output ports, as shown in Fig. 2c. A standard crossbar has five directions: North, East, South, West, and Local ports. The 2D mesh is a natural fit for such crossbar-based systems because each node can connect directly to its four cardinal neighbors and a local processing element.

Another typical application is die-to-die or card-to-card interconnects for chiplets, multi-die packages, and server-scale systems. The interconnection interfaces are usually placed along the edges of the die or card, as shown in Fig. 2e. These connections can be built as either unidirectional or bidirectional links, using technologies such as on-chip wires, AIB [22] or BoW [23] interfaces, and optical or electrical cables.

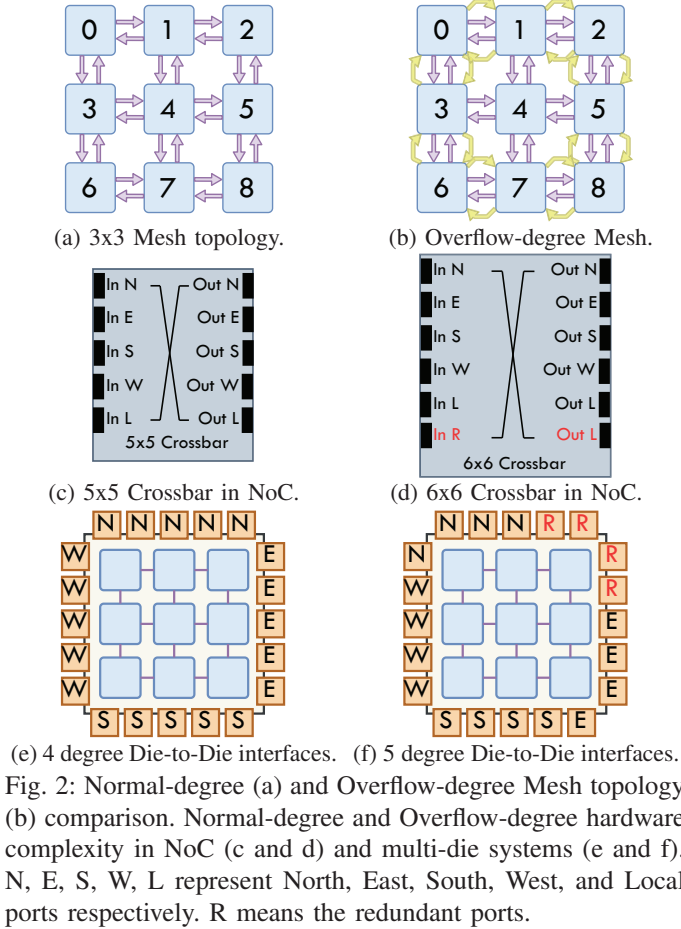


Fig. 2: Normal-degree (a) and Overflow-degree Mesh topology (b) comparison. Normal-degree and Overflow-degree hardware complexity in NoC (c and d) and multi-die systems (e and f). N, E, S, W, L represent North, East, South, West, and Local ports respectively. R means the redundant ports.

Unlike symmetric topologies such as Torus [15] and Hypercube [16] whose nodes are uniformly connected, the 2D mesh has different types of nodes with varying degrees of connectivity. Specifically, corner nodes have a degree of 2, edge nodes have a degree of 3, and inner nodes have a degree of 4, as shown in Fig. 2a. This non-uniform connectivity can

lead to performance bottlenecks, especially when the network is heavily loaded. Ruche [17] adds express channels [18] to reduce long-distance hops but introduces extra hardware complexity and fails to solve the low-degree issue of edge nodes. RingRoad [19] improves mesh by adding multi-rings to decouple different traffic streams but faces the same connectivity limitations. Other proposals such as Switch-Less Dragonfly [20] and FRED [21] use on-wafer switches to connect multiple nodes to build a fully connected topology.

One direct way to mitigate the limited degree of edge nodes is to add duplicated links on each side of the mesh, as depicted in Fig. 2b. However, it can lead “overflow” degrees for some edge nodes such as Node 1, 3, 5, and 7, where extra links remain underutilized. In router-based designs such as Network-On-Chip (NoC) and Network-On-Package (NoP) [3], crossbar complexity increases quadratically from  $O(N^2)$  to  $O((N + 1)^2)$ , as shown in Fig. 2c and 2d. The crossbar complexity is mainly determined by the number of input and output ports. For example, a 5-port crossbar requires 25 switches while a 6-port crossbar requires 36, significantly impacting the system performance and power consumption.

For the die-to-die interconnects, such as chiplet-based multi-die systems [4], an overflow-degree design introduces additional I/O pads and SerDes transceivers, as shown in Fig. 2e and 2f. These redundant ports are not always active but consume valuable area budget along the die edge, such as the red IO R in Fig. 2f. Therefore, the full-degree EquiMesh is a superior choice for high-radix topologies, as it balances performance and hardware complexity by utilizing existing router ports or IO area to compensate for insufficient degrees of edge nodes.

### B. AllReduce Algorithms

AllReduce in Fig. 3—along with its two constituent phases, ReduceScatter and AllGather—is a widely used collective communication operation in distributed systems, especially in current hybrid-parallel LLM systems [24]. AllReduce aggregates data across all participating nodes and then distributes the aggregated result back to every node. In the ReduceScatter phase, nodes first perform a reduction across the group and then scatter disjoint partitions of the reduced result to the participants. Subsequently, AllGather collects these partitions from all nodes through broadcasts, allowing each node to reconstruct the complete reduced tensor.

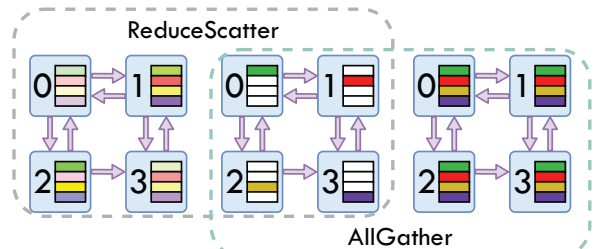


Fig. 3: Process of AllReduce: ReduceScatter and AllGather.

A substantial body of work has proposed AllReduce algorithms tailored to mesh topology, targeting lower latency

or higher bandwidth utilization. Hierarchical Ring [25] decomposes communication into different dimensions and uses blocked ring algorithm to perform AllReduce on general topologies. 2D Mesh algorithm [32] implements ReduceScatter and AllGather using ring algorithm on X and Y dimensions separately to fully utilize bandwidth in both dimensions. MultiTree [26] proposes a tree-based AllReduce algorithm with fewer steps and higher bandwidth utilization. TTO [27] builds non-interlaced trees spatially combining chunk pipelining to reduce latency. TACOS [28] offers a topology-agnostic method to automatically generate AllGather algorithms for any given topology. TidalMesh [29] proposes a mesh-specific AllReduce algorithm that overlaps AllGather and ReduceScatter from different chunks to reduce overall latency. Collectively, these schemes advance AllReduce performance on meshes, with some approaching the theoretical latency lower bound. Nevertheless, they remain fundamentally constrained by the low-degree edge nodes inherent to mesh topologies.

Moreover, these studies [26]–[28] do not directly generate communication schedules for ReduceScatter. They typically use the reverse process of AllGather for ReduceScatter, based on the conventional assumption that network links are bidirectional. However, if a topology implements unidirectional links on each side, simply reversing the AllGather process is unfeasible for ReduceScatter due to the absence of the necessary reversed links. While previous algorithms like MultiTree and TACOS do not directly generate communication schedules for ReduceScatter, simply reversing the AllGather process without considering data dependencies and topology can lead to incorrect results or inefficient communication patterns.

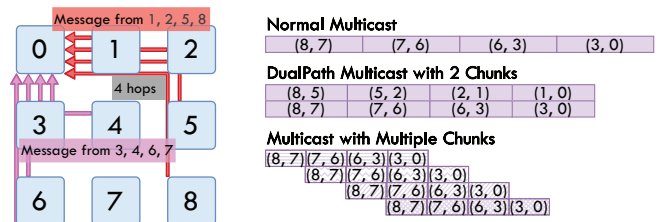
### III. ENHANCED MESH TOPOLOGY

#### A. AllGather Analysis

AllGather constitutes one phase of AllReduce, while ReduceScatter is typically regarded as its reverse [26]–[28]. Hence, we focus on AllGather to identify the AllReduce bottleneck on mesh topologies.

As shown in Fig. 4a, consider Node 0 as an example. During AllGather, Node 0 must receive data from all other nodes (Node 1 to 8). However, it has only two incoming links, (1, 0) and (3, 0), whereas Node 3 has three incoming links and Node 4 has four. This disparity indicates that the minimum node degree bounds the effective bandwidth of AllGather; when the message size is large and bandwidth-dominated, low-degree nodes become the bottleneck. Performance can be significantly improved, approaching the theoretical optimum, if these critical links are fully utilized without redundant transmissions throughout the AllGather phase.

Communication distance, determined by the mesh diameter, also impacts latency. Under a per-hop allocation model, the minimum latency of AllGather is bounded by the length of the critical longest path. For an  $N \times N$  square mesh, where each node holds  $MSize$  data and each link has bandwidth  $B$ , the bandwidth bound is  $\left\lceil \frac{N^2-1}{2} \right\rceil \times \frac{MSize}{B}$ . This implies that corner nodes with a degree of two must receive data from all



(a) 2D Mesh AllGather. (b) 2D Mesh AllGather Chunk Analysis.

Fig. 4: 2D Mesh AllGather Analysis. (We choose a 3x3 Mesh as an example with an analysis a multicast operation which is a part of Gather operations in completed AllGather process.)

other  $N^2 - 1$  nodes through only two links. The latency bound is  $2(N - 1) \times \frac{MSize}{B}$ , as data from the farthest node must traverse  $2(N - 1)$  hops to reach a corner node. These two bounds coincide when  $N$  equals 2 or 3, but the bandwidth bound becomes increasingly critical as  $N$  grows.

To better utilize critical links and reduce communication distance, prior work commonly splits large messages into chunks [27], [29]. We use a multicast from Node 8 to Node 0 to illustrate the chunk-based AllGather process in Fig. 4b. If the message remains undivided, it follows the XY route—(8, 7), (7, 6), (6, 3), (3, 0)—to reach Node 0. By contrast, if the message is split into two chunks, Chunk 0 and Chunk 1 can traverse different paths concurrently, roughly halving the latency through dual-path [31]. With even smaller chunks, these transmissions can be interleaved (overlapped) along a single path to further shorten the critical path. Because the minimum chunk size constrains the set of available paths and determines the critical-path latency for the AllGather operation, the chunk size should be chosen carefully to balance bandwidth and latency bounds for better performance.

#### B. EquiMesh Topology

We aim to alleviate the mesh bandwidth bottleneck by increasing the minimum node degree. The key idea is to add several additional links among edge nodes to compensate for their low connectivity without exceeding the link budget. Accordingly, we propose *EquiMesh*, a full-degree, mesh-based topology that ensures an equivalent number of input and output links for all nodes.

As shown in Fig. 5a, *EquiMesh* introduces two types of additional links on the mesh edges: (1) *EquiMesh Edge Links* (green arrows) connect edge nodes that are two hops apart via unidirectional links; and (2) *EquiMesh Corner Links* (yellow arrows) connect a corner node to a single neighboring node in one direction. These additional links form a unidirectional ring among the nodes on one edge of the 2D Mesh, bringing one extra degree of connectivity. In Fig. 5a, the top edge forms a single ring:  $0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 0$ . Corner nodes participate in the rings on both incident edges, thereby gaining two additional degrees. For instance, the corner Node 0 is connected across two edges, participating in both the top-edge ring and the vertical ring:  $0 \rightarrow 5 \rightarrow 15 \rightarrow 10 \rightarrow 0$ .

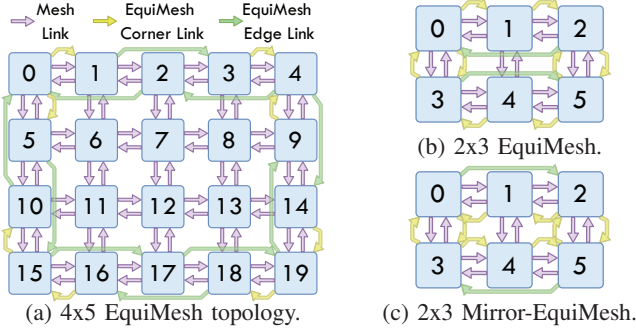


Fig. 5: EquiMesh Topology Examples.

---

#### Algorithm 1 EquiMesh Ring Generation on An Edge

---

**Require:** Edge length  $n \geq 2$ ; Boolean *anticlockwise*  
**Ensure:** Ring lists ( $L$ )

- 1:  $odds \leftarrow range[1, n - 2, 2]$ ;  $evens \leftarrow range[0, n - 2, 2]$
- 2: **if** *anticlockwise* **then**
- 3:    $L \leftarrow L_{evens} + Reverse(L_{odds})$ ; **return**  $L$
- 4: **else**
- 5:    $L \leftarrow L_{odds} + Reverse(L_{evens})$ ; **return**  $L$
- 6: **end if**

---

The EquiMesh ring generation algorithm for a single edge is listed in Algorithm 1. The edge length  $n$  is defined as the number of nodes situated along one edge of a 2D mesh. The algorithm generates an index list  $L$ , representing the sequence of the ring on the edge. Specifically, it creates two interleaved lists:  $L_{odd}$  for odd indices and  $L_{even}$  for even indices. Based on the *anticlockwise* boolean flag, it concatenates the lists to form a ring in either an anticlockwise or clockwise order. This approach ensures that each edge node is part of a single unidirectional ring without extra ports, regardless of whether the edge length is odd or even. Once the ring index list is established, the EquiMesh Edge Links and EquiMesh Corner Links are added to the original mesh based on the respective row or column offsets.

The direction of EquiMesh ring on each edge can be flexibly configured, as illustrated in Fig. 5b and 5c. For example, a  $2 \times 3$  EquiMesh topology may employ a clockwise direction on the top edge and an anticlockwise direction on the bottom edge, whereas the  $2 \times 3$  Mirror-EquiMesh chooses the opposite direction—hence the name “Mirror” topology. These directions can be configured independently for each edge without compromising the full-degree property of EquiMesh. However, the use of unidirectional rings introduces a challenge: ReduceScatter cannot be simply considered as a reverse process of AllGather because the required reverse-direction links may not exist in the topology. Therefore, the mirror topology is utilized in Section IV-B to resolve this issue.

#### IV. TOPOLOGY-AWARE ALLREDUCE

To adapt to the EquiMesh, we propose XTree, a topology-aware tree-based algorithm for the AllGather operation. Unlike MultiTree [26], which builds one tree per root node, XTree constructs multiple distinct trees for each chunk originating

---

#### Algorithm 2 XTree AllGather Algorithm (compressed)

---

**Require:** Topology graph  $G(V, E)$ , Chunk count  $C$   
**Ensure:** AllGather schedule *allgather\_schedule*

- 1: **for** each chunk  $c \in \{1, \dots, C\}$  **do**
- 2:   Initialize trees  $\{T_{v,c} \mid v \in V\}$  with roots at node  $v$ ;
- 3: **end for**
- 4:  $timestep \leftarrow 0$ ;
- 5: **while** there exists an incomplete tree **do**
- 6:    $timestep \leftarrow timestep + 1$ ;  $G'(V', E') \leftarrow G(V, E)$ ;
- 7:   Sort the trees  $\{T_{v,c}\}$  in descending order of the farthest distance to the remaining target nodes;
- 8:   **for** each selected tree  $T_{v,c}$  **do**
- 9:     **if**  $\exists e = (u, w) \in E'$  s.t.  $u \in T_{v,c}$  and  $w \notin T_{v,c}$  **then**
- 10:       Add  $w$  and  $e$  to  $T_{v,c}$ ; Remove  $e$  from  $E'$ ;
- 11:       Append  $e$  to *allgather\_schedule*( $timestep$ ); **break**;
- 12:     **end if**
- 13:   **end for**
- 14: **end while**

---

from each node. Consequently, the total number of trees is equal to the number of nodes multiplied by the number of chunks.

Algorithm 2 presents the pseudocode for the XTree AllGather algorithm. The inputs include a topology graph  $G(V, E)$  with nodes set  $V$  and edges set  $E$ , along with the chunk number  $C$ . The output is the AllGather schedule *allgather\_schedule*. The algorithm first initializes  $C$  trees for each node in the graph as  $B$ . It then iterates over discrete *timesteps* (Line 6) until all spanning trees are complete. At each timestep, the algorithm creates a residual graph copy  $G'(V', E')$  to track currently available links. It sorts the existing trees by the farthest distance from the nodes already included in each tree to the target nodes. It then enters a loop that continues as long as available edges remain in  $E'$ , meaning the remaining links are idle and available for use. Inside this loop, it iterates over the sorted trees to check for available edges connecting a node already within the tree to a target node not yet included. If such an edge exists, it adds the new node to the tree, removes the edge from  $E'$ , and records the edge in the AllGather schedule for the current time step. This procedure repeats until no further edges can be assigned in the current step or all trees are completed.

##### A. XTree for AllGather

Fig. 6a illustrates the XTree AllGather trees generated on a  $2 \times 3$  EquiMesh topology. To fully utilize link bandwidth and distribute data from other  $n - 1$  nodes, the chunk number should be a multiple of the topology’s minimum node degree. This ensures that the total communication load is balanced across all parallel links, resulting in a tree depth that is a multiple of  $n - 1$ , where each timestep’s duration is the ratio of chunk size to link bandwidth. For a full four-degree EquiMesh, XTree sets the chunk count to 4, generating trees with a depth of 5.

When constructing the trees, XTree prioritizes those with a larger remaining “future depth”, referring to the farther remaining targets. This strategy promotes balanced growth and reduces idle time across the network. The farthest distance can be overlapped by four chunks without increasing the overall tree depth. Moreover, the XTree algorithm is topology-agnostic: its greedy-like strategy finds available links without relying on topology-specific structural features.



Fig. 6: XTree Link Allocation on a  $2 \times 3$  EquiMesh Topology. (The colors in this figure correspond to the links in Fig. 5.)

### Algorithm 3 MirrorXTree AllReduce Algorithm

**Require:** Topology graph  $G(V, E)$ , Chunk count  $C$   
**Ensure:**  $reducescatter\_schedule$  and  $allgather\_schedule$   
1:  $\bar{G}(V, \bar{E}) \leftarrow \text{Mirror\_Topology\_Graph}(G(V, E))$ ;  
2:  $reducescatter\_schedule \leftarrow \text{Reverse}(\text{XTree}(\bar{G}(V, \bar{E}), C))$ ;  
3:  $allgather\_schedule \leftarrow \text{XTree}(G(V, E), C)$ ;  
4: **return** ( $reducescatter\_schedule, allgather\_schedule$ );

### B. MirrorXTree for AllReduce

Unlike the standard 2D mesh topology, where all links are bidirectional, the EquiMesh topology forms unidirectional rings along its edges. This design complicates ReduceScatter, because it can no longer be obtained by simply reversing the AllGather schedule, as specific reverse-direction links do not exist in the topology. To address this issue, we propose MirrorXTree, which enables efficient AllReduce on EquiMesh.

As discussed in Section III-B, EquiMesh allows for the flexible configuration of ring directions on each edge. MirrorXTree algorithm leverages this flexibility by constructing a mirror configuration in which the ring directions are reversed. This construction guarantees that for every directed link used by AllGather, a corresponding reverse-direction link exists for ReduceScatter, as formalized in Algorithm 3.

For example, to build ReduceScatter trees for the  $2 \times 3$  EquiMesh topology (Fig. 5b), we first construct a logical  $2 \times 3$  Mirror-EquiMesh topology (Fig. 5c) to generate its AllGather trees. We then reverse the direction of each link in the generated AllGather trees to obtain the ReduceScatter trees for the original  $2 \times 3$  EquiMesh topology, as shown in Fig. 6b.

Note that the ReduceScatter and AllGather trees for the same root and chunk are not strictly symmetric, as EquiMesh and Mirror-EquiMesh differ in edge-ring directions. Nevertheless, both schedules occupy the links in a similar manner, enabling

TABLE I: System Configuration

Parameters		Configurations
PEs	Number of Arrays	$5 \times 11, 8 \times 8$
	MAC Array	$512 \times 512$ (32-bit) at 1GHz
Network	Parallelism	SP + TP
	Link Bandwidth and Latency	128 GB/s, 20 ns per hop

AllReduce to execute the ReduceScatter and AllGather phases sequentially without requiring additional overlap scheduling.

## V. EVALUATION

### A. Experimental Setup

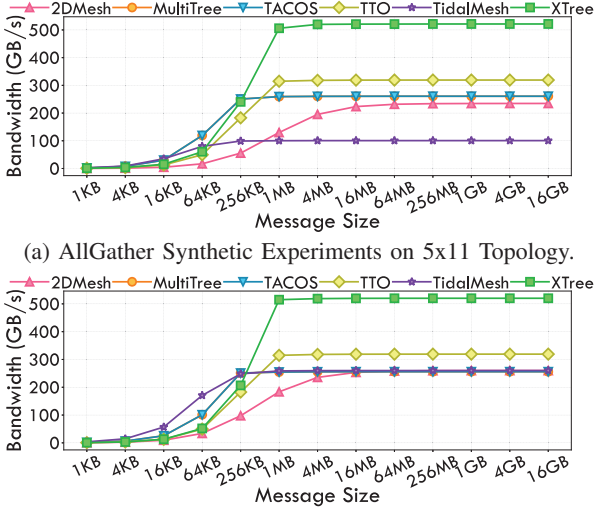
We developed an event-driven simulator to evaluate the performance of our proposed algorithms and topology, comparing them with existing methods using an  $\alpha$ - $\beta$  communication model like TACOS with an analytical computation model considering peak performance. The experimental settings are listed in Table I. We compare our hardware-software co-design compared with following popular AllReduce algorithms:

- 2DMesh: A hierarchical simultaneous ring algorithm that executes ReduceScatter on X and Y dimensions, followed by AllGather on Y and X dimensions [13], [32].
- MultiTree: Constructs one tree for each node for AllGather and uses the reverse tree for ReduceScatter [26].
- TACOS: We enhance MultiTree algorithm achieving the bandwidth-bound performance to simulate TACOS [28].
- TTO: Generates three trees, where a corner node does not participate in the communication [27].
- TidalMesh: Reduces latency by overlapping ReduceScatter and AllGather across different data chunks [29].

### B. Communication Performance

By leveraging its full-degree architecture, EquiMesh achieves up to  $2 \times$  effective bandwidth for AllGather compared to the

state-of-the-art (SOTA) algorithms. Even though TidalMesh overlaps ReduceScatter and AllGather across different chunks to improve AllReduce bandwidth utilization, it remains fundamentally constrained by the degree limitations of the underlying mesh topology. As shown in Figure 7, across both square and rectangular meshes (including “even-shaped” and “uneven-shaped” configurations), EquiMesh achieves nearly  $2\times$  the effective AllGather bandwidth of competing algorithms once the bandwidth is saturated. We define effective bandwidth as the communication size per node divided by the collective completion time, a metric consistent with prior work [26], [28].



(a) AllGather Synthetic Experiments on 5x11 Topology.  
(b) AllGather Synthetic Experiments on 8x8 Topology.  
Fig. 7: AllGather Synthetic Experiments.

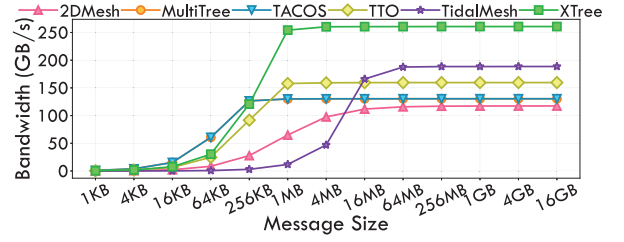
EquiMesh also achieves a higher effective bandwidth for AllReduce, as its full-degree architecture ensures there are no idle links during the AllReduce procedure. As shown in Figure 8, EquiMesh achieves nearly  $1.2\times$  improvement in effective bandwidth compared to the baselines once the bandwidth is saturated. While TidalMesh overlaps across different chunks of ReduceScatter and AllGather to boost its AllReduce performance relative to its AllGather, it remains constrained by the mesh’s physical degree limitations. Consequently, EquiMesh still maintains a  $1.2\times$  speedup over TidalMesh by eliminating the remaining topological bottlenecks.

### C. End-to-End Performance

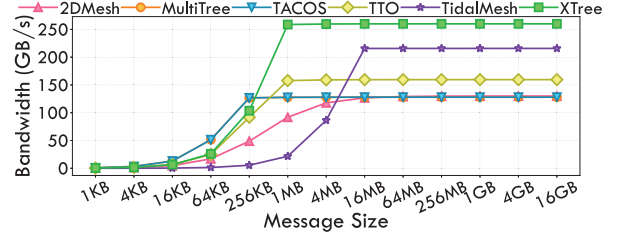
To evaluate our design in real-world settings, we conducted end-to-end training experiments on four LLMs—GPT-NeoX-20B, OPT-30B, Qwen3-32B, and Llama-3-70B—representing a diverse range of model sizes and architectures. As shown in Fig. 9, EquiMesh achieves a  $1.35\times$ – $1.39\times$  speedup during the LLM forward pass and a  $1.06\times$ – $1.23\times$  speedup during the backward pass, where computation–communication overlap reduces the fraction of time attributable to collectives.

## VI. CONCLUSION

To address the non-uniform node degree of classical mesh topology, we propose EquiMesh, a full-degree topology that introduces no extra arbitration complexity. Leveraging EquiMesh,

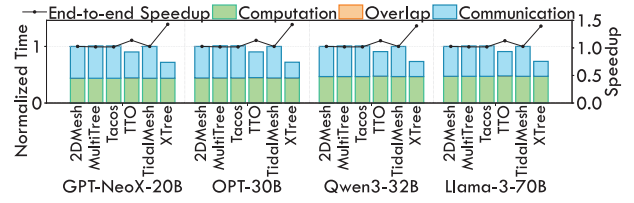


(a) AllReduce Synthetic Experiments on  $5\times 11$  Topology.

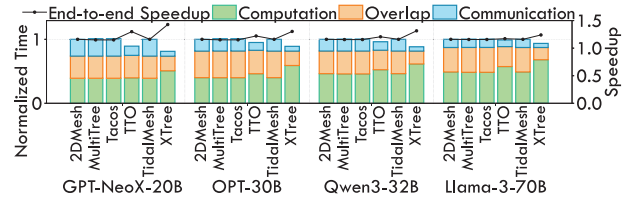


(b) AllReduce Synthetic Experiments on  $8\times 8$  Topology.

Fig. 8: AllReduce Synthetic Experiments.



(a) LLM Forward Performance.



(b) LLM Backward Performance.

Fig. 9: End-to-End LLM Performance on  $8\times 8$  Topology.

we propose XTree, a topology-aware algorithm that fully exploits EquiMesh bandwidth via chunk-count adaptation. We further propose MirrorXTree to support ReduceScatter and AllReduce on top of XTree, enabling efficient communication even without full bidirectional links. Overall, our evaluation shows that EquiMesh with XTree and MirrorXTree achieves nearly  $2\times$  and  $1.2\times$  higher effective bandwidth than state-of-the-art mesh-based topology-algorithm co-designs for AllGather and AllReduce, respectively, and delivers geometric-mean speedups of  $1.37\times$  (forward) and  $1.13\times$  (backward) in end-to-end LLM training.

## ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China (No. 62402411), the Guangdong Basic and Applied Basic Research Foundation (No. 2023A1515110353), the Guangdong Science and Technology Department (No. 2025B1212150003), and the Guangdong Provincial Project (No. 2023QN10X252).

## REFERENCES

- [1] A. Sodani, R. Gramunt, J. Corbal, H.-S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.-C. Liu, "Knights Landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016.
- [2] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 609–622.
- [3] 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, 2019, p. 14–27.
- [4] E. Talpes, D. D. Sarma, D. Williams, S. Arora, T. Kunjan, B. Floering, A. Jalote, C. Hsiong, C. Poorna, V. Samant, J. Sicilia, A. K. Nivarti, R. Ramachandran, T. Fischer, B. Herzberg, B. McGee, G. Venkataramanan, and P. Banon, "The Microarchitecture of Dojo, Tesla's Exa-Scale Computer," *IEEE Micro*, vol. 43, no. 3, pp. 31–39, 2023.
- [5] H. Ltaief, Y. Hong, L. Wilson, M. Jacquelin, M. Ravasi, and D. E. Keyes, "Scaling the "Memory Wall" for Multi-Dimensional Seismic Processing with Algebraic Compression on Cerebras CS-2 Systems," in ser. SC '23. Association for Computing Machinery, 2023.
- [6] J. Vasiljevic, L. Bajic, D. Capalija, S. Sokorac, D. Ignjatovic, L. Bajic, M. Trajkovic, I. Hamer, I. Matosevic, A. Cejkov, U. Aydonat, T. Zhou, S. Z. Gilani, A. Paiva, J. Chu, D. Maksimovic, S. A. Chin, Z. Moudallal, A. Rakhmati, S. Nijjar, A. Bhullar, B. Drazic, C. Lee, J. Sun, K.-M. Kwong, J. Connolly, M. Dooley, H. Farooq, J. Y. T. Chen, M. Walker, K. Dabiri, K. Mabee, R. S. Lal, N. Rajatheva, R. Retnamma, S. Karodi, D. Rosen, E. Munoz, A. Lewycky, A. Knezevic, R. Kim, A. Rui, A. Drouillard, and D. Thompson, "Compute Substrate for Software 2.0," *IEEE Micro*, vol. 41, no. 2, pp. 50–55, 2021.
- [7] C. Zhao, C. Deng, C. Ruan, D. Dai, H. Gao, J. Li, L. Zhang, P. Huang, S. Zhou, S. Ma, W. Liang, Y. He, Y. Wang, Y. Liu, and Y. Wei, "Insights into DeepSeek-V3: Scaling Challenges and Reflections on Hardware for AI Architectures," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1731–1745.
- [8] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism," 2020.
- [9] V. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing Activation Recomputation in Large Transformer Models," 2022.
- [10] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021.
- [11] Y. Hao, Z. Lai, W. Wang, S. Li, W. Liu, K. Ge, and D. Li, "HSDP: Accelerating Large-Scale Model Training via Efficient Sharded Data Parallelism," in *2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2024, pp. 116–125.
- [12] H. Liao, B. Liu, X. Chen, Z. Guo, C. Cheng, J. Wang, X. Chen, P. Dong, R. Meng, W. Liu, Z. Zhou, Z. Zhang, Y. Gai, C. Qian, Y. Xiong, Z. Cheng, J. Xia, Y. Ma, X. Chen, W. Du, S. Xiao, C. Li, Y. Qin, L. Xiong, Z. Yu, L. Chen, L. Chen, B. Wang, P. Wu, J. Gao, X. Li, J. He, S. Yan, and B. McColl, "UB-Mesh: A Hierarchically Localized ND-FullMesh Datacenter Network Architecture," 2025.
- [13] S. Kumar and N. Jouppi, "Highly Available Data Parallel ML Training on Mesh Networks," *arXiv preprint arXiv:2011.03605*, 2020.
- [14] J. M. Cámara, M. Moretó, E. Vallejo, R. Beivide, J. Miguel-Alonso, C. Martínez, and J. Navaridas, "Twisted Torus Topologies for Enhanced Interconnection Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1765–1778, 2010.
- [15] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. A. Patterson, "TPU V4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023.
- [16] S.-Y. Cheng and J.-H. Chuang, "Variate Hypercube—A New Interconnection Network Topology for Large Scale Multicomputer," in *Proceedings of 1994 International Conference on Parallel and Distributed Systems*, 1994, pp. 703–708.
- [17] D. C. Jung and M. Taylor, "Evaluating Ruche Networks: Physically Scalable, Cost-Effective, Bandwidth-Flexible NoCs," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1035–1048.
- [18] W. Dally, "Express Cubes: Improving the Performance of K-Ary N-Cube Interconnection Networks," *IEEE Transactions on Computers*, vol. 40, no. 9, pp. 1016–1023, 1991.
- [19] Y. Feng, W. Li, and K. Ma, "Ring Road: A Scalable Polar-Coordinate-Based 2D Network-on-Chip Architecture," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 871–884.
- [20] Y. Feng and K. Ma, "Switch-Less Dragonfly on Wafers: A Scalable Interconnection Architecture Based on Wafer-Scale Integration," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 1–17.
- [21] S. Rashidi, W. Won, S. Srinivasan, P. Gupta, and T. Krishna, "FRED: A Wafer-Scale Fabric for 3D Parallel DNN Training," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 34–48.
- [22] C. Liu, J. Botimer, and Z. Zhang, "A 256Gb/s/mm-Shoreline AIB-Compatible 16nm FinFET CMOS Chiplet for 2.5D Integration with Stratix 10 FPGA on EMIB and Tiling on Silicon Interposer," in *2021 IEEE Custom Integrated Circuits Conference (CICC)*, 2021, pp. 1–2.
- [23] S. Ardlan, R. Farjadrad, M. Kuemerle, K. Poulton, S. Subramaniam, and B. Vinnakota, "An Open Inter-Chiplet Communication Link: Bunch of Wires (BoW)," *IEEE Micro*, vol. 41, no. 1, pp. 54–60, 2021.
- [24] L. Qin, J. Cui, W. Cai, and J. Huang, "Chimera: Communication Fusion for Hybrid Parallelism in Large Language Models," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 498–513.
- [25] S. Shi, Q. Wang, and X. Chu, "Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 949–957.
- [26] 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.
- [27] S. Laskar, P. Majhi, S. Kim, F. Mahmud, A. Muzahid, and E. J. Kim, "Enhancing Collective Communication in MCM Accelerators for Deep Learning Training," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 1–16.
- [28] W. Won, M. Elavazhagan, S. Srinivasan, S. Gupta, and T. Krishna, "TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 856–870.
- [29] D. Lim and J. Kim, "TidalMesh: Topology-Driven AllReduce Collective Communication for Mesh Topology," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025, pp. 1526–1540.
- [30] N. E. Jerger, T. Krishna, and L.-S. Peh, "On-Chip Networks." Springer Nature, 2022.
- [31] H. Chen, P. Chen, X. Luo, S. Huai, and W. Liu, "LAMP: Load-Balanced Multipath Parallel Transmission in Point-to-Point NoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5232–5245, 2022.
- [32] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image Classification at Supercomputer Scale," 2018.