

**SOFTWARE-HARDWARE OPTIMIZATIONS FOR EFFICIENT COLLECTIVE
COMMUNICATIONS IN DISTRIBUTED MACHINE LEARNING PLATFORMS**

A Dissertation
Presented to
The Academic Faculty

By

William Jonghoon Won

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science
College of Computing

Georgia Institute of Technology

December 2025

© William Jonghoon Won 2025

SOFTWARE-HARDWARE OPTIMIZATIONS FOR EFFICIENT COLLECTIVE COMMUNICATIONS IN DISTRIBUTED MACHINE LEARNING PLATFORMS

Thesis committee:

Dr. Tushar Krishna
School of Electrical and Computer
Engineering
Georgia Institute of Technology

Dr. Manya Ghobadi
Department of Electrical Engineering and
Computer Science
Massachusetts Institute of Technology

Dr. Yingyan (Celine) Lin
School of Computer Science
Georgia Institute of Technology

Dr. Bradford Beckmann
Research and Advanced Development
Advanced Micro Devices

Dr. Divya Mahajan
School of Computer Science
Georgia Institute of Technology

Date approved: December 3, 2025

We're here to put a dent in the universe.

Steve Jobs

To my mother, in loving memory

ACKNOWLEDGMENTS

As I conclude my Ph.D. journey, I am deeply aware that this accomplishment would not have been possible without the support of so many people. I cannot mention everyone individually, so this section is necessarily a concise acknowledgment of those who have been most influential.

First of all, I would like to express my sincere gratitude to my advisor, Dr. Tushar Krishna. This entire Ph.D. program—from beginning to end—would not have been possible without his guidance, tutelage, and support. Dr. Krishna has shaped me as a researcher, providing invaluable insights, advice, and mentorship while demonstrating extraordinary perseverance. His support extended far beyond research: I am truly grateful for the many opportunities to travel across the nation, visit renowned institutions, network with outstanding researchers, and, in the process, grow as a person and as a researcher.

I am also deeply thankful to my thesis committee members, Dr. Brad Beckmann, Dr. Manya Ghobadi, Dr. Celine Lin, and Dr. Divya Mahajan. I appreciate not only your guidance in shaping this thesis but also your kindness and encouragement. Dissertation proposals and defenses are demanding endeavors, but your tremendous support and kind words made these challenges far more manageable and enjoyable.

Next, I would like to thank all the members of the Synergy Lab. My first appreciation goes to the senior members: Dr. Hyoukjun Kwon, Dr. Mayank Parasar, Dr. Anand Samajdar, Dr. Eric Qin, Dr. Felix Kao, Dr. Saeed Rashidi, Dr. Srikant Bharadwaj, and Matthew Denton. I fondly remember the many moments of fun—from K-pop and hiking to trips to Black Rock Mountain. I would like to give special thanks to Hyoukjun and Saeed. Hyoukjun supported me even before I started my Ph.D., helping me gain admission to Georgia Tech and the Synergy Lab. I am sincerely grateful to Saeed for serving as my Ph.D. mentor; none of my research would have been possible without his guidance. I also thank the other Synergy Lab members and Georgia Tech collaborators, including Dr. Geonhwa Jeong, Dr.

Taekyung Heo, Dr. Joongun Park, Dr. Raveesh Garg, Dr. Zishen Wan, Dr. Divya Kadiyala, Jinsun Yoo, Jianming Tong, Hanjiang Wu, Abhimanyu Bambhaniya, Canlin Zhang, Jamin Seo, Ziwei Li, Huan Xu, Anirudh Itagi, Yangyu Chen, Mingyu Lee, Ritik Raj, Changhai Man, Difei Cao, Hanchen Yang, Akshat Ramachandran, and Hao Kang. Your collaboration, insights, and countless fun conversations made my Ph.D. journey richer and more enjoyable. There are many stories I could share, but they are too large to fit in the margin.

I also extend my thanks to my industry collaborators. My first internship was with Dr. Sudarshan Srinivasan at Intel. He taught me much about machine learning and collective communication. This thesis would not have been possible without his mentorship, for which I am profoundly grateful. I am also thankful to my past and current collaborators at AMD: Dr. Tuan Ta, Dr. Vinay Ramakrishnaiah, Dr. Ruchi Shah, Dr. Furkan Eris, Dr. Moumita Dey, Dr. David Sidler, and Dr. Kishore Punniyamurthy. I have learned immensely from each of you and deeply appreciate your support.

I am also grateful to Dr. Justin Zhang for welcoming me when I first arrived in the U.S. and for all the conversations and camaraderie we shared as roommates in Atlanta. I wouldn't forget about it.

Special thanks to all my friends in Korea. From middle and high school, all the experiences with you were unforgettable and helped shape who I am today. I am truly grateful for all the fun, mischief, and memories we shared.

Lastly, my deepest and sincerest gratitude goes to my family for their unconditional support, encouragement, and understanding. My younger sister, Jongeun, whose maturity and insight I greatly admire, has been a source of guidance throughout this journey. To my father, Changjae, I am forever indebted for your unwavering support and understanding. To my mother, Mija, I love you deeply—this thesis is as much yours as it is mine.

Indeed, I truly realize: “the journey is the reward.”

TABLE OF CONTENTS

Acknowledgments	v
List of Tables	xv
List of Figures	xvii
List of Acronyms	xxii
Summary	xxiv
Chapter 1: Introduction	1
1.1 Challenges	2
1.1.1 Collective Communication	2
1.1.2 Design Space of Distributed Machine Learning	2
1.2 Thesis Contributions	4
1.2.1 ASTRA-sim2.0: Modeling Hierarchical Networks for Distributed Machine Learning at Scale	4
1.2.2 LIBRA: Enabling Workload-Aware Multi-Dimensional Network Topol- ogy Optimization for Distributed Machine Learning	4
1.2.3 TACOS: Topology-Aware Collective Algorithm Synthesizer for Dis- tributed Machine Learning	5
1.2.4 PCCL: Process Group-Aware Scalable and Generic Collective Al- gorithm Synthesizer	5

1.3	Thesis Statement	5
1.4	Thesis Overview	5
Chapter 2: Background		7
2.1	Distributed Machine Learning	7
2.1.1	Neural Processing Unit	7
2.1.2	Parallelization Strategy	7
2.1.3	Synchronous Training and Inference	8
2.2	Collective Communication Pattern	8
2.2.1	Collective Communication Pattern	8
2.2.2	Common Collective Communication Patterns	10
2.3	Collective Communication Algorithm	10
2.3.1	Collective Communication Library	10
2.3.2	Textbook Collective Communication Algorithm	11
2.3.3	Topology-Aware Collective Algorithm	12
Chapter 3: ASTRA-sim2.0: Modeling Hierarchical Networks for Distributed Machine Learning at Scale		15
3.1	Motivation	15
3.2	Background	17
3.2.1	Multi-dimensional Network	17
3.2.2	ASTRA-sim	17
3.3	Challenges	18
3.3.1	Ability to Model Multi-Dimensional Networks	19

3.4	ASTRA-sim2.0	20
3.4.1	Multi-dimensional Network Representation	21
3.4.2	Flow-Based Analytical Network Modeling	23
3.5	Methodology	27
3.5.1	Validation of Analytical Network Backend	27
3.5.2	Simulation Time Speedup	27
3.5.3	Compute Model	28
3.6	Evaluations	28
3.6.1	Flow-Based versus Packet-Based Network Modeling	28
3.6.2	Comparison without Modeling Correlation	29
3.6.3	Correlating Analytical and ns-3	29
3.6.4	Comparison with Network Congestion	30
3.6.5	Simulation Time Speedup	34
3.6.6	Conventional System versus Wafer-Scale System	36
3.7	Related Work	40
3.8	Conclusion	41

Chapter 4: LIBRA: Enabling Workload-Aware Multi-Dimensional Network Topology Optimization for Distributed Machine Learning 42

4.1	Motivation	42
4.2	Background	45
4.2.1	Physical Connotation to Multi-Dimensional Networks	45
4.2.2	Hierarchical Collective Algorithm	46
4.2.3	Hybrid Parallelism	47

4.2.4	Training Loop	49
4.3	Challenges	49
4.3.1	Technology Constraints of a Network Dimension	49
4.3.2	Opportunity of Multi-dimensional Networks	50
4.3.3	Design-time Consideration of Network Bandwidth	51
4.3.4	Problem Statement	54
4.4	LIBRA	54
4.4.1	Multi-dimensional Network Representation	55
4.4.2	Modeling Distributed Training	56
4.4.3	Modeling Network Cost	59
4.4.4	Optimizing Multi-dimensional Network	60
4.4.5	Optimization Scheme	60
4.4.6	Optimization for Multiple Workloads	61
4.5	Methodology	61
4.5.1	Simulation Infrastructure	61
4.5.2	Experimental Setup	62
4.6	Evaluations	63
4.6.1	Sweeping Large Model Training	63
4.6.2	Non-Transformer Workloads	67
4.6.3	Topology Exploration	68
4.6.4	Optimizing for Multiple Workloads	68
4.6.5	Cost Model Sensitivity Analysis	69
4.6.6	LIBRA with Runtime Optimizations	70

4.6.7	Co-optimizing Network and Parallelization	71
4.7	Related Work	72
4.7.1	Multi-dimensional Networks	72
4.7.2	Design-Time Optimization	72
4.7.3	Co-Optimization of Network and Workload	73
4.8	Conclusion	73
Chapter 5:	TACOS: Topology-Aware Collective Algorithm Synthesizer for Dis-	
	tributed Machine Learning	74
5.1	Motivation	74
5.2	Background	78
5.2.1	Collective Algorithm Synthesizer	78
5.3	Challenges	79
5.3.1	Importance of Topology-aware Collective Algorithm	79
5.3.2	Heterogeneity, Asymmetry, and Scale of ML Systems	80
5.3.3	Designing Topology-aware Collective Algorithm	81
5.3.4	Topology-aware Collective Synthesizer	81
5.4	TACOS	82
5.4.1	Time-Expanded Network	82
5.4.2	Representing Collective Algorithm using TEN	84
5.4.3	TACOS Approach to Collective Algorithm Synthesis	86
5.4.4	Network Utilization Maximizing Matching Algorithm	88
5.4.5	End-to-End TACOS Collective Algorithm Synthesis	89
5.4.6	Collective with Reduction Operation	92

5.4.7	TACOS for Heterogeneous Network	93
5.4.8	TACOS for Switch-based Network	94
5.4.9	TACOS Synthesis Result	95
5.5	Methodology	95
5.5.1	Baseline Collective Algorithms and Synthesizers	95
5.5.2	Target Topologies and Synthesis Environment	96
5.5.3	Simulation Infrastructure	97
5.5.4	Network Simulation Backend	97
5.6	Evaluations	98
5.6.1	TACOS Synthesis Result	98
5.6.2	Analysis of TACOS-Synthesized Algorithm	98
5.6.3	Scalability Analysis	106
5.6.4	End-to-End Application	107
5.7	Related Work	108
5.7.1	Time-Expanded Network	108
5.7.2	Customized Collective Communication Algorithm	109
5.7.3	Collective Algorithm Synthesizer	109
5.8	Conclusion	111
Chapter 6: PCCL: Process Group-Aware Scalable and Generic Collective Algorithm Synthesizer		113
6.1	Motivation	113
6.2	Background	118
6.2.1	Process Group	118

6.3	Challenges	119
6.3.1	Scalability	119
6.3.2	Generic Topology Support	120
6.3.3	Generic Collective Support	120
6.3.4	Process Group Awareness	120
6.3.5	Problem Statement	121
6.4	PCCL	121
6.4.1	Condition	121
6.4.2	TEN Representation	122
6.4.3	BFS Pathfinding Algorithm	123
6.4.4	Synthesizing Collective Algorithm	126
6.4.5	Reduction Operations	129
6.4.6	Heterogeneous Networks	130
6.4.7	Modeling Switches	132
6.4.8	Translating Synthesis Results	132
6.5	Methodology	133
6.5.1	Experimental Infrastructure	133
6.5.2	Baseline Collectives	133
6.6	Evaluations	133
6.6.1	Scalability	133
6.6.2	Supporting Generic Topology	136
6.6.3	Supporting Generic Collective	136
6.6.4	Synthesis with Process Group	137

6.7	Related Work	141
6.7.1	Collective Algorithm Synthesizers	141
6.7.2	Customized Collective Algorithms	143
6.8	Conclusion	144
Chapter 7: Conclusions and Future Works		145
7.1	Summary of Contributions	145
7.2	Future Directions	146
7.2.1	Dynamic Collective Algorithm Synthesis	147
7.2.2	Collective Algorithm Synthesis for Latency-Sensitive Collectives	148
7.2.3	Network Topology–Collective Algorithm Co-Design	149
7.2.4	Full-Stack Optimization of Distributed ML Platforms	149
References		150
Vita		165

LIST OF TABLES

2.1	Collectives required by each parallelization strategy.	10
3.1	Network building blocks and its corresponding topology-aware collective communication algorithm.	22
3.2	Target training workloads and their characteristics.	37
3.3	Target wafer-scale and conventional (multi-dimensional) topologies.	37
3.4	Message size in MiB per each dimension and collective time when running an 1 GB All-Gather collective.	39
4.1	Cost model for network cost evaluation in LIBRA.	59
4.2	Workloads specifications used for analysis.	62
4.3	Multi-dimensional Topologies used for analysis.	63
5.1	Various All-Reduce algorithms and their target topologies.	75
5.2	Qualitative comparison of collective algorithm synthesizers.	76
5.3	Common model and data parallelization strategies and their required collective communication patterns.	78
5.4	Topologies evaluated for TACOS.	97
5.5	All-Reduce collective time (with synthesis time in parentheses for both TACOS and TACCL) for a multi-node system.	101
6.1	Qualitative comparison of existing collective algorithm synthesizers.	116

6.2 Supported collective patterns by each collective algorithm synthesizers. . . 116

LIST OF FIGURES

1.1	Contributions of this dissertation.	3
2.1	Definition of MPI collective communication patterns. Each square denotes an NPU, whereas each circle denotes a chunk.	9
2.2	Textbook All-Reduce algorithms and their traffic patterns.	11
2.3	Examples of topology-aware and topology-unaware All-Gather collective algorithms.	13
2.4	Multi-dimensional network building blocks, and their corresponding topology-aware All-Reduce algorithm.	14
3.1	An example definition of network dimension versus hierarchy.	17
3.2	Overview of the proposed infrastructure for modeling next-generation distributed ML platforms.	20
3.3	Proposed multi-dimensional network topology representation in ASTRA-sim2.0.	21
3.4	Example of congestion-unaware network modeling.	24
3.5	Example of congestion-aware network modeling.	26
3.6	Analytical network backend validation over real-system measurements.	27
3.7	Comparing reported bus bandwidth of analytical and ns-3 without correlation.	29
3.8	Comparing reported bus bandwidth of analytical and ns-3 with network correlation.	30

3.9	Analytical versus ns-3 modeling comparison for 32-NPU fully-connected topology.	31
3.10	Analytical versus ns-3 modeling comparison for 8-NPU ring topology.	31
3.11	Analytical versus ns-3 modeling comparison for 16-NPU two-dimensional mesh topology.	32
3.12	Analytical versus ns-3 modeling comparison when running TACOS-synthesized collective algorithm.	33
3.13	Simulation speedup of congestion-aware analytical model over ns-3 for 8-NPU fully-connected topology.	34
3.14	Simulation speedup of congestion-aware analytical model over ns-3 for 32-NPU fully-connected topology.	35
3.15	Simulation speedup of congestion-aware analytical model over ns-3 for 8-NPU ring topology.	35
3.16	Simulation speedup of congestion-aware analytical model over ns-3 for 16-NPU two-dimensional mesh topology.	36
3.17	Wafer-scale versus conventional (multi-dimensional) systems profiling with 512 NPUs.	38
3.18	Wafer-scale versus conventional system scalability comparison.	40
4.1	Communication sizes for ML model training across 1,024 NPUs.	43
4.2	An abstract view of multi-dimensional networks and the physical connotation.	45
4.3	All-Reduce example on a 3×2 two-dimensional network.	48
4.4	Example training loops.	50
4.5	Running All-Reduce with four chunks on three-dimensional networks with different bandwidth allocations.	52
4.6	The normalized end-to-end training time of MSFT-1T on 2–4-dimensional topologies.	53
4.7	The architecture of LIBRA.	54

4.8	Examples of 2–3-dimensional networks and their corresponding names in the notation used.	55
4.9	Example cost modeling of a three-NPU switch network.	60
4.10	End-to-end training speedup over the baseline EqualBW.	65
4.11	Performance-per-cost benefit of LIBRA over the EqualBW baseline.	66
4.12	Speedup and performance-per-cost analysis of ResNet-50 and DLRM.	67
4.13	Speedup and performance-per-cost analysis of MSFT-1T.	68
4.14	Speedup over EqualBW and slowdown over LIBRA-optimized network of various DNN models.	69
4.15	Normalized performance-per-cost benefit of PerfPerCostOptBW.	69
4.16	Normalized performance and performance-per-cost of LIBRA with Themis runtime optimizer.	70
4.17	Normalized performance and performance-per-cost with LIBRA and TACOS.	71
4.18	Normalized speedup of MSFT-1T across various parallelization strategies.	71
5.1	Heat map of total message size transferred over each link, when running 1 GiB All-Reduce using textbook algorithms over different network topologies.	76
5.2	Effect of topology-aware collective algorithms to the All-Reduce bandwidth.	79
5.3	Overview of the existing CCLs and proposed TACOS architecture.	82
5.4	An example TEN data structure using unidirectional ring topology.	83
5.5	TEN representation of a homogeneous, asymmetric 3-NPU topology.	83
5.6	Representing unidirectional All-Gather algorithm using TEN data structure.	85
5.7	Network Utilization Maximizing Matching algorithm.	87

5.8	End-to-end overview of how TACOS constructs an All-Gather algorithm for a homogeneous, asymmetric 4-NPU topology.	90
5.9	Distinct target topologies with 4 NPUs but different numbers of links, and their All-Gather synthesis results over TEN using TACOS.	91
5.10	Synthesis of collectives with reduction operations (e.g., Reduce-Scatter).	92
5.11	An example representation of a heterogeneous network topology in TEN.	93
5.12	Example unwinding of a 4-NPU switch network into a direct-connect topology.	94
5.13	An example All-Gather collective algorithm synthesized by TACOS over a homogeneous 3×3 two-dimensional mesh topology.	98
5.14	All-Reduce bandwidth and link utilization heat map over various network topologies.	100
5.15	TACOS All-Reduce bandwidth and link utilization compared to BlueConnect and Themis.	102
5.16	All-Reduce bandwidth of TACOS and relative speedups compared to Themis, MultiTree, and C-Cube.	103
5.17	Network utilization of TACOS-synthesized and ring algorithms during the execution of an All-Reduce.	105
5.18	Synthesis time of TACOS and TACCL for various-sized homogeneous two-dimensional mesh and three-dimensional hypercube topologies.	106
5.19	End-to-end training time of GNMT, ResNet-50, and Turing-NLG	107
5.20	End-to-end training time breakdown of ResNet-50 and MSFT-1T over a 1,024-NPU three-dimensional torus.	107
6.1	An example limitation of TACOS when synthesizing a unicast pattern.	114
6.2	Visualization of process groups over a six-NPU cluster.	119
6.3	Defining collectives patterns in a list of conditions.	122
6.4	BFS search algorithm to find the path of a chunk.	125

6.5	Synthesizing a All-Gather collective algorithm for a process group.	128
6.6	Example synthesis of a Reduce operation.	129
6.7	TEN representation of a heterogeneous network topology.	130
6.8	Removing TEN links in a heterogeneous network.	131
6.9	All-to-All synthesis time of PCCL for small two-dimensional mesh.	134
6.10	PCCL synthesis time of All-to-All algorithm of size 8–512 MiB.	135
6.11	All-to-All bandwidth of PCCL versus CCLs and collective speedup over heterogeneous 2D switch topology.	136
6.12	Normalized All-to-All bandwidth when the entire two-dimensional mesh cluster is executing a All-to-All collective.	137
6.13	PCCL-synthesized collective algorithm of two process groups.	138
6.14	Normalized All-to-All bandwidth of PCCL-synthesized algorithm over the baseline direct, as the two-dimensional mesh size and the number of process groups gradually increase.	138
6.15	Normalized link utilization heat map of PCCL-synthesized versus direct collective algorithms.	139
6.16	Network bandwidth utilization of PCCL-synthesized and direct algorithms over time.	140
6.17	Normalized All-to-All bandwidth over CCLs, when the number of process groups increase.	141

LIST OF ACRONYMS

AI	Artificial Intelligence
ASIC	Application-Specific Integrated Circuit
BFS	Breadth-First Search
CCL	Collective Communication Library
CPU	Central Processing Unit
DBT	Double Binary Tree
DI	Direct
DLRM	Deep Learning Recommendation Model
DMA	Direct Memory Access
DNN	Deep Neural Network
DP	Data Parallelism
EP	Expert Parallelism
FC	Fully-Connected
FSDP	Fully Sharded Data Parallel
GPU	Graphics Processing Unit
HC	Hypercube
HO	Homogeneous
HP	Hybrid Parallelism
HPC	High-Performance Computing
HT	Heterogeneous
ILP	Integer Linear Programming
ITL	Inter-Token Latency

LLM Large Language Model
LP Linear Programming
MCM Multi-Chip Module
ML Machine Learning
MLP Multi-Layer Perceptron
MoE Mixture-of-Experts
MP Model Parallelism
MPI Message Passing Interface
MWU Multiplicative Weight Update
NCCL NVIDIA Collective Communication Library
NIC Network Interface Card
NoC Network-on-Chip
NP Nondeterministic Polynomial
NPU Neural Processing Unit
PP Pipeline Parallelism
QP Quadratic Programming
RCCL ROCm Collective Communication Library
RDMA Remote Direct Memory Access
RHD Recursive Halving-Doubling
RI Ring
SMT Satisfiability Modulo Theories
SW Switch
TCO Total Cost of Ownership
TEN Time-Expanded Network
TP Tensor Parallelism
TPU Tensor Processing Unit
TTFT Time to First Token
XML Extensible Markup Language
ZeRO Zero Redundancy Optimizer

SUMMARY

Foundation machine learning (ML) models have emerged as one of the most prominent applications in modern computing, exemplified by mixture-of-experts-based large language models. The immense resource demands of these models have driven the development of large-scale, high-performance computing platforms tailored for artificial intelligence workloads. In such distributed platforms, both model parameters and data are partitioned and processed across numerous neural processing units, requiring frequent synchronization of activations and gradients through collective communication operations. As collective communication constitutes a primary bottleneck in distributed ML, optimizing its efficiency remains a critical research challenge.

This dissertation explores software-hardware optimizations for collective communications to better understand the tightly coupled design space of networking in distributed ML platforms. First, it introduces ASTRA-sim2.0, an end-to-end simulation and modeling framework that enables comprehensive design space exploration of distributed ML platforms with arbitrary parallelization strategies and multi-dimensional networks. Second, it presents LIBRA, which enhances the bandwidth utilization of hierarchical collective communication algorithms by optimizing multi-dimensional network topologies via analytical modeling. Finally, the dissertation proposes two collective communication algorithm synthesizers, TACOS and PCCL, which automatically generate optimized collective communication algorithms for arbitrary network topologies through algorithmic approaches. Together, the dissertation underscores the significance of judicious software-hardware approaches in achieving efficient collective communication for large-scale distributed ML platforms.

CHAPTER 1

INTRODUCTION

Generative artificial intelligence (AI) models have substantially increased the demand for training and serving large-scale workloads. Large language models (LLMs) with billions of parameters (i.e., weights) exemplify this trend [1, 2]. The mixture-of-experts (MoE) paradigm further enables massive model scaling while keeping computational requirements manageable [3, 4]. For instance, state-of-the-art MoE-based LLMs often contain trillions of parameters [5, 6]. Such models power some of today’s most demanding applications, including OpenAI ChatGPT [7], Google Gemini [8], and Microsoft Copilot [9].

The computational requirements for machine learning (ML) model training have increased by an average factor of $4.7\times$ per year [10]. Given the massive scale of these workloads, training or serving large ML models on a single compute device is impractical [11]. Large models cannot fit within the memory of a single device, and even when they do, the computational demand necessitates distributed ML. For example, GPT-3 [1] would take 355 years to train on a single NVIDIA V100 GPU [12]. This limitation has driven the widespread adoption of distributed ML approaches, where multiple compute devices each handle a portion of the model and data [13].

These massive resource requirements and the distributed ML paradigm have fueled the development of specialized high-performance computing (HPC) platforms tailored for AI, often called AI supercomputers. Such platforms are particularly important for large deep neural network (DNN) models, including LLMs, due to their substantial compute and memory demands [11]. Examples include Google Cloud TPU [14], Meta Research Super-Cluster [15], Cerebras CS-2 [16], and Tenstorrent Galaxy [17]. These systems integrate multiple neural processing units (NPU)s—compute devices for ML such as graphics processing units (GPU)s, central processing units (CPU)s, tensor processing units (TPU)s, or

custom application-specific integrated circuits (ASICs)—at their endpoints. The NPUs are tightly interconnected via custom high-speed fabrics to enable both scale-up and scale-out, reaching thousands or even hundreds of thousands of NPUs.

1.1 Challenges

1.1.1 Collective Communication

As discussed, modern foundation models and other large models often have a memory footprint during training and inference that exceeds the capacity of a single NPU [11]. To address this, both the model and the training dataset must be sharded and distributed across multiple NPUs.

When multiple NPUs process distinct shards of models and data, they must periodically synchronize computation results, such as forward and backward pass activations and weight gradients [18]. Thus, it is inevitable that devices communicate and synchronize data, including forward activations, weight updates, and input gradients [19]. These synchronization steps impose substantial communication overhead among NPUs and are typically implemented through collective communication operations (collectives) [20], as defined by the message passing interface (MPI) [21].

Communication represents a major challenge for distributed ML systems because models and training data are dispersed across devices, requiring frequent synchronization [19]. Indeed, at the scale of modern AI clusters and workloads, collective communication has become a primary bottleneck in distributed ML execution, affecting both training and inference [22, 23, 23, 24, 25].

1.1.2 Design Space of Distributed Machine Learning

The importance of optimizing collective communication in distributed ML continues to grow. However, the design space of distributed ML is complex due to intertwined software–hardware interactions. Designing an efficient distributed ML system involves many

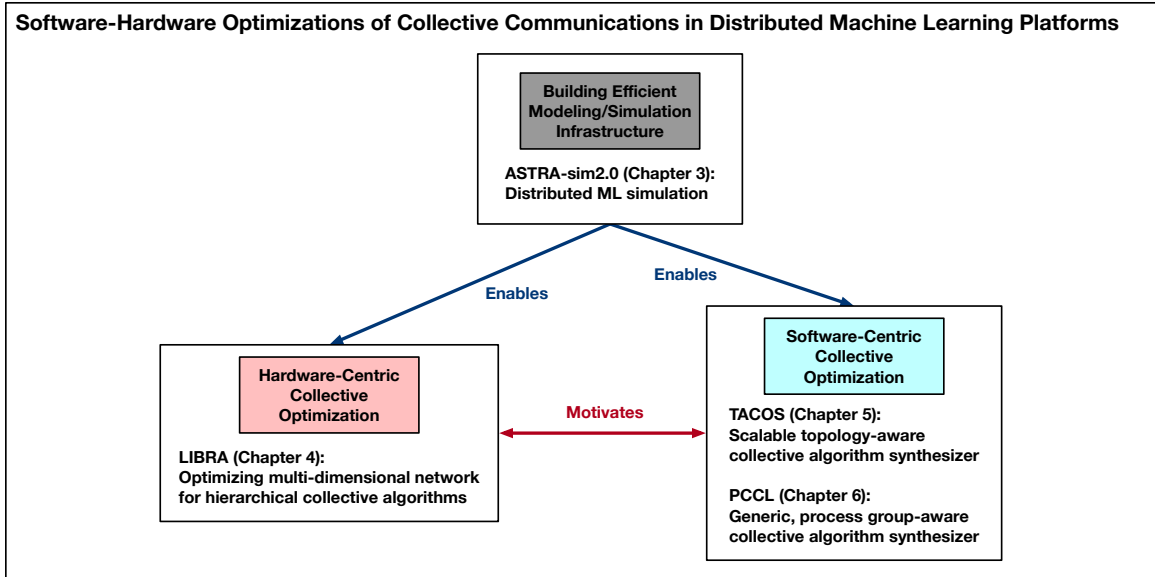


Figure 1.1: Contributions of this dissertation.

interdependent choices, including parallelization strategies, NPU performance, memory bandwidth, network topology and bandwidth, scheduling policies, and other parameters. Frontier distributed ML models often require massive resources, sometimes exceeding 100,000 NPUs [26], with training lasting months, which further enlarges and complicates the design space.

Navigating this space requires careful software–hardware co-design considerations. A full-stack modeling and simulation infrastructure at scale is essential to enable research in distributed ML. Software optimizations must account for hardware characteristics. For example, the performance of collective communication depends not only on the chosen algorithm but also on the underlying network topology; ignoring topology prevents full optimization for a target system. Similarly, hardware-centric techniques require a deep understanding of software aspects, such as the deployed collective communication algorithms and the workload’s parallelization strategies.

1.2 Thesis Contributions

To this end, this dissertation examines the design space of collective communication for distributed ML using combined software–hardware approaches. The contributions of this dissertation are summarized in Figure 1.1. First, it develops a scalable modeling and simulation infrastructure for full-stack distributed ML. Next, it proposes optimizations for multi-dimensional network topologies—especially improving bandwidth utilization—by accounting for the collective algorithms used on the platform. Finally, it explores the automatic synthesis of collective communication algorithms through a detailed analysis of the underlying hardware and network topology.

1.2.1 ASTRA-sim2.0: Modeling Hierarchical Networks for Distributed Machine Learning at Scale

In Chapter 3, we motivate the need for a full-stack modeling and simulation infrastructure for distributed ML platforms as a foundational research enabler for this dissertation. We introduce ASTRA-sim2.0, a simulation infrastructure for distributed ML platforms at scale.

1.2.2 LIBRA: Enabling Workload-Aware Multi-Dimensional Network Topology Optimization for Distributed Machine Learning

In Chapter 4, we explore hardware-centric approaches for software-aware optimization. We introduce a hierarchical collective communication algorithm and propose LIBRA, an infrastructure for optimizing multi-dimensional network topologies to maximize bandwidth utilization.

1.2.3 TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning

In Chapter 5, we shift to software-centric optimization with hardware awareness. We argue that manually designing topology-aware collective algorithms is prohibitive and motivate the need for an automated topology-aware collective algorithm synthesizer. We introduce TACOS, a highly scalable synthesizer for topology-aware collective algorithms.

1.2.4 PCCL: Process Group-Aware Scalable and Generic Collective Algorithm Synthesizer

In Chapter 6, we extend the requirement for a topology-aware collective synthesizer to support arbitrary collective patterns and process-group awareness. We propose PCCL, which supports arbitrary network topologies and collective patterns and maximizes network utilization through process-group awareness.

1.3 Thesis Statement

Software and hardware optimizations tailored for collective communication can enhance the end-to-end performance of distributed machine learning platforms.

1.4 Thesis Overview

The rest of this dissertation is organized as follows:

- In Chapter 2, we review the background knowledge relevant to this dissertation.
- In Chapter 3, we introduce ASTRA-sim2.0, a full-stack simulation infrastructure for distributed ML.
- In Chapter 4, we present LIBRA, an infrastructure for optimizing multi-dimensional network topologies to maximize bandwidth utilization.

- In Chapter 5, we describe TACOS, a highly scalable, topology-aware collective algorithm synthesizer.
- In Chapter 6, we propose PCCL, a topology-aware collective algorithm synthesizer that supports arbitrary network topologies, collective patterns, and process groups.
- In Chapter 7, we conclude this dissertation by summarizing the key contributions and outlining potential future directions.

CHAPTER 2

BACKGROUND

2.1 Distributed Machine Learning

2.1.1 Neural Processing Unit

In this dissertation, we use the term neural processing unit (NPU) as a general abstraction encompassing various compute devices used in machine learning (ML). NPU includes graphics processing units (GPUs), tensor processing units (TPUs), and custom ML accelerators.

2.1.2 Parallelization Strategy

Each parameter, including model weights and input data, is distributed across NPUs. The parallelization strategy defines how computations and data are distributed across the cluster [27]. The two most pervasive parallelism strategies are data parallelism (DP) and model parallelism (MP). In MP, there are two common methods to distribute the model parameters: tensor parallelism (TP) and pipeline parallelism (PP).

MP partitions the ML model across a subset of NPUs, while DP distributes input data across multiple NPUs [13]. DP distributes the mini-batch across NPUs and synchronizes weight gradients during the backward pass [13, 27]. MP, on the other hand, distributes a model evenly across NPUs and communicates forward activation and input gradients [13]. Within MP, TP divides the model vertically and distributes it across NPUs, reducing the memory requirement of each NPU during execution [13]. PP distributes model layers across nodes and processes micro-batches in a pipelined manner [28, 29]. Other parallelization strategies are also actively being investigated [11, 30, 31].

MP and DP are orthogonal patterns; therefore, MP and DP can be used simultaneously

in a hybrid-parallel scheme [32].

2.1.3 Synchronous Training and Inference

When models and data are distributed across NPUs, it is crucial to decide when and how to synchronize such distributed information across them. The asynchronous approach, as the name suggests, communicates among NPUs in an asynchronous manner. Therefore, asynchronous training suffers from the convergence problem [25] and is more complex to implement and maintain [13].

The most common approach is, therefore, synchronous distributed ML. In this mechanism, all nodes work on their own data and synchronize the distributed information altogether before proceeding to the next iteration, usually in the form of collective communications [27, 33].

2.2 Collective Communication Pattern

2.2.1 Collective Communication Pattern

Message passing interface (MPI) defines several synchronization patterns among NPUs [21]. These are known as collective communication patterns (collectives, for short) and are extensively used in distributed ML clusters [18, 20].

Figure 2.1 illustrates the common MPI collective patterns used in distributed ML platforms. Each circle represents a chunk, which is a logical unit of network transfers in collective communication. The precondition specifies the initial buffer contents of NPUs, while the postcondition describes the final buffer status after the collective operation. Although not shown in Figure 2.1, All-to-Allv is a generalized version of All-to-All where each NPU can have different numbers of chunks in the pre/postcondition, common in MoE-based LLMs [34].

Some collective patterns (Reduce, Reduce-Scatter, and All-Reduce) require the reduction of chunks, typically using arithmetic addition in the context of distributed ML.

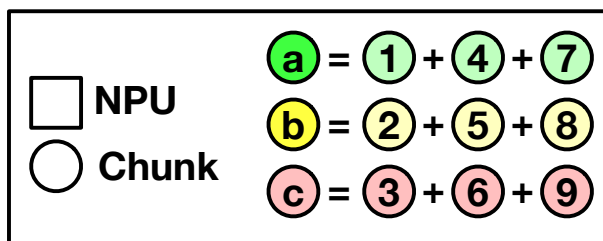
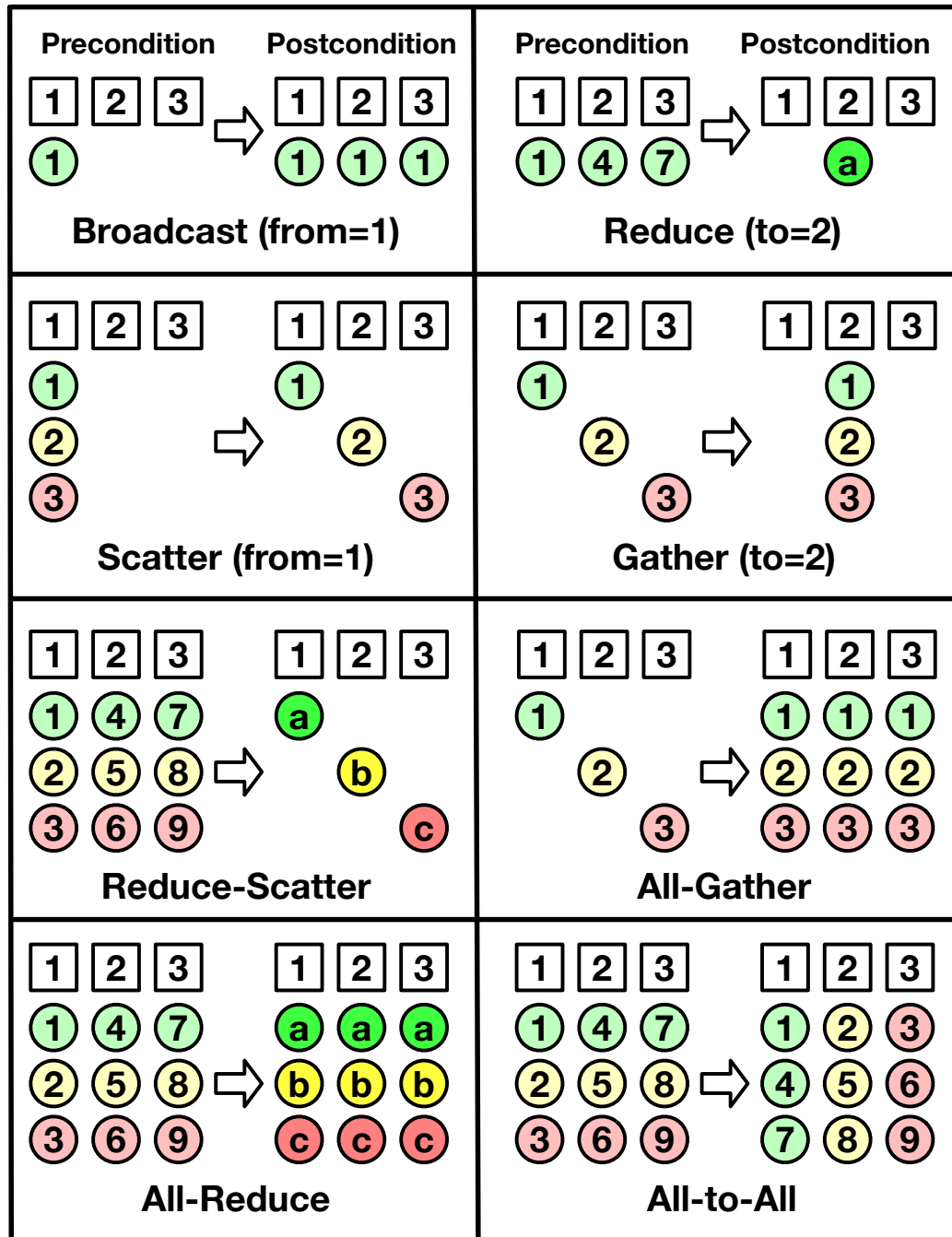


Figure 2.1: Definition of MPI collective communication patterns. Each square denotes an NPU, whereas each circle denotes a chunk.

Table 2.1: Collectives required by each parallelization strategy.

Parallelism	Reduce-Scatter	All-Gather	All-Reduce	All-to-All	Pt-to-Pt
Data			✓		
Tensor	✓	✓			
Expert				✓	
Pipeline					✓

2.2.2 Common Collective Communication Patterns

Different ML model architectures and parallelization strategies require different sets of collective patterns to be executed. Such a paradigm is summarized in Table 2.1.

Required collective patterns change depending on the workload type and the parallelization strategy. For training, with a synchronous training approach, the most pervasive collective pattern is All-Reduce [22]. The All-Reduce can be conceptualized as two sequential stages: Reduce-Scatter followed by All-Gather, and it is the most prevalent pattern in synchronous NPU-to-NPU ML executions [22].

Mixture-of-experts (MoE) LLM models are an example class of workloads that require specific collective communication patterns. MoE-based models require All-to-All (and sometimes All-to-Allv) collectives for their use of expert parallelism (EP) [5, 34]. Although not shown in Table 2.1, embedding table lookup in deep learning recommendation models (DLRM) [35] is another example of a model requiring the use of the All-to-All collective pattern.

2.3 Collective Communication Algorithm

2.3.1 Collective Communication Library

Real systems execute collectives using collective communication libraries (CCLs). NVIDIA Collective Communication Library (NCCL) [36], AMD ROCm Collective Communication Library (RCCL) [37], and Intel oneAPI Collective Communications Library (oneCCL) [38] are examples of GPU-based CCLs currently in use.

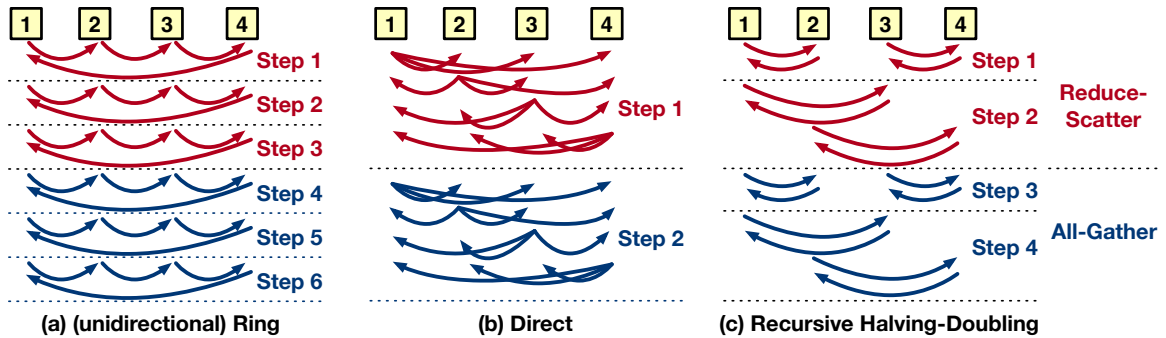


Figure 2.2: Textbook All-Reduce algorithms and their traffic patterns.

CCLs execute collective communications by implementing collective communication algorithms. A collective communication algorithm defines how each chunk should traverse (i.e., be sent and received) over the network to execute a target collective pattern [39]. Collective algorithms define the static path for each chunk during the execution of collective communication. For a given collective communication pattern, there can be several distinct routing algorithms to implement it.

Collective communication algorithms are often also called collectives for short. In this dissertation, therefore, we use the term collective liberally to represent both collective communication patterns and collective communication algorithms.

2.3.2 Textbook Collective Communication Algorithm

CCLs often come up with out-of-the-box implementations for commonly used collectives. Such algorithms often assume simple traffic patterns but are optimized for specific network types and collective patterns. In this dissertation, textbook collective communication algorithms denote such baseline collective algorithms.

As an example, the textbook All-Reduce algorithms—ring, direct, and recursive halving-doubling—are illustrated in Figure 2.2. Specifically, the traffic pattern of each collective algorithm is shown. In Reduce-Scatter, each red arrow represents sending and adding a chunk to the local data. For All-Gather, each blue arrow signifies forwarding a chunk. Direct algorithms are often also called one-shot or all-pairs algorithms [40, 41], and are used

interchangeably throughout this dissertation.

The ring algorithm assumes NPUs are logically connected in a ring. Each NPU sends a chunk to its neighbor while receiving a chunk from its other neighbor. This process is repeated for $2 \times (N - 1)$ steps, where N is the number of NPUs, until every NPU receives all chunks to satisfy their All-Reduce postcondition. Direct [27], recursive halving-doubling [42], and double binary tree [43] are additional examples of All-Reduce collective algorithms.

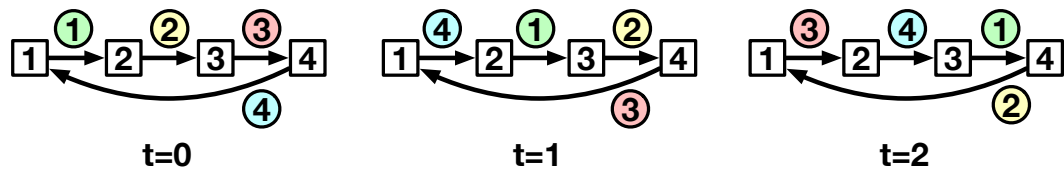
The intuition for having different algorithms for the same collective pattern is to minimize contention on distinct physical network topologies. To enhance network utilization, a collective can be decomposed into multiple smaller chunks which can be run in parallel [19].

2.3.3 Topology-Aware Collective Algorithm

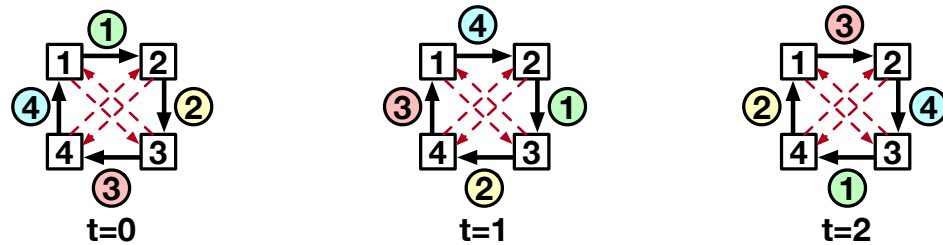
As discussed, Figure 2.2 depicts the traffic pattern of textbook collective algorithms. Depending on the network connectivity, however, each step may encounter link congestion and underutilization. When these patterns are mapped over their preferred topologies, they can efficiently utilize links in the topology every cycle without any contention. However, executing a collective algorithm on its unpreferred physical topology leads to contention (i.e., oversubscription) on certain links and low or even no utilization (i.e., undersubscription) on others.

Note that the physical topology in Figure 2.3(a) is also a unidirectional ring. Consequently, the ring collective algorithm utilizes 100% of the available network bandwidth and incurs no network congestion. Thus, Figure 2.3(a) is a topology-aware (unidirectional) ring algorithm for a ring topology. In this case, the ring algorithm fully leverages the physical network without network congestion, yielding optimal collective performance [44].

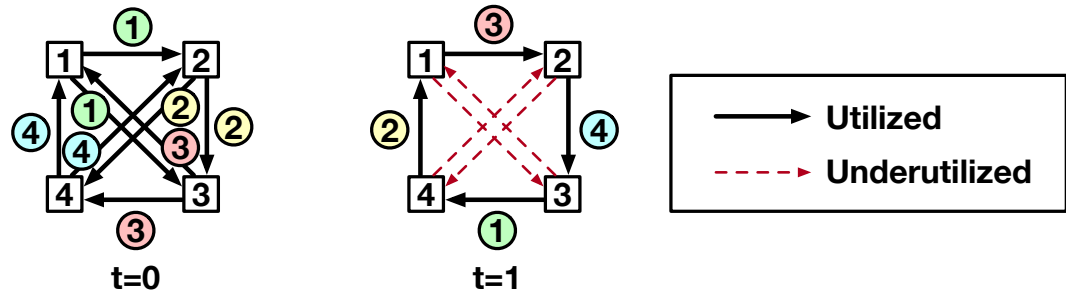
In contrast, Figure 2.3(b) shows the same ring algorithm executed over an arbitrary physical network with eight links. Although it performs a valid All-Gather collective, it



(a) Unidirectional Ring All-Gather collective algorithm over a unidirectional Ring topology.



(b) Unidirectional Ring All-Gather collective algorithm over a custom 8-link topology.



(c) Topology-aware All-Gather collective algorithm over a custom 8-link topology.

Figure 2.3: Examples of topology-aware and topology-unaware All-Gather collective algorithms.

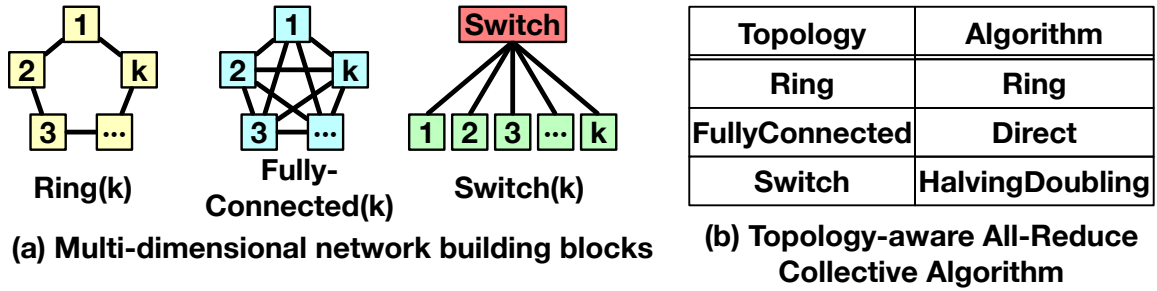


Figure 2.4: Multi-dimensional network building blocks, and their corresponding topology-aware All-Reduce algorithm.

underutilizes the available network resources and therefore does not achieve optimal performance. In this context, the ring algorithm is not topology-aware. Therefore, Figure 2.3(b) depicts that the ring algorithm is not topology-aware over this example custom topology, resulting in network underutilization. Figure 2.3(c) shows an example of a topology-aware All-Gather algorithm tailored for the same custom network, achieving a 50% speedup compared to the topology-unaware ring algorithm.

As shown in this example, we define a topology-aware collective communication algorithm as a traffic pattern that maximizes network resource utilization, minimizes underutilization, and thereby achieves maximized collective performance. For example, ring, direct, and recursive halving-doubling are commonly used All-Reduce algorithms [42]. These algorithms are topology-aware collective algorithms designed for ring, fully-connected, and switch networks, respectively, ensuring that they do not introduce link contention when running on their respective physical topologies. Figure 2.4 lists common network building blocks and their corresponding topology-aware collective algorithms.

CHAPTER 3

ASTRA-SIM2.0: MODELING HIERARCHICAL NETWORKS FOR DISTRIBUTED MACHINE LEARNING AT SCALE

3.1 Motivation

As discussed, having a full-stack modeling infrastructure for the design space of distributed ML is imperative. It is one of the research enabler to model and optimize the performance of collective communication in distributed ML platfors. This chapter discusses the development of ASTRA-sim2.0, full-stack end-to-end modeling and simulation infrastructure for distributed ML.

ASTRA-sim [27, 45] is an existing open-source infrastructure. ASTRA-sim aims to model the complete software-hardware co-design stack of distributed training systems. Notably, ASTRA-sim only targets training workloads. ASTRA-sim is a promising tool for exploring the design space of distributed training systems and has been leveraged by several recent works [19, 25, 46, 47, 48, 49].

However, as discussed above, ASTRA-sim has limitations and can be extended to model the full-stack design space of distributed ML. In this chapter, we identify limitations in ASTRA-sim that restrict it from supporting arbitrary end-to-end workloads, including the modeling of hierarchical network topologies at scale.

On the software end, there has been a growing interest in new parallelism strategies, both hand-designed such as fully sharded data parallel (FSDP) [50], 3D hybrid parallelism [51, 52], zero redundancy optimizer (ZeRO) [11], expert parallelism (EP) [53], and synthesized [54, 55]. These strategies enable the training and inference of large models, splitting data sets, parameters and optimizer state, while optimizing for communication [23, 33]. ASTRA-sim did not have a strong motivation to support arbitrary parallelism when it

was proposed as there were a handful of parallelism strategies such as data parallelism [56], tensor parallelism, and simple hybrid [32], and only supports distributed ML training for that reason.

The hardware landscape for distributed ML has been evolving rapidly as well. State-of-the-art systems extensively deploy multi-dimensional network topologies with hierarchical bandwidths to interconnect NPUs [57, 58, 59, 60, 61]. This is because increasing the aggregated network bandwidth per NPU through a single dimension is fundamentally limited by the link technology the network is leveraging (e.g., current NVLink [62] offers up to 450 GiB/s). Naively scaling out through network interface cards (NICs) is also not practical due to engineering limitations such as dollar-cost, power, and thermal problems. Meanwhile, wafer-scale systems [63, 64] tackle the communication problem by fabricating NPU chipllets on a large-wafer with low-dimensional, high on-chip networking, then scaling out such wafers using NICs. In order to study these technology-driven network landscapes, there is a need for a mechanism to represent and study arbitrary multi-dimensional topologies at scale, with different shapes and bandwidth configurations. ASTRA-sim natively uses the Garnet simulator [65] from gem5 as its network layer, which has limitations in modeling such platforms.

In this chapter, we address the aforementioned limitations of ASTRA-sim and enhance it by supporting hierarchical network simulation at scale using flow-based analytical modeling. Flow-based analytical modeling was able to achieve a 0.23% mean difference compared to a packet-based ns-3 simulator [66] while achieving up to a $169,443\times$ speedup for a 32-NPU system. Using these enhancements, we present case studies to deliver key insights about future platforms. We compared conventional multi-dimensional and wafer-scale systems and found that with appropriate collective scheduling and parallelization strategy designs, conventional systems can match wafer-scale systems' performance, whereas wafer-scale shows up to $2.51\times$ better collective time when scaled.

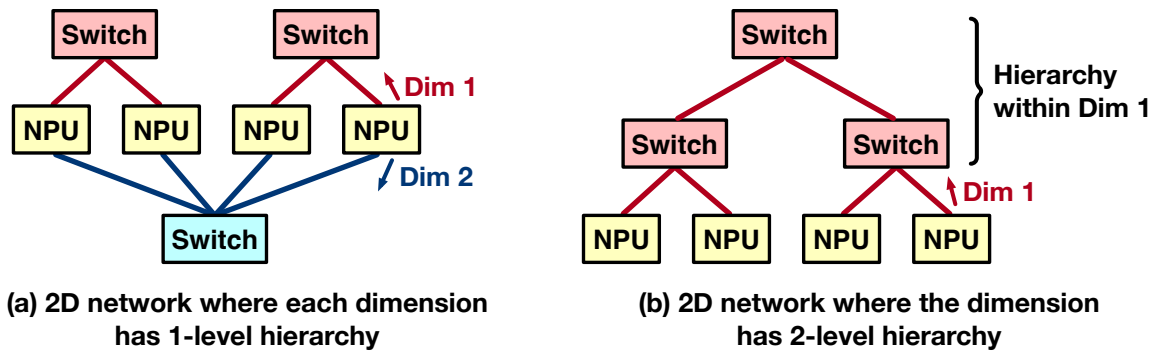


Figure 3.1: An example definition of network dimension versus hierarchy.

3.2 Background

3.2.1 Multi-dimensional Network

A multi-dimensional network is defined as a network topology where each NPU has multiple independent connectivity options, which can be accessed in parallel to communicate with other NPUs. This concept is equivalent to the definition of multi-rail networks [67, 68].

It is worth noting that, in our terminology, a network dimension is distinct from adding a hierarchy within a network dimension. Each dimension of the network is directly accessible to the NPU via explicit ports and pins in parallel. In contrast, within a dimension, the implementation may choose to use a hierarchy, such as a hierarchy of switches instead of a single large switch.

This distinction is depicted in Figure 3.1. While both topologies utilize three physical switches, Figure 3.1(a) represents a two-dimensional network since NPUs have two distinct switch-based networks accessible in parallel, whereas Figure 3.1(b) showcases a one-dimensional topology using a two-level switch hierarchy within dimension one.

3.2.2 ASTRA-sim

Vast design choices of distributed machine learning, combined with diverse hardware configurations, create an enormous software-hardware design space for distributed ML. Such

an enormous design space cannot be solely explored by only leveraging physical systems, especially at scale. Therefore, a simulation-based mechanism to quickly model and profile distributed ML platforms is necessary for design-space exploration.

ASTRA-sim [27] is a distributed training simulation framework that exactly addresses this demand. ASTRA-sim concentrates on the modeling of distributed training. It codifies the complex software-hardware search space of distributed training platforms via three abstraction layers: (i) workload, (ii) system, and (iii) network.

The workload layer lets the user describe and define target DNN models, target parallelization strategies, and training loops. It implements the DNN model, its parallelization strategy such as DP, TP, and compute-communication ordering.

The system layer implements collective communication algorithms, schedules compute and communication operations, and manages compute-communication overlap. The system layer provides various collective communication algorithm implementations (e.g., All-Reduce, All-to-All) and also manages pipelining and scheduling of communication operations. Compute times are fed in via external NPU models [69] or real system measurements.

Finally, the network layer models the hardware-software components of the network and simulates the traffic issued by the system layer. Communication times are computed using a network simulator. The default simulator is Garnet [65] from gem5. It reports detailed system- and network-level behaviors as well as end-to-end training throughput.

3.3 Challenges

Even though the ASTRA-sim framework has allowed brisk navigation of the distributed training search space [19, 25], the tool as-is does not meet the demand to capture more complex target platforms. Evidently, ASTRA-sim only targets distributed training of ML models, for example.

In this section, we motivate the need to extend the ASTRA-sim toolchain to enable

modeling of state-of-the-art and futuristic distributed ML systems. Specifically, we call for the ability to model multi-dimensional hierarchical networks, especially at scale.

3.3.1 Ability to Model Multi-Dimensional Networks

From the necessity to distribute and synchronize models and data across devices, large-scale distributed ML is usually communication-bound [22, 23, 33], as discussed in previous chapters. Therefore, in order to maximize ML execution performance, state-of-the-art systems mix and match a plethora of networking technologies. This usually results in a system having multi-dimensional network topologies with heterogeneous bandwidth configurations [19]. As an instance, NVIDIA DGX-A100 [58] exploits a two-dimensional network topology whose first dimension is NVIDIA NVLink [62], then scaled out using InfiniBand [70] or Ethernet [71, 72] technologies. The Google Cloud TPUv4 [59] leverages a three-dimensional torus where each inter-core interconnect runs at 448 Gib/s [73].

Although ASTRA-sim can, in principle, target multi-dimensional networks, it only supports a limited set of pre-defined network topologies—two- and three-dimensional torus. In order to study different topologies, one must implement both a new network topology in Garnet and its corresponding topology-aware hierarchical collective algorithm, which significantly undermines ASTRA-sim’s strength of swift distributed training system modeling and performance analysis.

Therefore, it is necessary to attach a more powerful network backend to the ASTRA-sim framework for rapid design-space exploration of state-of-the-art and futuristic distributed ML platforms. It must define a systematic mechanism to represent arbitrary multi-dimensional network topologies at scale. With such notation, the user can swiftly represent arbitrary multi-dimensional networks, instead of manually implementing network topology files and their corresponding collective communication algorithms.

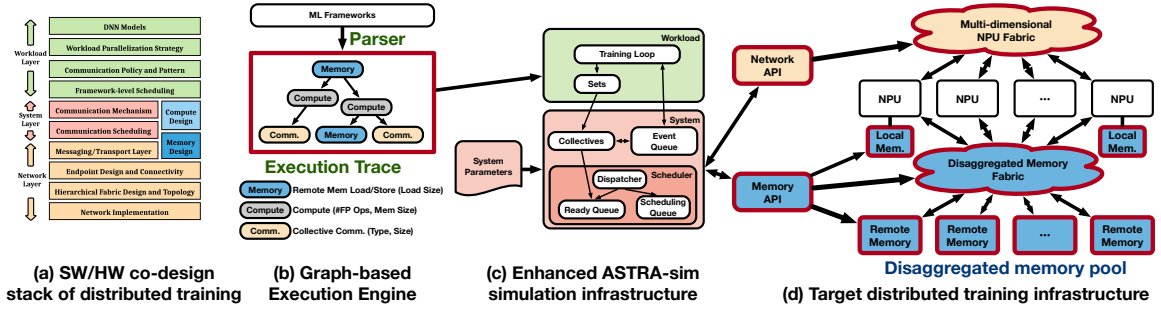


Figure 3.2: Overview of the proposed infrastructure for modeling next-generation distributed ML platforms.

3.4 ASTRA-sim2.0

In this section, we introduce the ASTRA-sim2.0 approaches to overcome the aforementioned ASTRA-sim limitations to model the full-stack design space of distributed ML. We introduce the new features we added to ASTRA-sim and describe how they are implemented. The intricate design space of distributed ML and the updated ASTRA-sim2.0 infrastructure is summarized in Figure 3.2. Specifically, the components extended in ASTRA-sim2.0 from the original ASTRA-sim to model emerging platforms are marked in bold.

Although not discussed in this dissertation in detail, we added arbitrary workload and remote memory models to ASTRA-sim2.0. Specifically, we added arbitrary parallelism support by encoding parallelism strategies as execution traces and developing a parser to translate these into compute and communication tasks with dependencies. For the memory models, we augmented ASTRA-sim with the ability to model local and networked remote pooled memories.

Rather, this dissertation discusses the details of the network modeling. For network support, we developed a taxonomy to define hierarchical topologies and created a flow-based analytical model to estimate performance when running topology-aware collective algorithms over the physical topology.

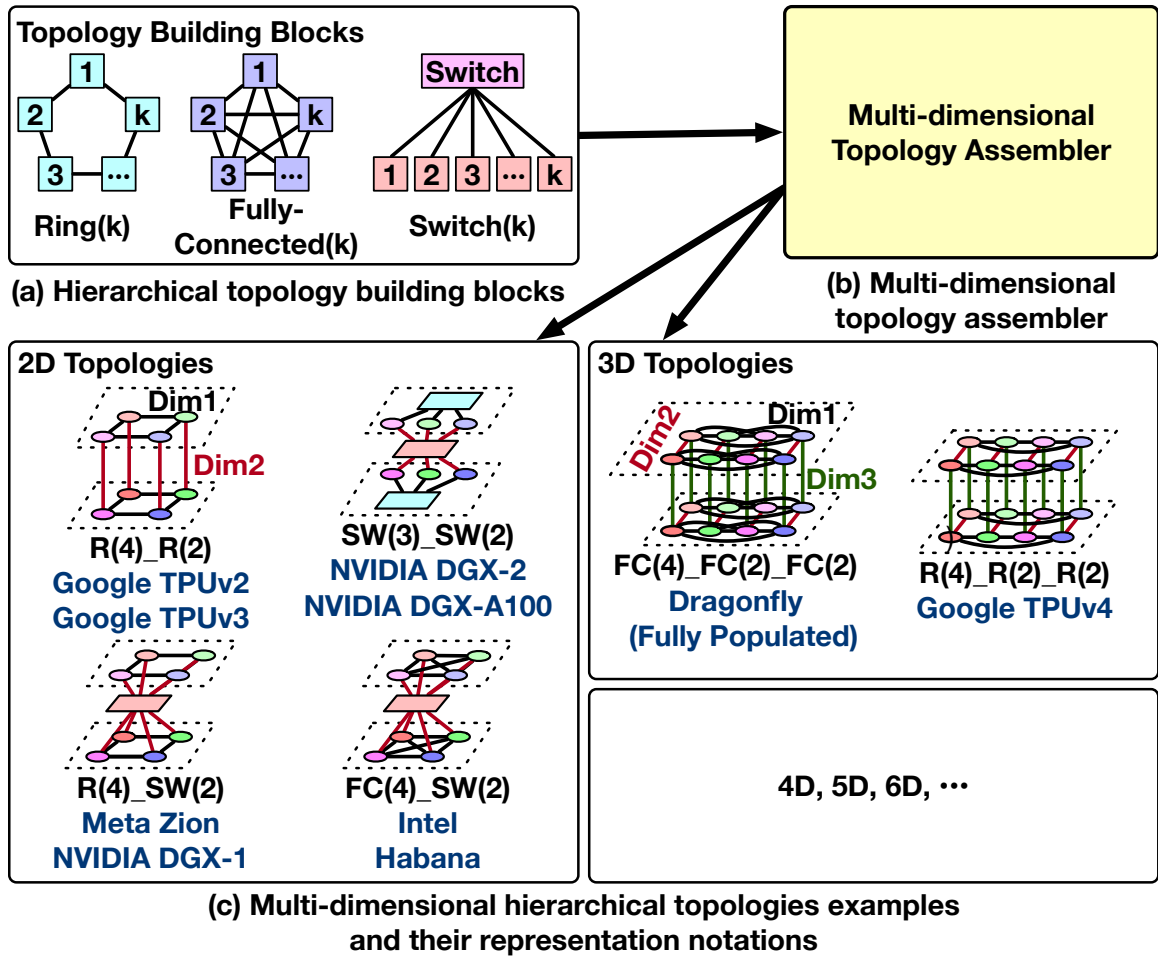


Figure 3.3: Proposed multi-dimensional network topology representation in ASTRA-sim2.0.

3.4.1 Multi-dimensional Network Representation

In order for users to quickly target arbitrary multi-dimensional network topologies, it is crucial to design a generic notation to represent such multi-dimensional shapes. In this chapter, we propose a taxonomy that constructs a multi-dimensional topology by stacking network building blocks in a hierarchical manner.

Figure 3.3(a) shows the three network building blocks for hierarchical network topologies utilized in this chapter: ring (R or RI), fully-connected (FC), and switch (SW). $RI(k)$ connects k NPUs in a ring shape (i.e., two connections per NPU). $FC(k)$, on the other hand, offers one-shot all-to-all connectivity among all pairs of NPUs. Finally, $SW(k)$ connects

Table 3.1: Network building blocks and its corresponding topology-aware collective communication algorithm.

Network Building Block	Topology-Aware Collective Algorithm
Ring	Ring [74]
Fully-Connected	Direct [42]
Switch	Recursive Halving-Doubling [42]

all k NPUs using an external switch fabric.

We chose these three as the network building blocks because they have corresponding well-known topology-aware collective algorithms, as summarized in the previous chapter Section 2.3.3. They are summarized in Table 3.1. Even if the underlying system uses other topologies, they are logically reduced into one of these building blocks in the CCL [36, 38]. This is a unique feature of distributed ML platforms.

Multi-dimensional topologies can be generated by assembling these blocks in an arbitrary hierarchical manner, as illustrated in Figure 3.3(b). Multi-dimensional network topologies are created by stacking network building blocks. A handful of example constructed topologies are shown in Figure 3.3(c). Examples of multi-dimensional hierarchical topologies, their shape notations, and corresponding distributed ML frameworks are shown.

RI(4)_RI(2) simply denotes a two-dimensional torus with eight NPUs in total, where the first dimension is RI(4) and two such dimension-one networks are interconnected using an RI(2) topology. RI(4)_SW(2), on the other hand, has the same dimension one, but planes are scaled out using an external switch instead. An example three-dimensional topology from Figure 3.3(c) is FC(4)_FC(2)_FC(2), a fully-populated Dragonfly [75] topology with 16 NPUs. Also shown is RI(4)_RI(2)_RI(2), where the NPU placement is equivalent, but the topologies connecting them are substituted with ring, thereby resulting in a three-dimensional torus instead.

The number of network dimensions or the order of the building blocks is not restricted; thus, arbitrary-dimensional networks can easily be represented using the same notation.

Note that each example topology listed in Figure 3.3(c) corresponds to some state-of-the-art distributed ML platforms, demonstrating the power of our proposed representation in modeling the design space.

3.4.2 Flow-Based Analytical Network Modeling

We implemented a new analytical network backend and ported it to the ASTRA-sim2.0 framework to support arbitrary multi-dimensional network topologies. The following points summarize why an analytical equation-based network was sufficient for our purpose:

- There is a need for first-order design-space exploration (topology shape and bandwidth) of the target system at scale.
- Given the scale (thousands to tens of thousands of NPUs) of state-of-the-art and futuristic systems and ML models, cycle-level and packet-based simulation is too slow to be practical.
- Analytical equation-based modeling shows comparable results to packet-based simulations, and in fact closely matches real system measurements for a small system, as we show later.

To model communication between NPUs, the ASTRA-sim2.0 frontend delegates the network backend to simulate such communication and requests the backend to invoke a callback function to notify that the transmission is completed. This protocol is defined in the form of network application programming interface (API) methods [47], as in Figure 3.2(d).

Congestion-Unaware Modeling

First, we implement a fully congestion-unaware analytical network modeling. Whenever a communication flow is initiated by the workload layer of ASTRA-sim2.0, the congestion-unaware analytical network backend leverages a simple alpha-beta equation [76] to esti-

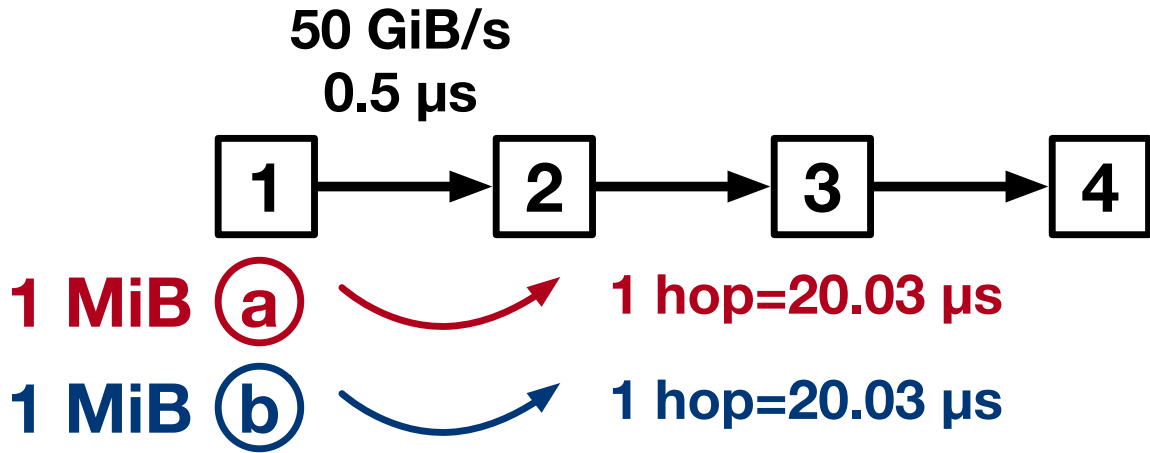


Figure 3.4: Example of congestion-unaware network modeling.

mate the communication delay instead of simulating actual packet-based network behaviors:

$$\text{Time} = (\text{LinkLatency} \times \text{Hops}) + \frac{\text{MessageSize}}{\text{LinkBandwidth}}$$

The analytical equation can be amended to consider other effects, such as wire propagation delay, as desired. For example, complex system and network optimizations, such as remote memory management or in-network collective communication, can further be captured by equations.

An example of congestion-unaware modeling is shown in Figure 3.4. The network models two flows, each of size 1 MiB, being transmitted from NPU 1 to NPU 4. Each link has 50 GiB/s bandwidth and 0.5 μs latency. Per the equation, each flow takes:

$$\text{Time} = (0.5\mu s \times 1 \text{ hop}) + \frac{1 \text{ MiB}}{50 \text{ GiB/s}} = 20.03 \mu s$$

As congestion-unaware network modeling is completely congestion-unaware, the two flows do not contend over the network. Therefore, the simulated time of the two flows is still 20.03 μs.

Congestion-Aware Modeling

This variation of the analytical modeling introduces first-order congestion modeling to overcome the limitations of the fully congestion-unaware modeling. In this implementation, instead of modeling end-to-end communication of a network communication, each flow is simulated link-by-link. When two flows are contending to use the same link, only one flow can pass through the link, while all other chunks are queued. The link only gets freed after the serialization delay:

$$\text{Serialization Delay} = \frac{\text{MessageSize}}{\text{LinkBandwidth}}$$

The chunk arrives at the next destination after accounting for both contention and serialization delays:

$$\text{Chunk Arrives Next NPU} = \text{LinkLatency} + \frac{\text{MessageSize}}{\text{LinkBandwidth}}$$

Figure 3.5 showcases the same example with congestion-aware modeling. As two flows are contending to use the same link 1→2, only one flow gets to use this link, while the other flow is queued. The other flow can only start leveraging this link after the serialization delay:

$$\text{Link Becomes Free} = \frac{1 \text{ MiB}}{50 \text{ GiB/s}} = 19.53 \mu\text{s}$$

Therefore, the simulation ends at 39.56 μs when the delayed chunk b reaches its destination NPU 2.

Notably, both implementations of the analytical network modeling run at the communication flow level. Unlike packet-based simulators such as Garnet or ns-3, the flow-based analytical model does not generate or simulate individual packets from the communication

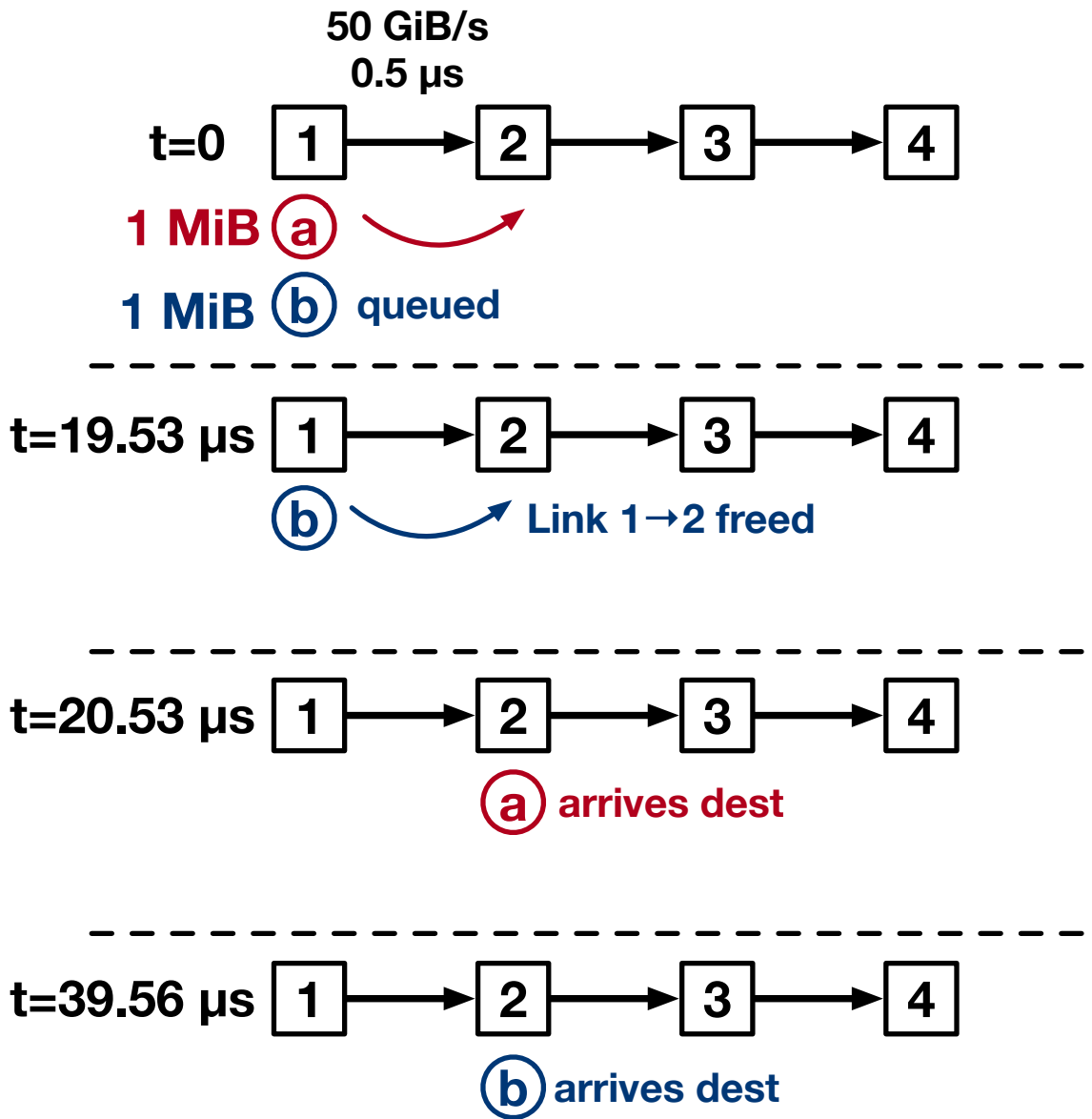


Figure 3.5: Example of congestion-aware network modeling.

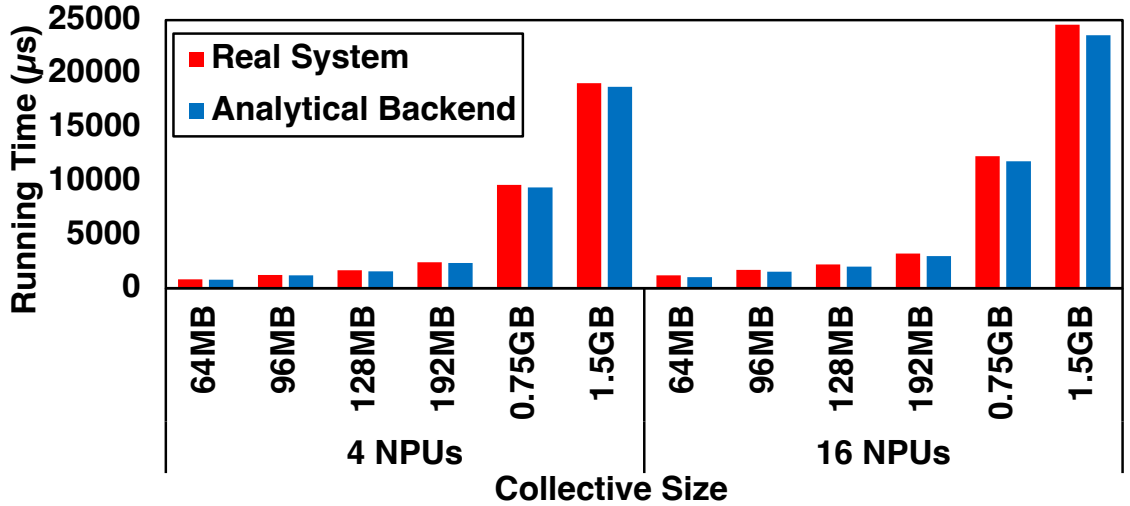


Figure 3.6: Analytical network backend validation over real-system measurements.

flow and models each flow analytically. Consequently, the flow-based analytical modeling showcases orders-of-magnitude better scalability and modeling time than packet-based simulations, as discussed next.

3.5 Methodology

3.5.1 Validation of Analytical Network Backend

In order to demonstrate the accuracy of the analytical network backend, we constructed two real systems and compared the running times of various-sized All-Reduce operations. The two real systems used NCCL v2.4.6 [36], which consist of 4 and 16 NVIDIA V100 GPUs [77] with a ring topology and 150 GiB/s NVLink [62] among the GPUs. Figure 3.6 shows the results. We ran 64 MiB to 1.5 GiB All-Reduce operations, and the results suggested that the mean error of the simulation across all configurations was 5%.

3.5.2 Simulation Time Speedup

To measure the simulation time improvement, we ran a 1 MiB All-Reduce simulation on a three-dimensional torus with 64 NPUs ($4 \times 4 \times 4$). On Garnet-based ASTRA-sim2.0, the

simulation took 21.42 minutes to finish. For the same configuration, the analytical backend took only 1.70 seconds, showing a $756\times$ speedup in simulation runtime. Furthermore, the analytical backend supported a three-dimensional torus with 4,096 NPUs ($16\times 16\times 16$) in just 3.14 seconds. This nearly three-orders-of-magnitude speedup shows the analytical network backend’s ability to profile large-scale systems quickly.

3.5.3 Compute Model

For all our experiments, we assumed an NPU compute power of 234 TFLOPS, as observed from the measurements of an A100 GPU [58].

3.6 Evaluations

In this section, we present comprehensive case studies that showcase the extended capabilities of ASTRA-sim2.0 and provide meaningful insights into the design space.

3.6.1 Flow-Based versus Packet-Based Network Modeling

We first analyze and establish the fidelity of analytical, flow-based modeling by comparing the results against a packet-based, event-driven network simulator, ns-3 [66]. Unlike the flow-based analytical modeling proposed in this chapter, ns-3 models packet-level behavior, as its backend creates and simulates individual packets from each network flow. This includes the overhead of packetization itself, as well as congestion control and arbitration protocols, packet spraying and load balancing, and protocols for scale-out networks such as remote direct memory access (RDMA).

It is important to note that we chose to compare the flow-based analytical model against the ns-3 simulator because it is the only packet-based network simulator seamlessly integrated with the ASTRA-sim2.0 infrastructure, though we acknowledge that ns-3 is specifically designed for scale-out networks. The analysis conducted here is one example of a comparison against packet-based network modeling, and we anticipate further comparisons

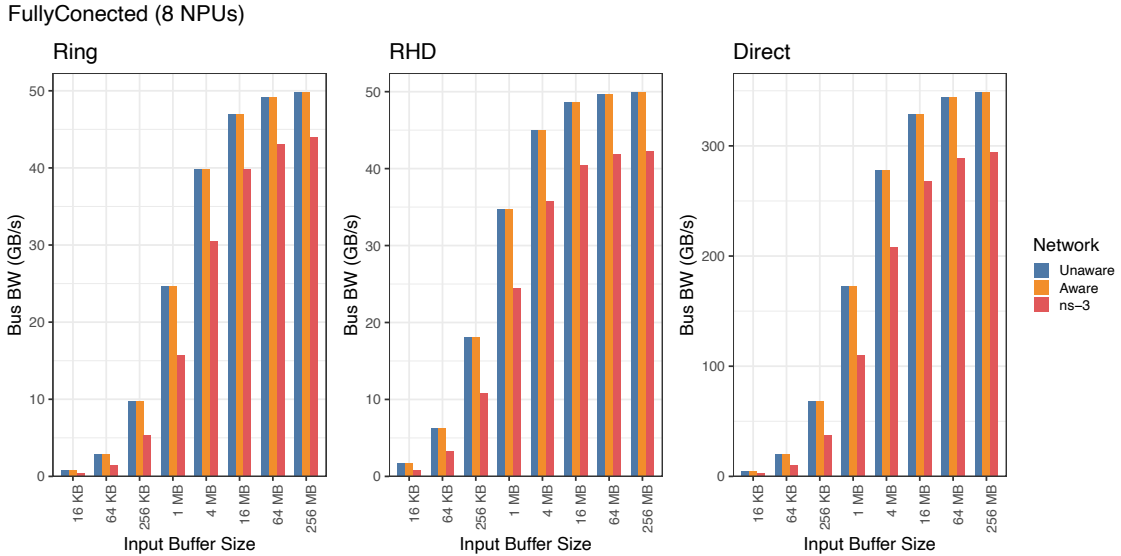


Figure 3.7: Comparing reported bus bandwidth of analytical and ns-3 without correlation.

using packet-based simulators for scale-up networks as well.

3.6.2 Comparison without Modeling Correlation

We first evaluated the modeling capabilities of the flow-based analytical network backend using a fully-connected network topology, which offers full bisection bandwidth and raises no network contention regardless of the collective algorithm. We evaluated a topology with 8 NPUs, $0.5 \mu\text{s}$ link latency, and 50 GiB/s bandwidth, and measured the bus bandwidth of the All-Gather collective pattern whose input buffer size spanned 16 KiB–256 MiB.

The comparison against ns-3 is shown in Figure 3.7. Because the topology offers full bisection bandwidth, congestion-unaware and congestion-aware analytical backends reported identical modeling results. Due to the additional overhead modeled by ns-3, it reported lower bus bandwidth—39.68% smaller on average.

3.6.3 Correlating Analytical and ns-3

To correlate the two network modeling backends, we conducted two example runs: a 1 KiB collective to correlate link latency and a 256 MiB collective to correlate link bandwidth.

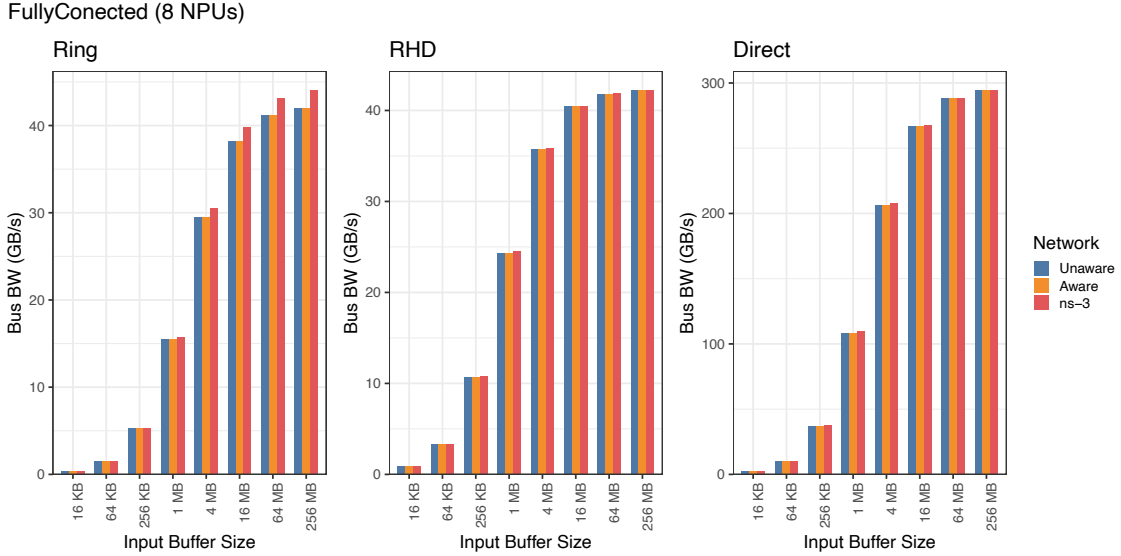


Figure 3.8: Comparing reported bus bandwidth of analytical and ns-3 with network correlation.

From these measurements, we set the analytical model to use $2.00\times$ larger link latency and $0.85\times$ smaller link bandwidth compared to ns-3, and reran the experiment shown in Figure 3.7.

After adjusting the link latency and bandwidth factors in the analytical backend, we observed that the difference between the analytical and packet-based modeling was reduced to 0.26% on average, as shown in Figure 3.8.

We conducted the same experiment for a 32-NPU target cluster, with results shown in Figure 3.9. The average difference between the analytical and ns-3 modeling was 0.23%.

3.6.4 Comparison with Network Congestion

We also evaluated a scenario where the network does not offer full bisection bandwidth, resulting in congestion depending on the deployed collective algorithm. We evaluated a ring topology with eight NPUs, with the same baseline link latency of $0.5 \mu\text{s}$ and 50 GiB/s bandwidth, later adjusted for analytical modeling with the above correlation factors.

Ring collective algorithms on a ring topology are topology-aware and do not generate network congestion. Therefore, the measurements observed in Figure 3.10 show iden-

FullyConnected (32 NPUs)

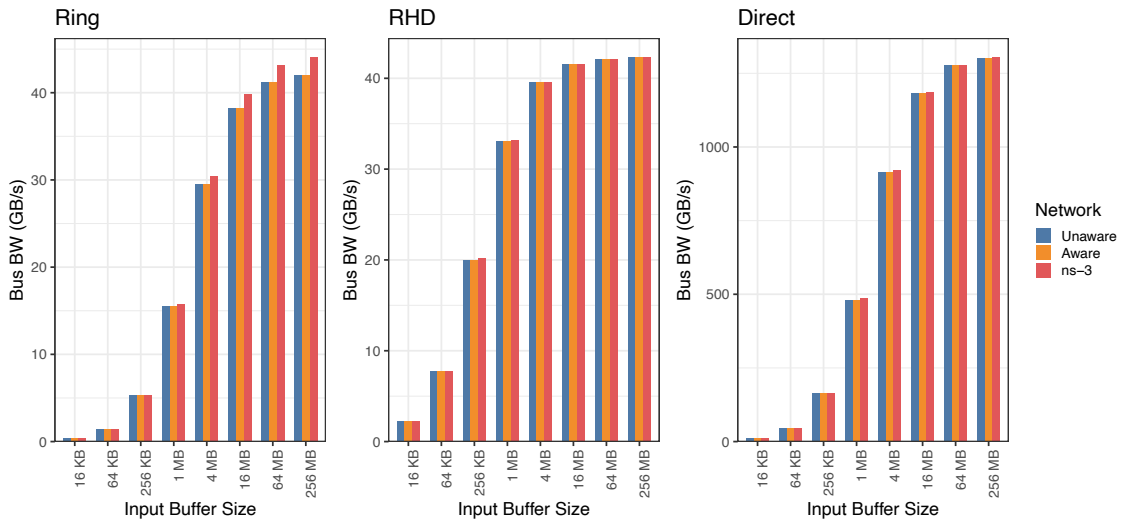


Figure 3.9: Analytical versus ns-3 modeling comparison for 32-NPU fully-connected topology.

Ring (8 NPUs)

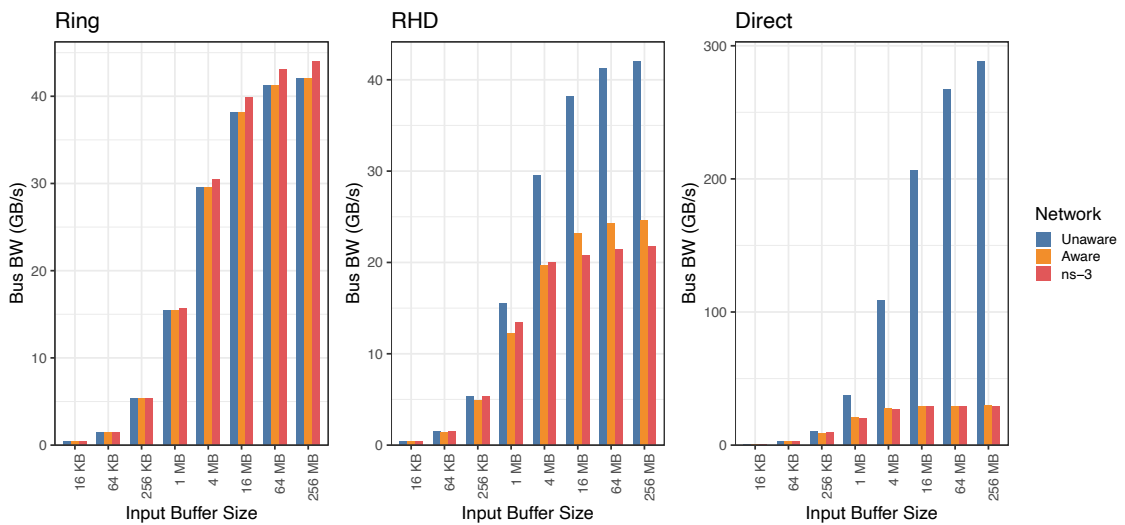


Figure 3.10: Analytical versus ns-3 modeling comparison for 8-NPU ring topology.

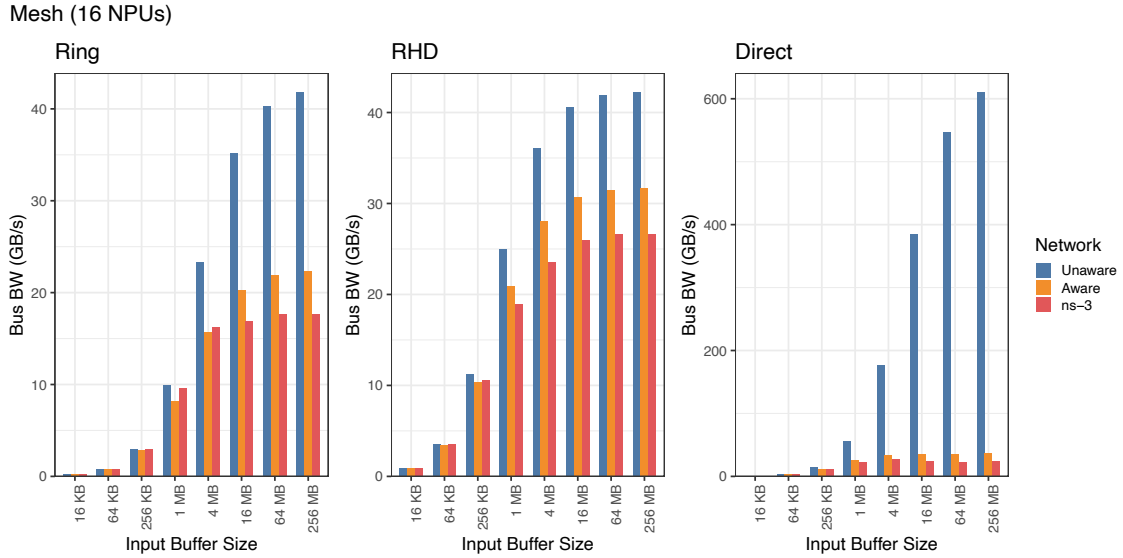


Figure 3.11: Analytical versus ns-3 modeling comparison for 16-NPU two-dimensional mesh topology.

tical trends, as expected. However, recursive halving-doubling and double binary tree algorithms generate congestion on a ring topology. Because congestion-unaware modeling cannot capture congestion effects, it reported overly high, idealized bus bandwidth. Congestion-aware modeling, on the other hand, captured congestion effects and showed comparable results to ns-3. The average difference between the two was 2.71% (13.29% max).

We conducted the same experiment on a 4×4 two-dimensional mesh topology, where all algorithms generate network congestion. The results are shown in Figure 3.11. As expected, congestion-unaware modeling consistently reported idealized performance, whereas congestion-aware modeling produced comparable results to packet-based modeling, with a mean difference of 6.71% (55.17% max).

To further confirm that the observed trends are driven by network congestion, we ran a TACOS-synthesized collective algorithm, which generates congestion-free, topology-aware patterns (TACOS is discussed in Chapter 5). As shown in Figure 3.12, the analytical simulation differed from ns-3 by only 0.45% on average, even on a two-dimensional mesh topology, but without congestion.

Mesh (16 NPUs)

TACOS

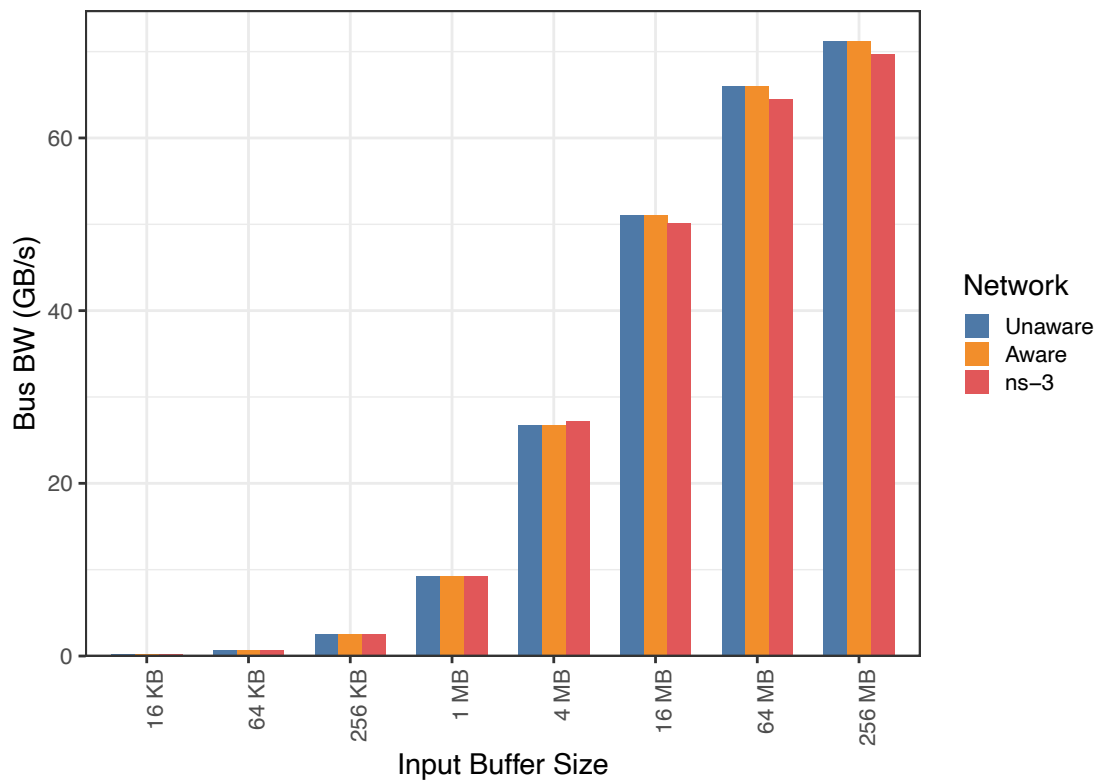


Figure 3.12: Analytical versus ns-3 modeling comparison when running TACOS-synthesized collective algorithm.

FullyConnected (8 NPUs)

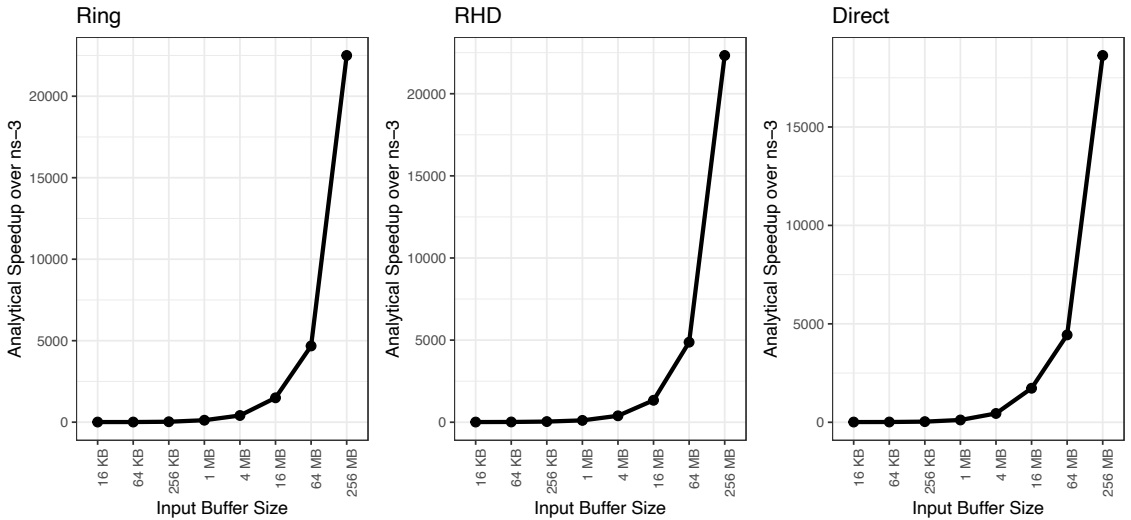


Figure 3.13: Simulation speedup of congestion-aware analytical model over ns-3 for 8-NPU fully-connected topology.

3.6.5 Simulation Time Speedup

Finally, we compare the modeling and simulation wall-clock time of the flow-based analytical model with the packet-based ns-3 simulator.

First, we measured the simulation time of both the congestion-aware analytical model and ns-3 while simulating All-Gather with input buffer sizes ranging from 16 KiB to 256 MiB. Figure 3.13 reports the speedup of the analytical model over ns-3. Because increasing the collective size does not increase the number of network flows, analytical modeling time remained constant regardless of collective size. However, packet-based simulation must inject and model more packets as the collective size increases, resulting in a significant slowdown of $22,330\times$ max.

We conducted the same experiment on a 32-NPU fully-connected topology. The results are shown in Figure 3.14. The slowdown of ns-3 became more pronounced, up to $148,150\times$, as the system scaled by $4\times$. Analytical modeling provided five orders-of-magnitude faster simulation time compared to ns-3 as collective size and system scale increased.

FullyConnected (32 NPUs)

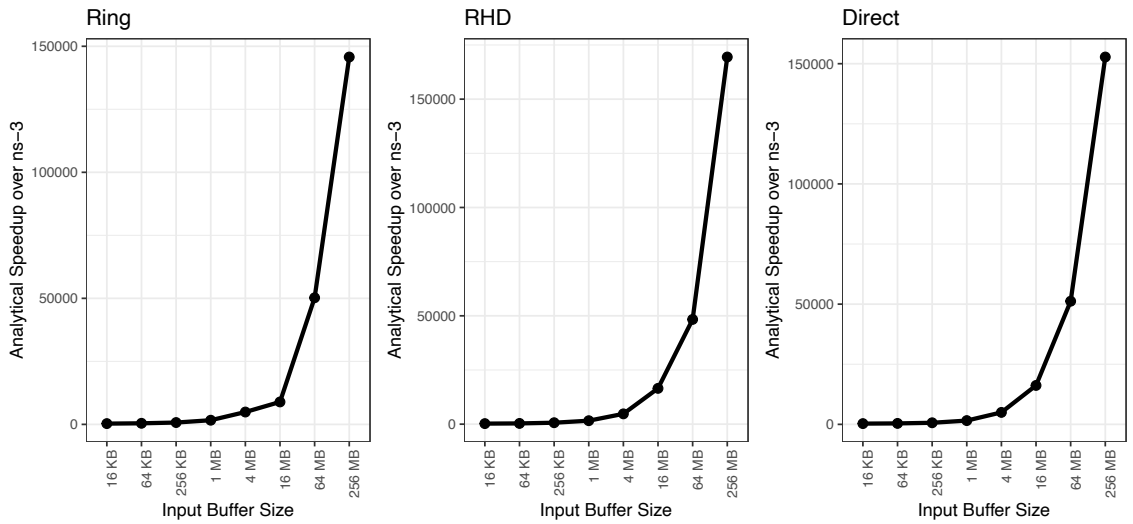


Figure 3.14: Simulation speedup of congestion-aware analytical model over ns-3 for 32-NPU fully-connected topology.

Ring (8 NPUs)

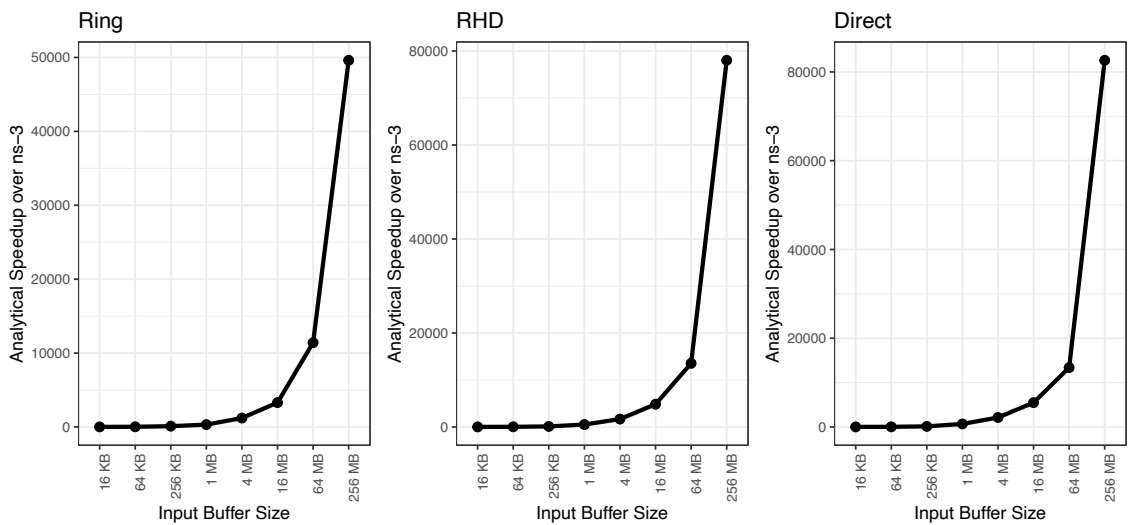


Figure 3.15: Simulation speedup of congestion-aware analytical model over ns-3 for 8-NPU ring topology.

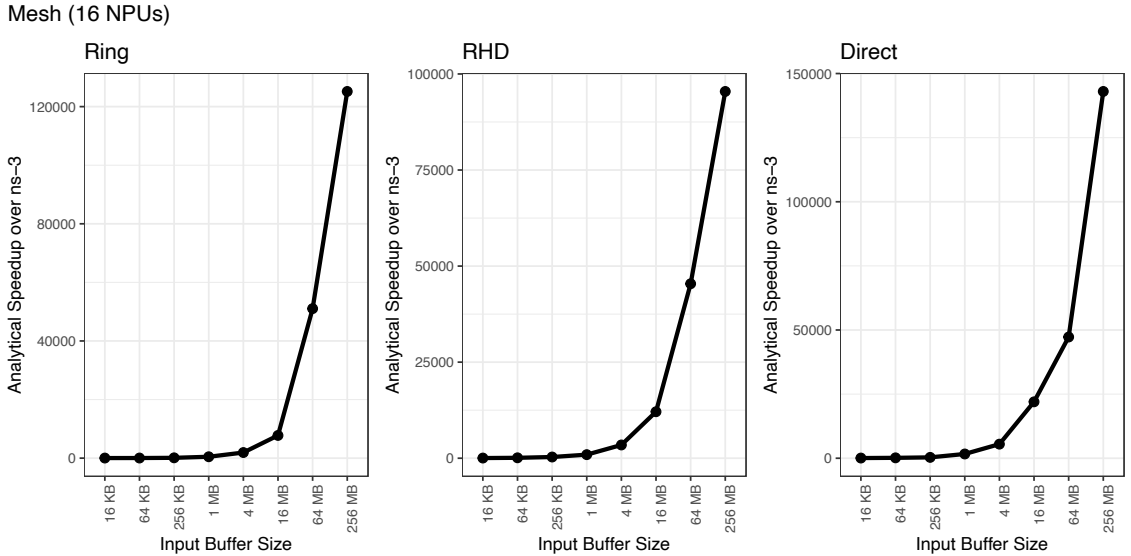


Figure 3.16: Simulation speedup of congestion-aware analytical model over ns-3 for 16-NPU two-dimensional mesh topology.

We also performed the speedup analysis on an 8-NPU ring topology, with results shown in Figure 3.15. Although the system size remained eight NPUs, compared to the congestion-free case in Figure 3.13, modeling congestion slowed down ns-3 simulation substantially. The speedup of the analytical modeling was up to $71,225\times$.

Finally, the same experiment on a 16-NPU 4×4 two-dimensional mesh topology—where all algorithms generate network congestion—is shown in Figure 3.16, where the speedup of up to $129,171\times$ is measured.

3.6.6 Conventional System versus Wafer-Scale System

Wafer-scale systems feature a number of NPUs connected using low-dimensional but high-bandwidth on-chip interconnection networks [63, 64, 78]. Meanwhile, conventional systems [58, 60] have multi-dimensional hierarchical topologies with various networking techniques, including on-chip, scale-up, and scale-out via NICs. We compare the two distinct approaches by abstracting these systems.

First, the target distributed training workloads and their characteristics are summarized in Table 3.2. The number of parameters for the DLRM workload only includes multi-layer

Table 3.2: Target training workloads and their characteristics.

Workload	Number of Parameters	TP Size	DP Size
DLRM	57 million (MLP layers)	1,024	1,024
GPT-3	175 billion	16	64
MSFT-1T	1 trillion	128	8

Table 3.3: Target wafer-scale and conventional (multi-dimensional) topologies.

Topology	Shape	NPU Size	BW (GiB/s)
W-1D	Switch	512	350, 500, 600
W-2D	Switch_Switch	32×16	250_250
Conv-3D	Ring_FC_Switch	16×8×4	200_100_50
Conv-4D	Ring_FC_Ring_Switch	2×8×8×4	250_200_100_50

perceptron (MLP) layers.

Also, the target experimental topologies with 512 NPUs are summarized in Table 3.3. Conventional system parameters are borrowed from [57, 58], and wafer-scale parameters are borrowed from [78, 79]. For the wafer-scale proxy, we created three one-dimensional topologies with 300, 500, and 600 GiB/s bandwidth (W-1D), and a two-dimensional topology with 250_250 GiB/s bandwidth (W-2D) to model futuristic wafer systems [78, 79]. For conventional systems, we created three- and four-dimensional topologies (Conv-3D and Conv-4D) using on-chip, scale-up, and scale-out interconnections borrowed from [58, 57, 19].

The normalized runtimes of a single 1 GiB All-Reduce, as well as real workloads, are shown in Figure 3.17. W-1/2D- x denotes a wafer-scale system that leverages a one-/two-dimensional topology with x GiB/s bandwidth. Conv-3D/4D denotes multi-dimensional conventional systems. Normalized training time breakdown with baseline hierarchical scheduling is shown on the left.

When a topology is multi-dimensional, complex behaviors such as pipelining bubbles or unbalanced network bandwidth result in low bandwidth resource utilization and sub-optimal performance [19]. Having only one dimension, W-1D yields the overall best performance. However, when comparing W-1D-350 and Conv-4D (600 GiB/s per NPU)

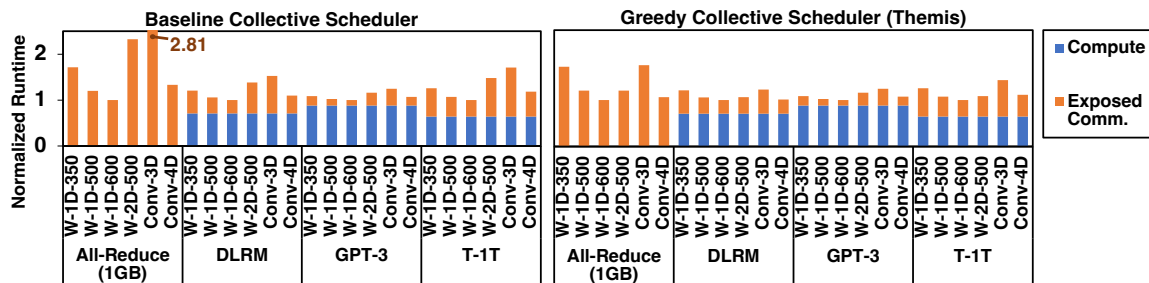


Figure 3.17: Wafer-scale versus conventional (multi-dimensional) systems profiling with 512 NPUs.

specifically, Conv-4D is able to drive more bandwidth per NPU and shows better performance despite being multi-dimensional.

Impact of Scheduling

Next, we study the impact of scheduling. Themis is a greedy scheduling policy for collectives that aims to balance the load across multiple dimensions to achieve near-optimal bandwidth utilization [19]. The results with the greedy collective scheduler (Themis) are shown on the right in Figure 3.17.

W-1D topologies, already being only one-dimension, show no gain from smart scheduling. However, W-2D, Conv-3D, and Conv-4D, being multi-dimensional, heavily benefit from the Themis scheduler. It is worth noting that for a single All-Reduce and DLRM, conventional systems with the Themis scheduler show identical results compared to their corresponding wafer-scale systems with equivalent bandwidth per NPU. Considering the complexity and cost of building a system on a single wafer, such results provide a glimpse of the possible advantage of using the conventional hierarchical approach in performance-per-cost aspects. Meanwhile, for GPT-3 and MSFT-1T, wafer-scale systems still maintain better training time. For hybrid parallelism on conventional systems, TP and DP span over some (and not every) dimension and utilize only that bandwidth, whereas for wafer-scale systems, every communication runs on full on-wafer bandwidth. This emphasizes the importance of appropriate parallelization strategies and the need to co-design them with

Table 3.4: Message size in MiB per each dimension and collective time when running an 1 GB All-Gather collective.

System Size	NPU	Dim 1	Dim 2	Dim 3	Dim 4	Collective Time (μ s)
2_8_8_4	512	1024	896	112	12	4,392.85
2_8_8_8	1,024	1024	896	112	14	4,392.85
2_8_8_16	2,048	1024	896	112	15	4,392.85
2_8_8_32	4,096	1024	896	112	15.5	4,392.85
4_8_8_4	1,024	1536	448	56	6	2,212.60
8_8_8_4	2,048	1792	224	28	3	1,753.48
16_8_8_4	4,096	1920	112	14	1.5	1,879.17

underlying topologies for conventional hierarchical systems.

Impact of Scaling using Wafer-Scale Systems

Traditional systems scale their infrastructure by a scale-out approach, meaning they attach more nodes to the last-dimension NICs. On the contrary, wafer-scale technologies let the framework scale up the system by increasing the number of NPUs on-chip (dimension one) while maintaining the number of scale-out nodes equally.

To measure the impact, we took the Conv-4D topology from Table 3.3, set the on-chip (i.e., dimension one) bandwidth to 1,000 GiB/s to model wafer-scale systems [78, 79], and set it as a baseline. Then, we scaled the platform up to 4,096 nodes and measured the 1 GiB All-Reduce time.

The results are shown in Table 3.4. Conventional scale-out increases the dimension four (NIC) message size, but the impact is marginal, thereby showing identical collective time. Scaling over the wafer, however, significantly increases on-wafer (dimension one) communication size while dramatically cutting down other dimensions’ load. As long as the system has enough on-wafer bandwidth, collective time decreases due to this effect, showing up to a $2.51\times$ speedup over the corresponding scale-out mechanism. Once the on-wafer dimension becomes the bottleneck, the collective time starts to bounce and increase again, as can be seen from the 16_8_8_4 system.

The end-to-end training time breakdown of GPT-3 and MSFT-1T is also shown in Fig-

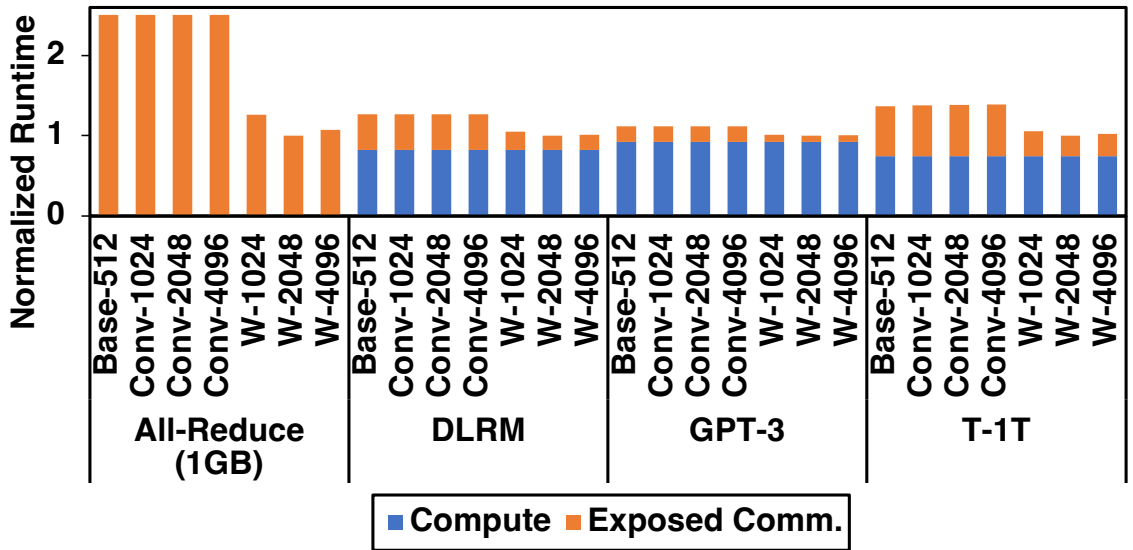


Figure 3.18: Wafer-scale versus conventional system scalability comparison.

Figure 3.18, showing the equivalent trend in the end-to-end regime. Base-512 denotes the baseline 2.8.8.4 (512 NPU) system. Conv- k denotes conventional (NIC scale-out) framework with k NPUs, whereas W- k denotes wafer-scale (on-chip; dimension one) scaling with k NPUs.

3.7 Related Work

Several simulators are available for modeling distributed systems running general-purpose workloads in the community, including SST [80], ZSim [81], and Mohammad et al. [82]. These simulators often require a trade-off between simulation accuracy, simulation speed, and engineering effort. Several models and simulators have been proposed to optimize communication performance in HPC platforms, such as LogGOPSim [83] and SMPI [84].

This chapter builds upon recent observations [19, 27, 85, 86] that the compute-memory-communication characteristics of distributed ML execution can be abstracted and captured using a combination of analytical and simulation-based methods without requiring a general-purpose simulator.

To the best of our knowledge, this is the first simulator that enables the execution of arbitrary distributed ML execution traces on next-generation platforms with multi-dimensional (scale-up and scale-out) topologies.

3.8 Conclusion

As deep learning models and input data continue to scale at an unprecedented rate, it has become inevitable to move towards distributed ML platforms to fit the models and increase ML throughput. State-of-the-art distributed ML systems are adopting emerging approaches and techniques such as wafer-scale nodes, multi-dimensional network topologies, and optimized parallelization strategies. This results in a complex software-hardware co-design stack, necessitating a modeling and simulation infrastructure for design space exploration. In this chapter, we motivate the need for rapidly modeling and profiling advanced distributed ML platforms for large ML models. This chapter introduces ASTRA-sim2.0, which extends the open-source ASTRA-sim infrastructure with capabilities to model state-of-the-art and emerging distributed ML models and platforms. Specifically, we enable ASTRA-sim to implement a parameterizable multi-dimensional heterogeneous topology generation infrastructure with the capability to simulate target systems at scale through analytical performance estimation. On top of such capabilities, we conduct comprehensive case studies targeting emerging distributed models and platforms. We conduct a comprehensive end-to-end, full-stack co-design space exploration of distributed ML. By empowering efficient navigation of the intricate design space of distributed ML, ASTRA-sim2.0 can provide meaningful first-order insights to system designers, thereby supporting the development of futuristic ML platforms at scale.

CHAPTER 4

LIBRA: ENABLING WORKLOAD-AWARE MULTI-DIMENSIONAL NETWORK TOPOLOGY OPTIMIZATION FOR DISTRIBUTED MACHINE LEARNING

4.1 Motivation

As explained in the backgrounds of this dissertation, collective communication among NPUs is a requirement for distributed ML. This chapter specifically aims at distributed ML training, which in nature necessitates collectives of large sizes. Consequently, this chapter discusses optimizing bandwidth-bound collective communications.

Figure 4.1 shows the communication sizes for ML model training. FP16 was used as the data type, and the parallelization strategy of Turing-NLG and smaller workloads is data parallel (minibatch size of 32), while GPT-3 and MSFT-1T use both tensor and data parallelism. As shown in Figure 4.1, the total communication size for large model training can span gigabytes to terabytes. Consequently, communication becomes one of the major bottlenecks in distributed training [22, 23, 23, 24, 25]. This naturally necessitates driving higher network bandwidth resources per NPU to efficiently process the massive volume of communication during training.

Therefore, AI clusters in data centers today most commonly leverage two-dimensional network fabrics to achieve high network bandwidth, namely scale-up and scale-out. In the first dimension, they employ custom rack-to-rack links (e.g., NVLink [87], Infinity Fabric [88], or XeLink [89]) to scale up a node, enabling peer-to-peer communication among NPUs. The second dimension is the conventional scale-out fabric, which employs NICs using technologies such as Ethernet [71] or InfiniBand [90]. However, as AI models continue to scale at an unprecedented rate [91], next-generation AI systems demand even higher network bandwidth, and addressing this challenge remains an open problem. A naive ap-

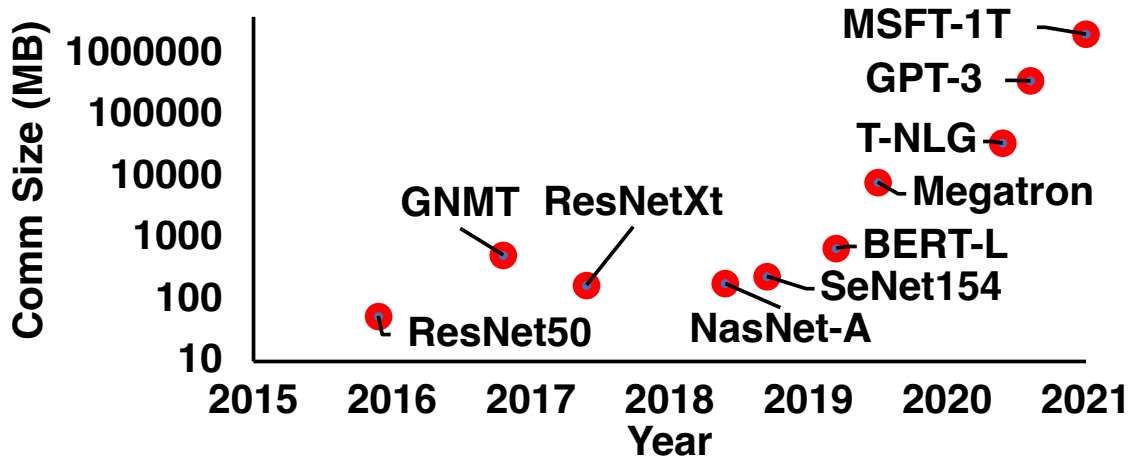


Figure 4.1: Communication sizes for ML model training across 1,024 NPUs.

proach would be to increase the individual bandwidth of the two dimensions. However, it is pivotal to note that physical and management constraints, such as limited pin counts allotted per network dimension, electrical serialization-deserialization constraints, signaling, and power consumption, cap the potential network bandwidth achievable per network dimension [92]. For example, while we can expect some further increase in bandwidth across generations of scale-up and scale-out links, NVLink technology today only reaches a maximum of 450 GiB/s [87]. This is true even when transitioning to alternative technologies that may offer higher bandwidth, such as advanced wafer-scale packaging [79, 93, 94] or photonics [95, 96], since additional physical constraints like thermals and reliability still apply, as well as expensive manufacturing costs, at least today. In summary, there is no single technology that can provide high network bandwidth within a single network dimension. This is evident from the diverse fabrics used in AI systems today: Intel leverages XeLink and InfiniBand technologies [97], NVIDIA uses NVLink, NVSwitch, and InfiniBand NICs [98, 99, 100], and Cerebras [16] and Tesla [101] utilize wafer-scale technologies.

We believe that a promising approach to enhance the bandwidth per NPU is to (i) explicitly add more network dimensions and (ii) leverage a mixture of fabric technologies. An abstract view of such multi-dimensional fabrics for AI systems is demonstrated in Fig-

ure 4.2, as well as its physical connotation assigned to 2–4-dimensional networks. By having multiple network dimensions, we can overcome the constraint of limited bandwidth of each network dimension and drive overall higher network bandwidth per NPU. For example, Google Cloud TPUv4 [95] employs a three-dimensional point-to-point electrical network augmented by a photonic network dimension in order to drive more network resources [95]. Even when employing a multi-dimensional network to allocate equal bandwidth resources per NPU, it can still achieve a higher performance-per-cost design point. This is attributable to the load-reducing nature of the multi-rail communication algorithm, which enables substituting expensive network resources with more cost-efficient technologies.

This multi-dimensional network scheme opens up a new fabric architecture optimization problem that this chapter aims to solve: how to determine bandwidth distribution across different dimensions at design time, under diverse technology-driven dollar cost and bandwidth constraints, while enabling high performance across multiple AI workloads. As we show in this chapter, judicious optimization of network bandwidth at design time is necessary to enable co-optimization and scheduling opportunities at runtime. Otherwise, bandwidth resources augmented by leveraging multi-dimensional topologies may be underutilized, resulting in inefficiencies in communication and training slowdown [19]. Resource allocation design is always a challenging problem, akin to other design challenges like sizing scratchpads or caches. However, unlike general-purpose HPC applications, communication patterns (e.g., communication types, sizes, and the group of NPUs involved) for ML applications can be predetermined when the parallelization strategy of a workload is set. Therefore, focusing on AI workloads enables us to adopt a more systematic workload-aware approach.

To this end, we propose LIBRA: leveraging intelligent bandwidth resource allocation for multi-dimensional networks. LIBRA also symbolizes balance in Latin, aligning with this chapter of judiciously allocating and harmonizing bandwidth resources across dimen-

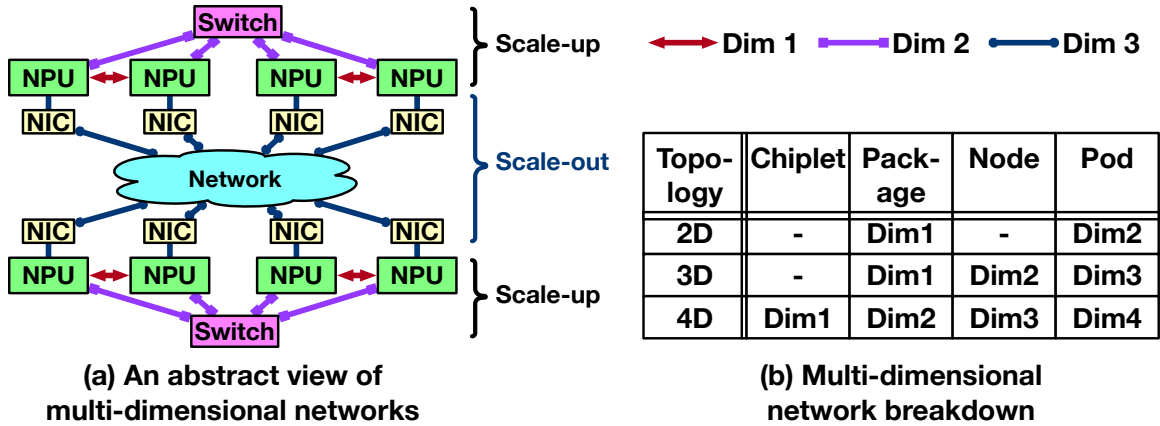


Figure 4.2: An abstract view of multi-dimensional networks and the physical connotation.

sions. LIBRA is a workload-aware, multi-dimensional network optimization framework at design time. LIBRA yields an optimized network design targeted for a specific set of workloads. We use the term design time to contrast LIBRA, as a toolchain, with runtime-based methodologies (e.g., schedulers or load balancers).

Given a set of target DNN models, fabric technology options, and design constraints, LIBRA can swiftly estimate the training performance and propose the optimal network bandwidth design point that maximizes performance or perf-per-cost. Through case studies, we demonstrate that LIBRA can be used for two primary purposes: (i) optimizing the multi-dimensional network architecture for a family of target workloads, and (ii) exploring co-optimization opportunities, such as designing networks alongside target workload parallelizations or runtime-based training scheduling techniques. To the best of our knowledge, LIBRA is the first workload-aware, constrained-optimization framework for AI systems fabric design.

4.2 Background

4.2.1 Physical Connotation to Multi-Dimensional Networks

An abstract view of the multi-dimensional network is illustrated in Figure 4.2(a), which depicts an example of a switch-based two-dimensional network. In Figure 4.2(b), physical

technology connotations are assigned to each network dimension as an illustrative example for 2–4-dimensional networks. This example physical connotation is used for evaluation purposes in this paper. Still, LIBRA remains flexible and supports arbitrary dimensionalities.

We introduce chiplet, package, node, and pod, inspired by emerging technology and system trends. Chiplet represents an NPU chip. A package consists of one or more chiplets interconnected via multi-chip module (MCM) packaging [94, 102]. A pod is an inter-server scale-out unit used in some platforms today, typically interconnected through NICs [103]. A node constructs a server unit by connecting multiple packages through an inter-board network.

4.2.2 Hierarchical Collective Algorithm

There exist several textbook topology-aware collective communication algorithms to execute these communication patterns. A handful of examples of textbook topology-aware All-Reduce collective algorithms include ring [74], double binary tree [104], and recursive halving-doubling [42]. However, when the underlying network topology is multi-dimensional, such textbook algorithms would not perform optimally, as the logical topology each algorithm assumes mismatches the physical one. For instance, the direct collective algorithm performs well on a fully-connected network, but the physical connectivity of multi-dimensional networks often does not meet such expectations. Consequently, heavy network contention and oversubscription over low-bandwidth links can occur, leading to significant underutilization of network bandwidth. To address this issue, multi-rail collective algorithms have been proposed to fully leverage the resources of multi-dimensional networks [105, 27], which is the approach adopted in LIBRA. A multi-rail collective algorithm capitalizes on the inherent nature of multi-dimensional networks, where basic network building blocks are stacked up. Therefore, it executes textbook collective algorithms in sequence.

For example, to perform an All-Reduce collective on an N -dimensional network:

- Run Reduce-Scatter on dimension 1. Then, run Reduce-Scatter on dimension 2. These Reduce-Scatter jobs, in ascending order, continue up to dimension N (N Reduce-Scatter stages).
- Perform All-Gather on dimension N , then execute All-Gather on dimension $N - 1$. This All-Gather stage continues, in descending order, down to dimension 1 (N All-Gather stages).

In summary, the multi-rail All-Reduce collective algorithm involves a total of $2N$ stages. Within each stage, each dimension utilizes its corresponding topology-aware collective algorithm. This approach guarantees that the overall collective operation runs in a contention-free manner.

An example All-Reduce on a 3×2 network is illustrated in Figure 4.3. Figure 4.3(a) shows the NPU placement. Figure 4.3(b)–(c) depicts the Reduce-Scatter phase, while Figure 4.3(d)–(e) shows the All-Gather phase. Figure 4.3(f) illustrates that the All-Reduce collective is finished. Each arrow represents the traffic of chunks received by NPU 1 for each stage.

4.2.3 Hybrid Parallelism

As discussed in the background, there are two main parallelization strategies: MP and DP. Specifically, MP includes TP and DP. TP divides the model and distributes it across NPUs, reducing the memory requirement of each NPU during training [13]. We use TP- n to denote that the model is sharded in an n -way. DP distributes the training dataset to boost the training throughput [13]. We use DP- n to indicate that the training dataset is split into n separate groups.

TP and DP are orthogonal and can be combined, resulting in a hybrid parallelism (HP) scheme. We use HP- (m, n) to denote the mixture of TP- m and DP- n . In the HP- (m, n)

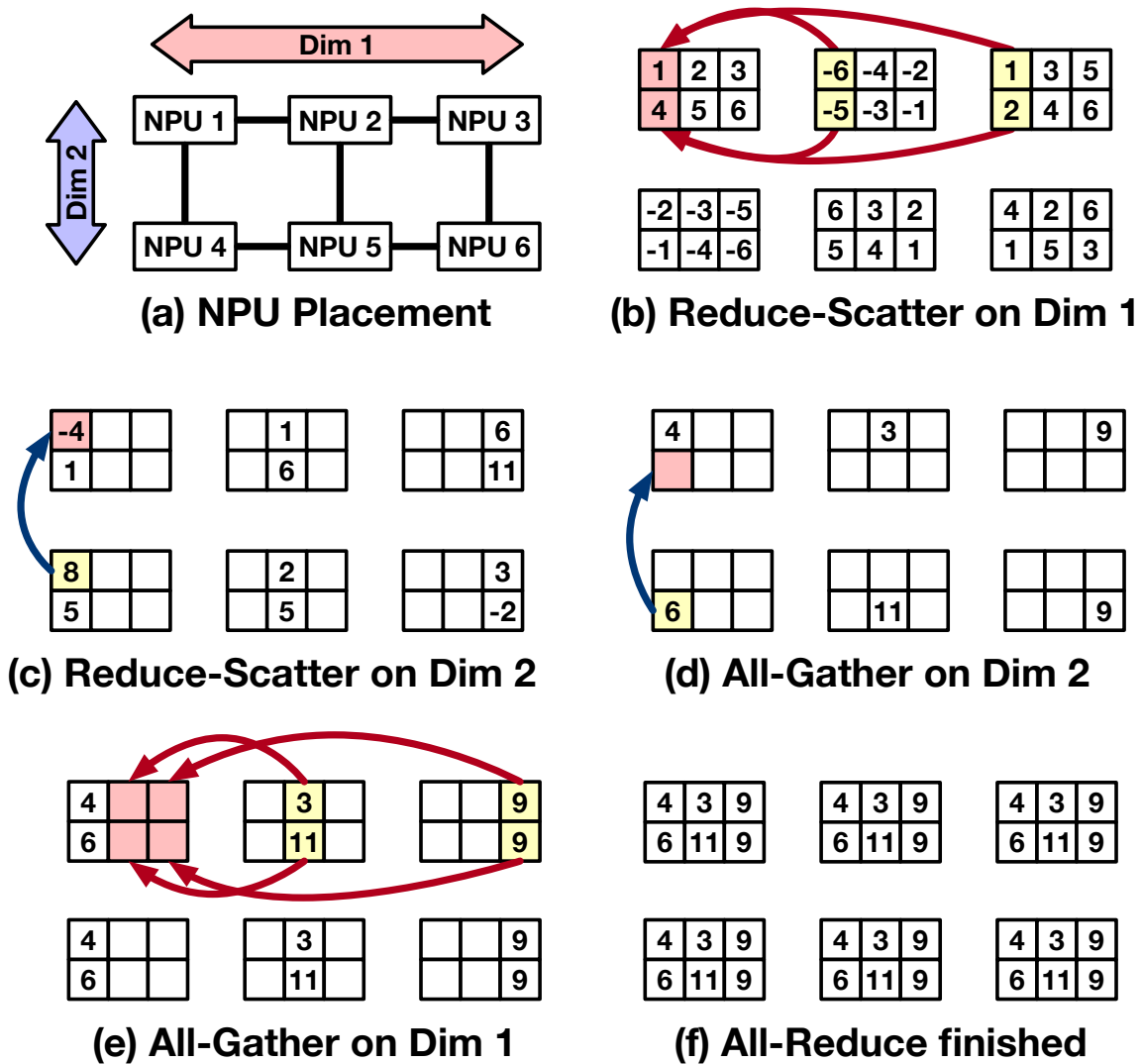


Figure 4.3: All-Reduce example on a 3×2 two-dimensional network.

setup, the training dataset is first split into n sets, and each set is fed into the group where the model is m -way sharded. Consequently, $\text{HP-}(m, n)$ requires a total of $m \times n$ NPUs. We leverage sophisticated hybrid parallelization schemes such as Megatron-LM [32], combined with the ZeRO-2 optimizer [11].

4.2.4 Training Loop

In addition to parallelization, the order of communication and computation must be clearly defined to execute distributed training. Such computation and communication ordering information is called a training loop [27].

Each parallelization strategy necessitates devices to communicate and synchronize dispersed information. For instance, TP requires input activations and input gradients to be communicated, while DP necessitates the synchronization of weight gradients. The training loop defines the ordering and scheduling of computation and communication during the training process.

As shown in Figure 4.4, we provide two examples of training loops for $\text{HP-}(2, 2)$ of a single-layer model. Figure 4.4(b) is a training loop without any overlap, namely No Overlap, denoting that there are no overlaps between the computation and communication stages. In contrast, Figure 4.4(c) is a training loop with TP and DP running concurrently, namely TP-DP Overlap. This training loop allows the concurrent processing of TP communication and DP computation and communication, resulting in a shorter training time.

4.3 Challenges

4.3.1 Technology Constraints of a Network Dimension

ML training platforms commonly utilize two-dimensional networks that leverage two technologies: (i) a high-bandwidth proprietary network to connect multiple NPUs within a server node (i.e., scale-up) and (ii) NICs to connect NPUs across server nodes (i.e., scale-out) [98, 106, 107, 108]. However, increasing the raw network bandwidth of each di-

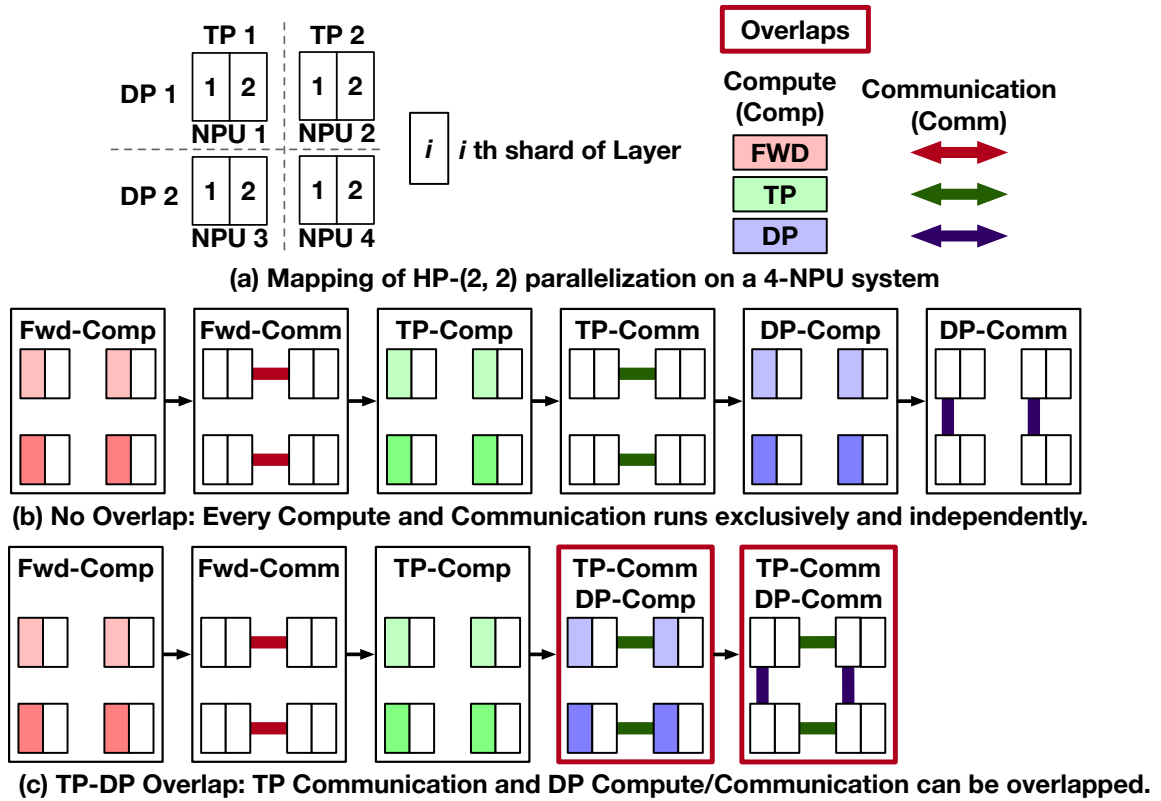


Figure 4.4: Example training loops.

mension in such setups is often challenging and expensive. This limitation arises due to the fundamental physical constraints imposed by the link technologies and their associated costs, such as pin counts, manufacturing, thermals, power, and area considerations [92]. For instance, the current NVLink technology [87] can provide up to 450 GiB/s per NPU, and the state-of-the-art NIC can drive up to 50 GiB/s [71].

4.3.2 Opportunity of Multi-dimensional Networks

In large model distributed training, a multi-dimensional topology composed of multiple network technologies can be an effective solution. There are two key benefits that such topologies provide in the context of DNN training.

Higher Aggregated Bandwidth

Simultaneously leveraging multiple network technologies, including chiplets [94] and photonics [95, 96], offers an opportunity to further increase aggregate bandwidth per NPU by introducing additional network dimensions between NPUs, either on a package [94] or on a board [95].

Higher Performance per Cost

As chunks are successively reduced throughout the stages of the multi-rail Reduce-Scatter phase, the volume of communication progressively decreases at each network dimension. This reduction in communication size is depicted in Figure 4.3. NPU 1 first receives four chunks from its neighbors through dimension 1. However, since these messages are reduced into a single chunk, the second Reduce-Scatter stage through dimension 2 only requires receiving one chunk. As a result, having more network dimensions significantly reduces the message size once it reaches higher network dimensions. This enables organizing a multi-dimensional network in a way that cheaper network technologies (i.e., scale-up) drive higher network bandwidth resources in lower network dimensions. Expensive network technologies (e.g., NICs or photonics) can establish higher network dimensions while providing limited bandwidth resources, benefiting from the reduced communication burden enabled by multi-dimensionality. Even when allocating equal bandwidth resources per NPU, this architectural design yields significant performance-per-cost improvements by substituting expensive technologies with more cost-effective resources.

4.3.3 Design-time Consideration of Network Bandwidth

Realizing a multi-dimensional architecture is a challenging task since the allocation of physical bandwidth across network dimensions at design time can significantly impact runtime performance. Consequently, appropriately sizing the bandwidth of each network dimension becomes crucial to avoid overprovisioning or underprovisioning network re-

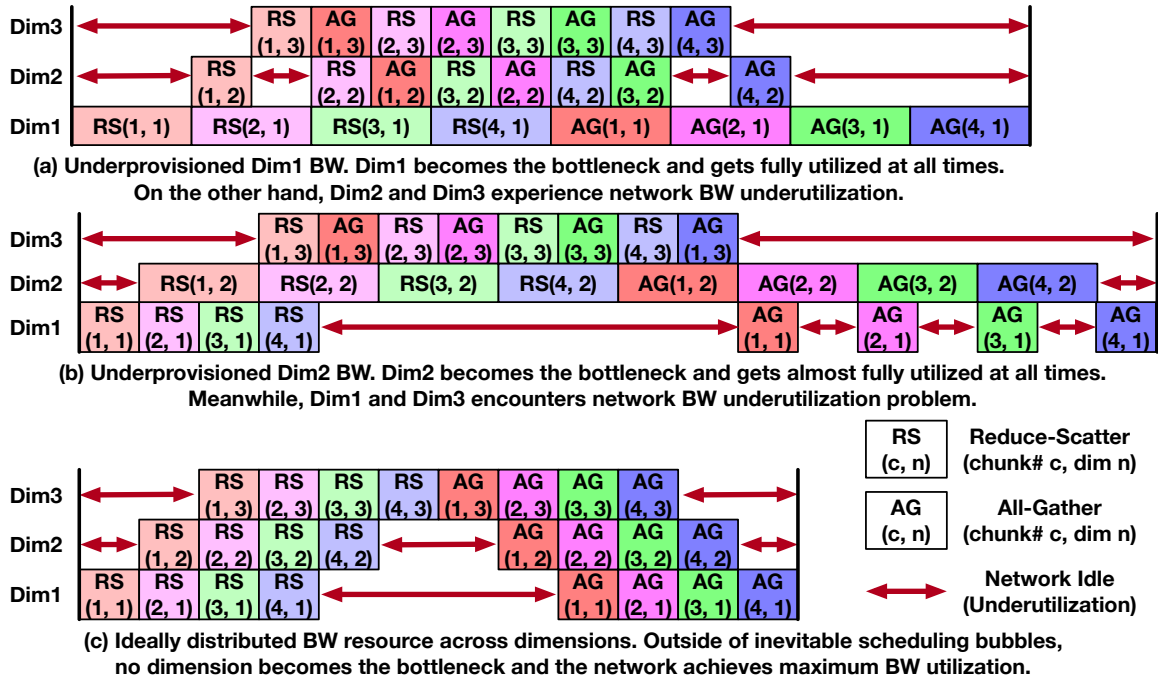


Figure 4.5: Running All-Reduce with four chunks on three-dimensional networks with different bandwidth allocations.

sources. This can be exemplified by the scenario in Figure 4.3. The payload size of dimension 2 is only a quarter of dimension 1's due to the reduction phase in dimension 1. Consequently, the bandwidth requirement of dimension 2 is only $\frac{1}{4}$ of dimension 1's. If the physical network bandwidth of dimension 2 is larger than $\frac{1}{4}$ of dimension 1's, then dimension 2's bandwidth resource is overprovisioned, leading to underutilization. Conversely, if dimension 2's bandwidth is less than $\frac{1}{4}$ of dimension 1's, dimension 2 becomes the bottleneck, and dimension 1's network resource will be underexploited. A collective communication usually consists of multiple chunks that run in a pipelined manner [109].

Figure 4.5 provides a generalized scenario using a three-dimensional network. In Figure 4.5(a), dimension 1's physical bandwidth is smaller than its requirement, making dimension 1 the communication bottleneck and leading to significant underutilization of other dimensions (dimension 2 and dimension 3). Similarly, Figure 4.5(b) depicts a case where dimension 2's bandwidth is underprovisioned, resulting in extensive network underutilization of dimension 1 and dimension 3. If we can judiciously design a multi-

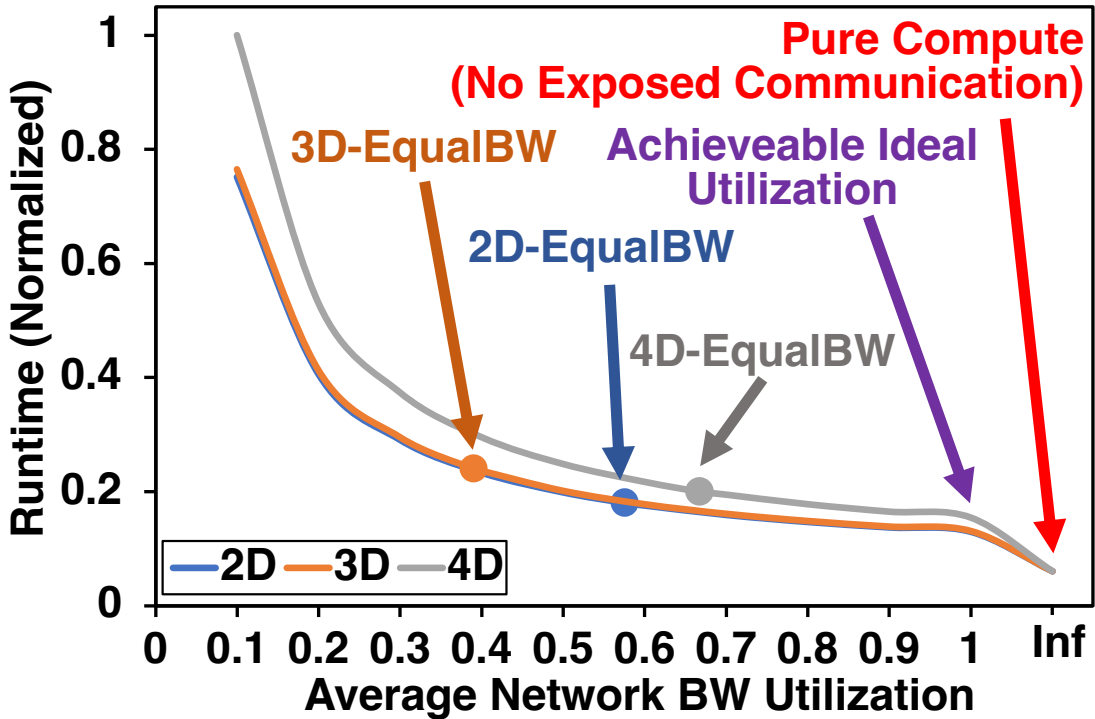


Figure 4.6: The normalized end-to-end training time of MSFT-1T on 2–4-dimensional topologies.

dimensional network with such considerations, the network utilization can be maximized, as shown in Figure 4.5(c).

The potential benefits of using workload-aware multi-dimensional networks are demonstrated in Figure 4.6. The plot shows the normalized runtime to train the MSFT-1T model, where each NPU has a 300 GiB/s aggregated bandwidth, but assuming different network bandwidth utilization ratios. Average network bandwidth utilization was 53.11% (66.74% max). In the baseline EqualBW configuration (i.e., all network dimensions have an equal amount of bandwidth, explained in Section 4.5.2), the network bandwidth utilization was only 57.53%, 39.02%, and 66.74% for two-, three-, and four-dimensional networks, respectively. By using a workload-aware network bandwidth configuration optimized for the target MSFT-1T model, we can maximize network bandwidth utilization and theoretically achieve training speedups of $1.39\times$, $1.83\times$, and $1.29\times$, respectively. We can achieve a $1.83\times$ maximum speedup if we can reach 100% bandwidth utilization. This result empha-

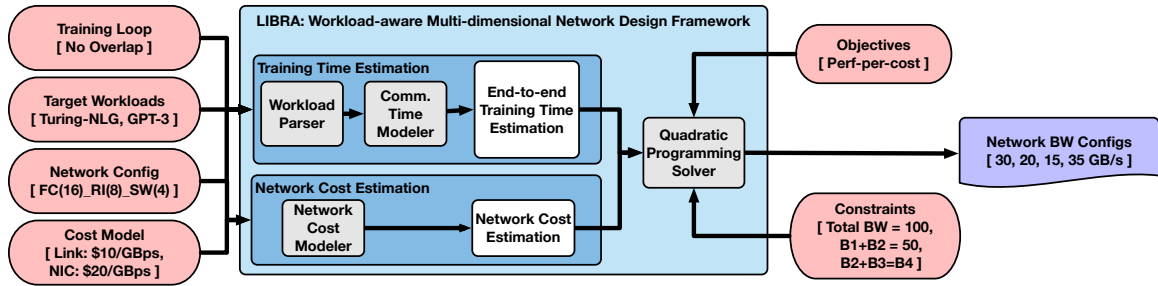


Figure 4.7: The architecture of LIBRA.

sizes the critical role of network bandwidth distribution across dimensions for AI collectives.

4.3.4 Problem Statement

LIBRA is a design-time framework to construct an HPC network optimized for AI training. Given a set of target workloads and a multi-dimensional network representation, LIBRA performs optimization to yield the best network bandwidth configuration. The optimization process aims to maximize specific objectives such as training performance or performance-per-cost. LIBRA, throughout optimization, ensures that the network bandwidth configuration adheres to user-defined design constraints, such as fixed bandwidth per NPU, bandwidth limits for a specific dimension, or limited total network cost.

4.4 LIBRA

In this section, we introduce LIBRA: a workload-aware, design-time multi-dimensional network optimization framework. We explain how LIBRA can model collective communications, large model distributed training, and optimize the network bandwidth of multi-dimensional topologies.

The architecture of LIBRA is summarized in Figure 4.7. LIBRA is a workload-aware multi-dimensional network bandwidth optimization framework. Inputs to the LIBRA framework are represented in oboards, with corresponding example values shown in square brackets. LIBRA estimates the end-to-end training time based on the provided network

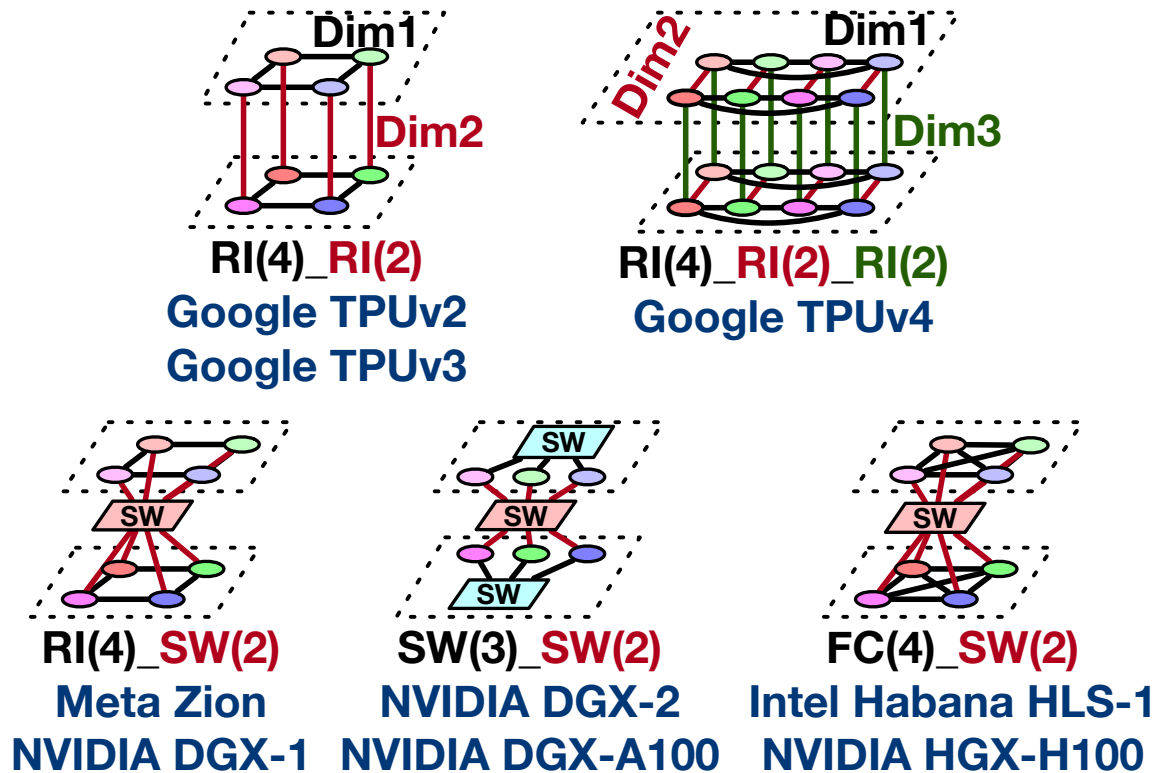


Figure 4.8: Examples of 2–3-dimensional networks and their corresponding names in the notation used.

shape, training loop, and target workloads. Additionally, it calculates the cost of a network using a specified cost model and network shape. Subsequently, LIBRA searches for the optimal network bandwidth configuration that maximizes the given objective while adhering to designated design constraints.

4.4.1 Multi-dimensional Network Representation

As proposed earlier in this dissertation, in this chapter, we also utilize the multi-dimensional network representation from Chapter 3. We adopt three unit topologies as building blocks for each network dimension: ring (RI), fully-connected (FC), and switch (SW). Multi-dimensional networks can be represented by stacking the building blocks together alongside the corresponding network size. For example, RI(4)_RI(4)_RI(4) represents a three-dimensional torus network with 64 NPUs.

To demonstrate the capability to capture HPC networks in deployment, Figure 4.8 depicts five examples of 2–3-dimensional networks captured in this taxonomy and their corresponding ML HPC clusters in use. To demonstrate the expressiveness of the representation in capturing real systems, corresponding ML clusters that utilize equivalent network shapes are also listed. Although not shown, the notation is flexible and can be leveraged to capture multi-dimensional networks with four dimensions or more.

4.4.2 Modeling Distributed Training

In order to optimize for the best bandwidth configuration of a multi-dimensional network, LIBRA estimates the end-to-end training time as a function of network bandwidth. To achieve this, LIBRA first models collective communications as a function of network bandwidth.

Collective Communication Modeling

Consider a scenario where we run an m -sized All-Reduce operation on a two-dimensional ($n_1 \times n_2$) network with B_1 and B_2 for its two dimensions' bandwidth, respectively. Mathematically, the nature of multi-rail All-Reduce results in the traffic volume of $\frac{2m(n_1-1)}{n_1}$ and $\frac{2m(n_2-1)}{n_1 n_2}$ transferred by each dimension, respectively. Taking into account the bandwidth of each dimension, and considering the bottlenecking dimension determines the collective time (as shown in Figure 4.5), the All-Reduce time is:

$$\max\left(\frac{2m(n_1-1)}{n_1 B_1}, \frac{2m(n_2-1)}{n_1 n_2 B_2}\right)$$

Here, we took a simple maximum of the delays of each dimension, rather than actually modeling pipeline flush delays or bubbles. Unlike other modeling equations, this equation is being used by LIBRA optimization in this chapter. As discussed in Figure 4.5, as the optimizer finds the design point closer to the ideal, the delay of each network dimension

saturates to a comparable point, and pipeline bubbles become negligible. Therefore, taking a simple max is an acceptable modeling approach, provided that the modeling will be used as an optimization objective.

Similarly, since the communication volume halves compared to All-Reduce, Reduce-Scatter and All-Gather can be modeled as:

$$\max\left(\frac{m(n_1 - 1)}{n_1 B_1}, \frac{m(n_2 - 1)}{n_1 n_2 B_2}\right)$$

Finally, due to the absence of chunk reduction, All-to-All time is computed as:

$$\max\left(\frac{m(n_1 - 1)}{n_1 B_1}, \frac{m(n_2 - 1)}{n_2 B_2}\right)$$

Such models can be further generalized as the network dimensionality increases. Ultimately, LIBRA estimates all communication times as functions where the sole parameter is the network bandwidth configuration.

End-to-End Training Modeling

The end-to-end model training can be captured by leveraging such communication representation. The training loop depicted in Figure 4.4(b) has no overlap among the computation and communication stages. Therefore, the end-to-end execution time is estimated by simply aggregating all compute and communication delays.

Specifically, the forward pass takes:

$$\sum_{l \in \text{layer}} (\text{Fwd_Compute}_l + \text{Fwd_Comm}_l)$$

Note that Fwd_Compute_l is bandwidth-independent and Fwd_Comm_l is a function of network bandwidth.

Similarly, the backward pass takes:

$$\sum_{l \in \text{layer}} (\text{TP_Compute}_l + \text{TP_Comm}_l + \text{DP_Compute}_l + \text{DP_Comm}_l)$$

If there are overlaps in complex training loops, they can easily be reflected in the end-to-end time estimation. For example, the training loop in Figure 4.4(c) assumes that TP_Compute is fully exposed, but overlapping TP_Comm with DP_Compute and DP_Comm is possible. Then, each layer’s backward pass will take:

$$\text{TP_Compute} + \max(\text{TP_Comm}, \text{DP_Compute} + \text{DP_Comm})$$

The end result is LIBRA estimating and capturing the training performance of a DNN model as a function of network bandwidth, which can then be optimized.

LIBRA Modeling

In the current LIBRA modeling, we disregarded the impact of link latencies or NPU performance implications (e.g., memory access bandwidth or reduction performance). Large-model training commonly involves large traffic volumes, making communication highly network bandwidth-bound [110]. This allows modeling communication performance in network bandwidth configuration to be a feasible option. In general, accurately modeling collective communications with more intricate equations has been the focus of recent endeavors, which is orthogonal to this dissertation since LIBRA can incorporate such modeling and still optimize for network bandwidth configurations.

In-Network Collective

The offloading of collectives to the network switches has been a subject of research [23] and commercial developments [70, 100]. In essence, in-network collective offloading effectively reduces the communication time of dimension i to $\frac{m}{n_1 n_2 \dots n_{i-1} B_i}$.

Table 4.1: Cost model for network cost evaluation in LIBRA.

(\$ per GiB/s)	Link	Switch	NIC
Inter-Chiplet	2.0	-	-
Inter-Package	4.0 – 5.2	13.0	-
Inter-Node	4.0 – 5.2	13.0	-
Inter-Pod	7.8 – 16	18.0 – 69.6	31.6 – 144

Parallelization Strategy

Intricate parallelization strategies, including PP, periodically involve non-collective communication patterns such as point-to-point message transfers. Such operations can still be captured in terms of network bandwidth (e.g., $\frac{m}{B_i}$) and can be leveraged in estimating end-to-end performance.

4.4.3 Modeling Network Cost

In addition to performance modeling, LIBRA can estimate the dollar cost of networks. This estimation process relies on the network cost model, which is provided as an input parameter to LIBRA by the user. The reason for this is that network costs can vary significantly based on the technology and specific vendors and may also change over time. For instance, the inter-node link costs can vary significantly across high-speed electrical [87, 89] to photonic networks [95].

To facilitate the analysis, we offer a default cost model, shown in Table 4.1. This cost model is derived from costs available in public citations. Dollar cost of each network component per GiB/s is shown. The values are derived from the most current available data [71, 103, 107, 111, 112, 113]. We used the lowest value of each entry for evaluation.

This default model is used for design-space exploration in this chapter. In this default cost model, a pod is defined as the unit for scale-out, meaning other network dimensions do not utilize NICs. For inter-chiplet networks, we assume chiplets are always connected peer-to-peer, thereby eliminating the need for switches. Figure 4.9 depicts an example cost analysis based on this default cost model shown in Table 4.1.

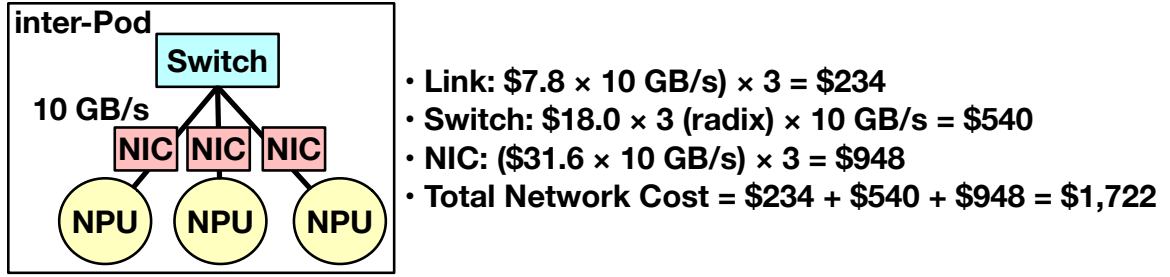


Figure 4.9: Example cost modeling of a three-NPU switch network.

4.4.4 Optimizing Multi-dimensional Network

With the provided modeling capabilities, LIBRA can optimize the network bandwidth for a target DNN training task, leading to the ideal network bandwidth allocation that achieves specific objectives, such as maximizing training performance. LIBRA utilizes a quadratic programming (QP) solver [114] to carry out the network optimization.

LIBRA harnesses the power of a QP solver to find an ideal network design point, providing the flexibility to represent linear and quadratic constraints as desired by the system designer. For instance, if there is a restriction on the total network bandwidth per NPU, it can be easily represented as $\sum_i B_i \leq B$, where B_i denotes the bandwidth of dimension i . Similarly, to limit the inter-pod dimension bandwidth to 50 GiB/s, this constraint is captured as $B_4 \leq 50 \text{ GiB/s}$. The representation of constraints is highly adaptable, allowing for the addition of various flexible constraints of interest, such as $B_1 + B_2 = 500 \text{ GiB/s}$ or $B_1 \geq B_2 \geq B_3$, $25 \leq B_3 \leq 150 \text{ GiB/s}$. Multiple constraints can be applied simultaneously, enabling a comprehensive and customized network optimization.

4.4.5 Optimization Scheme

LIBRA supports two optimization objectives to yield different network bandwidth design points, as discussed below.

PerfOptBW

The PerfOptBW optimization scheme aims to maximize the model training performance. LIBRA's QP solver disregards the network cost factor, and the solver's optimization objective is set to minimize the end-to-end training time.

PerfPerCostOptBW

PerfPerCostOptBW takes into account both network cost and performance. It multiplies end-to-end training time by the estimated network cost to measure the (reciprocal of) performance-per-cost metric, for which the QP solver inside LIBRA is then configured to minimize.

4.4.6 Optimization for Multiple Workloads

Furthermore, LIBRA has the capability to optimize a network that targets multiple workloads, accommodating real-world training scenarios. In practice, we envision that AI training clusters will be designed to handle a family of target workloads, not just a single target.

LIBRA is capable of designing a network aware of multiple target workloads. This is achieved by optimizing a weighted sum of the end-to-end training time of individual target workloads, where the weight represents the importance of the corresponding workload. This results in a network design point that is optimized toward multiple workloads. By accommodating multiple targets and considering their relative importance, LIBRA becomes a versatile tool for designing networks that can effectively handle various training scenarios.

4.5 Methodology

4.5.1 Simulation Infrastructure

LIBRA is a standalone framework aimed at aiding the design process for determining multi-dimensional network bandwidth configurations. However, to profile and measure

Table 4.2: Workloads specifications used for analysis.

Workload	Params	TP Size
Turing-NLG	17 billion	1
GPT-3	175 billion	16
MSFT-1T	1 trillion	128
DLRM	57 million (MLP layers only)	Across all NPUs
ResNet-50	25.6 million	1

the performance of networks designed by LIBRA, a simulation infrastructure is necessary to meticulously identify the practical implications.

As proposed in this dissertation, we utilized ASTRA-sim2.0, the distributed ML simulator presented in Chapter 3, to evaluate the training performance of LIBRA-generated networks. ASTRA-sim2.0 enables executing complex DNN workloads, modeling intricate network architectures, running various collective algorithms with chunk-level scheduling, and capturing compute–communication overlaps. ASTRA-sim2.0 has been validated over multiple real ML systems and shows an error rate of 2.8–11.4% [115].

4.5.2 Experimental Setup

Baseline

In this chapter, we use EqualBW as a strawman baseline. The EqualBW scheme distributes the given bandwidth resource B equally across all N dimensions, resulting in $\frac{B}{N}$ allocated to each dimension. EqualBW is chosen as the baseline because it is a workload-agnostic allocation that does not involve any performance-driven optimization.

Workloads

We selected three transformer-based LLMs and two non-LLMs (recommendation and vision) [116, 117, 35] for evaluations in this chapter. To distribute these models, we assumed ZeRO phase-2 optimizers [11]. The specifications of the selected models are provided in Table 4.2. All communications are split into 64 chunks per collective.

Table 4.3: Multi-dimensional Topologies used for analysis.

Name	Shape
4D-4K	RI(4)_FC(8)_RI(4)_SW(32)
3D-4K	RI(16)_FC(8)_SW(32)
3D-512	SW(16)_SW(8)_SW(4)
3D-1K	FC(8)_RI(16)_SW(8)
4D-2K	RI(4)_SW(4)_SW(8)_SW(16)
3D-Torus	RI(4)_RI(4)_RI(4)

Compute Model

The average efficacy of the A100 GPU [99] was measured to be 75% (i.e., 234 TFLOPS) and was used to estimate NPU compute times in the evaluation.

Network Topologies

We adopted different target multi-dimensional networks from related work [19], adjusting the last dimension size to scale the network size (ranging from 512 to 4,096 NPUs). The overall list of topologies evaluated in this chapter is summarized in Table 4.3. Across case studies, as a representative configuration, we chose the 4D-4K topology. For corresponding three-dimensional topology evaluations, we utilized the 3D-4K network, created by combining two ring dimensions of the 4D-4K topology.

4.6 Evaluations

We conducted case studies and design-space explorations of multi-dimensional networks to demonstrate diverse use cases of LIBRA. Due to space constraints, we highlight the most interesting design points in each study.

4.6.1 Sweeping Large Model Training

Here, we utilize LIBRA to design a network optimized for a specific training workload. In practice, LIBRA is envisioned to design networks optimized for an ensemble of training

models, as shown in Section 4.6.4.

We evaluated both 3D-4K and 4D-4K networks, sweeping the bandwidth per NPU over the range of 100–1,000 GiB/s, which covers the bandwidth budgets of recent ML clusters [95, 99, 100].

Performance

The training speedup over the baseline EqualBW configuration for Turing-NLG, GPT-3, and MSFT-1T models across PerfOptBW and PerfPerCostOptBW-optimized networks is depicted in Figure 4.10. Each point represents a LIBRA-optimized network for the target workload; e.g., GPT-3+3D denotes a three-dimensional network optimized for GPT-3. PerfOptBW maximizes training performance, while PerfPerCostOptBW maximizes performance-per-cost and can yield a speedup below 1.

On average, PerfOptBW achieves a $1.23\times$ speedup ($2.00\times$ max) over EqualBW. For GPT-3 on the 4D-4K topology, some dimension 2 bandwidth could not be fully utilized due to mismatched TP size, resulting in performance near the baseline. Nonetheless, PerfOptBW achieved a $4.58\times$ performance-per-cost improvement compared to the baseline for this case.

Performance-per-Cost

PerfPerCostOptBW achieves the highest performance-per-cost across all design points. On average, performance-per-cost improvement over EqualBW is $5.40\times$ ($12.24\times$ max) for PerfOptBW and $9.16\times$ ($13.02\times$ max) for PerfPerCostOptBW.

Key Insights

Larger models benefit more in raw performance, while smaller models gain higher performance-per-cost. Smaller models are less communication-critical, so optimizing the network has limited impact. As workload sizes increase, design-time considerations via LIBRA become

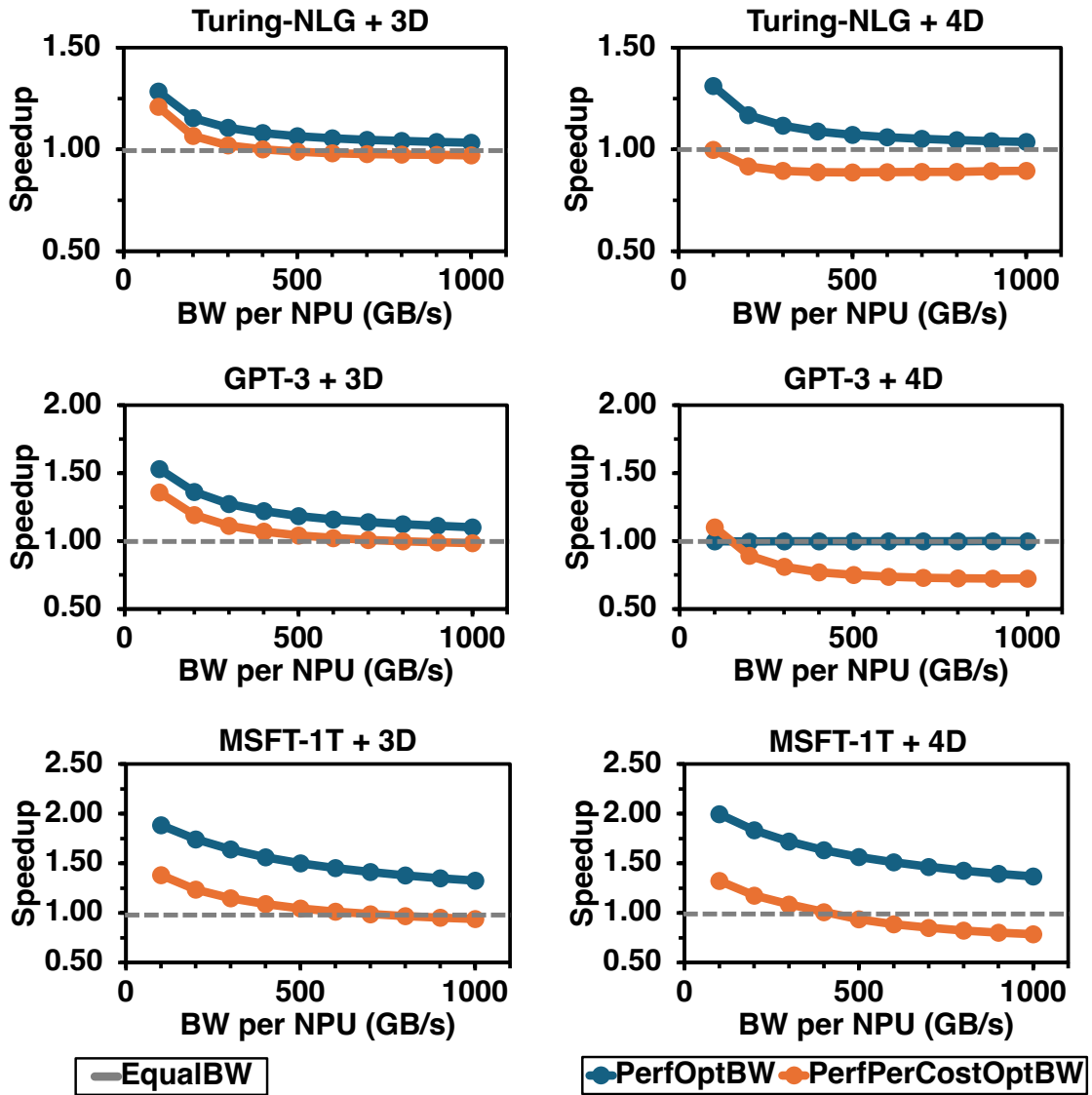


Figure 4.10: End-to-end training speedup over the baseline EqualBW.

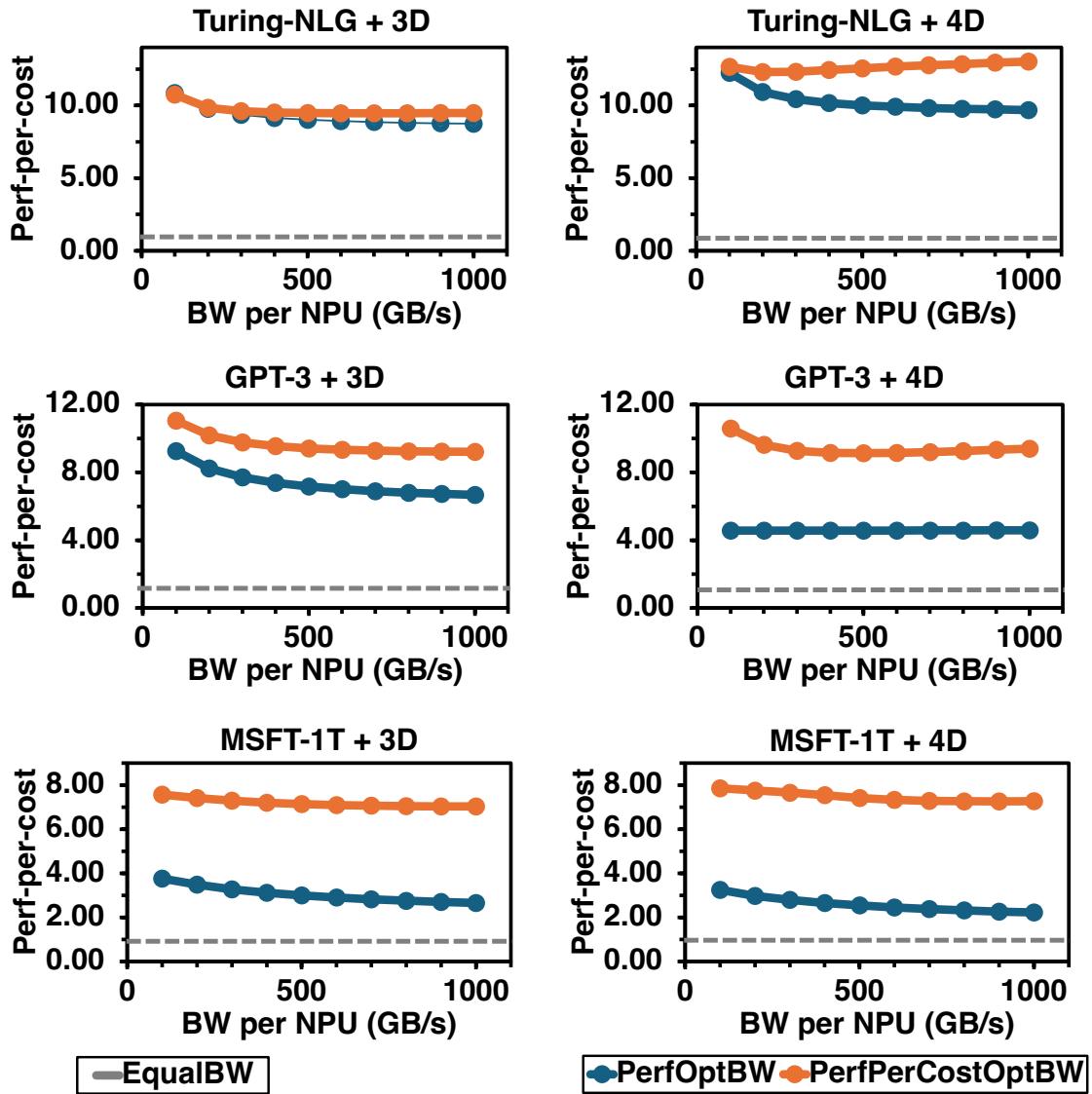


Figure 4.11: Performance-per-cost benefit of LIBRA over the EqualBW baseline.

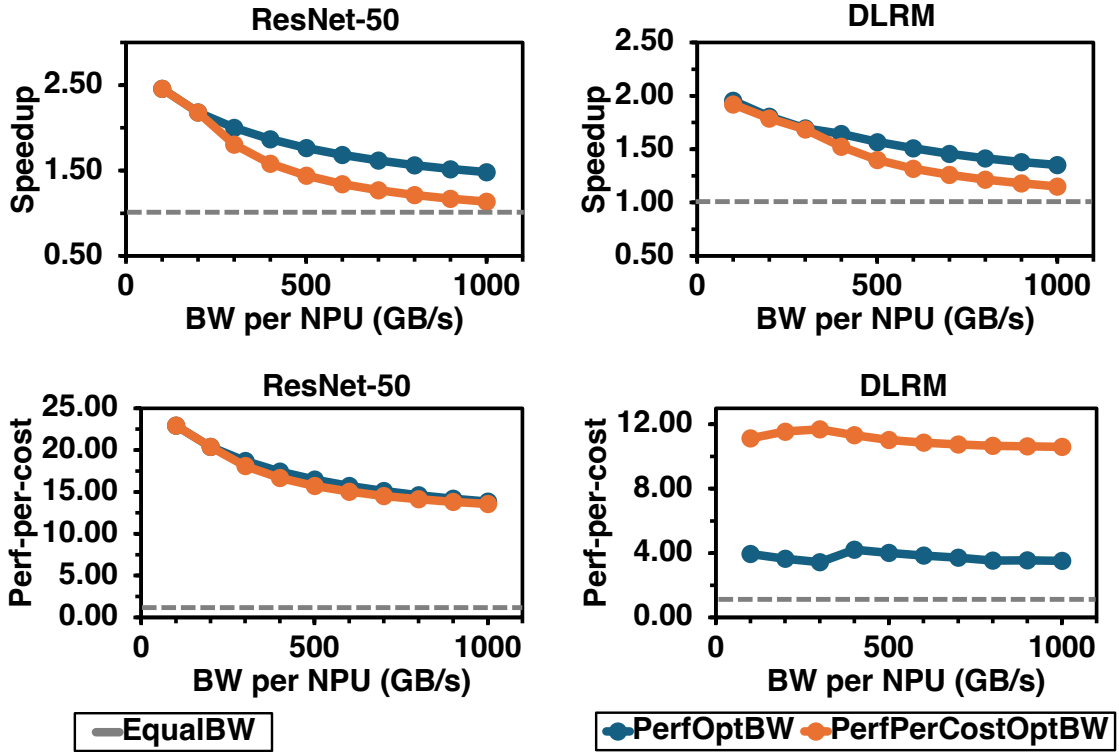


Figure 4.12: Speedup and performance-per-cost analysis of ResNet-50 and DLRM.

increasingly important.

4.6.2 Non-Transformer Workloads

LIBRA can optimize networks for non-transformer workloads without modification. For ResNet-50 and DLRM, we optimized the 4D-4K network using both PerfOptBW and PerfPerCostOptBW objectives. Results are normalized over the corresponding EqualBW baseline.

Due to ResNet-50’s smaller model size, training time is low, making the performance-per-cost metric heavily influenced by cost. PerfPerCostOptBW produced network designs similar in performance-per-cost to PerfOptBW, but $15.41\times$ cheaper on average.

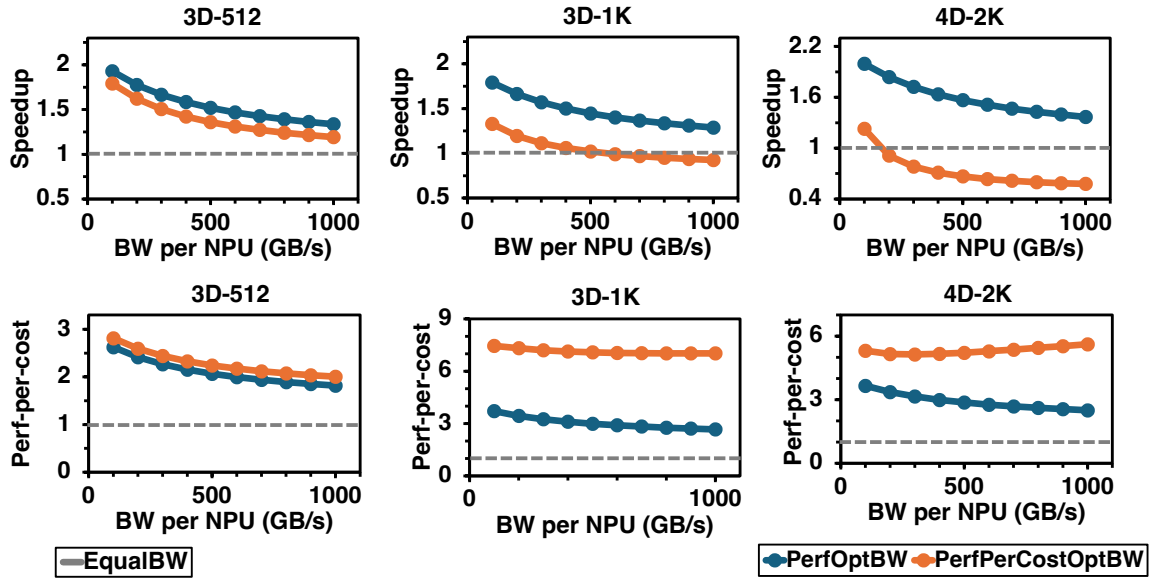


Figure 4.13: Speedup and performance-per-cost analysis of MSFT-1T.

4.6.3 Topology Exploration

To showcase LIBRA’s adaptability, we compared networks of different shapes and scales: 3D-512, 3D-1K, and 4D-2K, using the MSFT-1T workload. Results, normalized over the corresponding EqualBW baseline, demonstrate that LIBRA consistently improves both speedup and performance-per-cost across topologies.

4.6.4 Optimizing for Multiple Workloads

LIBRA can optimize a network considering a group of workloads instead of a single target. For a 4D-4K network at 1,000 GiB/s per NPU, we first optimized networks individually for each workload, then evaluated non-target workloads on these networks. Finally, we performed group-optimization using all target models simultaneously.

Figure 4.14 summarizes speedup over EqualBW (bars) and slowdown relative to individually optimized networks (dots). Red bars indicate performance for the group-optimized network. When optimized individually, slowdowns for non-target workloads reached up to $1.77\times$. With group optimization, the network achieved near-optimal performance across

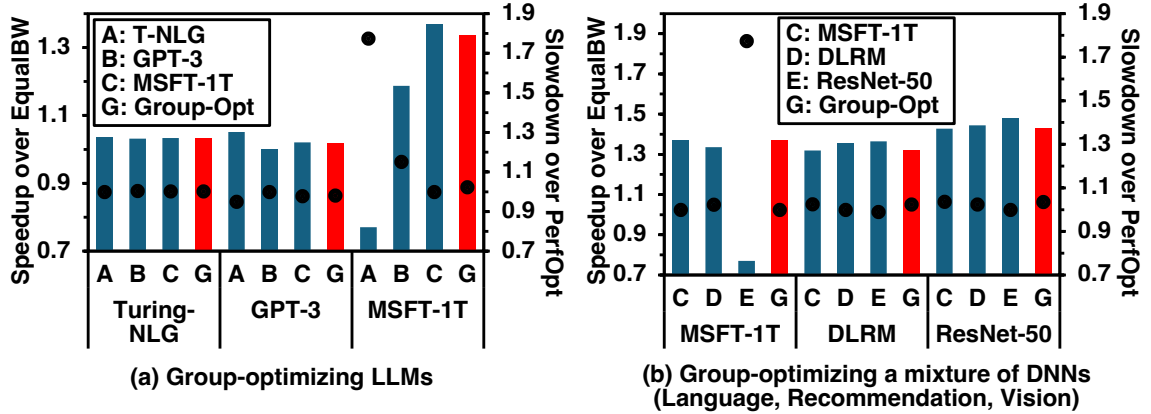


Figure 4.14: Speedup over EqualBW and slowdown over LIBRA-optimized network of various DNN models.

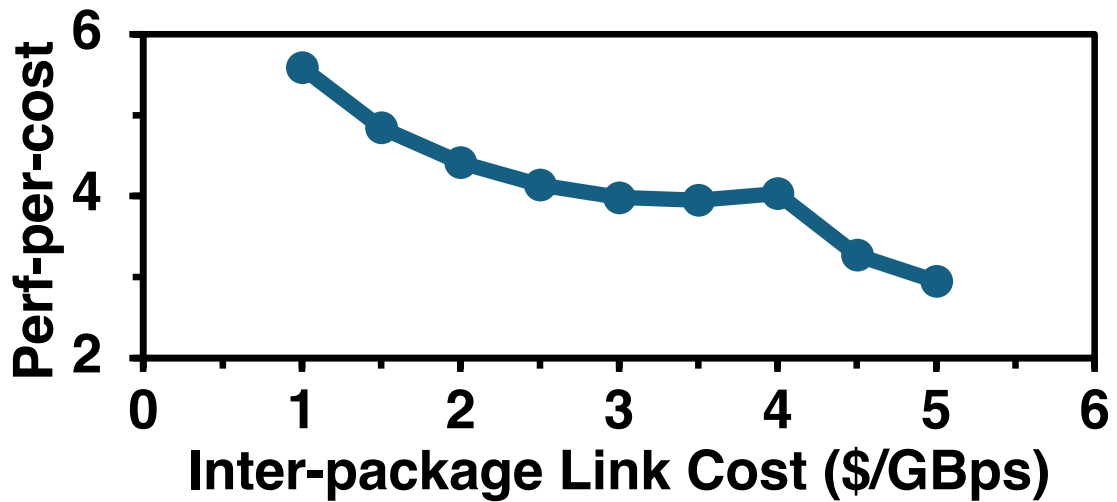


Figure 4.15: Normalized performance-per-cost benefit of PerfPerCostOptBW.

all workloads, with an average slowdown of only $1.01\times$.

4.6.5 Cost Model Sensitivity Analysis

LIBRA’s user-defined cost model allows flexible exploration of cost sensitivity. We varied inter-package link costs (\$1–5 per GiB/s) for the 4D-4K network with 1,000 GiB/s per NPU using PerfPerCostOptBW.

Average performance-per-cost improvement over EqualBW is $4.06\times$ ($5.59\times$ max), demonstrating LIBRA’s flexibility under varying cost assumptions.

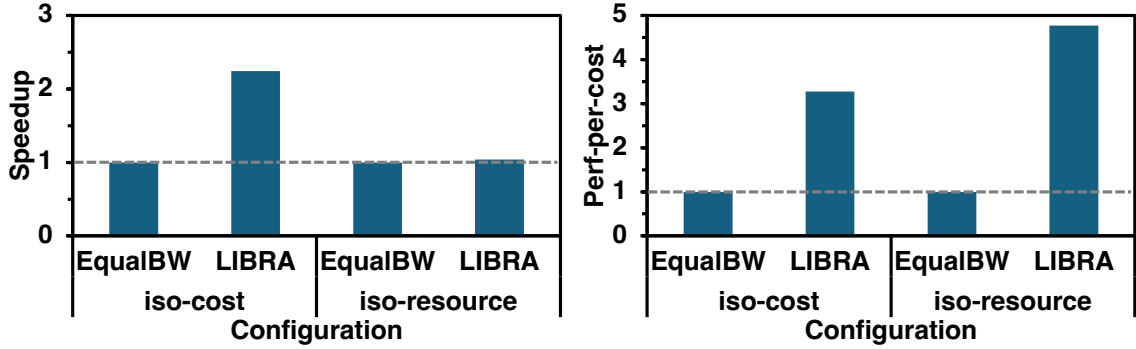


Figure 4.16: Normalized performance and performance-per-cost of LIBRA with Themis runtime optimizer.

4.6.6 LIBRA with Runtime Optimizations

Careful design-time network optimization with LIBRA enhances the benefits of runtime optimization techniques.

LIBRA+Themis

Themis [19] dynamically schedules data chunks over multi-dimensional networks. We trained GPT-3 on 4D-4K topology using EqualBW and LIBRA configurations, evaluating iso-cost (\$15M) and iso-resource (1,000 GiB/s per NPU) scenarios.

With iso-cost, LIBRA supported $5.05\times$ more bandwidth per NPU than EqualBW, yielding a $2.24\times$ training speedup even with Themis enabled for both. For iso-resource, LIBRA achieved $1.04\times$ better performance and $4.58\times$ lower network cost, resulting in $4.77\times$ improvement in performance-per-cost.

LIBRA+TACOS

TACOS (discussed in Chapter 5) automatically synthesizes topology-aware collective algorithms. A 1 GiB All-Reduce with eight chunks was executed on a three-dimensional torus network at 1,000 GiB/s per NPU.

LIBRA+TACOS showed $1.25\times$ and $1.08\times$ speedup over LIBRA-only and TACOS-

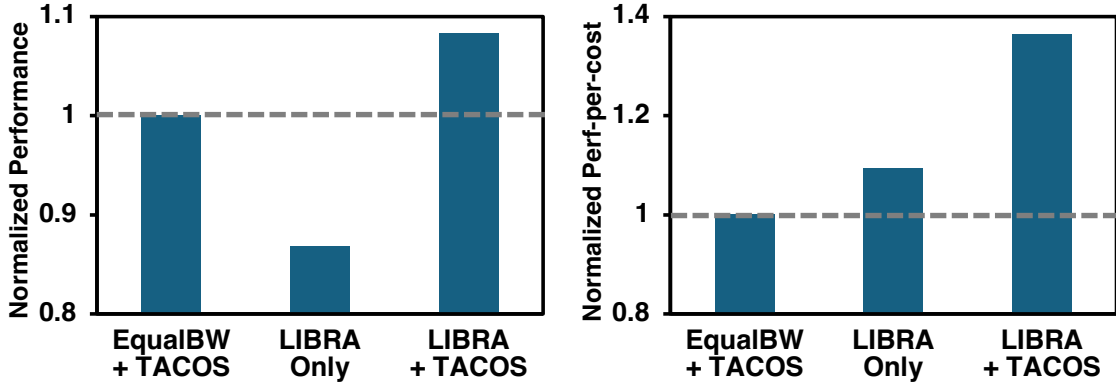


Figure 4.17: Normalized performance and performance-per-cost with LIBRA and TACOS.

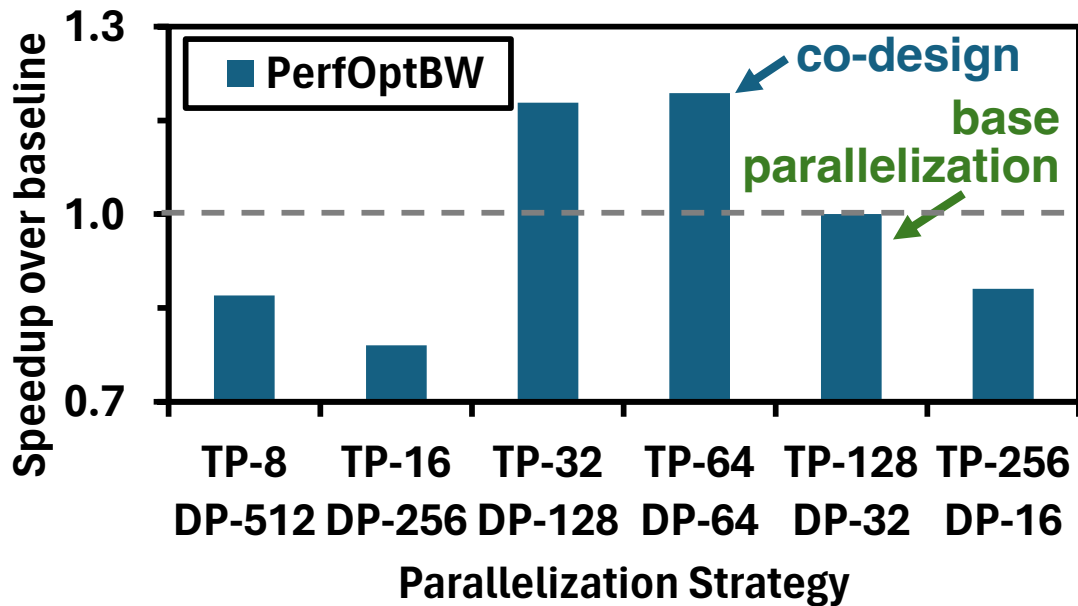


Figure 4.18: Normalized speedup of MSFT-1T across various parallelization strategies.

only, respectively. Due to LIBRA’s cost optimization, combined usage improved performance-per-cost by $1.36\times$ relative to TACOS-only.

4.6.7 Co-optimizing Network and Parallelization

LIBRA enables joint co-optimization of the network and parallelization strategy. We varied TP-DP sizes for the 4D-4K network with 1,000 GiB/s per NPU, assuming extended NPU memory via CXL [118] or CPU memory [53].

Figure 4.18 shows performance of PerfOptBW network normalized over EqualBW

with HP-(128, 32), varying parallelization from HP-(8, 512) to HP-(256, 16). HP-(64, 64) with its co-optimized network achieves peak training performance, with MSFT-1T training $1.19\times$ faster than the baseline.

Performance degrades when TP size falls below 32 due to the interplay of TP and DP communication. HP-(64, 64) allows LIBRA to reduce DP time while mitigating TP overhead, whereas other configurations incur higher total communication, limiting performance even with LIBRA support.

4.7 Related Work

4.7.1 Multi-dimensional Networks

Multi-dimensional networks have long been employed in general-purpose HPC platforms, including 3–6-dimensional networks [119, 120, 121]. These networks, however, were not explicitly designed for AI training workloads. In the AI domain, three-dimensional [99, 100, 17] and four-dimensional [14] networks are being actively explored to address the communication demands of large-scale models.

4.7.2 Design-Time Optimization

Collective communications differ fundamentally from point-to-point messaging: all NPUs communicate synchronously, and message sizes evolve throughout the operation. This complexity increases when using multi-dimensional algorithms.

Various topology-aware collective algorithms have been proposed [122, 123, 124, 125, 126, 127], but they are generally not AI-aware and do not consider specific workloads, parallelization strategies, training loops, or network constraints. To the best of our knowledge, LIBRA is the first framework targeting multi-dimensional network bandwidth optimization for AI workloads at design time.

Other approaches offload collectives to network switches [23, 70] or NICs [25], which are orthogonal to LIBRA. We incorporate these offload mechanisms into LIBRA’s model-

ing to identify the optimal network bandwidth configuration.

4.7.3 Co-Optimization of Network and Workload

AI HPC clusters are increasingly co-optimized with their target workloads due to the high resource demands. EFLOPS [128] proposes a novel network topology and collective algorithm to optimize All-Reduce performance, while ZionEx [108] focuses on accelerating DLRM training tasks. Our work further demonstrates that runtime-based optimization techniques achieve their full potential when combined with careful design-time network planning via LIBRA.

4.8 Conclusion

As model sizes in machine learning continue to scale, distributed training is necessary to accommodate model weights within each device and to reduce training time. However, this comes with the expense of increased communication overhead due to the exchange of gradients and activations, which become the critical bottleneck of the end-to-end training process. In this chapter, we motivate the design of multi-dimensional networks within machine learning systems as a cost-efficient mechanism to enhance overall network bandwidth. We also identify that optimal bandwidth allocation is pivotal for multi-dimensional networks to ensure efficient resource utilization. We introduce LIBRA, a framework specifically focused on optimizing multi-dimensional fabric architectures. LIBRA is a design-time framework to construct workload-aware multi-dimensional networks. To the best of our knowledge, LIBRA is the first framework in the field of ML to optimize the bandwidth of a multi-dimensional network, at design time, using target workload characteristics. LIBRA aims to maximize the training performance and performance-per-cost of a target set of DNN workloads. Through case studies, we demonstrate the value of LIBRA, both in architecting optimized fabrics under diverse constraints and in enabling co-optimization opportunities. The LIBRA artifact is publicly available at <https://github.com/astra-sim/libra>.

CHAPTER 5

TACOS: TOPOLOGY-AWARE COLLECTIVE ALGORITHM SYNTHESIZER FOR DISTRIBUTED MACHINE LEARNING

5.1 Motivation

As discussed in previous chapters, collective communication is a fundamental foundation of distributed ML platforms. Therefore, optimizing the performance of collective communication is pivotal, which requires judicious design of topology-aware collective algorithms.

However, it is important to note that different AI clusters employ diverse network topologies. For instance, point-to-point connections are used in Intel Gaudi [111] and Google TPU [60], hierarchical switches are utilized in NVIDIA HGX [107], mesh architectures are employed in Tesla Dojo [101] and Cerebras Andromeda [16], and Dragon-Fly [75] connections are used in Groq [129]. Moreover, these systems often incorporate multiple link technologies such as Intel XeLink [89], NVIDIA NVLink [62], AMD Infinity [130], InfiniBand [131], or even optical networks [60]. In summary, topologies in ML clusters commonly exhibit asymmetric (for example, NPUs at the center versus edge of a two-dimensional mesh have different degrees) shapes and heterogeneous link bandwidths (i.e., all links within a homogeneous topology have the same bandwidth). Considering that communication among NPUs is the main bottleneck of distributed ML [23, 24], orchestrating communication within these AI platforms remains an open research problem [19, 23, 132, 133].

Communication within AI workloads presents unique characteristics compared to traditional cloud workloads. In the latter, there can be heavy variability in the traffic pattern and volumes. This has naturally led to a set of past works on dynamically managing net-

Table 5.1: Various All-Reduce algorithms and their target topologies.

Topology-aware All-Reduce Algorithm	Preferred Physical Topology							
	Uni-dimensional			Multi-dimensional		Asymmetric		Any
	RI	FC	SW	Ho	Ht	Ho	Ht	Any
Ring [74]	✓							
Direct [27]		✓						
RHD [42]			✓					
DBT [43]			✓					
BlueConnect [105]	✓	✓	✓	△	△			
Themis [19]	✓	✓	✓	✓	✓			
TTO [134]				△		△		
C-Cube [135]		△		△		△		
TACOS	✓	✓	✓	✓	✓	✓	✓	✓

work congestion in data center fabrics [48]. In contrast, for ML models, both the traffic pattern and traffic volume are deterministic once the workload has been partitioned and mapped [52]. The traffic pattern is collective in nature (i.e., one-to-many and many-to-one). Given these characteristics, AI workload communication today is often routed statically via CCLs [36, 37, 38, 136]. These CCLs encompass a collection of predefined collective algorithms known as topology-aware collectives. As the name suggests, these algorithms are tailored for operation on specific network topologies [137]. For example, ring (RI) [74], direct (DI) [27], recursive halving-doubling (RHD) [42], and double binary tree (DBT) [43] are distinct All-Reduce collective algorithms tailored for fundamental network topologies (a.k.a. textbook collective algorithms). Table 5.1 exemplifies several predefined collective algorithms and their target topologies. Topology abbreviations denote RI (ring), FC (fully-connected), SW (switch), Ho (Homogeneous), and Ht (Heterogeneous). \triangle denotes the collective performance is closely tied to specific network configurations.

Naively deploying topology-aware collectives over non-preferred physical topologies could induce link congestions and underutilization of the network resources. This is demonstrated in Figure 5.1. Direct, recursive halving-doubling (RHD), and ring (also TACOS for comparison) over different network topology—fully-connected (FC), ring, two-dimensional mesh, and three-dimensional hypercube (HC)—are evaluated. Each cell at $(src, dest)$ de-

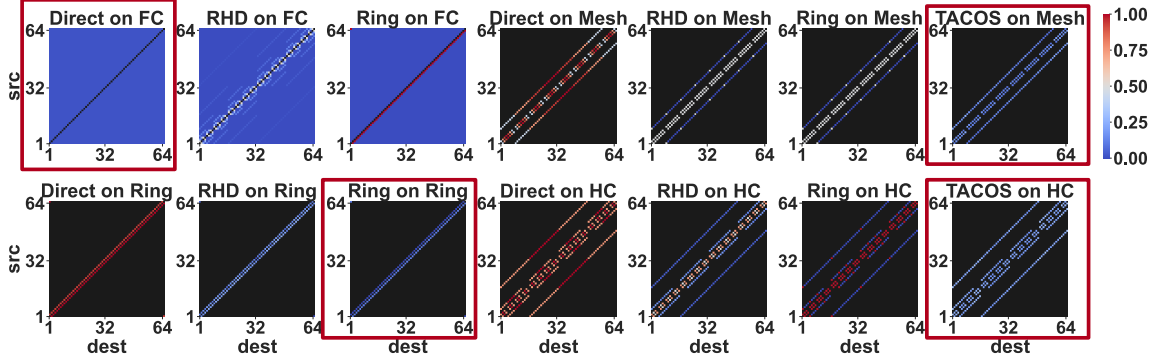


Figure 5.1: Heat map of total message size transferred over each link, when running 1 GiB All-Reduce using textbook algorithms over different network topologies.

Table 5.2: Qualitative comparison of collective algorithm synthesizers.

Framework	Network		Synthesis Mechanism		
	Asymmetric	Heterogeneous	Autonomous	Removes Congestion	Scalable
SCCL [133]			✓		
Blink [109]	✓		✓		
MultiTree [138]	✓		✓	✓	
TACCL [132]		△	△	△	
TACOS	✓	✓	✓	✓	✓

notes a link connecting NPU *src* to NPU *dest*. If there is no such link in the topology, the cell is marked black. All values are normalized to the largest value per topology. For every topology, scenarios running topology-aware collective algorithms are marked with a red box, which results in balanced, lowest overall loads (i.e., a cooler heat map). When textbook collective algorithms are executed over their non-preferred topologies, significant link underutilization and oversubscriptions occur. This is the reason CCLs today have heuristics built-in to switch among textbook collective algorithms. Unfortunately, beyond a limited set of textbook collective algorithms, there are no known solutions for arbitrary topologies. Given this complexity, when given an unknown topology, CCLs today default to employing the ring algorithm. Either one logical ring is mapped over the physical topology, or multiple parallel rings [19, 105]. But the problem of over/undersubscription remains. However, this approach may result in oversubscription and underutilization of certain links.

To address the aforementioned challenge, recent studies have demonstrated the feasi-

bility and effectiveness of autonomously synthesizing topology-aware collective algorithms when provided with a target physical network and collective patterns [109, 132, 133, 138]. These efforts are outlined in Table 5.2. Specifically, TACCL is marked \triangle as it requires assumptions (communication sketch) to model network heterogeneity and congestion. However, they are limited to supporting a specific subset of topologies, typically homogeneous or symmetric networks. Furthermore, they often overlook network congestion effects during the search, resulting in suboptimal synthesis results. A number of works treat collective synthesis as a global optimization problem, which poses NP-hard complexity to the search space, fundamentally limiting their scalability [139]. Consequently, their evaluations have been restricted to only tens of NPUs.

In this dissertation, we propose an autonomous collective synthesizer named TACOS: topology-aware collective algorithm synthesizer. Given a network topology and target collective pattern, TACOS autonomously synthesizes a static collective algorithm, which can be leveraged by CCLs. This is done by iteratively maximizing network resource utilization throughout the course of collective execution. TACOS showcases better quality synthesis results, orders of magnitude better scalability, and encompasses a larger class of topologies compared to previous endeavors. This is enabled by two features. Firstly, we frame collective algorithm synthesis as a link-chunk matching problem using a time-expanded network (TEN) representation, an acyclic directed graph integrating spatial and temporal dimensions [140, 141, 142]. Secondly, we employ a link-chunk matching algorithm to maximize network utilization and minimize congestion. Here is a summary of our contributions:

- TACOS brings the TEN representation into the domain of distributed ML, which is a widely adopted data structure for traffic and flow optimizations.
- TACOS accommodates a wide range of arbitrary (i.e., heterogeneous and asymmetric) topologies.

Table 5.3: Common model and data parallelization strategies and their required collective communication patterns.

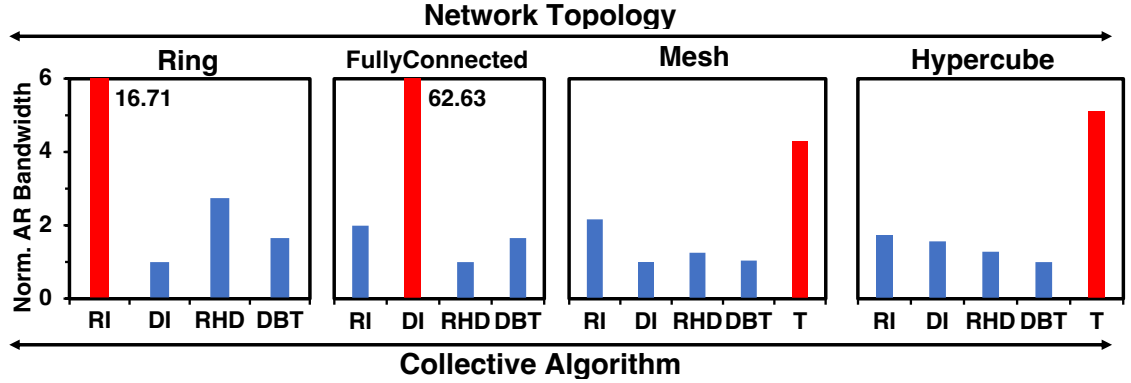
Parallelization	Reduce-Scatter	All-Gather	All-Reduce
Data Parallelism [27]			✓
Tensor Parallelism [27]			✓
FSDP [50]	✓	✓	
ZeRO [11]	✓	✓	
Hybrid [27]	✓	✓	✓

- TACOS considers network congestion effects throughout the synthesis process, resulting in high-quality search outcomes.
- TACOS features better scalability, surpassing previous works usually restricted to tens of NPUs. TACOS synthesized collective algorithms for a 40,000 NPU topology in 2.52 hours.

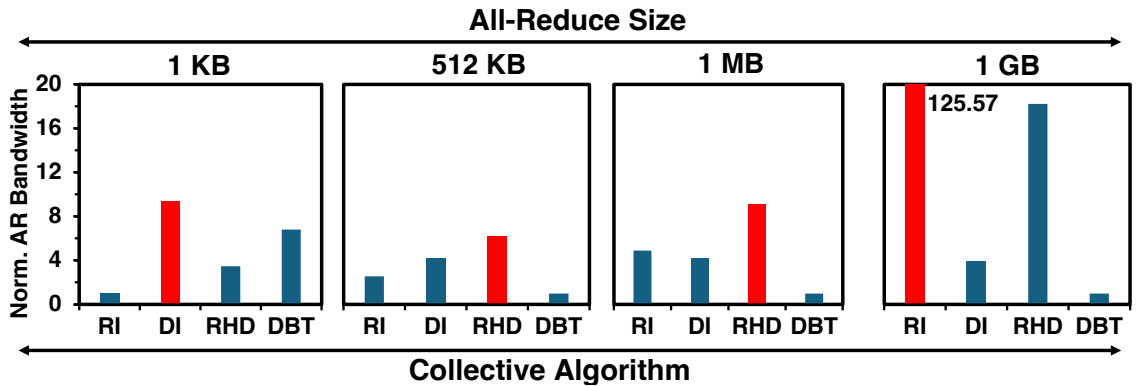
5.2 Background

5.2.1 Collective Algorithm Synthesizer

As discussed in previous chapters, executing topology-aware collective algorithms can maximize collective performance for a given target topology. However, manually designing such algorithms is costly, requiring significant engineering and validation effort [39, 133]. To address this challenge, collective algorithm synthesizers have been proposed [39, 109, 132, 133, 138, 143, 144]. Rather than relying on human experts, synthesizers are automated frameworks that generate topology-aware collective algorithms. A synthesizer takes the target network topology as input and autonomously produces optimized, topology-aware collective algorithms.



(a) All-Reduce bandwidth of baseline algorithms measured with 1GB All-Reduce on different topologies



(b) All-Reduce bandwidth of baseline algorithms measured on a Ring with different All-Reduce collective sizes

Figure 5.2: Effect of topology-aware collective algorithms to the All-Reduce bandwidth.

5.3 Challenges

5.3.1 Importance of Topology-aware Collective Algorithm

Table 5.3 shows some of the parallelism schemes used in practice today and the corresponding collectives. The physical characteristics of the network topology (including connectivity, link latencies, and link bandwidths) heavily influence network transmission behavior. We define a topology-aware collective algorithm as one intricately optimized alongside the underlying topology to achieve optimal collective performance [42, 145]. Table 5.1 summarizes common topology-aware algorithms and their target topologies.

This effect can be observed in the link subscription heat map in Figure 5.1. We also

quantify these effects in Figure 5.2. Figure 5.2(a) shows the All-Reduce bandwidth (i.e., collective size \div collective time) over distinct network topologies with 64 NPUs (using link $\alpha=0.5\mu\text{s}$, $1/\beta=50$ GiB/s), measured with 1 GiB collective (for two-dimensional mesh and three-dimensional hypercube, we also run the topology-aware algorithm synthesized by TACOS (T)). Results are normalized per each graph by the smallest number.

On the ring network, the topology-aware ring algorithm exhibited a $16.71\times$ higher All-Reduce bandwidth than the direct algorithm. Conversely, on the fully-connected topology, the direct algorithm demonstrated a performance improvement of $62.63\times$ over the ring algorithm. Figure 5.2(b) also demonstrates that the optimal topology-aware collective could change depending on the target collective pattern. Figure 5.2(b) showcases the All-Reduce bandwidth for different collective algorithms on a 128-NPU physical ring (link $\alpha=30\text{ns}$, $1/\beta=150$ GiB/s), measured with varying collective sizes. Unlike the 1 GiB All-Reduce case, for 1 KiB, the direct algorithm outperformed ring since each network transmission is latency-bound and prefers short-distance collective algorithms.

These results emphasize the importance of designing optimal topology-aware collective algorithms that encapsulate both communication patterns and network topology. Modern CCLs [36, 37, 38] implement a range of predefined basic collective algorithms, selecting them based on network features as shown in Figure 5.3(a). Unfortunately, this fundamentally limits the physical topologies they can efficiently support.

5.3.2 Heterogeneity, Asymmetry, and Scale of ML Systems

To provide maximum network resources, state-of-the-art distributed ML platforms are leveraging various network topologies and technologies, introducing bandwidth heterogeneity and asymmetry. Notable examples include asymmetric two-dimensional mesh topology with wafer-scale technologies as implemented by Cerebras CS-2 [16] and Tesla Dojo [101] systems. Heterogeneous multi-dimensional scale-up and scale-out switches in NVIDIA DGX clusters [62, 131], and the integration of scale-up and photonic technologies

by Google Cloud TPU [60, 95] are other examples. Moreover, the scale of recent ML clusters is also increasing to accommodate larger workloads, incorporating thousands to even tens of thousands of NPUs. This trend is evident in Google Cloud TPUv5 pods with 8,960 TPUs [146], the OpenAI cluster with 7,500 GPUs [147], and Meta’s Research Supercluster equipped with 24,000 GPUs [148].

5.3.3 Designing Topology-aware Collective Algorithm

Given the importance of topology-aware collective algorithms in distributed ML, designing them for commonly used networks is an active research field. For example, recent proposals devise All-Reduce algorithms for two-dimensional mesh [149] and Dragonfly [150] topologies. However, this process requires engineering and validation efforts for each new topology variant. Flattened Butterfly [151], MegaFly [152], SlimFly [153], and Tofu [121] are just a few examples that do not yet have specialized collective algorithms and default to baseline collective algorithms, which could lead to network resource underutilization. Even existing collective algorithms are often tuned under multiple assumptions, such as homogeneous link bandwidths or specific message sizes, making them susceptible to potential performance degradation in practice.

5.3.4 Topology-aware Collective Synthesizer

To address the challenges inherent in designing topology-aware collective algorithms manually, the development of an autonomous synthesis framework capable of automatically generating topology-aware collective algorithms is imperative. Such a toolchain can alleviate the burden on human experts to manually configure the routing of each chunk. Additionally, by automatically exploring the design space, these frameworks have the potential to generate higher-quality collective algorithms for specific targets. This is particularly crucial as network topologies become larger and more intricate, further complicating the task of design and optimization.

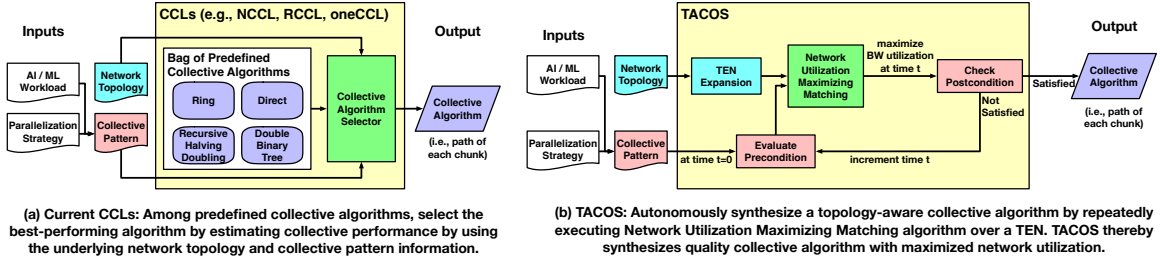


Figure 5.3: Overview of the existing CCLs and proposed TACOS architecture.

5.4 TACOS

In this section, we introduce TACOS and explain how it synthesizes topology-aware collective algorithms. TACOS architecture is summarized in Figure 5.3. Figure 5.3(a) shows that current CCLs execute collectives by selecting an algorithm from a set of predefined implementations. Figure 5.3(b) captures the high-level overview of the TACOS framework. The target network topology and collective pattern are provided as inputs. TACOS expands the network into a TEN and evaluates the collective precondition (i.e., which chunks are currently held by each NPU) for time $t=0$. Based on this information, TACOS employs a chunk-link matching algorithm that maximizes network resource utilization for the target time $t=0$. TACOS iteratively runs this matching process for successive time spans until the postcondition is satisfied (i.e., all NPUs have received every desired chunk). This procedure yields a topology-aware collective algorithm (i.e., static path of each chunk), which can then be utilized by CCLs in lieu of the predefined topology-unaware textbook algorithms.

5.4.1 Time-Expanded Network

We start introducing TACOS by bringing the notion of TEN to the distributed ML domain. TEN is widely used in flow optimization problems [141, 154, 155] because it enables judicious consideration of both network topology and timing information in a single data structure.

A TEN is a data structure that captures both spatial and temporal information of a

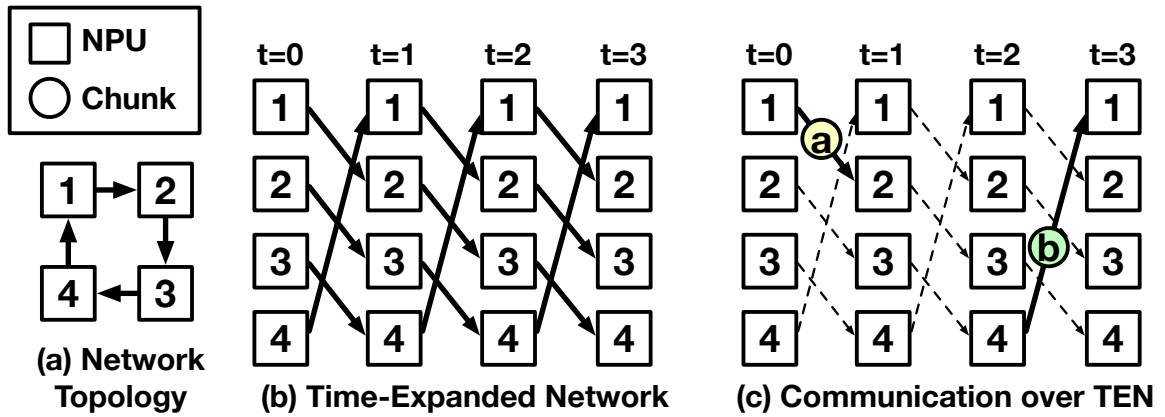


Figure 5.4: An example TEN data structure using unidirectional ring topology.

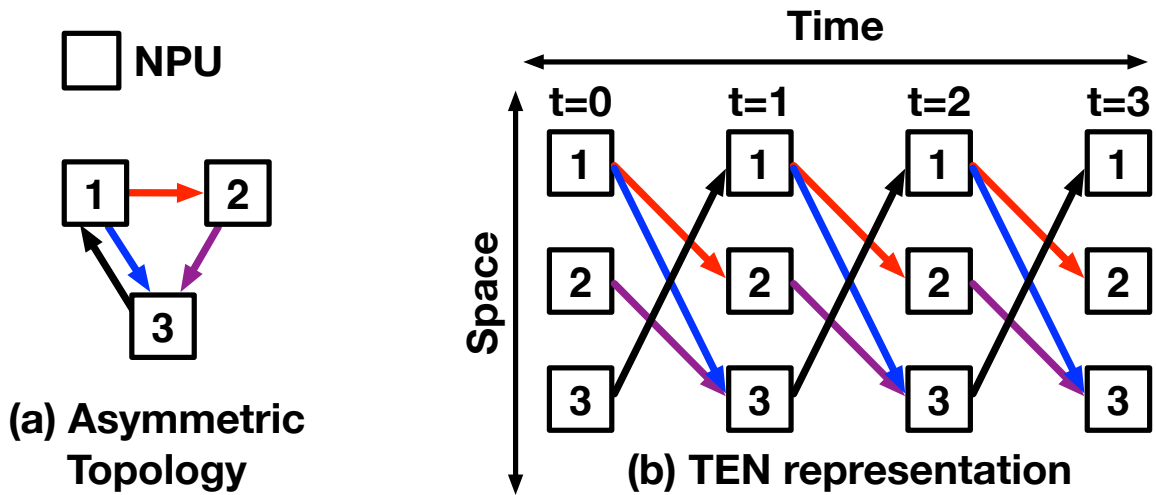


Figure 5.5: TEN representation of a homogeneous, asymmetric 3-NPU topology.

network in a unified representation. Figure 5.4 illustrates an example of a TEN representation of a four-NPU cluster. Specifically, Figure 5.4(a) is the spatial layout of a four-NPU unidirectional ring network. The TEN representation of this network topology is drawn in Figure 5.4(b), expanded up to time three. All endpoints in a network topology comprise a column, which is then duplicated across multiple timesteps (from $t = 0$ to $t = 3$ in this example). Spatial connectivities between devices are encoded as edges that span across timesteps. TEN enables intuitive modeling of network traffic. Figure 5.4(c) shows two such operations: two chunk transfer operations represented over the TEN. Chunk a is sent from NPU 1 to NPU 2 at $t = 0$, and chunk b is sent from NPU 4 to NPU 1 at $t = 2$.

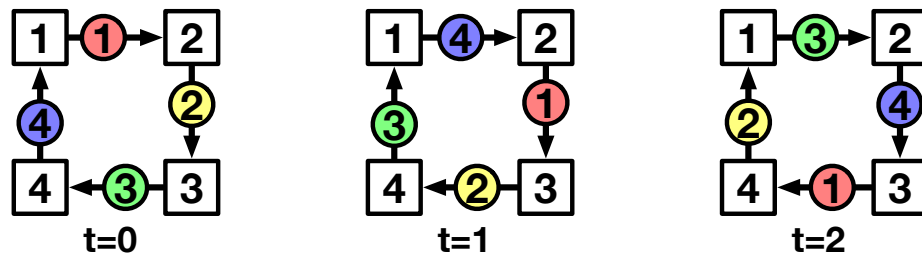
Another visual example of a homogeneous, asymmetric 3-NPU topology with 4 links is shown in Figure 5.5(a), as well as its TEN representation in Figure 5.5(b), up to $t = 3$. Each NPU in the base topology forms a column of vertices in the TEN representation, with each column corresponding to a unique time span. This column is then replicated across a time range (up to $t=3$ in this example). The topology exhibits four unidirectional connections: $1 \rightarrow 2$, $1 \rightarrow 3$, $2 \rightarrow 3$, and $3 \rightarrow 1$. These connections translate into distinct edges in the TEN representation, linking different time spans flowing from left to right.

This TEN representation effectively integrates the temporal and spatial information of the network communication behavior into a single data structure. Consider the scenario where a chunk starts being transmitted from NPU 1 to NPU 2 at $t=1$. It can be seen from the TEN that this chunk will reach its destination at $t=2$. Similarly, the absence of a direct link from NPU 3 to NPU 2 in the TEN illustrates that such a direct transmission is impossible due to the lack of the physical link in the network.

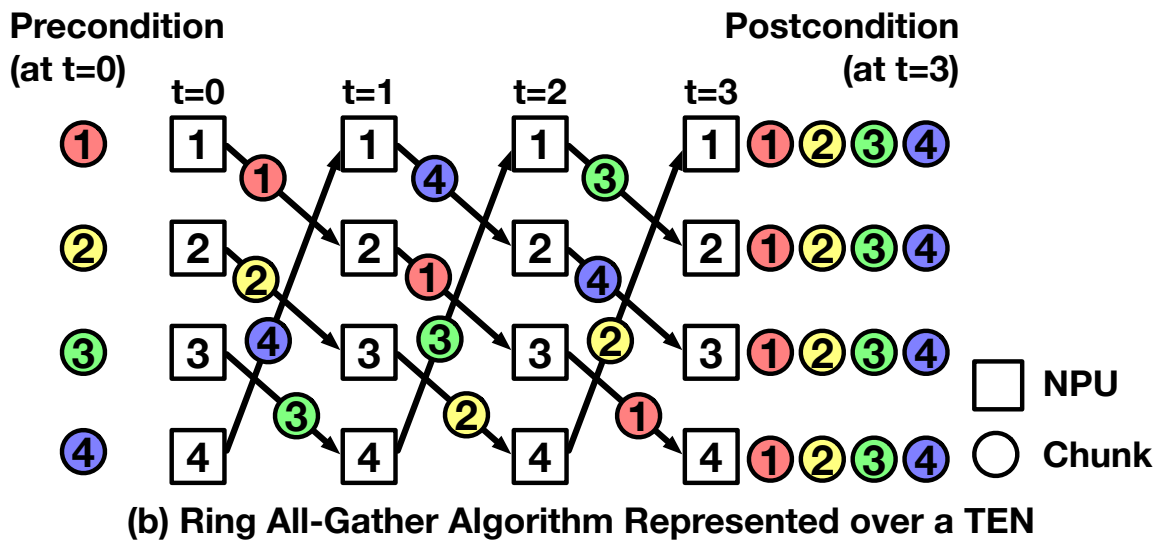
5.4.2 Representing Collective Algorithm using TEN

TEN offers a concise mechanism to represent collective algorithms in a single data structure. One example is shown in Figure 5.6. Depicted in Figure 5.6(a) is an unidirectional ring All-Gather algorithm laid over the ring topology across distinct time spans. The same collective algorithm is represented over the TEN representation in Figure 5.6(b). The collective algorithm is captured using the TEN-based representation. The leftmost segment of the TEN depicts the precondition (i.e., chunks initially held by each NPU) at $t=0$, while the rightmost segment represents the postcondition (i.e., chunks held at the end) at $t=3$.

In this visualization, a chunk transmission between two NPUs is symbolized by the chunk occupying a TEN link, which we term a link-chunk match. For this example, all links in the TEN structure correspond to matched chunks, indicating maximized resource utilization throughout the course of the collective execution. Furthermore, each link matches up to one chunk, eliminating network contention from multiple payloads assigned over a single



(a) Ring All-Gather Algorithm laid out over the Ring topology



(b) Ring All-Gather Algorithm Represented over a TEN

Figure 5.6: Representing unidirectional All-Gather algorithm using TEN data structure.

link at a time.

To the best of our knowledge, this dissertation is the first to introduce TEN to the domain of distributed ML and to represent collective algorithms atop it. We believe this formulation has value beyond the specific collective synthesis mechanism presented in this dissertation.

5.4.3 TACOS Approach to Collective Algorithm Synthesis

The TEN-based representation employed by TACOS enables it to frame the collective synthesis problem as a link-chunk match-making challenge, rather than as a global optimization problem. This strategy offers a significant advantage, as solving global optimizations (e.g., integer linear programming (ILP) formulations) is a nondeterministic polynomial (NP)-hard problem [139].

Instead, in order to synthesize a collective algorithm, TACOS aims to establish and maximize the number of link-chunk matches for individual chunks onto the TEN links. To synthesize the collective algorithm, TACOS first constructs a TEN for $t=0$ for the target network topology. It then proceeds to execute a link-chunk matching algorithm (described in Section 5.4.4) to maximize network utilization at $t=0$. The TEN is then expanded by one more time span, and the process repeats until all postconditions are met.

This iterative synthesis approach maximizes network utilization over the entire course of the collective execution, thereby producing a high-performance collective algorithm. Moreover, the framework inherently mitigates link congestion by allowing each TEN link to accommodate only one chunk. Importantly, TACOS supports arbitrary networks since any heterogeneous or asymmetric topologies can still be represented in TEN, eliminating the need for specific homogeneity or symmetry assumptions.

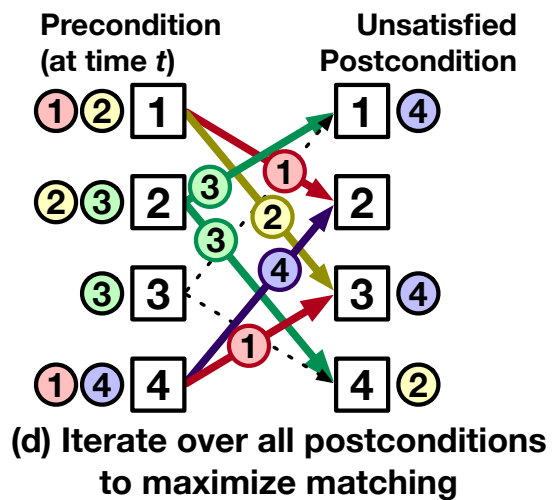
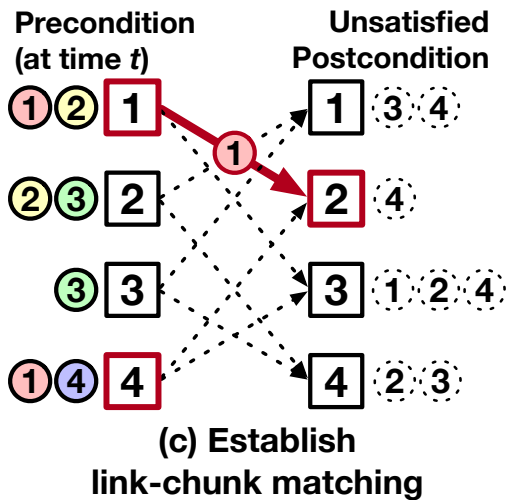
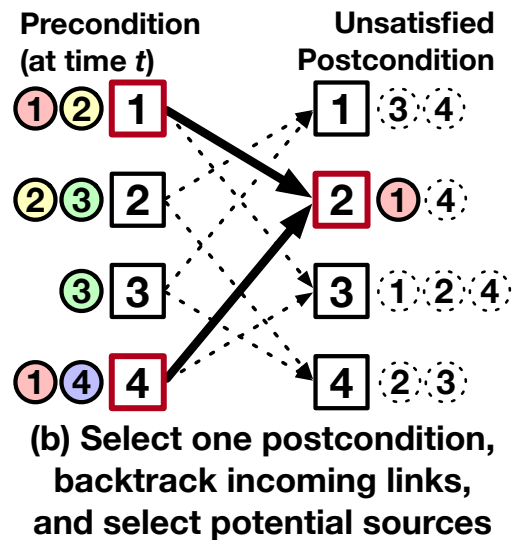
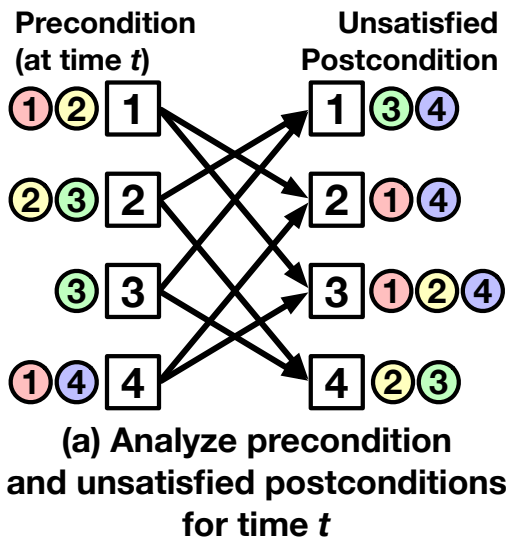


Figure 5.7: Network Utilization Maximizing Matching algorithm.

5.4.4 Network Utilization Maximizing Matching Algorithm

TACOS employs a swift network utilization maximizing matching algorithm to find link-chunk matches for a given time span t . This algorithm is summarized in Figure 5.7. For a given time span t of a TEN: Figure 5.7(a): the pre/postconditions of a collective communication are evaluated. Figure 5.7(b): an unsatisfied postcondition is randomly selected, and the destination NPU is backtracked over the TEN to identify candidate source NPUs capable of providing the requested chunk. Figure 5.7(c): one candidate is randomly chosen and a link-chunk match is made. Figure 5.7(d): steps (b)–(c) are repeated exhaustively over all unsatisfied postconditions to maximize the number of link-chunk matches, optimizing link utilization for the given time t .

The matching process begins in Figure 5.7(a), where TACOS evaluates the preconditions and unsatisfied postconditions of each NPU at the given time span t to determine the chunks requested to be received by each NPU. TACOS then randomly selects one unsatisfied postcondition to evaluate potential link-chunk matches. For instance, in Figure 5.7(b), NPU 2 and chunk 1 are selected. The algorithm then backtracks the TEN graph to identify a set of potential source NPUs capable of supplying the requested chunk, as represented by the bold arrows in the figure. These source NPUs are assessed to determine which NPUs currently possess and can provide the desired chunk. In the illustrated scenario, both NPU 1 and NPU 4 fulfill these criteria. From this pool of candidates, one source NPU is randomly selected, resulting in a successful link-chunk match as depicted in Figure 5.7(c). In this example, NPU 1 is chosen to cater to chunk 1, and the corresponding TEN link (NPU 1 to 2) becomes occupied by the established match. This sequence repeats for all unsatisfied postconditions, yielding the outcome in Figure 5.7(d).

The algorithm is summarized in Algorithm 1. For a given time span t , the matching algorithm exhaustively iterates over all unsatisfied postconditions and makes random selections whenever possible. Therefore, the algorithm tries to maximize the number of successful link-chunk matches, thereby ensuring nearly maximal utilization of network link

Algorithm 1 Utilization Maximizing Matching Algorithm

Condition: Provided TEN at time t

```
// Get unsatisfied postconds
for  $d \leftarrow \forall \text{NPUs}$  do
  for  $c \leftarrow \forall \text{Chunks}$  do
    if  $\text{postcond}[d][c] \ \&\& \ !\text{precond}[d][c]$  then
      Add  $(d, c)$  to  $\text{unsatisfied\_postcond}$ 

// Make link-chunk matches
 $\text{random\_shuffle}(\text{unsatisfied\_postcond})$ 
for  $(d, c) \leftarrow \forall \text{unsatisfied\_postcond}$  do
  // Select  $s$  NPU that can provide  $c$  to  $d$ 
   $s \leftarrow \text{backtrack}(\text{TEN}[t][*][d])$ 
   $\text{candidate\_srcs} \leftarrow \text{List}(\forall s: \text{precond}[s][c] \text{ is True})$ 
   $\text{chosen\_src} \leftarrow \text{random\_select}(\text{candidate\_srcs})$ 

  // Mark link-chunk match
   $\text{precond}[d][c] \leftarrow \text{mark True}$ 
   $\text{TEN}[t][\text{chosen\_src}][d] \leftarrow \text{mark } c$ 
```

resources at the given time t . Additionally, as only one chunk can be matched over a link, this algorithm effectively eliminates link congestion.

5.4.5 End-to-End TACOS Collective Algorithm Synthesis

To help understand the end-to-end synthesis process, Figure 5.8 provides an illustrative example synthesizing an All-Gather algorithm for a 4-NPU asymmetric network. Figure 5.8(a) depicts the target topology, while Figure 5.8(b) shows the initial TEN representation of the network for $t=0$. First, preconditions and postconditions are evaluated for $t=0$, as shown in Figure 5.8(c). Subsequently, the link-chunk matching algorithm is executed, resulting in maximized network utilization for $t=0$ as demonstrated in Figure 5.8(d). The preconditions and postconditions are updated for $t=1$, and since there are remaining unsatisfied postconditions, the TEN is expanded for one more time span, resulting in the structure depicted in Figure 5.8(e). This process repeats iteratively: the link-chunk matching

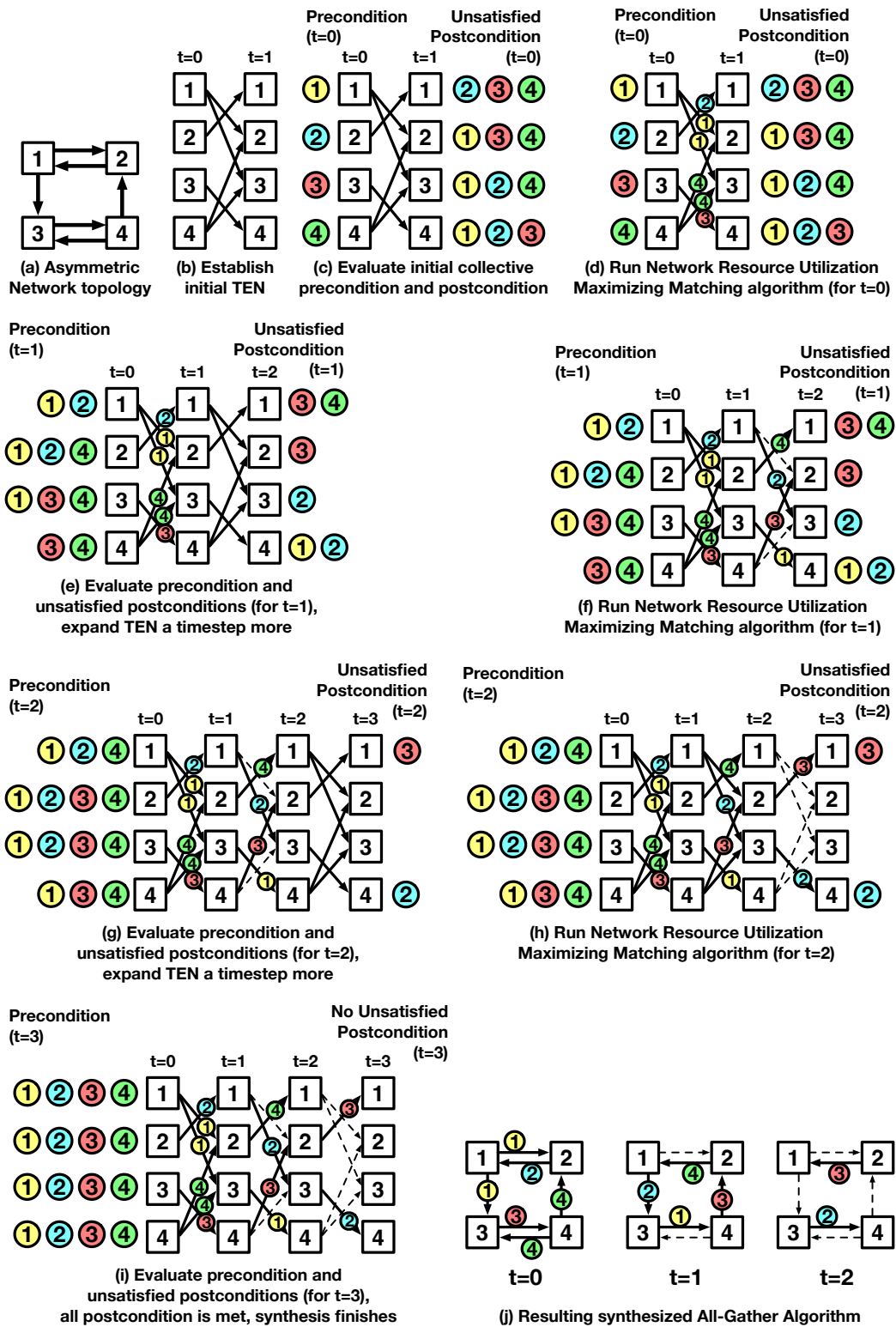


Figure 5.8: End-to-end overview of how TACOS constructs an All-Gather algorithm for a homogeneous, asymmetric 4-NPU topology.

Algorithm 2 TACOS End-to-End Synthesis

Condition: Provided target topology and collective

Initialize: TEN[$t = 0$], pre/postconds

while \exists unsatisfied_postconds **do**

 // Expand TEN by 1 time span and run matching

$t \leftarrow t + 1$

 Expand TEN[t]

 Utilization_Maximize_Matching(TEN[t]) (Figure 1)

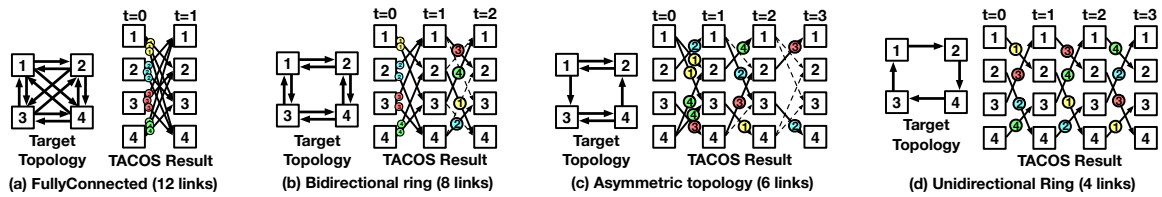


Figure 5.9: Distinct target topologies with 4 NPUs but different numbers of links, and their All-Gather synthesis results over TEN using TACOS.

algorithm is executed for $t=1$, once again maximizing network utilization, as shown in Figure 5.8(f). The pre/postcondition re-evaluation, TEN expansion, and link-chunk matching are repeated for $t=2$, as depicted in Figure 5.8(g–i). Once all postconditions are satisfied, the synthesis terminates at Figure 5.8(i). The resultant collective algorithm, laid out on the original network topology, is summarized in Figure 5.8(j), ready to be deployed by CCLs.

This end-to-end synthesis algorithm is summarized in Algorithm 2. We first initialize TEN and postconditions for $t = 0$. Until all postconditions are met, TACOS simply expands the TEN by one more time span and runs the utilization maximization matching process. Since TACOS only schedules communications between neighboring NPUs, the TACOS-synthesized algorithm is deadlock-free [138].

Figure 5.9 illustrates how TACOS synthesis results react to changes in the underlying topology. All target topologies in Figure 5.9(a)–(d) have 4 NPUs but different numbers of physical links connecting them. As the network connection becomes more sparse, TACOS is required to expand the TEN for more time spans to satisfy all collective postconditions.

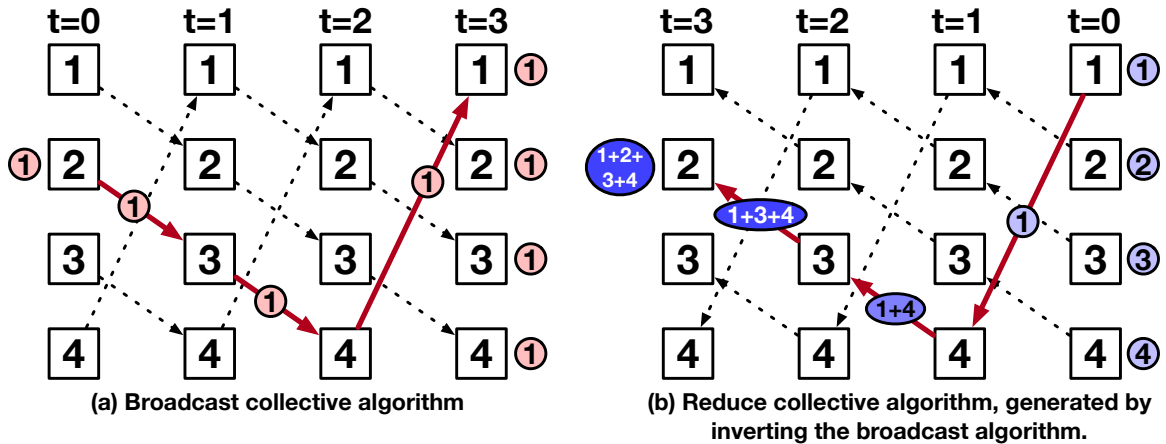


Figure 5.10: Synthesis of collectives with reduction operations (e.g., Reduce-Scatter).

Still, note that the matching algorithm was able to maximize the network utilization for every time span. For the fully-connected topology, all postconditions can be satisfied in a single shot, effectively yielding a direct algorithm. As connectivity becomes scarcer, the execution of the collective takes longer. It is important to note, however, that TACOS remains effective in maximizing link utilization for each discrete time span, resulting in optimal collective algorithms with minimized collective time.

5.4.6 Collective with Reduction Operation

Certain collectives, such as Reduce-Scatter or All-Reduce, involve reduction operations among chunks. TACOS handles such collectives by synthesizing their non-combining counterpart collective patterns.

To synthesize Reduce collective, all $(src, dest)$ links are inverted into $(dest, src)$ and the corresponding Broadcast algorithm gets synthesized. By reversing the synthesized algorithm back, TACOS can obtain the Reduce algorithm. An illustrative example with Broadcast and Reduce collectives is shown in Figure 5.10. Figure 5.10(a) shows that TACOS first constructs a TEN whose $(src, dest)$ are reversed to $(dest, src)$ (reversed link directions) and synthesizes corresponding non-reduction collectives. Then, TACOS reverses the synthesized collectives back to get the final combining collectives, as shown

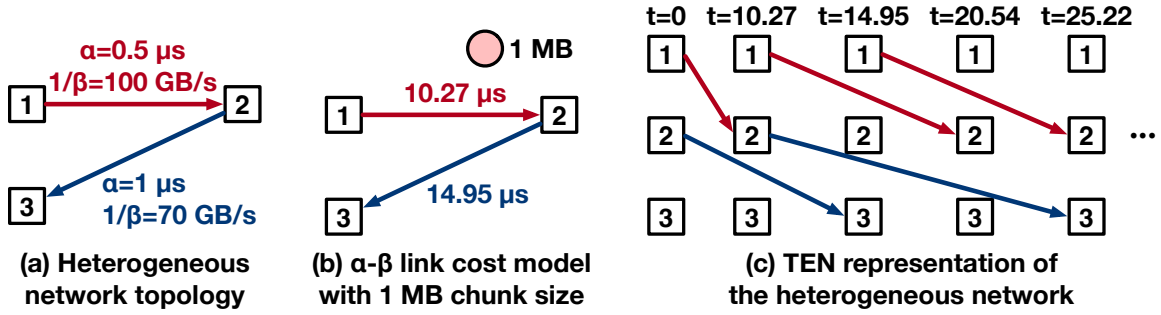


Figure 5.11: An example representation of a heterogeneous network topology in TEN.

in Figure 5.10(b).

Similarly, search for a Reduce-Scatter algorithm can be accomplished by synthesizing an All-Gather algorithm and reversing the sender-receiver of all communication pairs. An All-Reduce operation can be synthesized by combining a Reduce-Scatter followed by an All-Gather.

5.4.7 TACOS for Heterogeneous Network

The basics of conducting link-chunk matching over TEN remain consistent even for heterogeneous networks. To delineate heterogeneous link costs, we adopt the α - β model [19, 44, 76, 132, 133]. α characterizes the latency of the link, whereas β represents the reciprocal of link bandwidth (i.e., serialization delay per unit message size). Given a chunk size n , we can translate each link transmission cost into a single number, $\alpha + \beta \times n$. Once the link cost is set, a TEN representation can be drawn.

Figure 5.11 shows this process. The network configuration in Figure 5.11(a) features a heterogeneous 3-NPU topology. Each link delay is derived using the α - β model in Figure 5.11(b) (assuming a 1 MiB chunk in this scenario). Figure 5.11(c) shows the resultant TEN representation, expanded up to $t=25.22 \mu\text{s}$, which can then be absorbed by the TACOS framework. Although we have not considered NPU or memory performance during synthesis in this dissertation, these factors can be also incorporated into the α and β costs of each link.

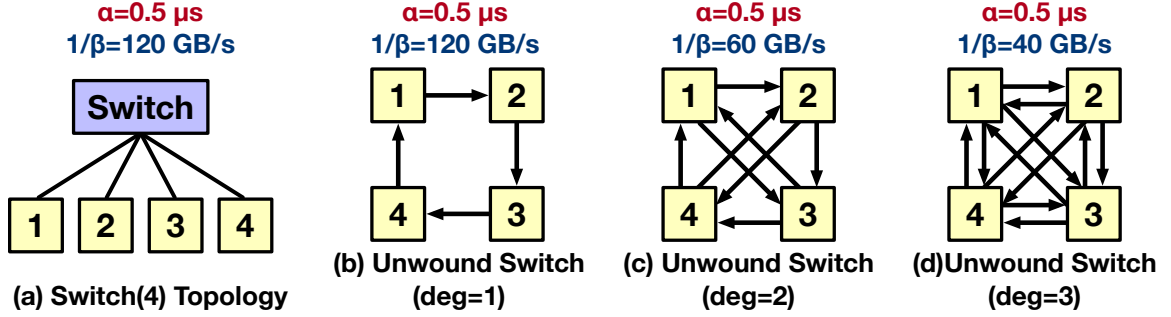


Figure 5.12: Example unwinding of a 4-NPU switch network into a direct-connect topology.

When conducting link-chunk matching, multiple links with different transmission costs may emerge as potential matches due to network heterogeneity. To minimize collective time, TACOS prioritizes the link with the lowest cost when making link-chunk matches.

5.4.8 TACOS for Switch-based Network

A switch topology offers versatile NPU connectivity, but unregulated use leads to sub-optimal performance due to network contentions or even deadlocks. To address this, we propose a mechanism to unfold switch networks into fixed, point-to-point connections. With a degree d unwinding approach for an N -NPU switch, we establish d outgoing links from each NPU i , connecting to NPUs $(i+1)$, $(i+2)$, \dots , $(i+d) \pmod{N}$. During this transition, the cost α associated with each point-to-point link remains consistent. However, due to shared link bandwidth, the β cost multiplies by $d \times$. Figure 5.12(a) illustrates a 4-NPU switch while Figure 5.12(b)–(d) depict its corresponding $d=1$ – 3 unwinding. Once the switch is unwound, the topology can be represented in TEN and used by TACOS.

This unwinding scheme neither covers all potential point-to-point configurations nor dynamic switch connectivity across TEN time spans. Notably, $d=1$ and $d=N-1$ unwinding are suitable for bandwidth and latency-critical collective synthesis, respectively [132]. However, developing a more flexible switch unwinding methodology remains a subject for future investigation.

5.4.9 TACOS Synthesis Result

MSCCLang [40] introduces an extensible markup language (XML)-based representation to capture customized collective communication algorithms, mainly using `send`, `receive`, and `reduce` operations. Such an MSCCLang XML representation can readily be deployed and executed on GPU-based systems as a drop-in replacement for CCL-based implementations. We propose using the same XML-based representation to capture TACOS-synthesized customized collective algorithms. A TACOS-synthesized operation consists of chunk transfers and reductions, which can be represented in the MSCCLang XML format without any modification. This also allows TACOS-generated collective algorithms to be executed on real GPU-based systems without specific modifications. However, we emphasize that TACOS proposes a generalized algorithm to synthesize topology-aware collective algorithms and does not make any GPU-specific considerations or assumptions. Other target platforms can also represent TACOS synthesis output in a suitable form.

5.5 Methodology

5.5.1 Baseline Collective Algorithms and Synthesizers

For performance evaluations, we compare TACOS against the following baseline All-Reduce algorithms.

- Ring, Direct: These are textbook collective algorithms showing traffic patterns explained in previous chapters. In particular, ring currently serves as the default algorithm across most CCLs [36, 37, 38].
- Recursive halving-doubling, Double binary tree: We also assessed recursive halving-doubling [42] and double binary tree [43] (in Table 5.5) since they are only suited for networks with a power-of-two number of NPUs.
- BlueConnect, Themis, C-Cube: These are topology-aware collective algorithms man-

ually crafted to accommodate specific network topologies. BlueConnect [105] develops a multi-rail All-Reduce algorithm for a symmetric hierarchical network. Themis [19] further optimizes BlueConnect through improved chunk-level scheduling. Meanwhile, C-Cube [135] manually maps two binary trees on DGX-1 and executes two tree-based All-Reduce algorithms in parallel.

- **MultiTree, TACCL:** MultiTree [138] synthesizes collective algorithms for homogeneous networks via spanning tree construction, while TACCL [132] employs an ILP-based approach for symmetric heterogeneous topologies. C-Cube and MultiTree implementations are not public, and we used their reported numbers in the paper for an apples-to-apples comparison. Since TACCL offers limited topology options, we implemented a TACCL-like baseline by integrating its ILP formulation over our TEN representation.
- **Ideal:** Finally, to show TACOS’ absolute synthesis quality, we also augmented all results with theoretically ideal collective performance. This can be derived from the topology diameter (the minimum latency for the farthest two NPUs to communicate) for α costs, and the bottleneck serialization delay (injection and ejection time of all chunks) for β .

$$\text{Ideal} = \frac{\text{CollectiveSize} \times 2(n - 1)/n}{\min_{N \in \forall \text{NPU}}(\text{BW}_N)} + \text{Diameter}$$

5.5.2 Target Topologies and Synthesis Environment

Table 5.4 lists the topologies studied, covering both homogeneous and heterogeneous networks, as well as both symmetric and asymmetric topologies. Unless otherwise specified, we set link $\alpha=0.5 \mu\text{s}$ and $1/\beta=50 \text{ GiB/s}$. TACOS synthesis time is measured using the Intel Xeon E5-2699v3 CPU.

Table 5.4: Topologies evaluated for TACOS.

Topology	Heterogenous	Asymmetric
Ring (RI)		
FullyConnected (FC)		
2D Torus		
3D Torus		
2D Mesh (Mesh)		✓
3D Hypercube (HC)		✓
2D Switch	✓	
3D Ring-FC-Switch	✓	
Dragonfly (Dragonfly)	✓	✓

5.5.3 Simulation Infrastructure

We use the ASTRA-sim2.0 distributed ML simulator [27, 156] to evaluate the performance of the baseline collective algorithms and their corresponding impact on full workloads. ASTRA-sim2.0 includes NCCL-validated implementations of the textbook collective algorithms. It also models chunking (i.e., breaking a large collective into smaller chunks) and its related effects in BlueConnect and Themis.

We validated ASTRA-sim2.0’s baseline All-Reduce implementations over two real systems: an NVLink-based, 8-GPU V100 server and a 32-TPUv3 (8×4) cluster. We observed ASTRA-sim2.0 to be 6–8% off on average, with the difference close to 1% for large message sizes [115].

5.5.4 Network Simulation Backend

To model the network oversubscription and congestion behaviors incurred by topology-unaware collectives, while enabling the simulation of large-scale network topologies, we enhanced the network modeling of ASTRA-sim2.0 by creating and attaching a congestion-aware analytical network simulation backend. The congestion-aware analytical backend simulates a message transfer by simulating the send and receive operations at the link granularity. Each link is equipped with message queues and can process (i.e., send and receive) only one message at a time. For example, if two messages contend for the same link, only

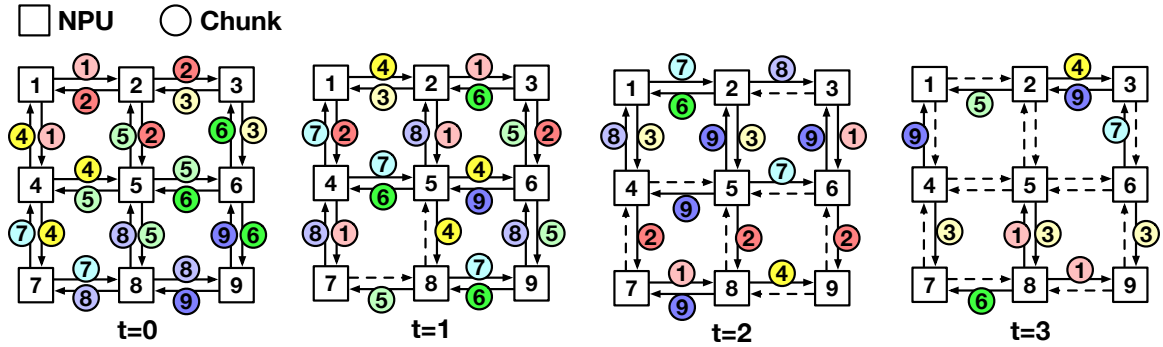


Figure 5.13: An example All-Gather collective algorithm synthesized by TACOS over a homogeneous 3×3 two-dimensional mesh topology.

one message is sent out in a first-come, first-served order. Such simulation infrastructure enables the first-order modeling of network queuing and congestion effects for networks at scale.

5.6 Evaluations

5.6.1 TACOS Synthesis Result

We begin our evaluation with a visual representation of a TACOS-synthesized algorithm. Specifically, we synthesized an All-Gather algorithm for a 3×3 two-dimensional mesh network, as shown in Figure 5.13. The synthesized algorithm effectively avoids network contentions, demonstrating TACOS' ability to autonomously alleviate link congestion during synthesis.

5.6.2 Analysis of TACOS-Synthesized Algorithm

Modern distributed ML clusters typically employ symmetric topologies, whether homogeneous [60, 95, 146] or heterogeneous [19, 107, 111]. This section highlights the benefits of TACOS in handling these regular, symmetric topologies, compared to baseline collective algorithms and existing solutions.

Topology Exploration

We synthesized topology-aware All-Reduce algorithms for three distinct systems: a Dragonfly topology (4×5) with link bandwidths of [400, 200] GiB/s (per dimension), a 2D switch (8×4) with [300, 25] GiB/s, and 3D Ring-FC-Switch ($2 \times 4 \times 8$) with [200, 100, 50] GiB/s. Notably, the Dragonfly topology is both asymmetric and heterogeneous, while switch and 3D Ring-FC-Switch are symmetric topologies.

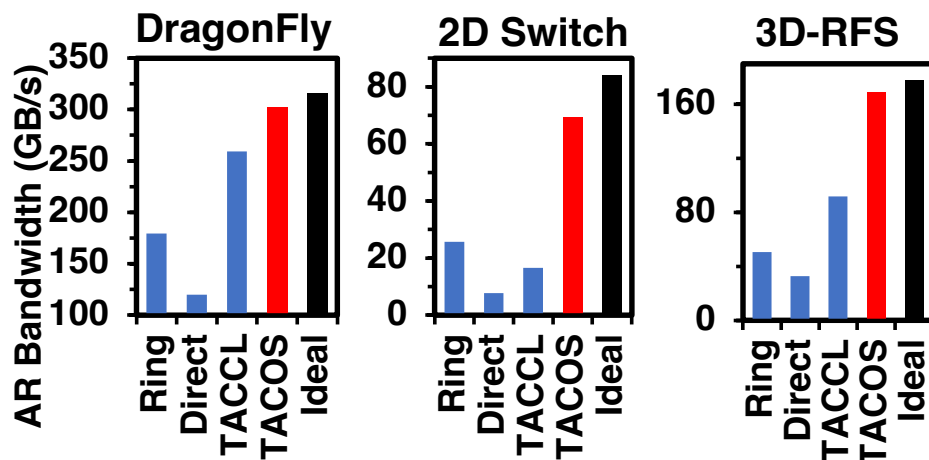
Figure 5.14(a) illustrates the All-Reduce bandwidth (i.e., collective size \div collective time) synthesized by TACOS, alongside ring, direct, and TACCL-synthesized algorithms. TACOS effectively synthesizes collective algorithms, resulting in an average speedup of $2.56\times$ over the baseline algorithms, including symmetric topologies. Moreover, thanks to its congestion-free search mechanism, TACOS even outperforms TACCL, consistently achieving more than 90% efficiency compared to the theoretical upper bound.

The heat map in Figure 5.14(b) shows the average link utilization across all links for the Dragonfly and 3D Ring-FC-Switch networks. Topology-unaware basic algorithms lead to significant oversubscription on certain links, leaving others underutilized. However, the topology-aware collective algorithms synthesized by TACOS improve overall link utilization, distributing traffic more evenly across all links. TACOS achieved 90.84% efficiency compared to the theoretical ideal.

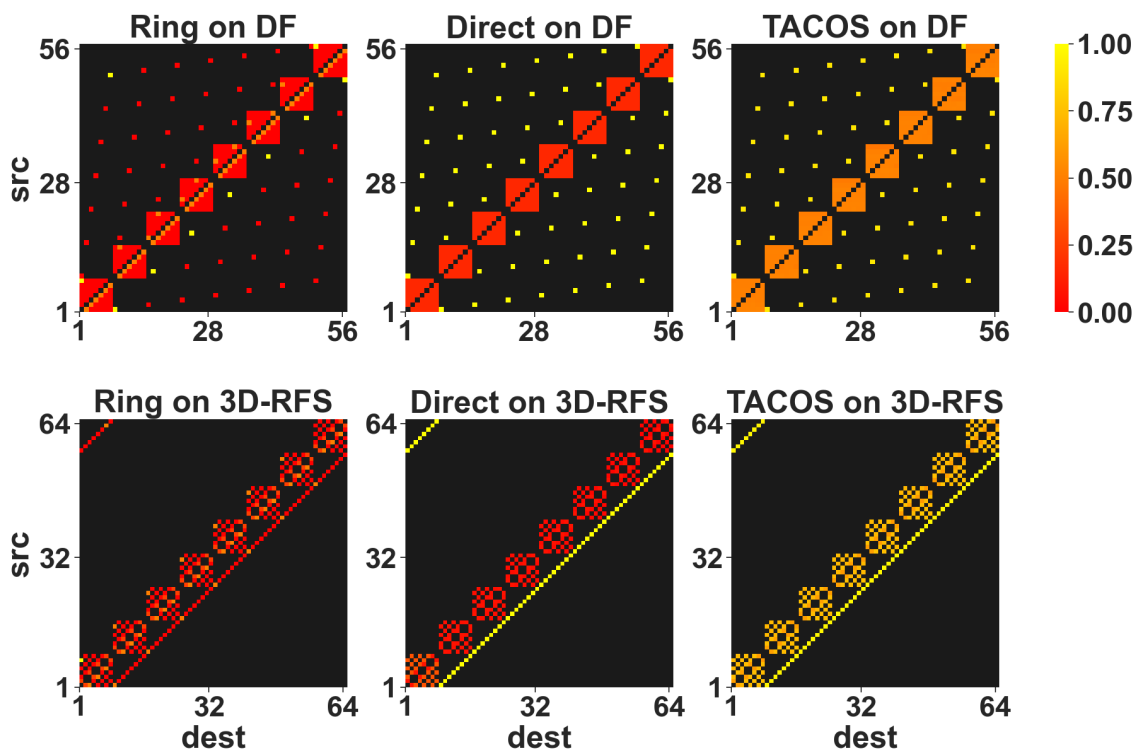
Multi-node Analysis

To demonstrate TACOS' applicability for multi-node, symmetric AI clusters, we focus on the 3D Ring-FC-Switch topology and scale it by increasing the size of the last dimension (i.e., node). For instance, a 16-node system has $2 \times 4 \times 16 = 128$ NPUs.

Table 5.5 provides a comprehensive summary of these evaluations, with 2 to 16 nodes (16 to 128 NPUs), normalized over TACOS. On average, TACOS achieved 75.88% efficiency compared to the theoretical ideal. For a 128-NPU cluster, TACCL incurred intractable synthesis times due to its NP-hard approach. On average, the collective algorithm



(a) All-Reduce bandwidth on heterogeneous topologies



(b) Network link utilization heat map

Figure 5.14: All-Reduce bandwidth and link utilization heat map over various network topologies.

Table 5.5: All-Reduce collective time (with synthesis time in parentheses for both TACOS and TACCL) for a multi-node system.

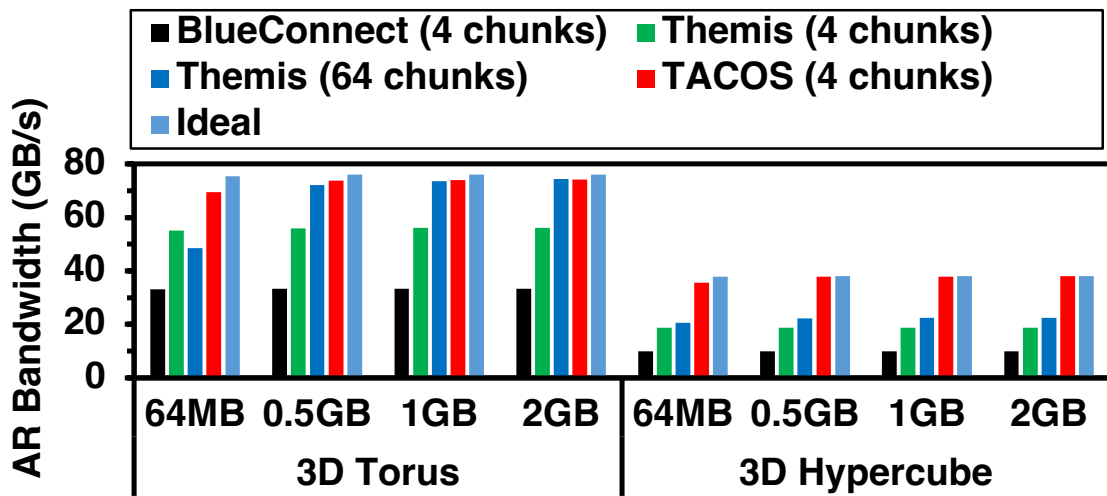
#NPU (#Nodes)	TACOS (ms)	TACCL (ms)	Ring	RHD	Direct	Ideal
16 (2)	1 (0.63)	2.89 (790)	7.14	5.27	4.04	1.00
32 (4)	1 (8.90)	4.10 (2001)	5.10	4.42	7.86	0.72
64 (8)	1 (97.92)	4.27 (7016)	4.80	5.83	16.84	0.68
128 (16)	1 (1080)	-	4.82	9.85	36.02	0.68

generated by TACOS exhibited a $5.39\times$ speedup over the ring baselines across all configurations. direct baselines suffered as the direct algorithm caused heavy network contention across the topology. TACOS achieved 75.88% efficiency relative to the theoretical ideal.

BlueConnect and Themis

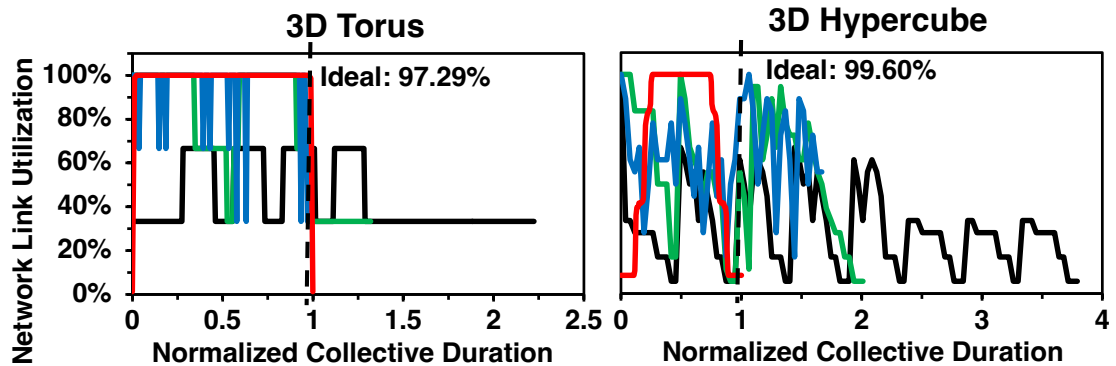
To evaluate TACOS performance against manually designed topology-aware collectives, we compare its results with the BlueConnect [44] and Themis [19] algorithms. The BlueConnect algorithm is designed for executing All-Reduce on symmetric multi-dimensional networks by applying Reduce-Scatter and All-Gather sequentially across each network dimension. Themis improves BlueConnect’s efficiency by allowing distinct chunks to traverse network dimensions in an arbitrary manner, thereby balancing the load. We assessed TACOS’ performance over a three-dimensional torus topology with $\alpha=0.7 \mu\text{s}$ and $1/\beta=25 \text{ GiB/s}$ links.

We measured TACOS’ benefits over Themis’ target topology, a symmetric 3D Torus. The All-Reduce bandwidth is summarized in Figure 5.15(a). TACOS achieved 95.90% efficiency relative to the ideal case, regardless of collective size, while optimizations like Themis may struggle with latency-critical collectives. Themis, using 64 chunks, achieved similar efficiency for large collectives, but this came at the cost of latency. For latency-critical small collectives, Themis’ efficiency dropped to 64.37%. When only 4 chunks per



All-Reduce Size

(a) Measured All-Reduce bandwidth



(b) Link utilization throughout collective duration

Figure 5.15: TACOS All-Reduce bandwidth and link utilization compared to BlueConnect and Themis.

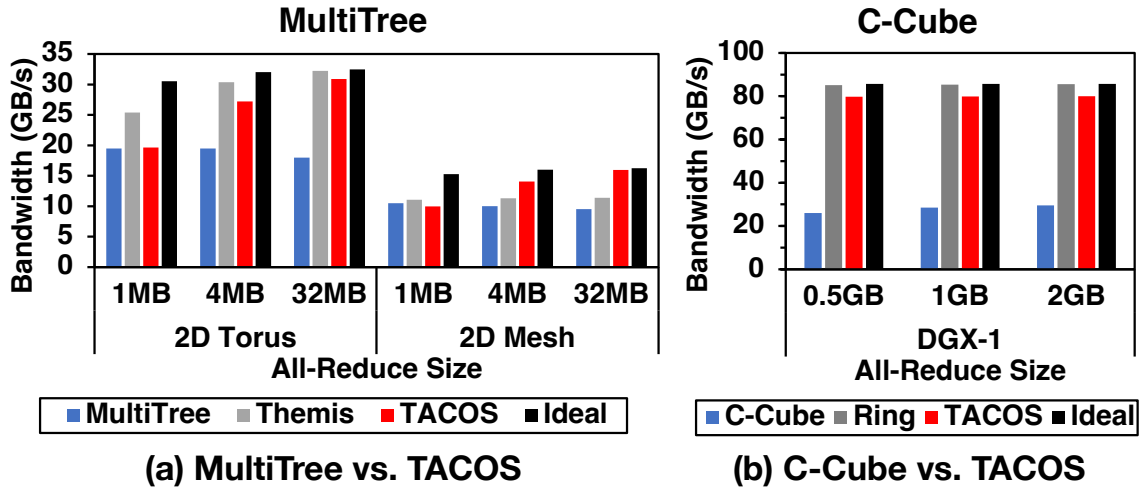


Figure 5.16: All-Reduce bandwidth of TACOS and relative speedups compared to Themis, MultiTree, and C-Cube.

collective were used to avoid this issue, TACOS consistently outperformed Themis. Furthermore, for Themis-unfriendly asymmetric topologies like the 3D Hypercube, Themis achieved only 49.43% efficiency, while TACOS reached 98.10% of the ideal case. This discrepancy arises because Themis cannot alter the path each chunk takes, limiting its ability to mitigate network contention when the optimal collective path for each network dimension is unknown. Consequently, TACOS demonstrated an average All-Reduce bandwidth $2.01 \times$ higher than Themis.

Figure 5.15(b) illustrates the link utilization during the collective execution, normalized by TACOS collective time. TACOS achieves 98.44% efficiency on average, regardless of topology shape. For the symmetric three-dimensional torus, both TACOS and Themis achieved nearly 100% utilization. However, in the three-dimensional hypercube topology, Themis experienced significant fluctuations in utilization due to network contention, which TACOS successfully avoided. We emphasize that TACOS achieved these results autonomously, without requiring any manual design efforts.

Comparison over MultiTree

We conducted a comparison between TACOS and MultiTree, a spanning tree construction-based collective synthesizer [138]. The evaluation used 2D Torus and two-dimensional mesh topologies with $\alpha=0.15 \mu\text{s}$ and $1/\beta=16 \text{ GiB/s}$ links. The summarized results are presented in Figure 5.16(a), alongside corresponding Themis results and the ideal upper bound.

Initially, TACOS demonstrated comparable performance for 1 MiB All-Reduce, but outperformed as the collective size increased, exhibiting an average speedup of $1.32\times$. This is because, unlike TACOS, which can schedule and overlap multiple chunks concurrently for maximized network resource utilization, MultiTree does not support chunk-level overlap [134]. For larger collectives with multiple chunks, MultiTree cannot exploit full network bandwidth, leading to diminished results. Consequently, MultiTree saturated after 1 MiB All-Reduce, while Themis and TACOS continued overlapping chunks, achieving 89.93% and 92.15% of theoretical efficiencies, respectively. Since Themis does not yield optimal results for asymmetric networks, for 2D Mesh, TACOS demonstrated a notable 82.60% efficiency. This underscores TACOS' ability to synthesize near-optimal collectives autonomously, regardless of target topology or collective size.

Comparison over C-Cube

C-Cube [135] manually lays out two binary trees over a DGX-1 topology and schedules two tree-based collectives concurrently. We modeled an equivalent DGX-1 topology with $\alpha=0.7 \mu\text{s}$ and $1/\beta=25 \text{ GiB/s}$ links. The comparison is illustrated in Figure 5.16(b), where TACOS achieved $2.86\times$ better performance on average. This can be attributed to C-Cube disabling 2 out of 6 available links per NPU to map two contention-free tree routes (as shown in Fig. 10(c) of [135]). Additionally, the remaining 4 links are not always fully utilized due to the tree-based approach for All-Reduce (i.e., links remain idle when there are no chunks to overlap), further reducing effective network utilization. Compared to the ideal

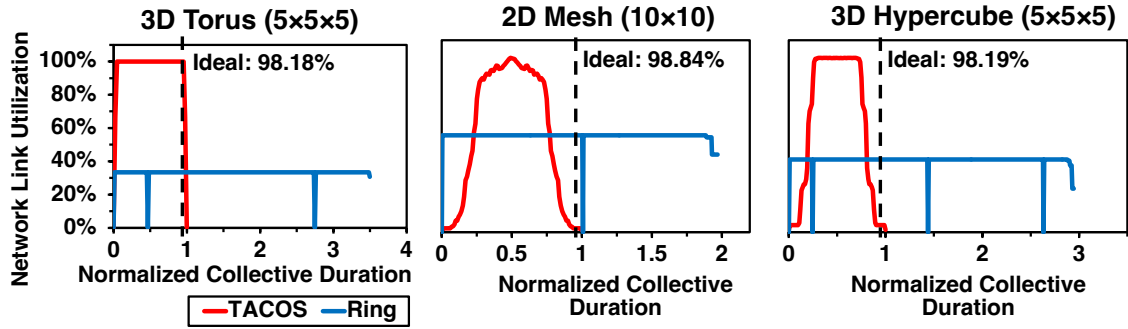


Figure 5.17: Network utilization of TACOS-synthesized and ring algorithms during the execution of an All-Reduce.

bound, C-Cube achieved 32.63% efficiency, while TACOS reached 93.26%, comparable to the Ring baseline at 99.61%.

TACOS consistently achieves near-optimal synthesis results, particularly for bandwidth-bound large collectives, resulting in an overall average efficiency of 85.25% relative to the ideal case.

Asymmetric Topology

To illustrate TACOS' efficacy over asymmetric networks, as shown in Figure 5.17, we measured link utilization during the execution of an All-Reduce collective over the homogeneous three-dimensional torus ($5 \times 5 \times 5$), two-dimensional mesh (10×10), and three-dimensional hypercube ($5 \times 5 \times 5$) topologies. The collective duration is normalized by TACOS collective time. On average, TACOS achieved 98.40% efficiency compared to the theoretical ideal.

While two-dimensional mesh and three-dimensional hypercube are asymmetric, three-dimensional torus is symmetric and shown for comparison. We also included the utilization results of the ring algorithm. It is noteworthy that the symmetric three-dimensional torus is highly suitable for collective communications, with TACOS achieving 100% utilization throughout the execution. However, the inherent asymmetry of two-dimensional mesh and three-dimensional hypercube topologies introduces inefficiencies, as not all chunks can

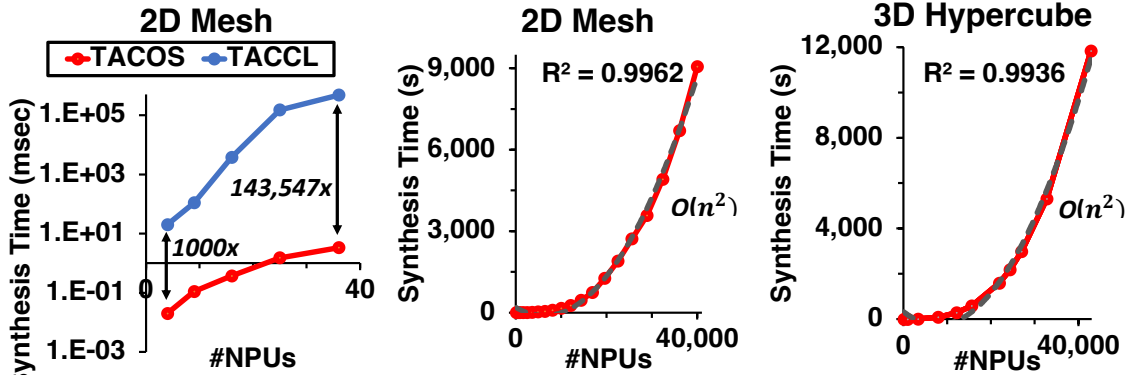


Figure 5.18: Synthesis time of TACOS and TACCL for various-sized homogeneous two-dimensional mesh and three-dimensional hypercube topologies.

depart or arrive simultaneously, causing some links to idle at the start or end of execution. This effect is visible at $t=3$ in Figure 5.13. Nonetheless, TACOS successfully maximizes link utilization after saturation. Across all scenarios, TACOS achieved 98.40% efficiency relative to the theoretical ideal.

5.6.3 Scalability Analysis

To demonstrate TACOS’ scalability, we synthesized All-Reduce algorithms for homogeneous two-dimensional mesh and three-dimensional hypercube topologies with up to 43,000 NPUs, utilizing 64 parallel threads, and measured the synthesis time. The results are summarized in Figure 5.18. TACOS used 64 parallel threads for synthesis, and TACOS synthesis time exhibited $O(n^2)$ complexity, where n is the number of NPUs.

Compared to TACCL for up to 36 NPUs, TACOS demonstrated significantly better scalability. Due to the NP-hard nature of ILP-based optimizers, the synthesis time gap between TACOS and TACCL increased from 10^3 to 10^5 as the topology size increased by $9\times$.

Further scalability analysis shows that TACOS synthesized an All-Reduce algorithm for a two-dimensional mesh topology with 40,000 NPUs in 2.52 hours and for a three-dimensional hypercube topology with 43,000 NPUs in 3.29 hours. The synthesis time scales quadratically with the number of NPUs, i.e., $O(n^2)$, where n is the number of NPUs.

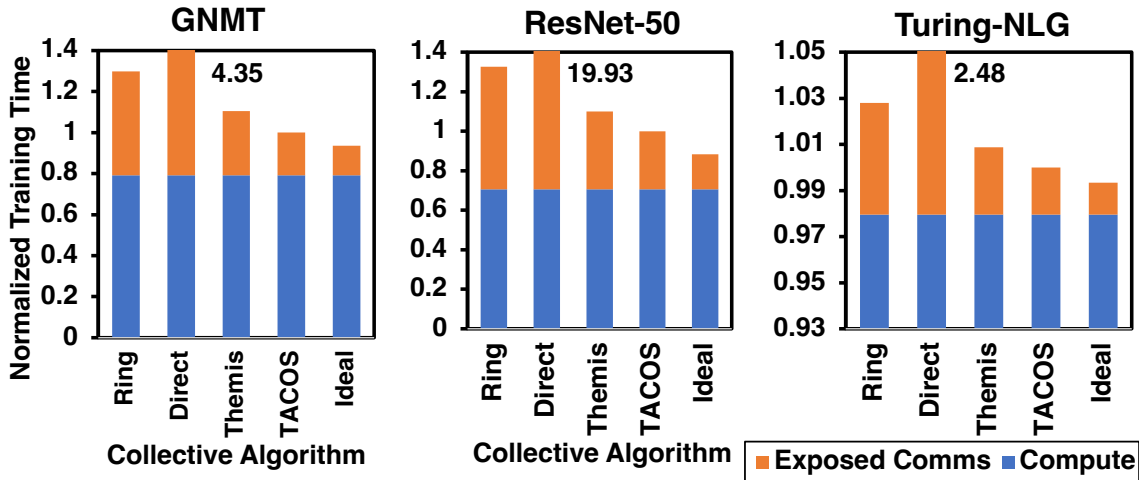


Figure 5.19: End-to-end training time of GNMT, ResNet-50, and Turing-NLG

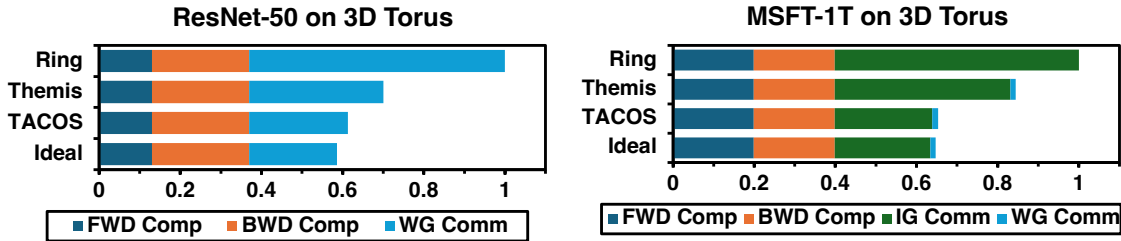


Figure 5.20: End-to-end training time breakdown of ResNet-50 and MSFT-1T over a 1,024-NPU three-dimensional torus.

This indicates that TACOS’ synthesis time is linear relative to the search space, which consists of $O(n)$ chunks and $\Theta(n)$ links.

5.6.4 End-to-End Application

Lastly, to elucidate TACOS’ implications in distributed ML, we conducted an evaluation of the end-to-end training performance of GNMT [157], ResNet-50 [117], and Turing-NLG [158], representative models of vision and LLM workloads. For models employing data parallelism, communication becomes exposed at the end of each training iteration [19]. GNMT was evaluated on a small, 8-node (64 NPU) 3D Ring-FC-Switch topology, whereas ResNet-50 and Turing-NLG were run over a larger 32-node (256 NPU) cluster. The normalized training time over TACOS is illustrated in Figure 5.19. GNMT was trained on a

64-NPU 3D-RFS system. ResNet-50 and Turing-NLG are trained on a 3D-RFS with 32 nodes. All results are normalized over the corresponding TACOS result. TACOS achieved 82.38% communication efficiency and 93.61% end training efficiency compared to theoretical bounds. Forward and backward pass computation, and exposed input/weight gradient communications are shown. All results are normalized over the Ring. TACOS achieved 97.32% efficiency over the theoretical ideal scenario.

Also, to model and showcase TACOS' applicability for larger ML clusters, we evaluated ResNet-50 and MSFT-1T [11] over a symmetric and homogeneous 3D Torus network with 1,024 NPUs. The breakdown of the normalized training time is depicted in Figure 5.20. Forward and backward pass computation, and exposed input/weight gradient communications are shown. All results are normalized over the ring baseline. TACOS achieved 97.32% efficiency over the theoretical ideal scenario.

Across all workloads, leveraging TACOS resulted in enhancements of $1.58\times$ and $1.21\times$ in end-to-end training performance over the baseline Ring and optimized Themis algorithms, respectively. Notably, compared to the theoretical upper bound, TACOS achieved the communication efficiency of 93.17%, thereby yielding an end-to-end efficiency of 97.32% compared to the ideal case.

5.7 Related Work

5.7.1 Time-Expanded Network

TEN is being leveraged in various fields that require communication or flow-based optimization. Notable examples include vehicle traffic management [154], logistics [141], and traffic signal optimization [155]. However, to the best of our knowledge, this work is the first attempt to bring TEN to distributed ML and collective communication with adequate optimization approaches.

5.7.2 Customized Collective Communication Algorithm

Collective Algorithm Representation

Some works have proposed domain-specific languages for depicting collective algorithms [40, 159]; however, these are oriented toward the manual design of human-crafted collective algorithms. Other works [132, 133] use custom representations.

Co-optimization of Collective and Topology

Previous efforts have addressed topology optimization for specific collective algorithms [160, 161]. Additionally, the co-optimization of collectives and network topology remains an active research domain [112, 138]. Given TACOS’s ability to perform synthesis on arbitrary topologies, it can harmonize with such endeavors.

Collective Algorithm Manual Design

Designing specialized collective algorithms for specific topologies is an ongoing topic of interest. Recent examples include three tree overlap for two-dimensional mesh [134] and PAARD for Dragonfly [150]. However, this approach requires engineering and validation efforts for each topology variant, which can be automated or augmented with synthesizers like TACOS.

5.7.3 Collective Algorithm Synthesizer

Solver-based

SCCL [133] synthesizes latency- and bandwidth-optimal collective algorithms by leveraging a satisfiability solver. To achieve this, SCCL captures the design space of collective communication in linear equations. To derive the linear relationships, SCCL assumes a k-synchronous collective algorithm, i.e., all NPUs executing identical transmissions clearly separated into steps and rounds. While SCCL guarantees optimality, this assumption only

holds for homogeneous, symmetric, single-node networks, meaning the SCCL approach cannot be extended beyond these scenarios. TACCL [132] overcomes SCCL’s limitations through an ILP approach instead of a satisfiability problem, removing the k-synchronous assumption. However, the ILP approach requires TACCL to capture the entire search space in a number of linear equations. Since network congestion effects cannot be captured in linear equations, they are completely ignored in the formulation. Also, because the equations must be predefined, TACCL assumed a 2D network and constructed formulations solely based on it to model heterogeneity. Although this assumption can be extended to multi-dimensional topologies, the number of cross-dimensional constraints would increase, further complicating the ILP search space. Therefore, while ILP guarantees the optimal solution of the provided formulation, as the equations themselves are limited, TACCL does not guarantee the optimality of the synthesized collective performance. Furthermore, TACCL had to assume the network is symmetrical to reduce the search space. With all these assumptions, both SCCL and TACCL synthesis efforts still scaled to only tens of NPUs.

Tree-based

Blink [109] constructs one-to-many spanning trees from a root to execute reduction and broadcast operations, effectively performing All-Reduce. The spanning tree construction process only takes network connectivity into account, thereby yielding sub-optimal results for heterogeneous topologies. To traverse the spanning tree upward and downward, all links are assumed to be bidirectional. Since the spanning tree is one-to-many, the approach is naturally detrimental for many-to-many collectives such as Reduce-Scatter or All-Gather, which are required by parallelization strategies like FSDP [50] or ZeRO [11]. Three tree overlap [134] optimizes Blink specifically for 2D Mesh topologies by constructing three spanning trees, all originating from edge NPUs. Although ideal for All-Reduce, three tree overlap still shares the inefficiency for many-to-many collectives. MultiTree [138] generates height-balanced multiple spanning trees originating from all NPUs, making it suit-

able for many-to-many collectives. However, MultiTree only takes network connectivity into account, disregarding network heterogeneity. Unlike TACOS’s link-chunk matching, which automatically considers multiple chunk overlaps, MultiTree does not allow concurrent chunks to overlap. Therefore, while both may show similar synthesis results for small, latency-critical collectives, TACOS can achieve higher network utilization for larger collectives consisting of multiple chunks.

5.8 Conclusion

The surge of artificial intelligence, particularly large language models, has driven the rapid development of large-scale machine learning clusters. Executing distributed models on these clusters is often constrained by communication overhead, making efficient utilization of available network resources crucial. As a result, the routing algorithm employed for collective communications (i.e., collective algorithms) plays a pivotal role in determining overall performance. Unfortunately, existing collective communication libraries for distributed machine learning are limited by a fixed set of basic collective algorithms. This limitation hinders communication optimization, especially in modern clusters with heterogeneous and asymmetric topologies. Furthermore, manually designing collective algorithms for all possible combinations of network topologies and collective patterns requires heavy engineering and validation efforts. To address these challenges, this chapter presents TACOS, an autonomous synthesizer capable of automatically generating topology-aware collective algorithms tailored to specific collective patterns and network topologies. We underscore the importance of topology-aware collective algorithms in distributed ML and emphasize the necessity of having an autonomous collective algorithm synthesizer that supports arbitrary topologies. We introduce TACOS, an automated framework for orchestrating collective algorithm synthesis. TACOS supports diverse networks and demonstrates near-optimal link utilization while showcasing polynomial-time scalability. TACOS is highly flexible, synthesizing an All-Reduce algorithm for a heterogeneous 128-NPU system in just 1.08

seconds, while achieving up to a $4.27\times$ performance improvement over state-of-the-art synthesizers. Additionally, TACOS demonstrates better scalability with polynomial synthesis times, in contrast to NP-hard approaches which only scale to systems with tens of NPUs. TACOS can synthesize for 40,000 NPUs in just 2.52 hours. TACOS artifact is publicly available at <https://github.com/astra-sim/tacos>.

CHAPTER 6

PCCL: PROCESS GROUP-AWARE SCALABLE AND GENERIC COLLECTIVE ALGORITHM SYNTHESIZER

6.1 Motivation

As iterated over previous chapters, topology-aware collective algorithm is a pivotal part of distributed ML. Notably, modern data center ML clusters comprise more than 100,000 NPUs [26]. As a result, collective communication among NPUs has become the most significant bottleneck in both ML training and inference [23, 24, 33].

Furthermore, in Chapter 5, we discussed that large-scale distributed ML systems employ different network technologies and topologies. As explained, collective communication algorithms define how traffic should be routed over the physical network to execute collective communications [39], and the optimal collective algorithm is highly dependent on the system’s underlying network topology [19, 137]. This further underscores the importance of topology-aware collective algorithms. For this reason, many CCLs [38, 162, 163] include customized collective algorithms tailored for their target topologies, and manually designed collective algorithms specialized for target networks have also been proposed [19, 105, 134, 150]. Unfortunately, manually designing topology-aware collective algorithms for every network topology is prohibitive, not only requiring expert knowledge but also incurring engineering and validation costs [39, 133]. This was a fundamental motivation for the development of TACOS in Chapter 5, as well as led to the development of other topology-aware collective algorithm synthesizers [109, 132, 133, 138, 143, 144]. These synthesizers take a target topology as input and autonomously generate topology-aware collective algorithms.

Although TACOS, discussed in Chapter 5, introduces an example of a scalable collec-

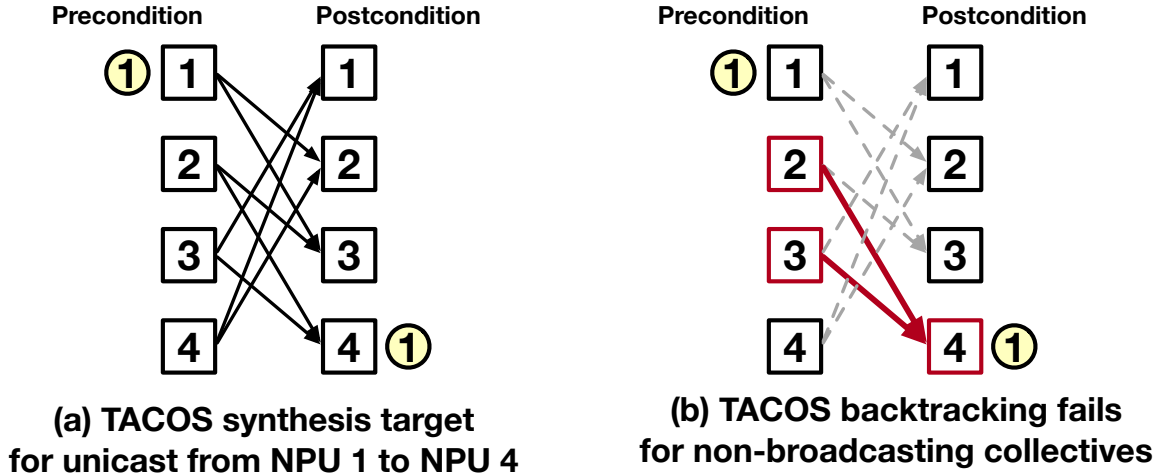


Figure 6.1: An example limitation of TACOS when synthesizing a unicast pattern.

Algorithm synthesizer, it comes with limitations, particularly in generality. An example is shown in Figure 6.1. A unicast pattern from NPU 1 to NPU 4 is provided as a synthesis target, as in Figure 6.1(a). However, as shown in Figure 6.1(b), the TACOS backtracking algorithm fails because neither NPU 2 nor NPU 3 can provide chunk 1, so no advancement in the synthesis can be made. TACOS is therefore only guaranteed to work for broadcasting collectives, such as All-Gather, despite its benefits in massive scalability and bandwidth usage maximization. To overcome the limitations of TACOS, we begin this chapter by motivating the following desired metrics for a collective synthesizer.

Scalability

The scale of ML clusters, which often comprise tens of thousands of NPUs, synthesizers must be scalable, i.e., support topologies with many NPUs. The pattern of collective communication calls change frequently as the underlying network topology and target workload change [164]. Even if collective synthesis is an offline process, if it takes hours for only tens of NPUs, the applicability would be limited.

Supporting Generic Topologies

Network topologies in ML clusters are often (i) heterogeneous, where links can vary in bandwidth and latency, and (ii) asymmetric, where each NPU may have differing numbers or shapes of connections [39]. Synthesizers should be able to model heterogeneous and asymmetric topologies.

Supporting Generic Collectives

Due to their use in both tensor and data parallelism, All-Reduce, Reduce-Scatter, and All-Gather are among the most common collective patterns in distributed ML [22]. However, MoE-based models require All-to-All (and sometimes All-to-Allv) collectives for their use of expert parallelism [5, 34]. These All-to-All communications have already emerged as a key bottleneck in MoE-based workloads [165].

Process Group Awareness

Parallelization strategies assign parts of the workload to subsets of NPUs. Collective communication is typically performed within these subsets, known as process groups [166]. It is uncommon for all NPUs in the cluster to participate in a single collective operation. Therefore, it is essential for a synthesizer to recognize and exploit process group information.

Table 6.1 shows a qualitative comparison of previously proposed collective synthesizers, including TACOS presented in Chapter 5. \triangle in scalability indicates linear programming, and \triangle in switch modeling indicates that the synthesizer unrolls switches into direct-connect topologies. Different synthesizers exhibit trade-offs in terms of scalability, supported target topologies and collective patterns, and process group awareness.

Table 6.2 further compares the synthesizers by comparing the supported collective patterns by each synthesizers. \triangle denotes synthesis is supported but is not scalable as the solution uses an NP solution. Notably, PCCL supports custom collectives, i.e., arbitrary

Table 6.1: Qualitative comparison of existing collective algorithm synthesizers.

Synthesizer	Scalable	Generic Topology	Generic Collective	Process Group Aware
SCCL [133]			✓	
TACCL [132]		✓	✓	
Blink [109]	△	✓		
MultiTree [138]	△			
ForestColl [143]	△	✓		
TE-CCL [144]	△	✓	✓	
TACOS	✓	✓		
PCCL	✓	✓	✓	✓

Table 6.2: Supported collective patterns by each collective algorithm synthesizers.

Synthesizer	Reduce-Scatter	All-Gather	All-Reduce	All-to-All	Custom
SCCL [133]	△	△	△	△	
TACCL [132]	△	△	△	△	
Blink [109]			✓		
MultiTree [138]	✓	✓	✓		
ForestColl [143]	✓	✓	✓		
TE-CCL [144]	△	△	△	✓	
TACOS	✓	✓	✓		
PCCL	✓	✓	✓	✓	✓

pre/postconditions such as multicasts, point-to-point, and All-to-Allv.

Unfortunately, as illustrated, existing synthesizers fail to meet one or more of the metrics defined above. Most of these synthesizers make trade-offs between scalability and generality of supported topologies and collective patterns. This is an artifact of the modeling limitations of existing synthesizers: for example, ILP-based solutions [132, 144] model heterogeneous networks but is limited in scalability since ILP is an NP problem [139]. Spanning tree-based approaches [109, 138, 143] improve the scalability albeit the tradeoff in their heterogeneous network and switch modeling capabilities. Moreover, none of the existing synthesizers exploit process group information; they assume that a single collective communication is executed across the entire network. Users may employ existing synthesizers by manually defining a smaller topology of interest (i.e., a subgraph) consisting only of the NPUs in the process group. However, this approach yields suboptimal synthesis results because the synthesizer cannot leverage available network resources outside the subgraph. For example, ILP-based synthesizers often assume symmetric network topologies and collectives to compensate for their scalability limitations, assumptions that can be invalidated by arbitrary process groups or collectives such as multicasts or All-to-Allv.

This chapter aims to develop a topology-aware collective synthesizer that is both scalable and generic, while leveraging process group information. We propose PCCL: a scalable, and generic, process group-aware collective communication library. PCCL achieves this objective by extending the foundations established by TACOS in Chapter 5. Specifically, PCCL adopts the TEN data structure, which integrates temporal and spatial information into a unified representation. Building on this foundation, PCCL employs a breadth-first search (BFS)-based pathfinding algorithm to synthesize generic collective algorithms while natively modeling heterogeneous and asymmetric networks with and without switches. Moreover, the BFS pathfinding algorithm naturally incorporates process group information into the synthesis process.

To summarize, this chapter makes the following contributions:

- We motivate the core requirements of collective algorithm synthesizers: scalability, support for generic topologies and collectives, and process group awareness.
- PCCL is a collective algorithm synthesizer that satisfies all these core objectives. PCCL is built on the TEN representation, which captures both temporal and spatial information of arbitrary network topologies.
- PCCL introduces a BFS pathfinding algorithm that is both scalable and generic. The BFS pathfinding algorithm supports arbitrary collectives, including All-to-All, and automatically incorporates process group information.
- Process group-aware PCCL synthesized an All-to-All algorithm $2.68\times$ faster for two-dimensional mesh topology, on average.
- PCCL All-to-All algorithm synthesis takes 11.68 minutes for an 512-NPU cluster, more than 3 orders of magnitude faster than a state-of-the-art synthesizer.

6.2 Background

6.2.1 Process Group

TACOS and other collective synthesizers assumed that a collective communication occurs on every NPUs on a given topology. In practice, since parallelization strategies distribute a job across subsets of NPUs, collective communication typically runs in a more localized fashion; not all NPUs in the cluster execute a single collective communication. Instead, a small set of NPUs in the cluster executes collective communication amongst themselves.

Process group is the term used to denote each such set of NPUs executing a collective communication [166]. As an example, Figure 6.2 shows two process groups over a six-NPU cluster. Chunk a , b , and c is defined as chunks $1 + 4 + 7$, $2 + 5 + 8$, and $3 + 6 + 9$, respectively. Process group $\{1, 2, 3\}$ are executing Reduce-Scatter, while process group $\{4, 5, 6\}$ are running All-Gather. NPUs $\{1, 2, 3\}$ execute Reduce-Scatter among themselves,

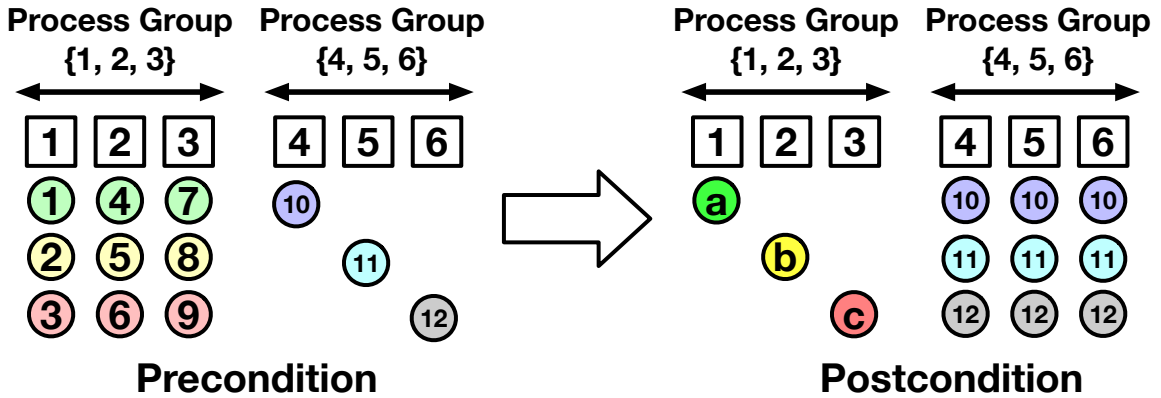


Figure 6.2: Visualization of process groups over a six-NPU cluster.

forming a process group. Another process group, composed of NPUs {4, 5, 6}, is running All-Gather.

6.3 Challenges

In this section, we motivate the key requirements that a collective algorithm synthesizer should satisfy to be practically deployable, and define the problem statement for PCCL.

6.3.1 Scalability

Modern ML clusters comprise tens to hundreds of thousands of NPUs [15, 26]. Over such a cluster, training and inference jobs are distributed using model and data parallelism strategies. Each model-parallel process group may include tens of NPUs [51], while the remaining NPUs are often organized into data-parallel process groups, resulting in a set of hundreds to even thousands of NPUs [167, 168]. Specifically, due to the changes in model architecture, network topology, optimization techniques, and hyperparameter tuning, collective patterns issued by the workload experiences frequent changes [164]. Consequently, it is critical for a synthesizer to support targets with hundreds to thousands of NPUs in a tractable time.

6.3.2 Generic Topology Support

AI supercomputers often utilize multiple networking technologies. On-package [169] high-bandwidth links, NPU-to-NPU direct memory access (DMA) links [89, 130, 170], scale-out interconnects [171, 172], even photonic networks [95], are all being leveraged within a single system. Due to such diverse network technology and connectivity options, the network topologies employed in ML clusters are captured as highly asymmetric and heterogeneous [39]. Therefore, the synthesizer should not be limited to symmetric and homogeneous networks and should support generic topology options.

6.3.3 Generic Collective Support

For example, MoE-based generative models [168, 173, 53] have gained their popularity since they can retain the computation requirement while massively increasing the model parameters [3], and it comes with the All-to-All communication cost to assign input tokens to appropriate experts [5]. In fact, All-to-All communication takes more than 60% of the total execution time as the cluster size increases [165]. However, it is surprising to note that no CCLs implement specific collective algorithms for the All-to-All pattern [38, 162, 163]. Instead, direct (i.e., pairwise send-recv) patterns are manually implemented using the CCL's send-recv operations.

6.3.4 Process Group Awareness

ML models are dispatched across the AI cluster through parallelization strategies, such as model and data parallelism. Collective communication often runs locally within the parallelization group, known as a process group. For example, All-Reduce is run across data parallel groups, and All-Gather is often executed across tensor parallel groups. Therefore, synthesizers must reflect this and should be process group aware, rather than synthesizing collective algorithms across the entire provided cluster.

6.3.5 Problem Statement

Design a collective algorithm synthesizer that autonomously generates topology-aware collective algorithms when the network topology is provided. Such a synthesizer should be:

- Scalable to hundreds to thousands of NPUs
- Support generic—heterogeneous and asymmetric—topologies
- Covers all collective patterns, including All-to-All
- Synthesize collective algorithms tailored for process groups

6.4 PCCL

This section discusses how PCCL synthesizes topology-aware collective algorithms. Specifically, we first define the notion of a collective condition. Then, we describe how a collective algorithm can be synthesized using a BFS pathfinding algorithm, initially targeting homogeneous networks. Finally, we generalize the synthesis process to support switch modeling and heterogeneous networks.

6.4.1 Condition

We first focus on collective patterns that do not require reduction operations (discussed later in Section 6.4.5), such as All-Gather or All-to-All. Chunks in these non-reduction collectives resides in only one NPU in the precondition. In the postcondition, a chunk may reside in either a single NPU (e.g., Scatter or All-to-All) or multiple NPUs (e.g., Broadcast or All-Gather).

We propose defining collective communication patterns using a condition-based representation. Preconditions and postconditions are NPU-centric—describing which chunks each NPU holds before and after collective communication, respectively. Meanwhile, the condition-based view is chunk-centric. Each condition specifies a chunk’s source NPU and

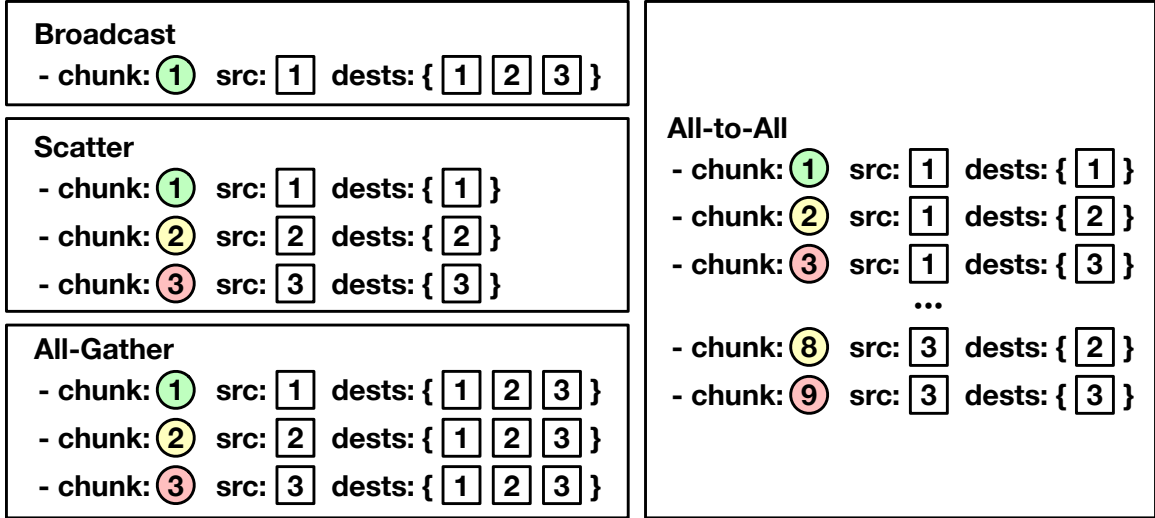


Figure 6.3: Defining collectives patterns in a list of conditions.

its set of destination NPUs. A collective communication pattern consists of one or more collective conditions.

Figure 6.3 shows examples of collective patterns expressed using condition-based notation. Each condition defines a chunk’s source NPU and destination NPUs. For instance, a Scatter contains three conditions, each describing a single source and destination pair for a chunk. All-Gather, on the other hand, also has three conditions, but each chunk is destined for multiple NPUs instead of just one.

6.4.2 TEN Representation

PCCL leverages the TEN representation to synthesize collective algorithms, which is already introduced in Chapter 5. In this section, we formally define the TEN structure and introduce three essential operations for processing a given TEN. These operations are illustrated in Algorithm 3. TEN is a three-dimensional boolean matrix: $TEN[t][s][d]$. A value of `true` at $TEN[t][s][d]$ indicates that there is a link from NPU s to NPU d at timestep t , meaning that NPU s can initiate a chunk transfer to d at that time. Given this representation, we define the following utility procedures:

- `NEXTDEVICES(TEN, npu, time)`: Returns the set of destination NPUs to which

Algorithm 3 TEN Functionality

Require: $TEN[ht][s][d]=\text{true}$: TEN has a link $s \rightarrow d$ at t

```
1: function NEXTDEVICES( $TEN, npu, time$ )
2:   return  $\{next: TEN[time][npu][next]=\text{true}\}$ 
3:
4: function AVAILABLE( $TEN, npu, time$ )
5:   if  $|\text{NEIGHBORS}(TEN, npu, time)| > 0$  then
6:     return true
7:   else
8:     return false
9:
10: function NEXTAVAILABLETIME( $TEN, npu, time$ )
11:    $t \leftarrow time$ 
12:   while not AVAILABLE( $TEN, npu, t$ ) do
13:      $t \leftarrow t + 1$ 
14:   return  $t$ 
```

npu can send a chunk at timestep $time$ (i.e., all d such that $TEN[time][npu][d]$ is true).

- AVAILABLE($TEN, npu, time$): Returns a boolean indicating whether npu is available to send a chunk at timestep $time$ (i.e., whether there exists at least one valid destination NPU at $time$).
- NEXTAVAILABLETIME($TEN, npu, time$): Returns the earliest time $t \geq time$ at which npu becomes available to initiate a chunk transfer. For example, NEXTAVAILABLETIME($TEN, npu, 0$) returns the first timestep at which npu is capable of sending out a chunk.

$npus$ in these TEN functionalities are later generalized as *devices* to accommodate network switches. This is explained in Section 6.4.7.

6.4.3 BFS Pathfinding Algorithm

With the TEN representation as a foundation, PCCL determines the paths to satisfy the condition. The objective of the BFS pathfinding algorithm is to determine the exact route

Algorithm 4 BFS Pathfinding Algorithm

Require: Network TEN , Condition c

Ensure: Synthesized path $paths$ for c

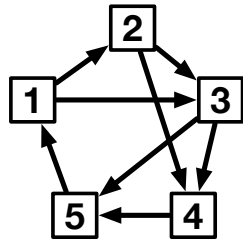
```
1:  $t \leftarrow \text{NEXTAVAILABLETIME}(TEN, c.src, 0)$ 
2:  $visited \leftarrow \{(c.src)\}$ 
3:  $paths \leftarrow \{c.src : []\}$ 
4: while  $c.dests \not\subseteq visited$  do
5:   for  $current$  in  $visited$  do
6:     for  $next$  in  $\text{NEXTDEVICES}(TEN, current, t)$  do
7:       if  $next \in visited$  then
8:         continue
9:       Add  $next$  to  $visited$ 
10:      Add  $(next : [paths[current], (t, next)])$  to  $paths$ 
11:       $t \leftarrow t + 1$ 
12: return  $paths$ 
```

for a chunk to travel from the source $c.src$ to all destinations in $c.dests$, where c is a collective condition.

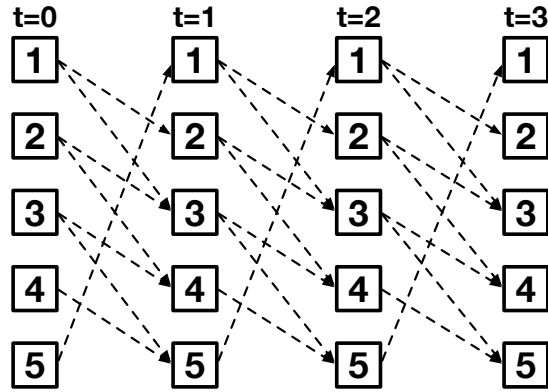
PCCL represents a path as an ordered list of tuples (t, n) , indicating that the chunk is sent from its current location to NPU n at timestep t . For example, consider $path = [(1, 2), (4, 3)]$ for a condition whose $c.src = 6$. This path indicates that the chunk is first transferred to NPU 2 from NPU 6 at $t = 1$, and then from NPU 2 to NPU 3 at $t = 4$.

PCCL pathfinding algorithm aims to construct a dictionary of such paths, $paths = \{dest : path\}$, where each entry maps the path of the chunk to reach $dest$ from $c.src$. The pseudocode of the procedure is presented in Algorithm 4, where a condition c is given and the goal is to find a path to each destination in $c.dests$. The algorithm initializes (i) t : the first available timestep for NPU $c.src$, and (ii) $visited$: the set of visited NPUs. It then performs a BFS search over the TEN to expand the $visited$ set until all $c.dests$ have been reached. Each time a new NPU $next$ is visited, it is added to the $paths$ dictionary by appending the edge from its predecessor npu , building upon the path already found for npu .

Figure 6.4 visually illustrates this process. Figure 6.4(a) shows an example target topology of an asymmetric 5-NPU network. Figure 6.4(b) displays the TEN representation ex-



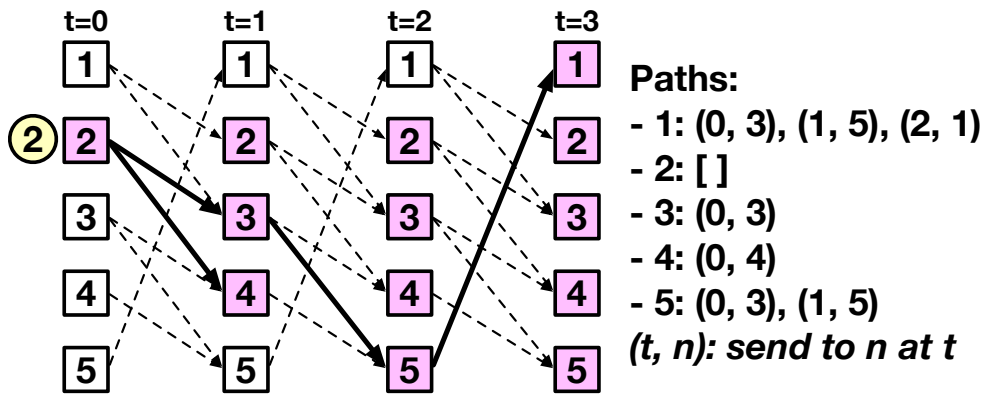
(a) Target 5-NPU topology with 8 unidirectional links



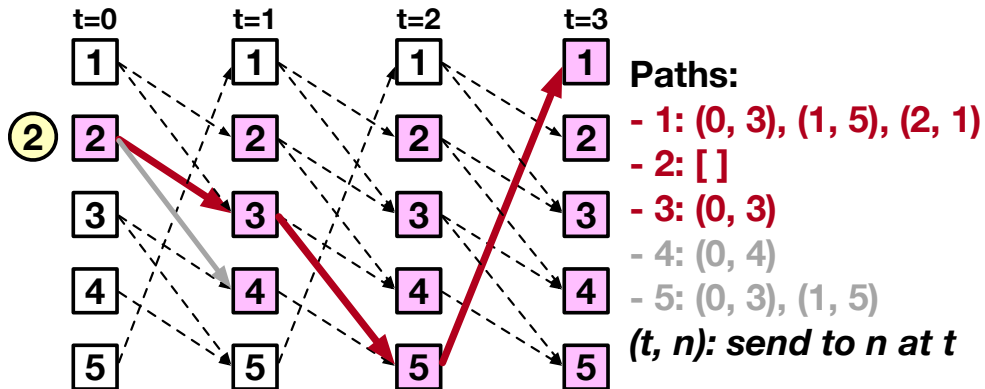
(b) TEN representation of (a)

- chunk: ② src: ② dests: { ① ② ③ }

(c) Target collective condition to run BFS pathfinding algorithm



(d) BFS pathfinding result after finding all dests of (c)



(e) Synthesized path of chunk 2, from NPU 2 to NPUs {1, 2, 3}.

Figure 6.4: BFS search algorithm to find the path of a chunk.

panded up to timestep 3. The target condition in Figure 6.4(c) specifies a target condition to find the route, that a chunk starts at NPU 2 and must reach NPUs $\{1, 2, 3\}$. Figure 6.4(d) shows the BFS traversal process: starting from source NPU 2, traversing the *TEN* until all destinations in *c.dests* are visited. Glimpsed in Figure 6.4(d) is how the BFS pathfinding algorithm may end up visiting more NPUs than the requested destinations of the condition. For example, the BFS result in Figure 6.4(d) visited all five NPUs, although the condition only requires visiting NPUs $\{1, 2, 3\}$. Consequently, not all paths constructed during the pathfinding process are meaningful, and only the useful paths should be retained to finalize the chunk’s actual path. Simply, the process iterates over the actual destinations of the condition *c.dests*, and selects the paths associated with them. Figure 6.4(e) shows the filtering result, illustrating the final chosen path of chunk 2 *TEN*[0][2][4] was filtered out, since such communication is meaningless given that 4 is not in the destination set of the condition. However, note that the chunk transfer to NPU 5 remains in the path—albeit not in the destination set. It is because NPU 5 acts as an intermediate node to forward the chunk to NPU 1, one of the destinations. This explains how the BFS pathfinding algorithm captures the process group information: it first tries to construct paths utilizing the entire network, then filters out only the meaningful paths to *c.dests*.

6.4.4 Synthesizing Collective Algorithm

Figure 6.4 visualized the synthesis process of a Broadcast algorithm for a chunk from NPU 2 to NPUs $\{1, 2, 3\}$. However, as shown in Figure 6.3, a collective pattern may consist of multiple conditions, unlike a simple Broadcast. Synthesizing collective algorithms for such patterns can be achieved by repeatedly applying the BFS pathfinding algorithm. Note that two chunks occupying the same *TEN* link lead to network congestion, as it indicates a conflict where multiple chunks attempt to use the same physical link at the same time. In other words, a specific *TEN* link can only be occupied by a single chunk. Therefore, to avoid link congestion in the resulting collective algorithm, any *TEN* links chosen during a

Algorithm 5 Synthesizing Collective Algorithm

Require: Network TEN , Conditions Set C

Ensure: Synthesized Collective Algorithm A

```
1: for condition  $c$  in  $C$  do
2:    $c.dist \leftarrow 0$ 
3:   for  $dest$  in  $c.dests$  do
4:      $c.dist \leftarrow \max(c.dist, \text{ShortestPath}(c.src, dest))$ 
5: Sort  $C$  in descending order by  $c.dist$ 
6:
7:  $A \leftarrow \{\}$ 
8: for  $c$  in  $C$  do
9:    $p \leftarrow \text{BFS}(TEN, c)$ 
10:  Add  $p$  to  $A$ 
11:  Remove  $p$  from  $TEN$ 
12: return  $A$ 
```

previous BFS pathfinding step are removed from subsequent BFS searches.

The pseudocode for this process is shown in Algorithm 5. Since multiple chunks must be mapped over the TEN, PCCL must first determine which chunk should have its path synthesized first. To assign the order, PCCL first computes a distance $dist$ to each condition c , defined as the maximum shortest-path distance between $c.src$ and $c.dests$. Then, it sorts the set of conditions C in descending order of $c.dist$. This strategy aims to maximize network resource utilization by assigning paths to chunks that must traverse the network for the longest duration first. Chunks that traverse shorter distances can then utilize the remaining unoccupied TEN links, thereby heuristically maximizing bandwidth utilization, as motivated in [132].

Figure 6.5 showcases an example by extending Figure 6.4 to the All-Gather collective among a process group $\{1, 2, 3\}$, based on the topology shown in Figure 6.4 Figure 6.5(a) shows the synthesized path for chunk 2, as presented in Figure 6.4(e). Since these links are occupied by chunk 2, they are removed from the TEN before conducting the next BFS pathfinding step, as illustrated in Figure 6.5(b). Subsequently, the BFS algorithm is run for chunk 3 in Figure 6.5(c), and the resulting links are also removed from the TEN, as depicted in Figure 6.5(d). Note that the number of available TEN links decreases as PCCL

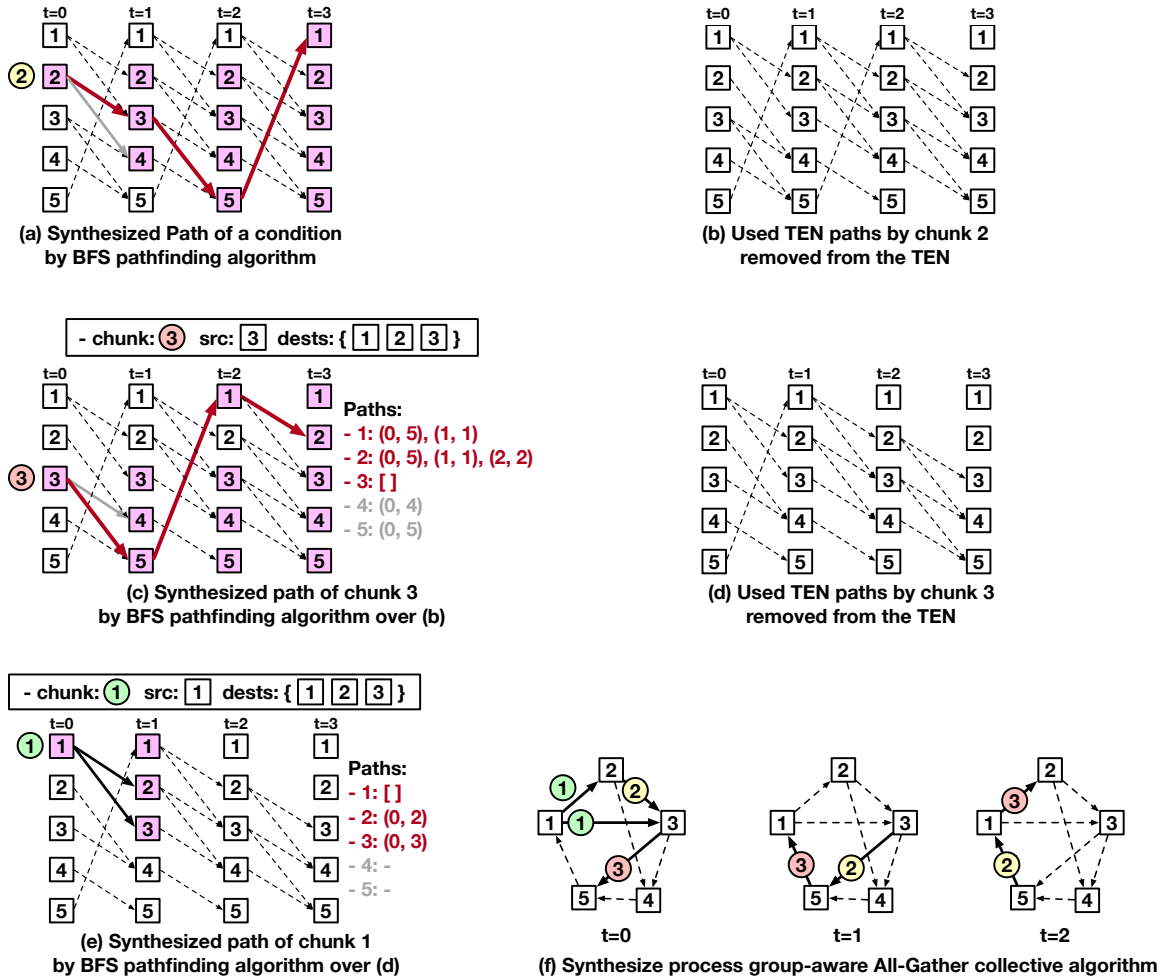


Figure 6.5: Synthesizing a All-Gather collective algorithm for a process group.

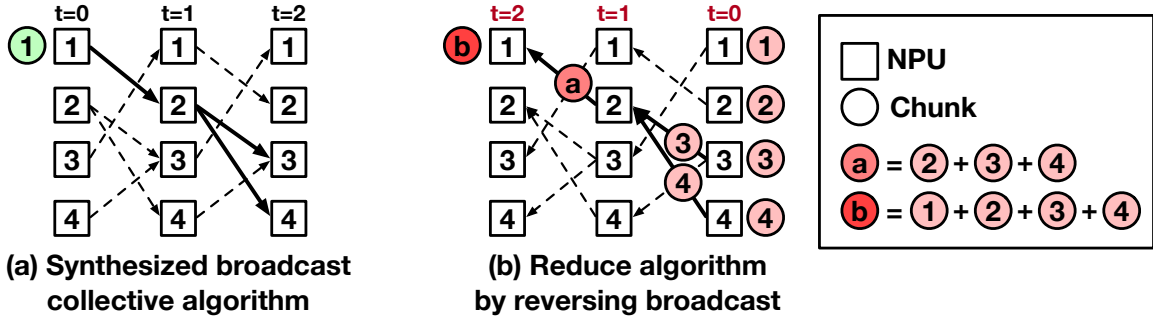


Figure 6.6: Example synthesis of a Reduce operation.

schedules more conditions across the network. Figure 6.5(e) shows the path for chunk 1, obtained through an additional BFS pathfinding phase. Finally, Figure 6.5(f) summarizes the synthesized topology-aware All-Gather algorithm, constructed through this iterative BFS pathfinding process.

Note that PCCL successfully synthesizes a process group-aware All-Gather algorithm. While Figure 6.5(f) illustrates the All-Gather operation among NPUs $\{1, 2, 3\}$, the synthesized algorithm flexibly utilizes network links outside this set—for example, the links $3 \rightarrow 5$ and $5 \rightarrow 1$ are used even though NPU 5 is not part of the source or destination NPUs. Furthermore, as PCCL assumes no specific characteristics of a condition, it is inherently generic and can be applied to any collective pattern, including All-to-All.

6.4.5 Reduction Operations

As already discussed in Section 5.4.6, collective patterns involving reductions can be supported by synthesizing their corresponding non-reduction collective algorithms. This follows the paradigm introduced in [39, 132, 133].

Figure 6.6(a) exemplifies synthesizing Broadcast collective algorithm for a 4-NPU cluster. To synthesize a Reduce algorithm, PCCL first generates the corresponding Broadcast algorithm, as shown in Figure 6.6(a). By reversing the direction of all transfers and applying reduction operations, the Reduce algorithm can be automatically constructed, as depicted in Figure 6.6(b). Similarly, Reduce-Scatter can be synthesized by reversing

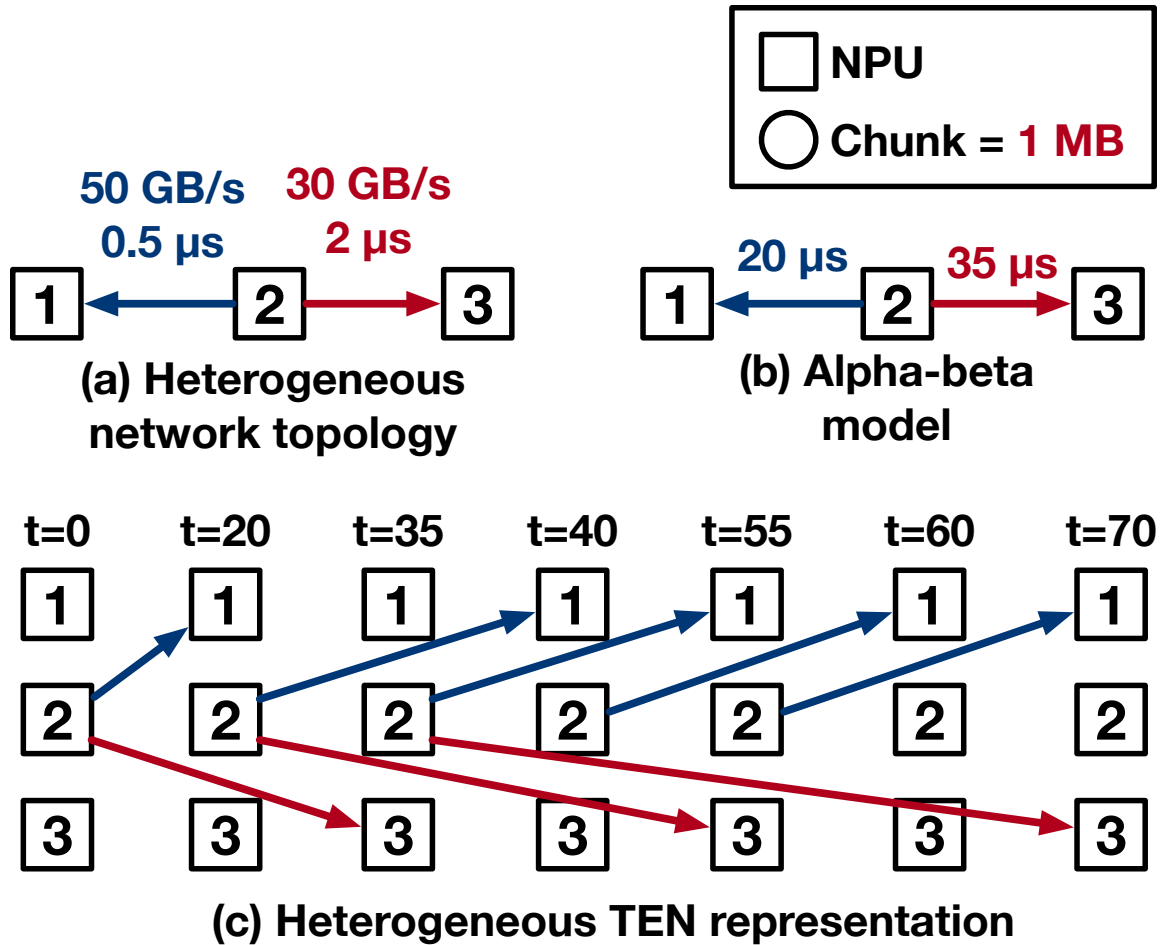


Figure 6.7: TEN representation of a heterogeneous network topology.

All-Gather, and All-Reduce is realized by composing Reduce-Scatter followed by All-Gather [132].

6.4.6 Heterogeneous Networks

PCCL leverages the α - β network model [76] to support heterogeneous networks, as suggested in [132, 39], and explained in Section 5.4.7. The α - β model estimates the transfer time of a link as $\alpha + (m \times \beta)$, where α represents the link latency, β is the reciprocal of the link bandwidth, and m is the message size (i.e., chunk size for the synthesizers).

Figure 6.7(a) depicts a heterogeneous network topology with two distinct links of different bandwidths and latencies. For such a network, the α - β model is used to capture the

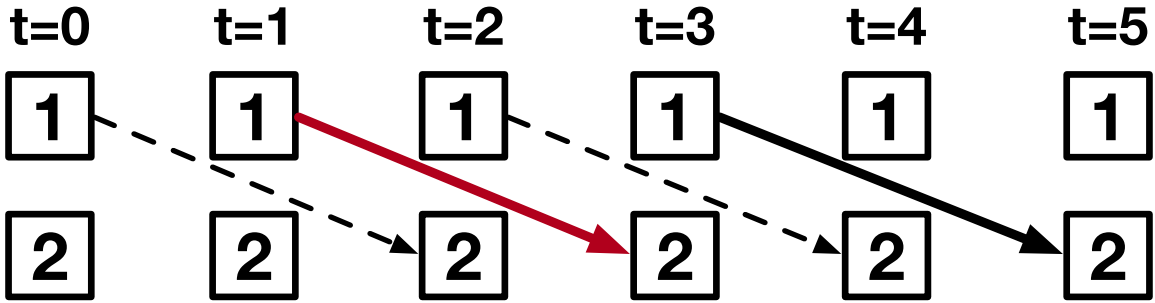


Figure 6.8: Removing TEN links in a heterogeneous network.

transfer time of each link as a single value, as illustrated in Figure 6.7(b). Specifically, a chunk size of 1 MiB is used in this application of the α - β model. Figure 6.7(c) shows the TEN representation corresponding to Figure 6.7(b), where the timesteps in μs are from the α - β model. Note that the timesteps reflect the timing information from the α - β model.

For heterogeneous TEN representations, PCCL’s BFS pathfinding is applicable with only marginal modifications. Such modifications mostly arise from the timing considerations when a chunk arrives at the next NPU, and which TEN links to disable to avoid network congestion.

Since each link can have a different transfer time, PCCL must track the exact arrival time of each chunk at every device. As a result, the *visited* set is extended to include the *time* information representing the timestep at which the chunk reaches each device. During the BFS process, whenever a *current* device is processed, it must be ensured that the chunk has actually arrived at this source.

Additional care must also be taken when removing TEN links during subsequent BFS pathfinding passes. When a TEN link is used, not only that specific link, but also all other links overlapping with its timestep must be removed to avoid network congestion. This scenario is illustrated in Figure 6.8. If a TEN link from $t = 1$ to $t = 3$ is taken, other TEN links overlapping with this timestep (e.g., $TEN[0][1][2]$ and $TEN[2][1][2]$) must be disabled to prevent network congestion. If a chunk is sent at $t = 1$ from NPU 1 to 2, $TEN[0][1][2]$ and $TEN[2][1][2]$ are also disabled to avoid network congestion.

6.4.7 Modeling Switches

Switch modeling remains an open question in collective synthesizers today. Most past works, including TACOS in Chapter 5, they unroll a switch into direct-connect links[39, 132, 143]. Unfortunately, this limits modeling switch-specific considerations (e.g., finite buffers, or unicast versus multicast support). In PCCL, while we inherently support unrolling, we also add explicit support to model switches via two classes of TEN nodes: NPUs and switches. We track the number of chunks buffered at each switch node, and the BFS pathfinding algorithm skips visiting a TEN node at a given timestep if the switch node exceeds the provided buffer size. Furthermore, if the switch node does not support multicast, the BFS algorithm visits only one next neighbor when the node type is a switch. Even under this restriction, as the BFS algorithm visits other nodes in subsequent timesteps (i.e., rather than visiting all next neighbors at once, it visits next nodes one by one), the algorithm can still successfully synthesize a collective algorithm.

6.4.8 Translating Synthesis Results

As already discussed in Section 5.4.9, it is important to note that PCCL does not specifically target GPU-centric networking, but rather contributes to the algorithmic foundations for synthesizing arbitrary collective algorithms at scale. Nevertheless, we illustrate how PCCL synthesis results can be translated into other representations to execute on a specific target system by targeting a GPU-based system as an example. For GPU-based systems, we propose to use MSCCLang [40] and MSCCL++ [41] representations. Both frameworks enable the usage of scratch buffers and multiple threadblocks per GPU for concurrent communications. For further optimization, MSCCL++ even allows a single threadblock to concurrently put messages to multiple peers. Meanwhile, MSCCLang not only provides `send` and `receive` operations but also compound operations such as `receive-copy-send`, all suitable to represent PCCL chunk operations. Therefore, by leveraging MSCCLang and MSCCL++, PCCL synthesis results can be readily represented and executed on GPU sys-

tems without any modifications to the synthesizer itself.

6.5 Methodology

6.5.1 Experimental Infrastructure

We use ASTRA-sim2.0 from Chapter 3, a distributed ML systems simulator for the evaluation of this chapter [27, 156], similar to the setup of [39]. ASTRA-sim2.0 simulation has been validated over a real system comparison using a 128 NVIDIA H100 cluster with the accuracy of 97% [115].

6.5.2 Baseline Collectives

We mostly evaluate PCCL by targeting the All-to-All collective pattern, as (i) most synthesizers fail to synthesize such an algorithm, especially at scale, and (ii) no collective algorithm exists in CCLs. PCCL mechanism is still applicable to generic collective patterns and to showcase that, we also have All-to-Allv and All-Gather results as well. We use direct, a pairwise point-to-point send-receive algorithm, as the baseline collective algorithm to compare All-to-All performance, as such a mechanism is what CCLs use today.

6.6 Evaluations

In this section, we evaluate the performance and efficacy of the PCCL synthesizer. Specifically, we emphasize checking the four practical objectives for collective synthesizers discussed in Section 6.3: scalability, generic topology support, generic collective support, and process group awareness.

6.6.1 Scalability

Firstly, we evaluate the scalability of the PCCL synthesizer. For the scalability analysis, we evaluated All-to-All collective pattern as it has the largest search space and thereby

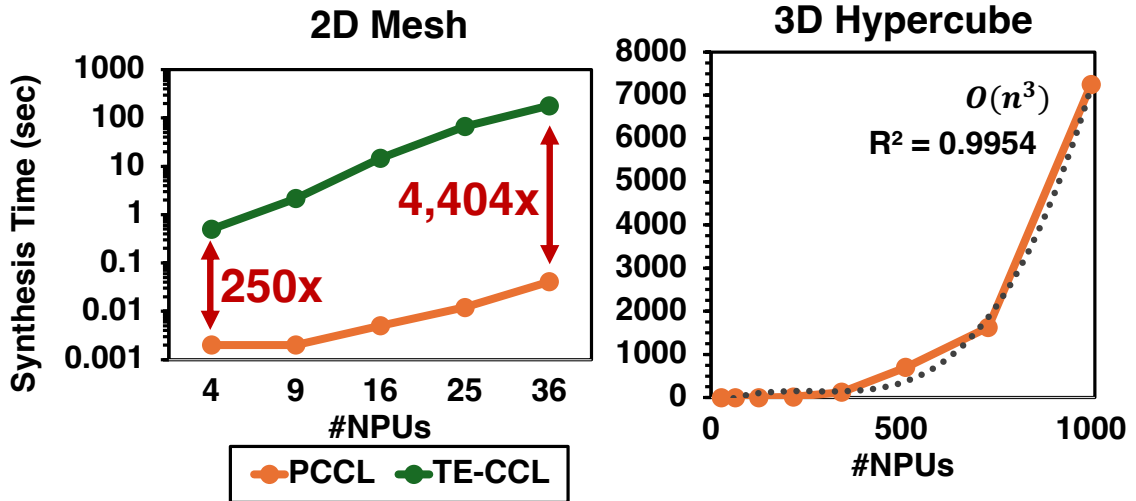


Figure 6.9: All-to-All synthesis time of PCCL for small two-dimensional mesh.

most synthesizers struggle to synthesize. Therefore, we mainly target TE-CCL [144], the state-of-the-art in All-to-All synthesis, for main scalability comparison.

Topology Size

We measured the All-to-All synthesis time of PCCL by increasing the size of two-dimensional mesh and three-dimensional hypercube target topologies. Figure 6.9 summarizes the observed synthesis time, compared with TE-CCL synthesis times. For a small 6×6 (36 NPU) topology, PCCL is already $4,404 \times$ faster over TE-CCL. Further topology scalability analysis of PCCL is also shown using three-dimensional hypercube.

Notably, only at 36-NPU cluster scale, PCCL was already more than 3 orders of magnitude faster than the state-of-the-art TE-CCL. Further scaling the target topology shows that PCCL can synthesize an All-to-All algorithm for a 512-NPU cluster in just 11.68 minutes, and 1,000-NPU cluster in 2.01 hours. The complexity to synthesize All-to-All algorithm was $O(n^3)$.

Specifically, we measured TE-CCL taking 3 minutes for a 36-NPU (6×6 two-dimensional mesh) target and more than 30 minutes for 49 NPUs. Although TE-CCL was able to synthesize a 256-GPU target in 25 minutes [144], Cao et al. [164] report that TE-CCL takes

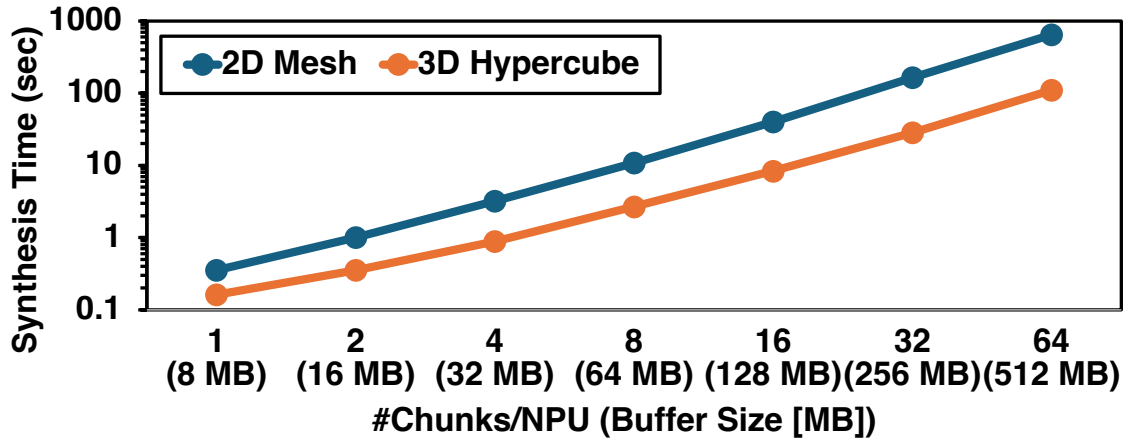


Figure 6.10: PCCL synthesis time of All-to-All algorithm of size 8–512 MiB.

4.4–49.5 minutes and 3.8 minutes–8.7 hours for 16- and 32-GPU systems, respectively. This indicates that TE-CCL synthesis time heavily depends on the synthesizer setup and hyperparameters, including the search policy, number of chunks per collective, and target network topologies. Given this large variability, we expect PCCL to demonstrate consistently better scalability than optimizer-based synthesizers.

Collective Size

We also measured the scalability of the target collective pattern, especially the sensitivity to the buffer size. Specifically, we synthesized All-to-All algorithms for an 8×8 two-dimensional mesh. The algorithm buffer size spans 8–512 MiB by fixing each chunk size to 128 KiB and increasing the number of chunks per NPU from 1 to 64.

The synthesis time is summarized in Figure 6.10. The target is 64-NPU two-dimensional mesh and three-dimensional hypercube topologies. Collective buffer size was adjusted by increasing the number of 128 KiB chunks per each NPU.

Notably, PCCL synthesized a 512 MiB All-to-All algorithm for a three-dimensional hypercube topology in 1.83 minutes. As this experiment is done by setting each chunk size to 128 KiB, the synthesis time can further be decreased by increasing the chunk size and reducing the number of chunks per NPU.

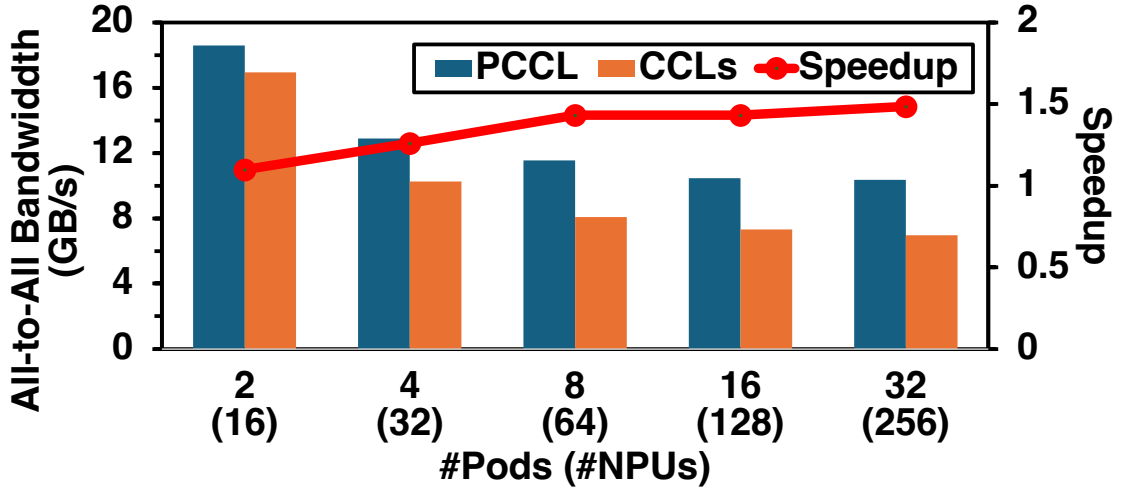


Figure 6.11: All-to-All bandwidth of PCCL versus CCLs and collective speedup over heterogeneous 2D switch topology.

6.6.2 Supporting Generic Topology

In Section 6.6.1, we already evaluated two homogeneous, asymmetric topologies: two-dimensional mesh and three-dimensional hypercube. In this section, we show the applicability of PCCL towards more classes of topologies: notably, 2D switch topology, which is heterogeneous.

Figure 6.11 showcases the same experiment by using a heterogeneous 2D switch topology: each node size is 8, and the cluster size spans 16–256 NPUs by increasing the number of nodes in the cluster. For this setup, PCCL showed consistent speedup over the baseline CCLs, $1.33\times$ on average.

6.6.3 Supporting Generic Collective

In this section, we explain the applicability of PCCL for generic collective patterns. First, we evaluate the performance of All-to-All pattern, with the largest search space, in more detail. Figure 6.12 compares the normalized All-to-All bandwidth of PCCL, CCLs, and state-of-the-art TE-CCL synthesizer. For small-sized Meshes, PCCL and TE-CCL showed comparable performance. However, TE-CCL synthesis breaks after 5×5 two-dimensional

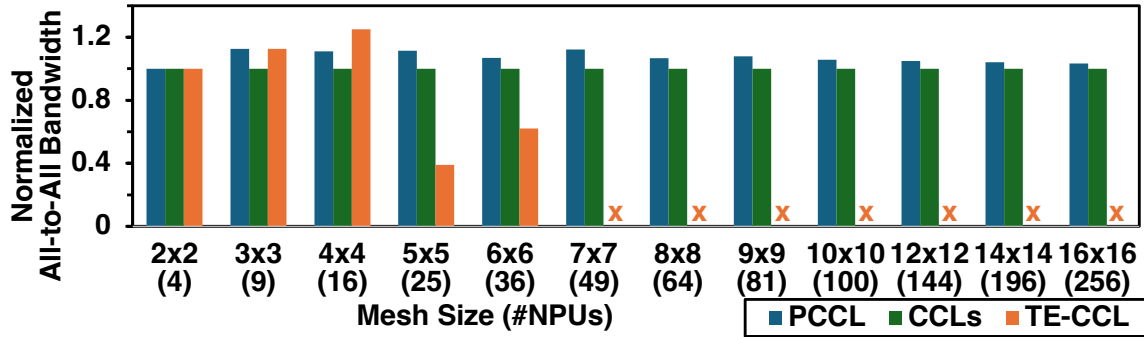


Figure 6.12: Normalized All-to-All bandwidth when the entire two-dimensional mesh cluster is executing a All-to-All collective.

mesh since it is set to search for the very first satisfiable All-to-All algorithm due to scalability considerations, which, even after this setup, is 3 orders of magnitude slower than PCCL (explained in Section 6.6.1). After 7×7 two-dimensional mesh, TE-CCL synthesis takes more than 30 minutes (explained in Section 6.6.1), and PCCL continuously show better performance than the baseline CCL algorithm.

6.6.4 Synthesis with Process Group

We then showcase the capabilities of PCCL with a simple synthesis example. Over a $3 \times$ two-dimensional mesh, we overlaid two process groups of size three each: group 1 running All-to-Allv (NPUs 0–2, with NPU 0 transmitting twice as much data as NPUs 1–2), and group 2 executing All-Gather (NPUs 6–8), with 2 chunks per collective. The synthesis result is depicted in Figure 6.13. PCCL generated a congestion-free collective algorithm over an asymmetric two-dimensional mesh topology. PCCL also supports arbitrary collectives, provided that pre-/postconditions are specified, as illustrated in this example with All-to-Allv. Finally, note the process group awareness of PCCL: (i) multiple process groups running independent collectives are naturally supported, and (ii) NPUs 3–5 and their associated links, even though not part of any process group, are actively leveraged by the PCCL-generated collective algorithm to maximize performance and resource utilization.

We also evaluate the benefits of process group-aware PCCL through All-to-All collec-

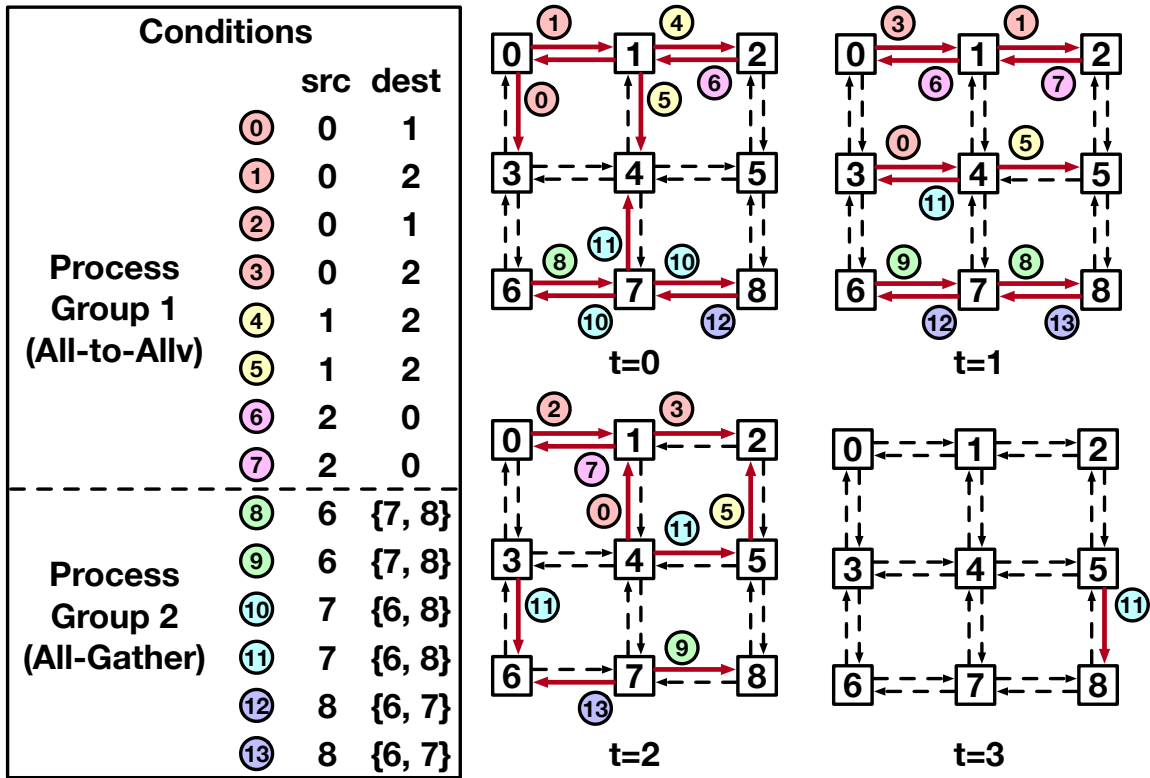


Figure 6.13: PCCL-synthesized collective algorithm of two process groups.

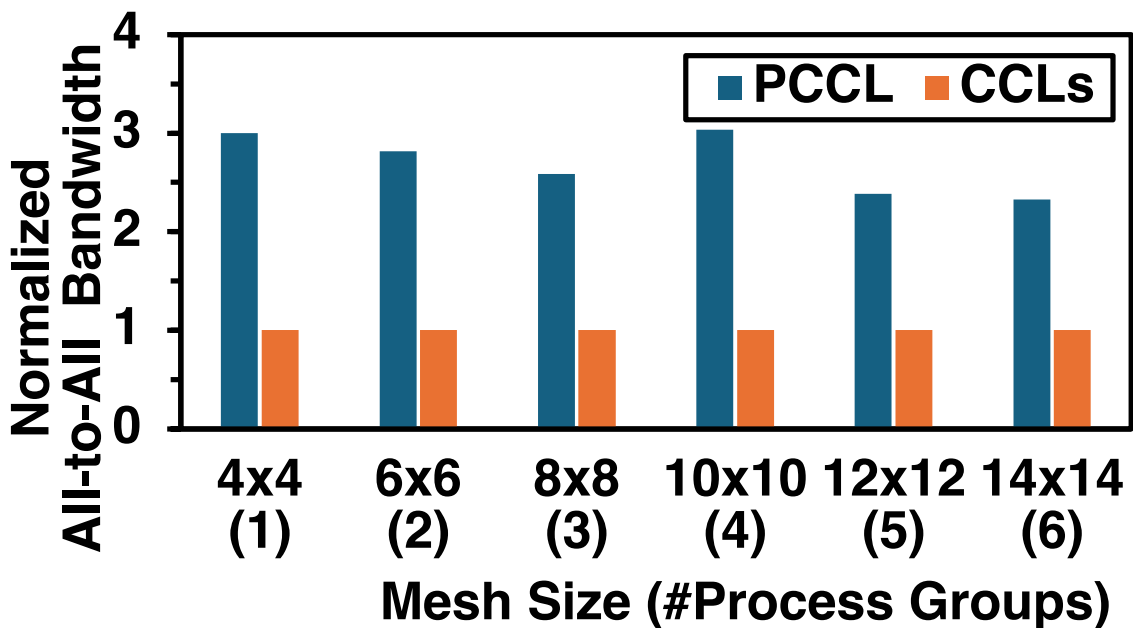


Figure 6.14: Normalized All-to-All bandwidth of PCCL-synthesized algorithm over the baseline direct, as the two-dimensional mesh size and the number of process groups gradually increase.

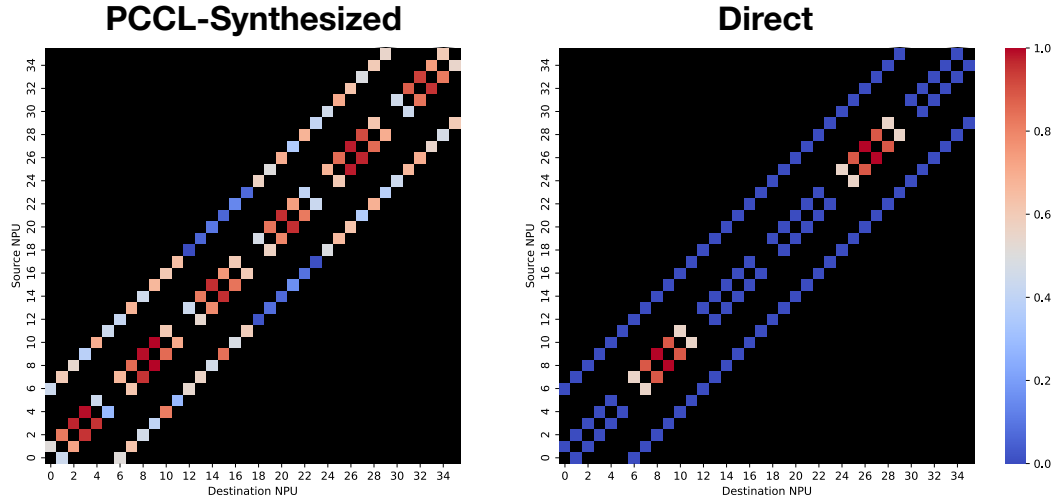


Figure 6.15: Normalized link utilization heat map of PCCL-synthesized versus direct collective algorithms.

tive. A large ML cluster with many NPUs often follows this scheme, where multiple groups of collectives are concurrently run, rather than the entire cluster executing a single collective communication. We measured the All-to-All bandwidth of pairwise direct algorithm (namely CCLs) as well as PCCL-generated algorithms. In doing so, we increased both the size of the target two-dimensional mesh topology as well as the concurrent numbers of process groups (we set the process group size equal to the two-dimensional mesh width in this experiment). The normalized algorithmic bandwidth is summarized in Figure 6.14. Notably, PCCL-synthesized algorithm showed $2.33\text{--}3.03\times$ speedup over the baseline direct ($2.68\times$ average).

This can be explained by the link utilization heat map captured in Figure 6.15, when two process groups are executing All-to-All amongst them. Unlike PCCL-synthesized All-to-All algorithm, direct fails to leverage the entire network outside the process group, resulting in $2.8\times$ speedup. When there are 2 process groups running All-to-All within themselves, PCCL still leverage the entire network resources to maximize the performance of both All-to-All executions. However, the traffic pattern generated by the direct algorithm only utilizes localized network resources, resulting in huge network underutilization. The same is applicable to all other previous collective algorithm synthesizers, as none of the synthe-

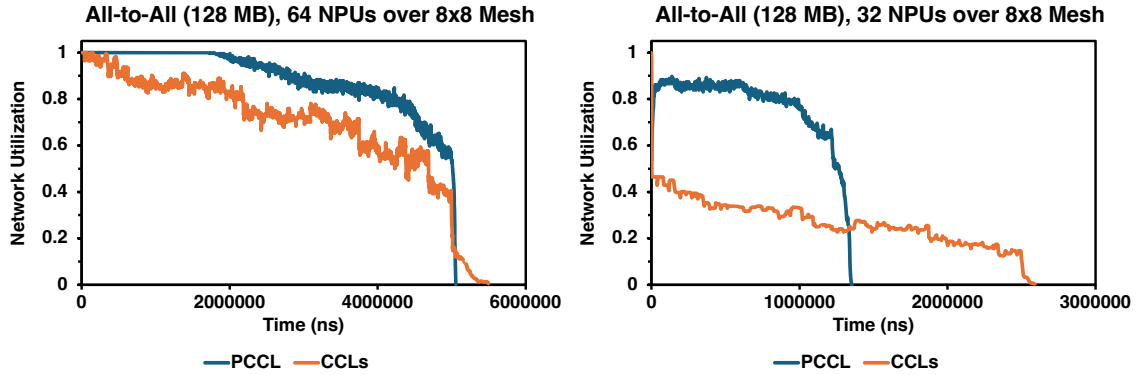


Figure 6.16: Network bandwidth utilization of PCCL-synthesized and direct algorithms over time.

sizers consider process group and only generate localized collective algorithms.

This is further explained in Figure 6.16, which depicts the network resource utilization over time. The values were measured by running 128 MiB All-to-All collective over a 8×8 two-dimensional mesh, with processing group of size 64 and 32, respectively. Even when running All-to-All over an entire topology (64 NPUs over a 8×8 two-dimensional mesh), PCCL still outperformed in terms of network resource utilization over the baseline direct algorithm, showcasing shorter collective time. However, even when the process group is smaller than the topology, unlike the baseline which underutilizes the network, PCCL still maximizes resource utilization, finishing the collective $1.88 \times$ faster.

The same trend is also exemplified by a sensitivity analysis shown in Figure 6.17. Here, we fixed the target topology to 8×8 two-dimensional mesh, and varied the number of concurrent 128 MiB All-to-All process groups, each of size 8. Since PCCL can leverage a lot of free network resources, when there was only one process group executing the collective, PCCL showed $3.05 \times$ speedup. As more network resources start to conflict across process groups, the benefit of PCCL-synthesized algorithm decreases as the concurrent process group increases.

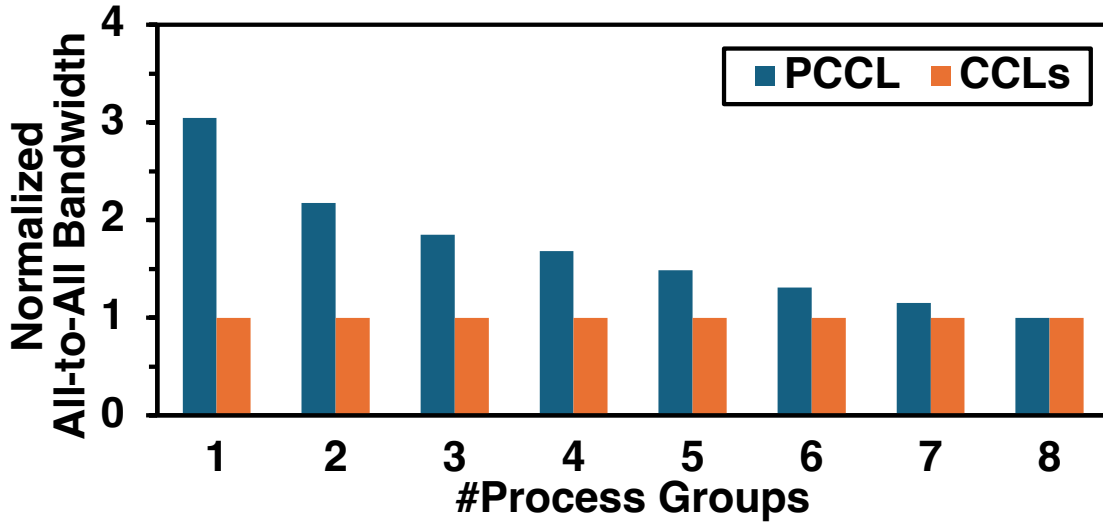


Figure 6.17: Normalized All-to-All bandwidth over CCLs, when the number of process groups increase.

6.7 Related Work

6.7.1 Collective Algorithm Synthesizers

Although we explained the specifics of the topology-aware collective algorithm synthesizers in Section 5.7, we reiterate them with the focus on comparison against PCCL. Existing synthesizers can largely be categorized into three groups based on their primary synthesis approach.

Tree-based

A spanning tree provides a routing structure for data chunks to be reduced and broadcast to and from the root [42]. Therefore, tree-based synthesizers try to generate spanning trees over the target network topology.

- Blink [109] constructs multiple physically disjoint spanning trees, allowing each tree to process a portion of the buffer in parallel. It uses multiplicative weight updates (MWU) and linear programming (LP) to pack as many disjoint trees as possible. Because it relies on LP, Blink offers moderate scalability—LP is polynomial

in theory, but the solver is superlinear in practice [174]. Additionally, Blink is restricted to direct-connect topologies (i.e., topologies without switches) and is limited to All-Reduce collectives.

- MultiTree [138] extends tree-based synthesis to support All-Gather and Reduce-Scatter by generating a spanning tree rooted at each NPU. It uses a greedy-based tree construction approach, which offers high scalability. However, MultiTree is limited to homogeneous network topologies.
- ForestColl [143] builds on the idea of generating spanning trees from all NPUs but introduces support for switch-based topologies, by eliminating switches through an LP-based transformation that substitutes switches with multiple direct-connect links. While this enables limited switch modeling, the reliance on LP constrains scalability. ForestColl is limited to symmetric topologies.

Optimizer-based

Spanning tree-based synthesis lacks generality for diverse collective patterns and network topologies, albeit a moderate benefit in scalability. Optimizer-based synthesizers use global optimization techniques to assign paths for each chunk, thereby supporting generic collectives including All-to-All. However, this generality comes at the cost of scalability, as the underlying optimization has NP time complexity [139].

- SCCL [133] models collectives using satisfiability modulo theories (SMT). While it supports all collective patterns, the SMT solver struggles with scalability as it is an NP-hard problem [175]. Moreover, SCCL only supports homogeneous and symmetric topologies and does not model switches.
- TACCL [132] improves upon SCCL by using integer ILP, modeling heterogeneous networks. Like ForestColl, it models switches by substituting them with multiple direct-connect links. TACCL suffers from limited scalability: ILP is NP-hard [139].

For example, TACCL failed to synthesize an All-to-All algorithm for just 16 NPUs within a 30-minute time limit.

- TE-CCL [144] employs a network multi-commodity flow formulation. This approach enables native modeling of switch devices, rather than reducing them to direct-connect edges. Additionally, TE-CCL transforms the All-to-All synthesis problem into LP, achieving practically moderate scalability for this specific pattern. However, for other collectives, including All-Reduce, TE-CCL still relies on solving ILPs, limiting its scalability. For example, TE-CCL managed to synthesize a 128-NPU All-to-All in 43 minutes, but required over 350 GB of memory.

Notably, no prior synthesizers leverage process group information; they all assume collective communication involves the entire cluster. Also, they model physical network switches by substituting them into logical, direct-connect links, blocking topology-aware collective algorithm synthesis. Finally, all synthesizers had to make several trade-offs between scalability and modeling generality.

6.7.2 Customized Collective Algorithms

Outside of collective synthesizers, several works manually design customized collective algorithms tailored to specific target topologies. BlueConnect [105] and Themis [19] propose collective algorithm optimizations tailored for symmetric multi-dimensional network topologies. three tree overlap [134] and PAARD [150] are specialized All-Reduce algorithms designed for two-dimensional mesh and Dragonfly [75] topologies, respectively. In parallel, MSCCLang [40] introduces a domain-specific language for manually defining customized collective algorithms. These approaches require engineering efforts and domain expertise, and they lack generality across diverse or evolving topologies. In contrast, synthesizers like PCCL can autonomously generate collective algorithms optimized for these target topologies.

6.8 Conclusion

Distributed machine learning has become increasingly important due to the massive scale of large-scale generative models. Both model parameters and data are distributed across many compute devices, which requires frequent collective communications to synchronize activations and parameter updates. Such collective communications have become a major bottleneck. While the performance of the collective algorithm depends on the physical network topology, the baseline collective algorithms in collective communication libraries are largely topology-agnostic. Therefore, designing and executing topology-aware collective algorithms is pivotal to optimizing collective communication, which is a major bottleneck in distributed ML. However, manual design of collective algorithms require domain expertise and costs engineering and validation efforts. Collective algorithm synthesizers address this inefficiency by automatically generating topology-aware collective algorithms. However, prior works have largely overlooked that collective communication typically occurs only among a subset of devices, known as process groups. Additionally, most existing synthesizers are limited in the range of target collective patterns they can generate. In this chapter, we propose PCCL, a scalable and generic framework for synthesizing topology-aware collective algorithms. PCCL is process group-aware and capable of generating near-optimal collective algorithms even when only a subset of devices participates in collective operations. PCCL leverages a BFS pathfinding algorithm over a TEN, resulting in a scalable, generic, and process group-aware synthesizer. PCCL synthesizes arbitrary collective patterns, including 512-NPU All-to-All synthesis in 11.68 minutes. PCCL with process group information synthesized on average $2.68\times$ faster All-to-All algorithm for a two-dimensional mesh topology, and it only took 11.68 minutes for a 512-NPU target All-to-All synthesis. PCCL artifact is publicly available at <https://github.com/astra-sim/pccl>.

CHAPTER 7

CONCLUSIONS AND FUTURE WORKS

7.1 Summary of Contributions

This dissertation highlights the need to optimize collective communication, a major bottleneck in today’s large-scale distributed ML platforms. We demonstrate that the design space of collective communication in distributed ML is complex, with tightly intertwined software and hardware facets. We then argue that software–hardware co-optimizations can significantly improve the performance and efficiency of distributed ML platforms.

To this end, this dissertation introduces four solutions for modeling and optimizing collective communication in distributed ML. First, we emphasize the need for end-to-end, full-stack modeling and simulation capabilities as a foundational research enabler to navigate the complex design space. We propose ASTRA-sim2.0, which supports modeling and simulating large-scale distributed ML execution with arbitrary parallelization strategies, multi-dimensional network topologies, and enhanced memory system modeling.

Next, we introduce LIBRA, a hardware-centric optimization framework tailored to specific collective communication algorithms. LIBRA underscores the importance of carefully designing multi-dimensional networks to maximize bandwidth utilization and provides a framework to automatically construct such topologies. By incorporating LIBRA’s design decisions, distributed ML platforms achieve improved performance and performance-per-cost.

We then motivate software-centric optimizations that account for target network topologies, as topology-aware collective algorithms are crucial for maximizing collective communication performance. To address this, we introduce TACOS, which represents collective algorithms using a TEN data structure, and develop a link–chunk matching algorithm to

maximize utilization. TACOS enables scalable synthesis of topology-aware collective algorithms. PCCL extends this approach to support more generic collective patterns, introducing a BFS-based pathfinding algorithm that supports arbitrary collective patterns and further improves performance through process group awareness.

By applying these software and hardware-centric optimizations, this dissertation enhances both the performance and cost efficiency of distributed ML platforms at scale.

In summary, this dissertation provides insights into modeling large-scale distributed ML platforms and demonstrates how such modeling can guide optimization of collective communication via software- and hardware-centric techniques. The design space of collective communication in distributed ML encompasses both software and hardware; consequently, effective solutions require hardware-aware software and software-aware hardware approaches.

7.2 Future Directions

Training massive-scale ML models has always been a challenge, as discussed in this dissertation. On the other hand, once large-scale generative ML models have been trained, their usefulness and adaptability make them among the most pivotal workloads today, as exemplified by multiple LLMs. As the adoption and usage of frontier ML models increase, the community has been shifting its interest from just large-scale ML training to also ML model inference, which poses intrinsically different challenges from model training. For example, the inference stage of LLM chatbots consists of two phases. First, the model must understand the user input sequence, called the prefill phase. Then, based on this comprehension, the model generates the response token by token, namely the decode phase. The token-by-token generation nature of the decode phase significantly reduces the size of collective communication, making it latency-sensitive rather than purely bandwidth-bound. Also, as specific use cases do not require general-purpose models, specialized, small-scale ML models are being employed to better optimize the total cost of ownership (TCO). This

trend further emphasizes the need for clusters that support smaller, latency-sensitive collective communications. Finally, with the user-facing nature of ML inference, inference use cases face additional latency-related objectives beyond simple throughput optimization, such as time-to-first-token (TTFT), inter-token latency (ITL), or end-to-end latency.

As ML workloads become even more popular, NPU hardware is evolving rapidly to meet not only the massive resource requirements but also to manage the TCO of training and serving ML workloads. For example, modern GPUs extensively adopt multi-chip module (MCM)-based architectures to increase the performance of individual sockets while providing higher network bandwidth and reduced communication latency. Unlike single-chip NPUs with symmetric, homogeneous scale-up networks, such advancements in hardware make the network topology intrinsically more asymmetric and heterogeneous. This trend emphasizes the necessity of designing topology-aware collective algorithms and reinforces the importance of collective algorithm synthesizers, as network topologies become increasingly complex and intricate. Furthermore, this trend warrants additional considerations when determining workload parallelization strategies or collective communication scheduling, opening interesting joint co-design opportunities in the distributed ML space.

As discussed above, the community is moving from simple ML model training optimization toward latency-sensitive collective communications, TCO optimization for ML inference, and more sophisticated software and hardware approaches that make the design space more complex. To this end, we highlight some of the future directions motivated by this research below.

7.2.1 Dynamic Collective Algorithm Synthesis

Both TACOS and PCCL assume a static target environment. In general, collective algorithm synthesis takes place before the actual ML workload execution to replace the algorithms implemented by CCLs, in part because collective algorithm synthesis is expensive at runtime. Consequently, synthesizers assume that the target network topology and col-

lective patterns are predetermined and do not change during ML execution, which is often the case when the parallelization strategy is chosen. However, as AI cluster size increases, TCO optimization becomes important, and inference use cases gain interest, dynamic behavior emerges. AI clusters run multiple inference workloads over a large AI cluster, which generates multiple independent process groups at runtime. As both network and compute devices may fail at runtime, especially in large-scale clusters, the target network topology can change during ML execution. As TACOS and PCCL demonstrate the possibility of scalable collective algorithm synthesis, expanding the synthesizers to account for dynamic collective behaviors at runtime is a viable future extension. For example, scalable synthesizers open opportunities for runtime optimization to recover from device failures by avoiding such devices or to optimize for dynamic process groups arising from multi-tenant ML inference.

7.2.2 Collective Algorithm Synthesis for Latency-Sensitive Collectives

In this dissertation, we motivated the need for a topology-aware collective algorithm synthesizer and proposed TACOS and PCCL. As discussed in the associated chapters, TACOS and PCCL approach synthesis by representing the target network topology using the TEN representation and aim to maximize network resource utilization by mapping chunks onto TEN links, either via a utilization-maximizing link–chunk matching algorithm or a greedy BFS-based pathfinding algorithm. While maximizing network resource utilization is critical for bandwidth-bound collectives, this metric alone is insufficient for latency-sensitive collectives. True latency-bound collectives require optimizations along the control path rather than the algorithmic mapping. For collectives with intermediate buffer sizes, however, considering both latency sensitivity and resource utilization becomes important. This is exemplified by modern CCLs, which include LL128, LL, and Simple protocols. The insights from this dissertation can be extended to guide the synthesis of topology-aware collective algorithms for collectives that are both latency- and bandwidth-sensitive.

7.2.3 Network Topology–Collective Algorithm Co-Design

This dissertation proposes two approaches to optimize collective communication performance: hardware-centric optimization for a target collective pattern, as in LIBRA, and software-centric optimization for a target network, as in TACOS and PCCL. The broader design space for co-designing and co-optimizing network hardware and collective algorithms remains largely unexplored. This is especially critical as hardware is evolving and naturally creating more heterogeneous and asymmetric topologies, as discussed above, and puts more optimization opportunities. The contributions of this dissertation can be combined to enable joint co-design of network topology and collective algorithms. For example, LIBRA could be extended to explore different network shapes and configurations, while TACOS or PCCL could synthesize the corresponding collective algorithms. All exposed parameters could then be co-optimized together, for instance, using reinforcement learning techniques commonly employed in neural architecture search.

7.2.4 Full-Stack Optimization of Distributed ML Platforms

The contributions of this dissertation primarily focus on collective communication performance, given its dominant impact at the scale of current distributed ML platforms. However, it also highlights that end-to-end performance depends on the highly intertwined design space of workloads, systems, and networks. For instance, changing a workload’s parallelization strategy can produce significantly different collective communication patterns, while altering the network topology may necessitate adapting parallelization strategies to maximize performance. Similarly, optimizing collective communication algorithms may require corresponding adjustments in network topology. Together, these insights indicate that full-stack optimization of distributed ML platforms remains an open challenge, and the techniques proposed in this dissertation provide a foundation for addressing this complex, large-scale optimization problem.

REFERENCES

- [1] T. B. Brown et al., “Language models are few-shot learners,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS ’20, 2020, ISBN: 9781713829546.
- [2] H. Touvron et al., “Llama: Open and efficient foundation language models,” in *arXiv:2302.13971 [cs.CL]*, 2023.
- [3] N. Shazeer et al., “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” in *arXiv:1701.06538 [cs.LG]*, 2017.
- [4] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, “A survey on mixture of experts in large language models,” 2025, pp. 1–20.
- [5] W. Fedus, B. Zoph, and N. Shazeer, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” *J. Mach. Learn. Res.*, vol. 23, no. 1, Jan. 2022.
- [6] Cerebras, *Cerebras Demonstrates Trillion Parameter Model Training on a Single CS-3 System - Cerebras*, <https://www.cerebras.ai/press-release/cerebras-demonstrates-trillion-parameter-model-training-on-a-single-cs-3-system>, 2024.
- [7] OpenAI, *Introducing chatgpt*, <https://openai.com/blog/chatgpt>, 2022.
- [8] Google DeepMind, *Welcome to the gemini era*, <https://deepmind.google/technologies/gemini>, 2023.
- [9] Microsoft, *Microsoft Copilot: Your everyday ai companion*, <https://copilot.microsoft.com>, 2023.
- [10] Epoch AI, *Key Trends and Figures in Machine Learning*, <https://epoch.ai/trends>, Accessed: 2025-04-11, 2023.
- [11] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, “ZeRO: memory optimizations toward training trillion parameter models,” in *Proceedings of the 2020 International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’20)*, 2020, ISBN: 9781728199986.
- [12] Meta, *Fully Sharded Data Parallel: Faster ai training with fewer gpus*, <https://engineering.fb.com/2021/07/15/open-source/fsdp>, 2021.

- [13] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, “A Survey on Distributed Machine Learning,” *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, 2020.
- [14] Google, *Cloud tpu*, <https://cloud.google.com/tpu>, 2021.
- [15] K. Lee and S. Sengupta, *Introducing the AI Research SuperCluster – Meta’s cutting-edge AI supercomputer for AI research*, <https://ai.meta.com/blog/ai-rsc>, 2022.
- [16] Cerebras Systems, *Cerebras Systems: Achieving industry best ai performance through a systems approach*, <https://cerebras.net/wp-content/uploads/2021/04/Cerebras-CS-2-Whitepaper.pdf>, 2021.
- [17] D. Patel, *Tenstorrent Wormhole Analysis - A Scale Out Architecture for Machine Learning That Could Put NVIDIA On Their Back Foot*, <https://www.semianalysis.com/p/tenstorrent-wormhole-analysis-a-scale>, 2021.
- [18] X. Zhao, Z. Zhang, and C. Wu, “Adapcc: Making collective communication in distributed machine learning adaptive,” in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, 2024, pp. 25–35.
- [19] 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 (ISCA ’22)*, 2022, pp. 581–596, ISBN: 9781450386104.
- [20] J. Yoo et al., “Towards a standardized representation for deep learning collective algorithms,” in *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2024, pp. 33–36.
- [21] M. 4.1, *Introduction and Overview*, <https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report/node114.htm>, 2023.
- [22] B. Klenk, N. Jiang, G. Thorson, and L. Dennison, “An in-network architecture for accelerating shared-memory multiprocessor collectives,” in *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA ’20)*, 2020, pp. 996–1009.
- [23] A. Sapio et al., “Scaling Distributed Machine Learning with In-Network Aggregation,” in *arXiv:1903.06701 [cs.DC]*, 2019.
- [24] H. Mikami, H. Suganuma, P. U-chupala, Y. Tanaka, and Y. Kageyama, “Massively Distributed SGD: Imagenet/resnet-50 training in a flash,” in *arXiv:1811.05233 [cs.LG]*, 2019.

- [25] S. Rashidi et al., “Enabling Compute-Communication Overlap in Distributed Deep Learning Training Platforms,” in *Proceedings of the 48th International Symposium on Computer Architecture (ISCA)*, 2021.
- [26] xAI, *Colossus*, <https://x.ai/colossus>, 2025.
- [27] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, “ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms,” in *Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '20)*, 2020, pp. 81–92.
- [28] Y. Huang et al., “GPipe: Efficient training of giant neural networks using pipeline parallelism,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [29] A. Harlap et al., “PipeDream: Fast and efficient pipeline parallel dnn training,” *arXiv:1806.03377 [cs.DC]*, 2018.
- [30] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, “ZeRO-Infinity: Breaking the gpu memory wall for extreme scale deep learning,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.
- [31] J. Ren et al., “ZeRO-Offload: Democratizing billion-scale model training,” in *2021 USENIX Annual Technical Conference (ATC)*, 2021.
- [32] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, “Megatron-LM: Training multi-billion parameter language models using model parallelism,” in *arXiv:1909.08053 [cs.CL]*, 2019.
- [33] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, “Accelerating Distributed Reinforcement Learning with In-Switch Computing,” in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19)*, 2019, pp. 279–291.
- [34] Y. Lei et al., “FAST: An efficient scheduler for all-to-all GPU communication,” in *arXiv:2505.09764*, version: 2, Oct. 10, 2025. *arXiv: 2505.09764[cs]*.
- [35] M. Naumov et al., “Deep learning recommendation model for personalization and recommendation systems,” in *arXiv:1906.00091 [cs.IR]*, 2019.
- [36] NVIDIA, *NVIDIA Collective Communication Library (NCCL)*, <https://developer.nvidia.com/nccl>, 2017.

- [37] AMD, *RCCL 2.18.3 Documentation*, <https://rocm.docs.amd.com/projects/rccl/en/latest>, 2023.
- [38] Intel, *Intel oneapi collective communications library*, <https://www.intel.com/content/www/us/en/developer/tools/oneapi/oneccl.html>, 2020.
- [39] 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.
- [40] M. Cowan, S. Maleki, M. Musuvathi, O. Saarikivi, and Y. Xiong, “Mscclang: Microsoft collective communication language,” in *ASPLOS 2023*, ser. ASPLOS 2023, 2023, pp. 502–514, ISBN: 9781450399166.
- [41] A. Shah et al., “MSCCL++: Rethinking GPU communication abstractions for cutting-edge AI applications,” in *arXiv:2504.09014*, Aug. 21, 2025. arXiv: 2504.09014[cs].
- [42] R. Thakur, R. Rabenseifner, and W. Gropp, “Optimization of Collective Communication Operations in MPICH,” *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, 2005.
- [43] S. Jeaugey, *Massively Scale Your Deep Learning Training with NCCL 2.4*, <https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4>, 2019.
- [44] W. Won, S. Rashidi, S. Srinivasan, and T. Krishna, “LIBRA: Enabling Workload-Aware Multi-Dimensional Network Topology Optimization for Distributed Training of Large AI Models,” in *Proceedings of the 2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '24)*, 2024, pp. 205–216.
- [45] *ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms*, 2020.
- [46] D. K. Kadiyala, S. Rashidi, T. Heo, A. R. Bambhaniya, T. Krishna, and A. Daglis, “COMET: A comprehensive cluster design methodology for distributed deep learning training,” *arXiv:2211.16648 [cs.DC]*, 2022.
- [47] S. Rashidi et al., “Scalable distributed training of recommendation models: An astra-sim + ns3 case-study with tcp/ip transport,” in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2020, pp. 33–42.
- [48] T. Khan, S. Rashidi, S. Sridharan, P. Shurpali, A. Akella, and T. Krishna, “Impact of RoCE Congestion Control Policies on Distributed Training of DNNs,” in *Pro-*

ceedings of the 2022 IEEE Symposium on High-Performance Interconnects (HOTI '22), 2022, pp. 39–48.

- [49] X. Hou, R. Xu, S. Ma, Q. Wang, W. Jiang, and H. Lu, “Co-designing the topology/algorithm to accelerate distributed training,” in *2021 IEEE Intl. Conf. on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 1010–1018.
- [50] M. Ott, S. Shleifer, M. Xu, P. Goyal, Q. Duval, and V. Caggiano, *Fully Sharded Data Parallel: faster AI training with fewer GPUs*, <https://engineering.fb.com/2021/07/15/open-source/fsdp>, 2021.
- [51] D. Narayanan et al., “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, 2021, ISBN: 9781450384421.
- [52] R. Majumder and J. Wang, *DeepSpeed: Extreme-scale model training for everyone*, <https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone>, 2020.
- [53] S. Rajbhandari et al., “DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation ai scale,” in *arXiv:2201.05596 [cs.LG]*, 2022.
- [54] Z. Jia, M. Zaharia, and A. Aiken, “Beyond data and model parallelism for deep neural networks,” *Proceedings of the 2019 Conference on Systems and Machine Learning (SysML)*, 2019.
- [55] C. Unger et al., “Unity: Accelerating dnn training through joint optimization of algebraic transformations and parallelization,” in *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022, pp. 267–284.
- [56] S. Li et al., “Pytorch distributed: Experiences on accelerating data parallel training,” *arXiv:2006.15704 [cs.DC]*, 2020.
- [57] *NVIDIA DGX Systems*.
- [58] NVIDIA, *NVIDIA DGX A100: The Universal System for AI Infrastructure*, 2021.
- [59] G. Cloud, *System Architecture - Cloud TPU*, 2022.
- [60] N. P. Jouppi et al., “A Domain-Specific Supercomputer for Training Deep Neural Networks,” *Commun. ACM*, vol. 63, no. 7, pp. 67–78, 2020.

- [61] ServeTheHome, *Intel Architecture Day 2021 Xe HPC Ponte Vecchio Xe Link*, 2021.
- [62] NVIDIA, *NVIDIA NVLink High-Speed GPU Interconnect*, <https://www.nvidia.com/en-us/design-visualization/nvlink-bridges>, 2022.
- [63] Cerebras, *Cerebras Systems: Achieving Industry Best AI Performance Through A Systems Approach*, 2021.
- [64] Tesla, *Tesla Dojo Technology: A Guide to Tesla's Configurable Floating Point Formats & Arithmetic*, 2022.
- [65] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: a detailed on-chip network model inside a full-system simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 33–42.
- [66] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds. 2010, pp. 15–34, ISBN: 978-3-642-12331-3.
- [67] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits, "Using multirail networks in high-performance clusters," in *Proceedings of the 2001 IEEE International Conference on Cluster Computing (CLUSTER '01)*, 2001, pp. 15–24.
- [68] N. Wolfe et al., "Preliminary performance analysis of multi-rail fat-tree networks," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID '17)*, 2017, pp. 258–261.
- [69] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "SCALE-Sim: Systolic cnn accelerator simulator," *arXiv:1811.02883 [cs.DC]*, 2018.
- [70] R. L. Graham et al., "Scalable hierarchical aggregation protocol (SHArP): A hardware architecture for efficient data reduction," in *Proceedings of the First Workshop on Optimization of Communication in HPC (COM-HPC '16)*, 2016, pp. 1–10.
- [71] NVIDIA, *ConnectX SmartNICs*, 2021.
- [72] Ethernet Technology Consortium, *800G Specification*, 2020.
- [73] T. P. Morgan, *Deep Dive on Google's Exascale TPuv4 AI Systems*, 2022.
- [74] E. Chan, R. van de Geijn, W. Gropp, and R. Thakur, "Collective Communication on Architectures That Support Simultaneous Communication over Multiple Links," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '06)*, 2006, pp. 2–11, ISBN: 1595931899.

- [75] J. Kim, W. J. Dally, S. Scott, and D. Abts, “Technology-Driven, Highly-Scalable Dragonfly Topology,” in *Proceedings of the 2008 International Symposium on Computer Architecture (ISCA '08)*, 2008, pp. 77–88.
- [76] R. W. Hockney, “The communication challenge for mpp: Intel paragon and meiko cs-2,” *Parallel Computing*, vol. 20, no. 3, pp. 389–398, 1994.
- [77] NVIDIA, *NVIDIA V100 Tensor Core GPU*, 2017.
- [78] S. Pal, *Scale-Out Packageless Processing*, 2021.
- [79] S. Pal, D. Petrisko, M. Tomei, P. Gupta, S. S. Iyer, and R. Kumar, “Architecting waferscale processors - a gpu case study,” in *Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA '19)*, 2019, pp. 250–263.
- [80] A. F. Rodrigues et al., “The structural simulation toolkit,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, 2011.
- [81] D. Sanchez and C. Kozyrakis, “ZSim: Fast and accurate microarchitectural simulation of thousand-core systems,” *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 475–486, 2013.
- [82] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim, “dist-gem5: Distributed simulation of computer clusters,” in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 153–162.
- [83] 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 (HPDC)*, 2010, pp. 597–604.
- [84] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, “Simulating mpi applications: The smpi approach,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2387–2400, 2017.
- [85] W. J. Robinson, F. Esposito, and M. A. Zuluaga, “DTS: A simulator to estimate the training time of distributed deep neural networks,” in *The 30th International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2022.
- [86] N. Ardalani, S. Pal, and P. Gupta, “DeepFlow: A cross-stack pathfinding framework for distributed ai systems,” *arXiv:2211.03309 [cs.AR]*, 2022.

- [87] NVIDIA, *Nvlink and nvswitch*, <https://www.nvidia.com/en-us/data-center/nvlink>, 2022.
- [88] AMD, *AMD Infinity Architecture*, <https://www.amd.com/en/technologies/infinity-architecture>, 2023.
- [89] I. Cutress, *Analyzing Intel’s Discrete Xe-HPC Graphics Disclosure: Ponte Vecchio, Rambo Cache, and Gelato*, <https://www.anandtech.com/show/15188/analyzing-intels-discrete-xe-hpc-graphics-disclosure-ponte-vecchio>, 2019.
- [90] NVIDIA, *The nvidia quantum infiniband platform*, <https://www.nvidia.com/en-us/networking/products/infiniband>, 2023.
- [91] J. Kaplan et al., “Scaling laws for neural language models,” in *arXiv:2001.08361 [cs.LG]*, 2020.
- [92] Cisco, *Co-packaged optics and an open ecosystem*, <https://blogs.cisco.com/sp/co-packaged-optics-and-an-open-ecosystem>, 2021.
- [93] A. Arunkumar et al., “MCM-GPU: Multi-chip-module gpus for continued performance scalability,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA’ 17)*, 2017, pp. 320–332.
- [94] Y. S. Shao et al., “Simba: Scaling deep-learning inference with multi-chip-module-based architecture,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’52)*, 2019, pp. 14–27.
- [95] N. Jouppi et al., “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 (ISCA ’23)*, 2023, ISBN: 9798400700958.
- [96] Lightmatter, *Passage: A wafer-scale, programmable photonic interconnect*, <https://lightmatter.co/products/passage>, 2023.
- [97] Intel, *Intel data center gpu max series technical overview*, <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-data-center-gpu-max-series-overview.html>, 2023.
- [98] NVIDIA, *NVIDIA DGX-2: The world’s most powerful deep learning system for the most complex ai challenges*, <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-1/dgx-2-datasheet-us-nvidia-955420-r2-web-new.pdf>, 2019.

- [99] NVIDIA, *Nvidia a100 tensor core gpu*, <https://www.nvidia.com/en-us/data-center/a100>, 2021.
- [100] NVIDIA, *Nvidia h100 tensor core gpu*, <https://www.nvidia.com/en-us/data-center/h100>, 2022.
- [101] T. P. Morgan, *Inside Tesla's Innovative And Homegrown Dojo AI Supercomputer*, <https://www.nextplatform.com/2022/08/23/inside-teslas-innovative-and-homegrown-dojo-ai-supercomputer>, 2022.
- [102] R. Smith, *Spotted At Hot Chips: Quad tile intel xe-hp gpu*, <https://www.anandtech.com/show/15996/spotted-at-hot-chips-quad-tile-intel-xehp-gpu>, 2020.
- [103] Intel, *Habana*, <https://habana.ai>, 2021.
- [104] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.
- [105] M. Cho, U. Finkler, D. Kung, and H. Hunter, "BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy," *Proceedings of the 2019 Conference on Systems and Machine Learning (SysML)*, pp. 241–251, 2019.
- [106] S. A. Mojumder et al., "Profiling dnn workloads on a volta-based dgx-1 system," in *Proceedings of the 2018 IEEE International Symposium on Workload Characterization (IISWC '18)*, 2018, pp. 122–133.
- [107] NVIDIA, *NVIDIA DGX SuperPOD: Instant Infrastructure for AI Leadership*, <https://resources.nvidia.com/en-us-auto-datacenter/nvpod-superpod-wp-09>, 2020.
- [108] D. Mudigere et al., "Software-hardware co-design for fast and scalable training of deep learning recommendation models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA '22)*, 2022, pp. 993–1011.
- [109] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and Generic Collectives for Distributed ML," in *Proceedings of the 2020 Machine Learning and Systems (MLSys '20)*, vol. 2, 2020, pp. 172–186.
- [110] J. Chen, S. Li, R. Gun, J. Yuan, and T. Hoefler, "AutoDDL: Automatic distributed deep learning with asymptotically optimal communication," in *arXiv:2301.06813 [cs.DC]*, 2023.
- [111] Intel, *Gaudi Training Platform White Paper*, <https://habana.ai/wp-content/uploads/2019/06/Habana-Gaudi-Training-Platform-whitepaper.pdf>, 2019.

- [112] W. Wang et al., “TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs,” in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*, 2023, pp. 739–767, ISBN: 978-1-939133-33-5.
- [113] M. Khani et al., “TeraRack: A tbps rack for machine learning training,” 2020.
- [114] Gurobi Optimization, *Gurobi optimizer*, <https://www.gurobi.com/solutions/gurobi-optimizer>, 2023.
- [115] ASTRA-sim, *ASTRA-sim Validation*, <https://astra-sim.github.io/astra-sim-docs/validation/validation.html>.
- [116] A. Vaswani et al., “Attention is all you need,” in *Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS '17)*, vol. 30, 2017.
- [117] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*, 2016, pp. 770–778.
- [118] M. Wagh, *Introducing Compute Express Link (CXL) 3.0: Expanding fabric capabilities and management*, https://www.openfabrics.org/wp-content/uploads/2023-workshop/2023-workshop-presentations/day-2/202_MWagh.pdf, 2023.
- [119] R. Alverson, D. Roweth, and L. Kaplan, “The gemini system interconnect,” in *Proceedings of the 18th IEEE Symposium on High Performance Interconnects (HOTI '10)*, 2010, pp. 83–87.
- [120] D. Chen et al., “Looking under the hood of the ibm blue gene/q network,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12)*, 2012.
- [121] Y. Ajima, S. Sumimoto, and T. Shimizu, “Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers,” *Computer*, vol. 42, no. 11, pp. 36–40, 2009.
- [122] A. J. Peña, R. G. C. Carvalho, J. Dinan, P. Balaji, R. Thakur, and W. Gropp, “Analysis of topology-dependent mpi performance on gemini networks,” in *Proceedings of the 20th European MPI Users' Group Meeting (EuroMPI '13)*, 2013, pp. 61–66.
- [123] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, “Multi-core and network aware mpi topology functions,” in *Proceedings of the 18th European MPI Users' Group Meeting (EuroMPI '11)*, 2011, pp. 50–60.
- [124] A. Faraj, S. Kumar, B. Smith, A. Mamidala, and J. Gunnels, “MPI Collective Communications on The Blue Gene/P Supercomputer: Algorithms and optimizations,”

in *Proceedings of the 17th IEEE Symposium on High Performance Interconnects (HOTI '09)*, 2009, pp. 63–72.

- [125] S. Kumar, A. Mamidala, P. Heidelberger, D. Chen, and D. Faraj, “Optimization of mpi collective operations on the ibm blue gene/q supercomputer,” *Int. J. High Perform. Comput. Appl.*, vol. 28, no. 4, pp. 450–464, 2014.
- [126] G. Almási et al., “Optimization of mpi collective communication on bluegene/l systems,” in *Proceedings of the 19th Annual International Conference on Supercomputing (ICS '05)*, 2005, pp. 253–262.
- [127] T. Adachi et al., “The design of ultra scalable mpi collective communication on the k computer,” *Computer Science - Research and Development*, vol. 28, 2013.
- [128] J. Dong et al., “EFLOPS: Algorithm and system co-design for a high performance distributed training platform,” in *Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA '20)*, 2020, pp. 610–622.
- [129] Groq, *GroqRack Compute Cluster*, <https://wow.groq.com/groqrack-compute-cluster>, 2024.
- [130] AMD, *AMD Infinity Fabric Link*, <https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/other/56978.pdf>, 2020.
- [131] NVIDIA, *InfiniBand Networking Solutions*, <https://www.nvidia.com/en-us/networking/products/infiniband>, 2024.
- [132] A. Shah et al., “TACCL: Guiding collective algorithm synthesis using communication sketches,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Apr. 2023, pp. 593–612, ISBN: 978-1-939133-33-5.
- [133] Z. Cai et al., “Synthesizing optimal collective algorithms,” in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPOPP '21, 2021, pp. 62–75, ISBN: 9781450382946.
- [134] 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.
- [135] S. Cho, H. Son, and J. Kim, “Logical/Physical Topology-Aware Collective Communication in Deep Learning Training,” in *Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA '23)*, 2023, pp. 56–68.

- [136] Microsoft, *Microsoft Collective Communication Library*, <https://github.com/microsoft/msccl>, 2023.
- [137] E. Gabrielyan and R. Hersch, “Network topology aware scheduling of collective communications,” in *Proceedings of the 10th International Conference on Telecommunications (ICT '03)*, 2003, pp. 1051–1058.
- [138] 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 *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA '21)*, 2021, pp. 181–194.
- [139] A. Paulus, M. Rolínek, V. Musil, B. Amos, and G. Martius, “Combopnet: Fit the right np-hard problem by learning integer programming constraints,” in *Proceedings of the 38th International Conference on Machine Learning (ICML '21)*, vol. 139, 2021, pp. 8443–8453.
- [140] E. Köhler, K. Langkau, and M. Skutella, “Time-Expanded Graphs for Flow-Dependent Transit Times,” in *Proceedings of the 2002 European Symposium on Algorithms (ESA '02)*, 2002, pp. 599–611, ISBN: 978-3-540-45749-7.
- [141] S. Belieres, M. Hewitt, N. Jozefowicz, and F. Semet, “A time-expanded network reduction matheuristic for the logistics service network design problem,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 147, 2021.
- [142] A. Tafreshian, M. Abdolmaleki, N. Masoud, and H. Wang, “Proactive shuttle dispatching in large-scale dynamic dial-a-ride systems,” *Transportation Research Part B: Methodological*, vol. 150, pp. 227–259, 2021.
- [143] L. Zhao, S. Maleki, Z. Yang, H. Pourreza, and A. Krishnamurthy, “Forestcoll: Throughput-optimal collective communications on heterogeneous network fabrics,” in *arXiv:2402.06787 [cs.NI]*, 2025.
- [144] X. Liu et al., “Rethinking machine learning collective communication as a multi-commodity flow problem,” in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24, 2024, pp. 16–37, ISBN: 9798400706141.
- [145] K. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda, “Designing topology-aware collective communication algorithms for large scale InfiniBand clusters: Case studies with Scatter and Gather,” in *Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW '10)*, 2010.

- [146] A. Vahdat and M. Lohmeyer, *Enabling next-generation AI workloads: Announcing TPU v5p and AI Hypercomputer*, <https://cloud.google.com/blog/products/ai-machine-learning/introducing-cloud-tpu-v5p-and-ai-hypercomputer>, 2023.
- [147] E. Sigler and B. Chess, *Scaling Kubernetes to 7,500 nodes*, <https://openai.com/research/scaling-kubernetes-to-7500-nodes>, 2021.
- [148] M. O. Kevin Lee Adi Gangidi, *Building Meta's GenAI Infrastructure*, <https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/>, 2024.
- [149] S. Kumar and N. Jouppi, "Highly Available Data Parallel ML training on Mesh Networks," in *arXiv:2011.03605 [cs.LG]*, 2020.
- [150] J. Ma, D. Dong, C. Li, K. Wu, and L. Xiao, "Paard: Proximity-aware all-reduce communication for dragonfly networks," in *2021 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 255–262.
- [151] J. Kim, J. Balfour, and W. Dally, "Flattened Butterfly Topology for On-Chip Networks," in *Proceedings of the 40th Annual Symposium on Microarchitecture (MICRO '07)*, 2007, pp. 172–182.
- [152] M. Flajslik, E. Borch, and M. A. Parker, "Megafly: A Topology for Exascale Systems," in *High Performance Computing*, 2018, pp. 289–310, ISBN: 978-3-319-92040-5.
- [153] M. Besta and T. Hoefler, "Slim Fly: A Cost Effective Low-Diameter Network Topology," in *arXiv:1912.08968 [cs.NI]*, 2020.
- [154] M. Ferrati and L. Pallottino, "A time expanded network based algorithm for safe and efficient distributed multi-agent coordination," in *Proceedings of the 52nd IEEE Conference on Decision and Control (CDC '13)*, 2013, pp. 2805–2810.
- [155] E. Köhler and M. Strehler, "Combining Static and Dynamic Models for Traffic Signal Optimization Inherent Load-dependent Travel Times in a Cyclically Time-expanded Network Model," *Procedia - Social and Behavioral Sciences*, vol. 54, pp. 1125–1134, 2012.
- [156] W. Won, T. Heo, S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale," in *Proceedings of the 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '23)*, 2023, pp. 283–294.

- [157] Y. Wu et al., “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” in *arXiv:1609.08144 [cs.CL]*, 2016.
- [158] C. Rosset, *Turing-NLG: A 17-billion-parameter language model by Microsoft*, <https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft>, 2020.
- [159] A. Jangda et al., “Breaking the Computation and Communication Abstraction Barrier in Distributed Machine Learning Workloads,” in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’22)*, 2022, pp. 402–416, ISBN: 9781450392051.
- [160] T. T. Nguyen and M. Wahib, “An Allreduce Algorithm and Network Co-design for Large-Scale Training of Distributed Deep Learning,” in *Proceedings of the 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid ’21)*, 2021, pp. 396–405.
- [161] L. Zhao et al., “Efficient Direct-Connect Topologies for Collective Communications,” in *arXiv:2202.03356 [cs.NI]*, 2023.
- [162] NVIDIA, *Nvidia collective communications library*, <https://developer.nvidia.com/nccl>, 2025.
- [163] AMD, *Rccl documentation*, <https://rocm.docs.amd.com/projects/rccl/en/docs-6.3.3/index.html>, 2025.
- [164] J. Cao et al., “SyCCL: Exploiting symmetry for efficient collective communication scheduling,” in *Proceedings of the ACM SIGCOMM 2025 Conference*, ser. SIGCOMM ’25, Aug. 27, 2025, pp. 645–662, ISBN: 979-8-4007-1524-2.
- [165] H. Huang et al., “Towards moe deployment: Mitigating inefficiencies in mixture-of-expert (moe) inference,” in *arXiv:2303.06182 [cs.DC]*, 2023.
- [166] S. Li et al., “Pytorch distributed: Experiences on accelerating data parallel training,” *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3005–3018, Aug. 2020.
- [167] J. W. Rae et al., “Scaling language models: Methods, analysis, and insights from training gopher,” in *arXiv:2112.11446 [cs.CL]*, 2022.
- [168] A. Chowdhery et al., “Palm: Scaling language modeling with pathways,” *J. Mach. Learn. Res.*, vol. 24, no. 1, Jan. 2023.
- [169] J. Selig, *The cerebras software development kit: A technical overview*, <https://f.hubspotusercontent30.net/hubfs/8968533/Cerebras%20SDK%20Technical%20Overview.pdf>

20Overview % 20White % 20Paper . pdf ? utm_campaign = Tech % 20Leadership % 20PR%202022&utm_source=SDK_WP, 2022.

- [170] NVIDIA, *NVLink and NVLink Switch*, <https://www.nvidia.com/en-us/data-center/nvlink/>, 2025.
- [171] ADC Telecommunications, *Fundamentals of Ethernet Technology*, https://www.adckcl.com/in/en/library/White_Papers/Enterprise/401270IN.pdf, 2009.
- [172] Mellanox Technologies, *InfiniBand Technology Overview*, https://network.nvidia.com/pdf/whitepapers/WP_InfiniBand_Technology_Overview.pdf, 2008.
- [173] S. Pichai and D. Hassabis, *Our next-generation model: Gemini 1.5*, <https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/>, 2024.
- [174] Z. Xiong, “High-probability polynomial-time complexity of restarted pdhg for linear programming,” in *arXiv:2501.00728 [math.OC]*, 2025.
- [175] E. Rakadjiev, T. Shimosawa, H. Mine, and S. Oshima, “Parallel smt solving and concurrent symbolic execution,” in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 3, 2015, pp. 17–26.

VITA

William Jonghoon Won received his B.S. in Computer Science and Engineering from Seoul National University in 2019 (summa cum laude). He joined the Georgia Institute of Technology (Georgia Tech) as a Ph.D. student in Computer Science in 2019, advised by Dr. Tushar Krishna, and received his M.S. in Computer Science (specializing in machine learning) from Georgia Tech in 2022.

William's research interests are in architecture, systems, and algorithms for machine learning, including software-hardware optimizations for distributed machine learning, training and inference of large-scale ML workloads, and modeling and simulation of distributed ML platforms.