SCALE-CCL: A Scalable Collective Communication Library for Wide-Area Distributed Training

Jiaheng Xiong, Qiaolun Zhang, Paolo Medagliani, Michele Ferrero, Xiaomin Liu, Meng Lian, Nicola Di Cicco, Baosen Zhao, Mëmëdhe Ibrahimi, Francesco Musumeci, Massimo Tornatore

December 1st , 2025

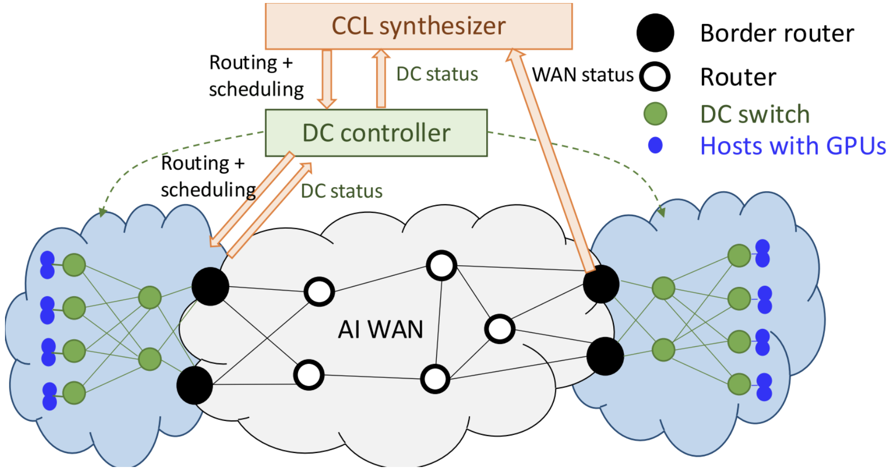
Outline

1. Background
2. Motivation
3. System Model
4. Problem Statement
5. Heuristic Algorithm
6. Results

Background

Across Data Center distributed Training:

1. Model sizes keep growing, and future workloads may exceed the capacity of a single datacenter
2. Compute and data resources are often distributed across multiple geographic regions
3. Current CCL algorithms don't adopt WAN scenario



Collective Communication Libraries (CCL): let multiple GPUs do **routing and scheduling of chunks** in a coordinated way so they can work together as a single, unified training system

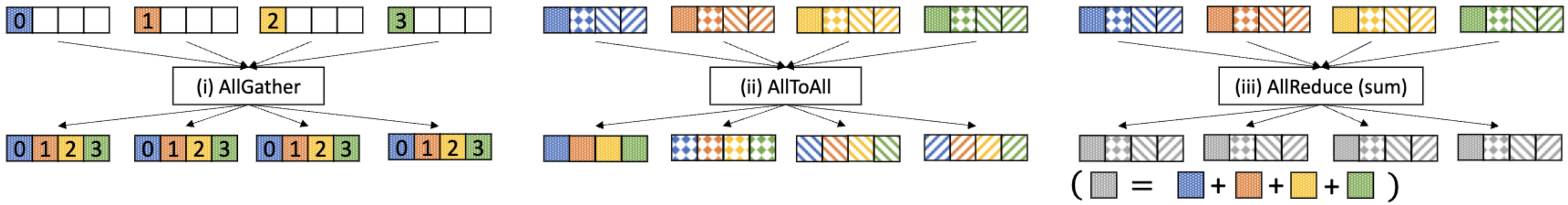
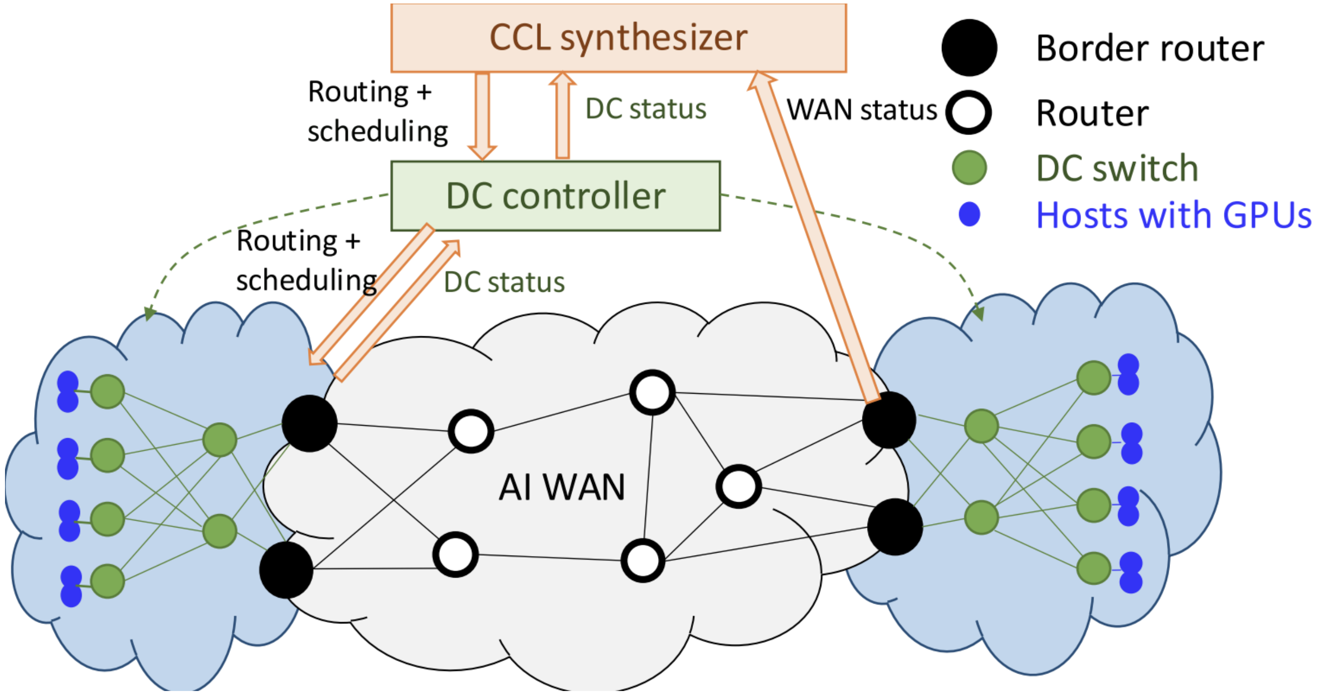


Fig 1: Example of Different Collective Communications [1]

[1] Shah, Aashaka, et al. "{TACCL}: Guiding collective algorithm synthesis using communication sketches." 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). 2023.

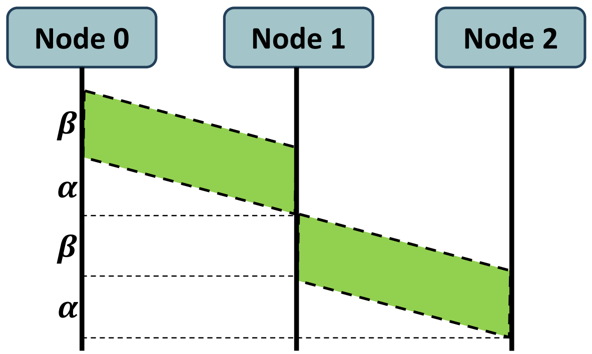
Motivation: Dynamic Environment



1. The link rate varies over time in WAN.
   - Traffic can surge from nearly idle to full link capacity within seconds
   - The fluctuation magnitude can reach $\pm$100% or even higher[1]
2. The CCL synthesizer needs to respond quickly
3. TECCL often responds at a timescale of tens or even hundreds of seconds. (Can not adopt WAN)

[1] Singh, Rachee, et al. "RADWAN: rate adaptive wide area network." Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 2018.

System Model (Different Transmission Type)

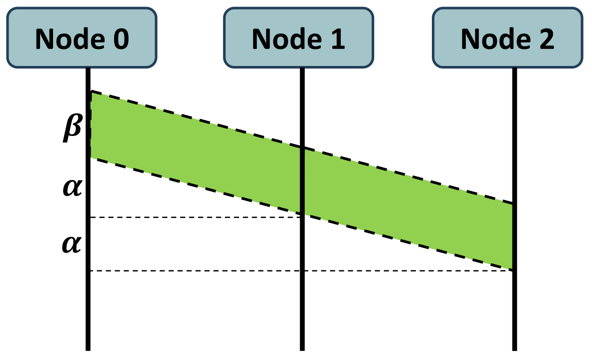


(a) Store and Forward

We model the transmission type of GPUs as Store and Forward

1. GPU-to-GPU transfers require receiving the entire chunk before forwarding it to the next hop (End to End flows)
2. This creates an end-to-end delay that accumulates propagation (α) and transmission time (β) on every hop

Transmission time: $T_{SF} = h(\alpha + \beta), h \text{ is the number of hops}$



(b) Cut-Through

We model the transmission type of Routers as Cut Through

1. A router can start forwarding as soon as the first part of the chunk arrives (packet level transmission from protocols)
2. End-to-end delay is dominated by propagation (α), with transmission time (β) determined by the bottleneck link only once

Transmission time: $T_{CT} = h\alpha + \beta$

Problem Statement

POLITECNICO MILANO 1863

Problem statement

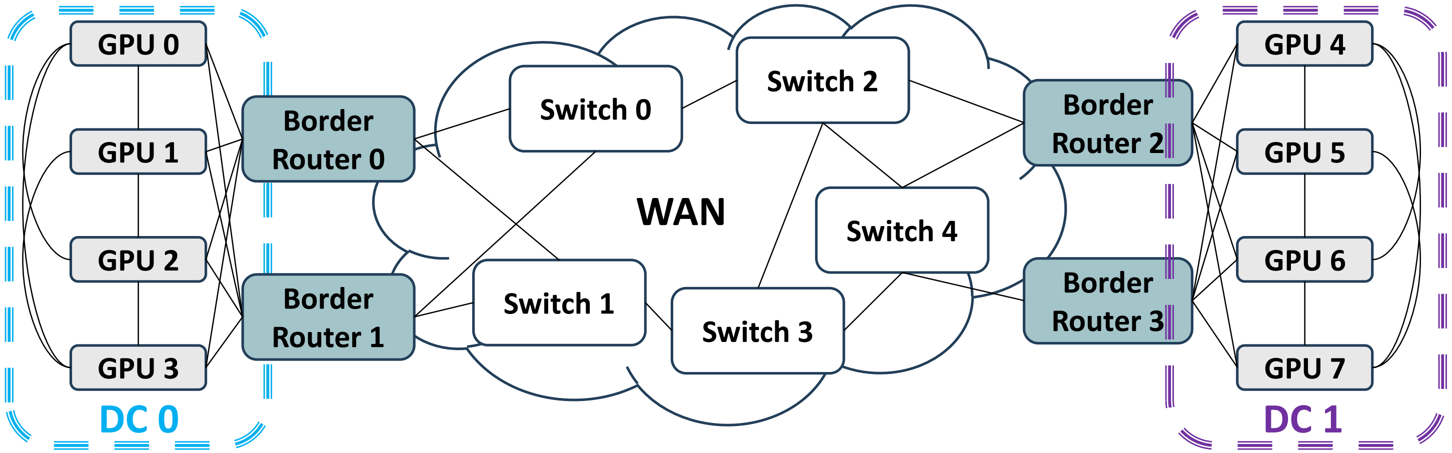
Given
- Network topology of multiple data centers
- Collective operation
- Chunk size
- Number of chunk

Decide
- Routing of subchunks
- Scheduling of forwarding chunks in one node

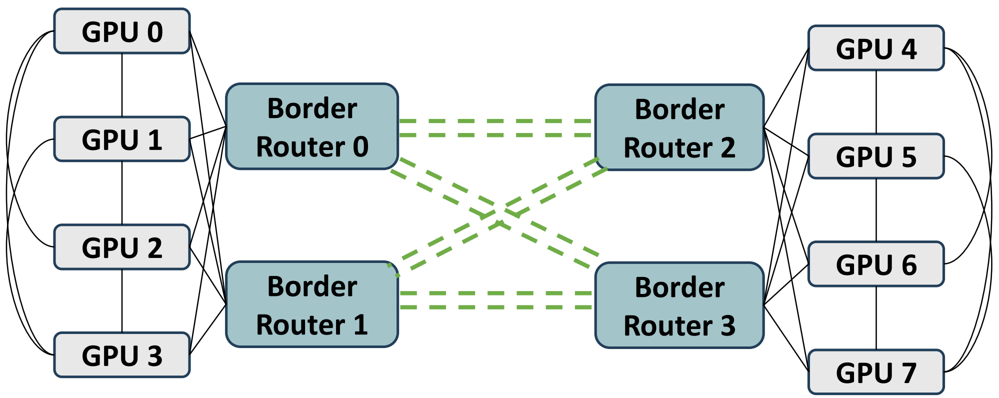
Objective
- Minimize the required to finish the collective

Constraints
- Link latency
- Node can only send a chunk after receiving it

System Model (Abstract Topology)



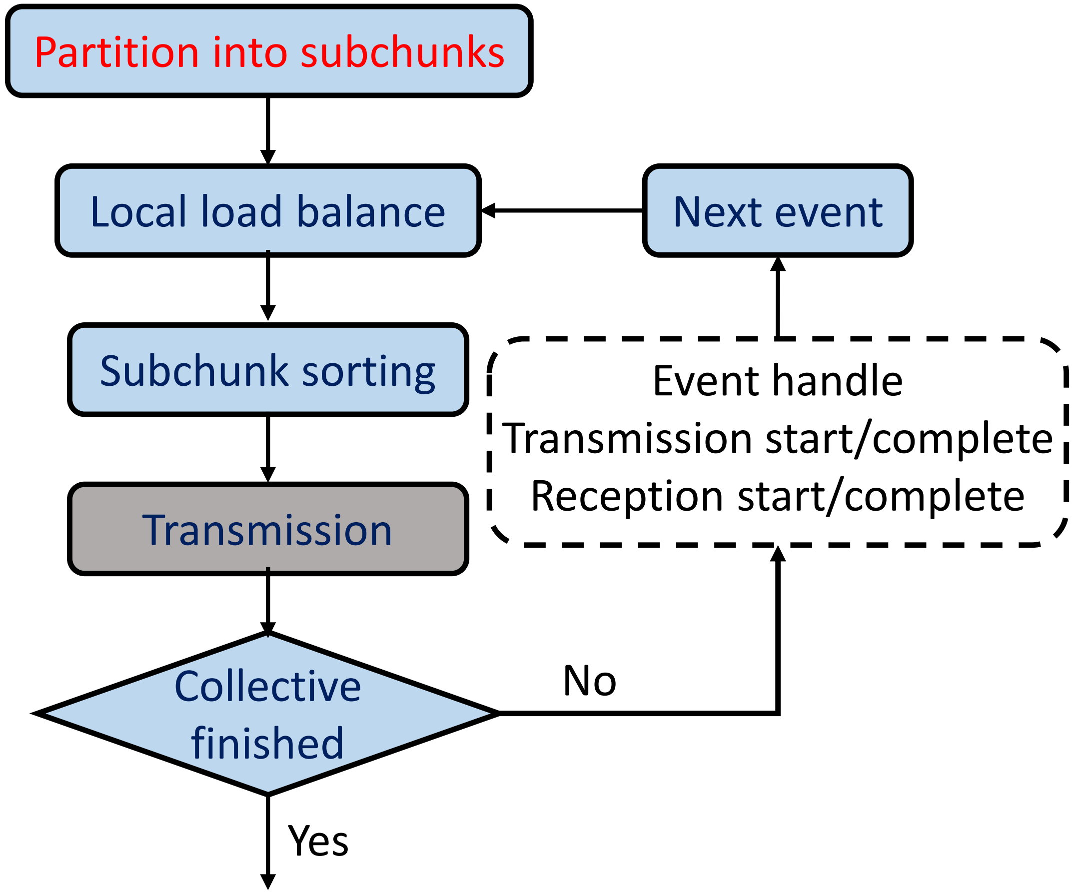
(a) Physical Topology

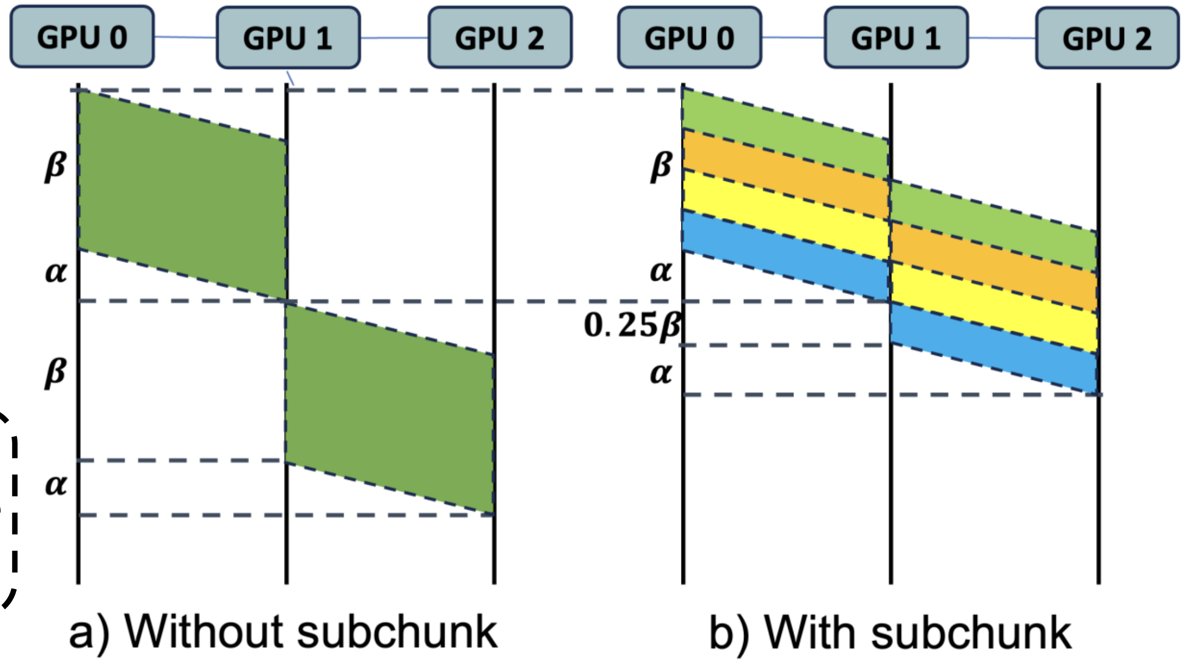


(b) Logical Topology

- We abstract the physical topology to logical topology as Fig. (b)
- Different field perform different function:
    1. **Intra-DC field:** leverages dense local connectivity to quickly diffuse subchunks among GPUs
    2. **WAN field:** forwards subchunks across border routers as early as possible toward the remote DC
    3. **Remote-DC field:** completes the final dissemination once subchunks arrive at the destination datacenter

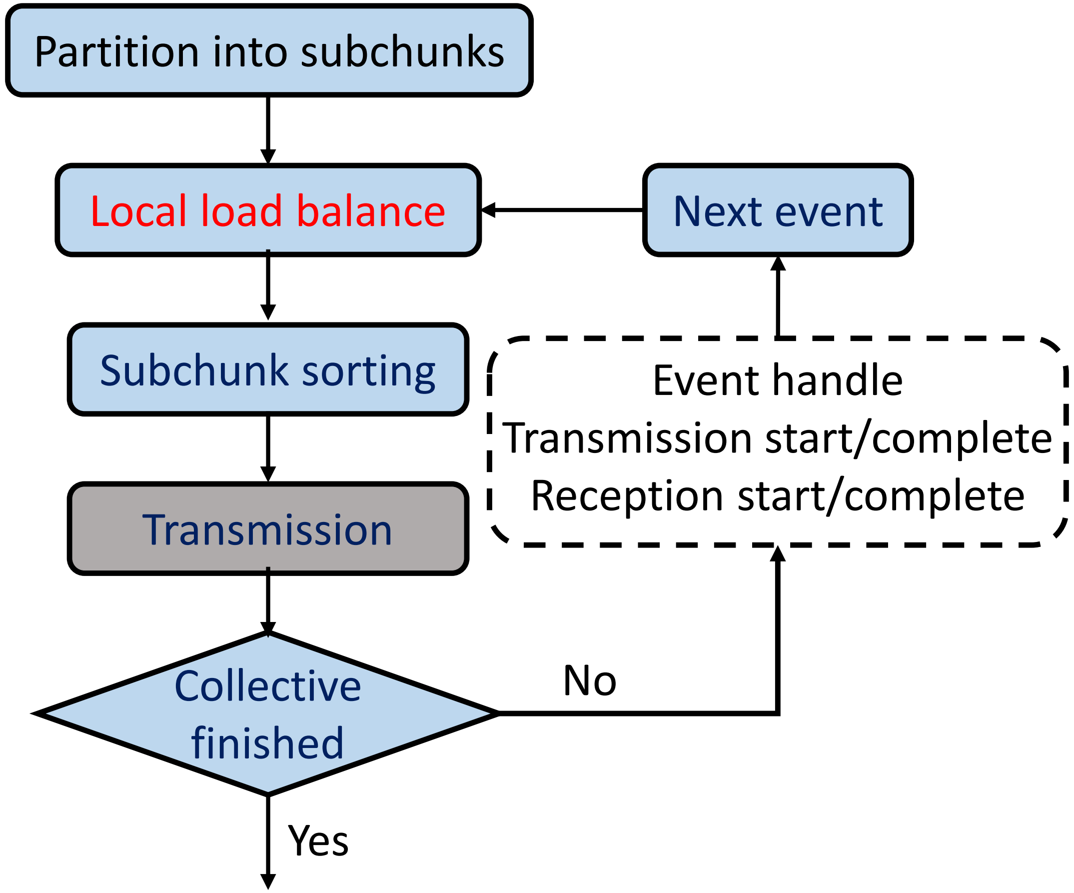
Flow Chart of the Proposed Algorithm

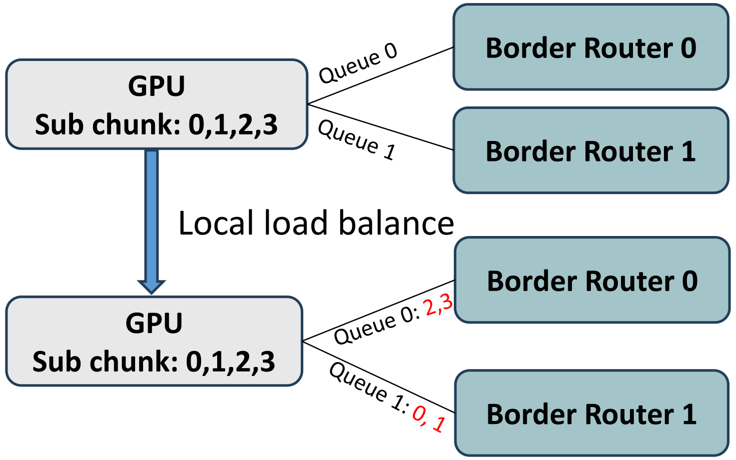




- Each chunk will be divided into multiple sub-chunks, and independent decisions will be made for each sub-chunk

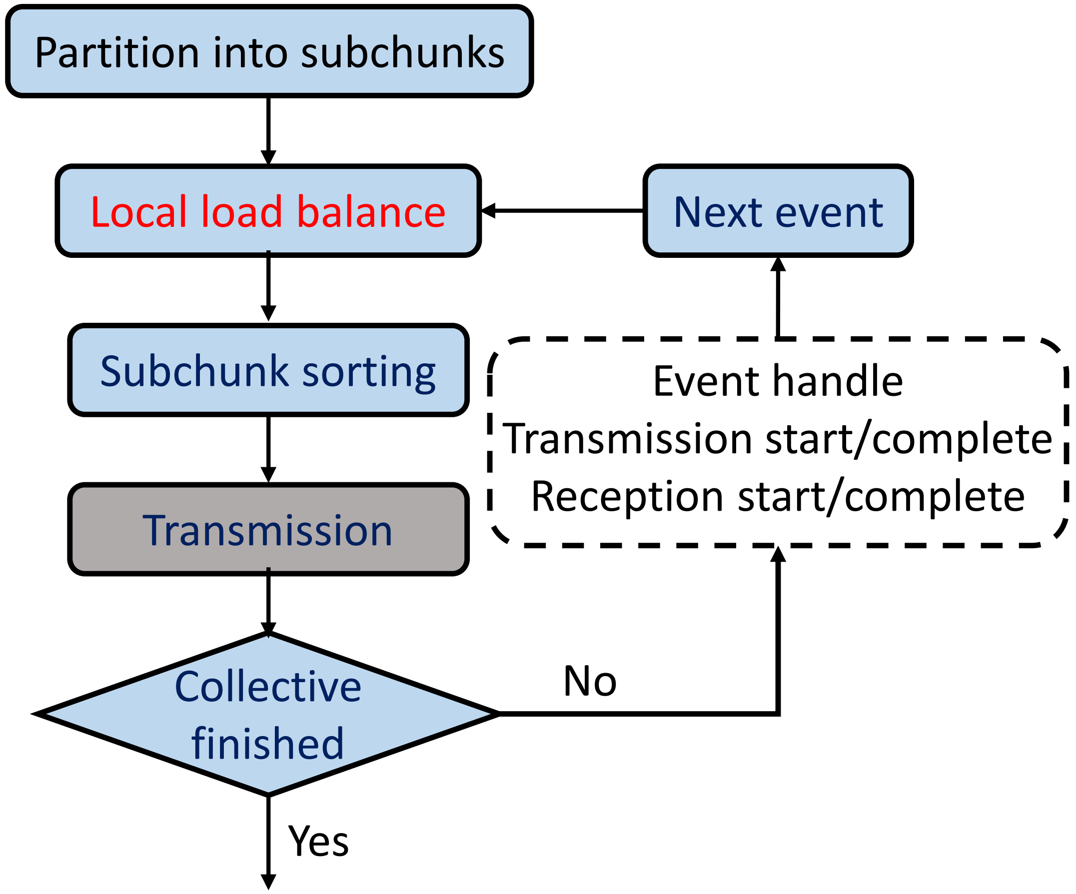
Flow Chart of the Proposed Algorithm

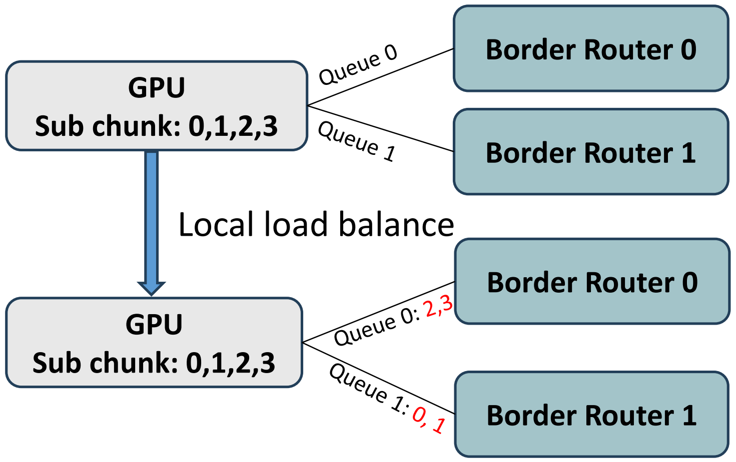




1. Local-Only Decisions: Nodes assign subchunks based on their own state and neighbors' known holdings.
2. Per-Neighbor Queues: Avoid redundant sends by tracking neighbor status and balancing load across queues.
3. Scope: Load balancing applies to inter-DC/WAN links; intra-DC uses broadcast only.

Flow Chart of the Proposed Algorithm

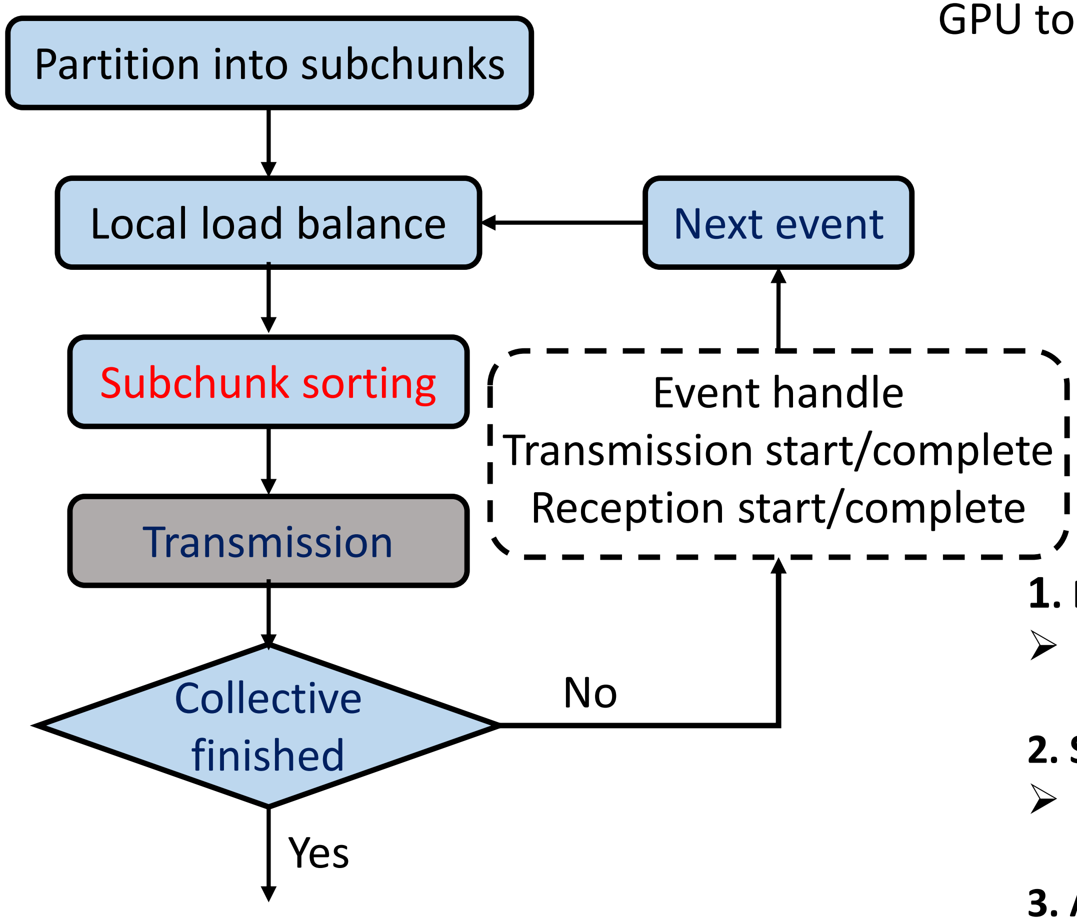




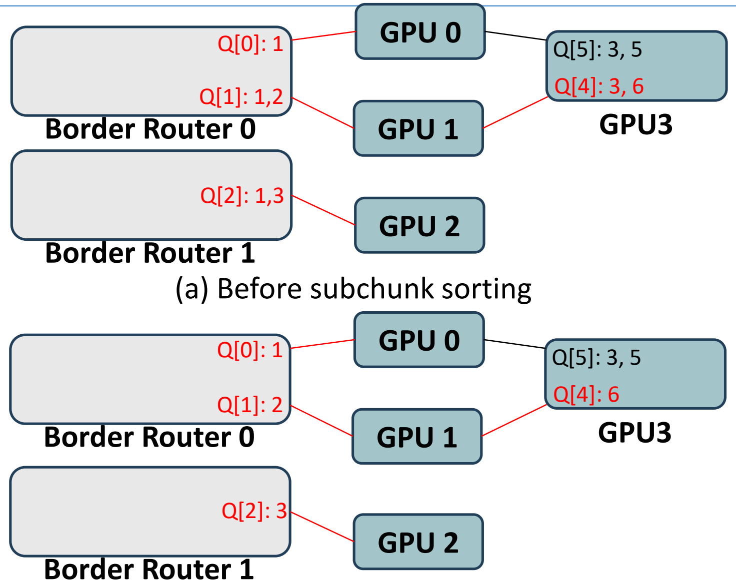
1. Intra-DC network: broadcast
2. **GPU to border router** link and **border router to border router** link: load balancing
   1. Local-Only Decisions: Nodes assign subchunks based on their own state and neighbors' known holdings
   2. Per-Neighbor Queues: Avoid redundant sends by tracking neighbor status and balancing load across queues

Flow Chart of the Proposed Algorithm

POLITECNICO
MILANO 1863



GPU to Border Router



(b) After subchunk sorting

1. Before Sorting — Redundancy Exists

➢ Subchunks like 1 and 3 appear in multiple queues (e.g., Q[1] of BR0 and Q[2] of BR1 both hold 1,3) → leads to redundant transmissions

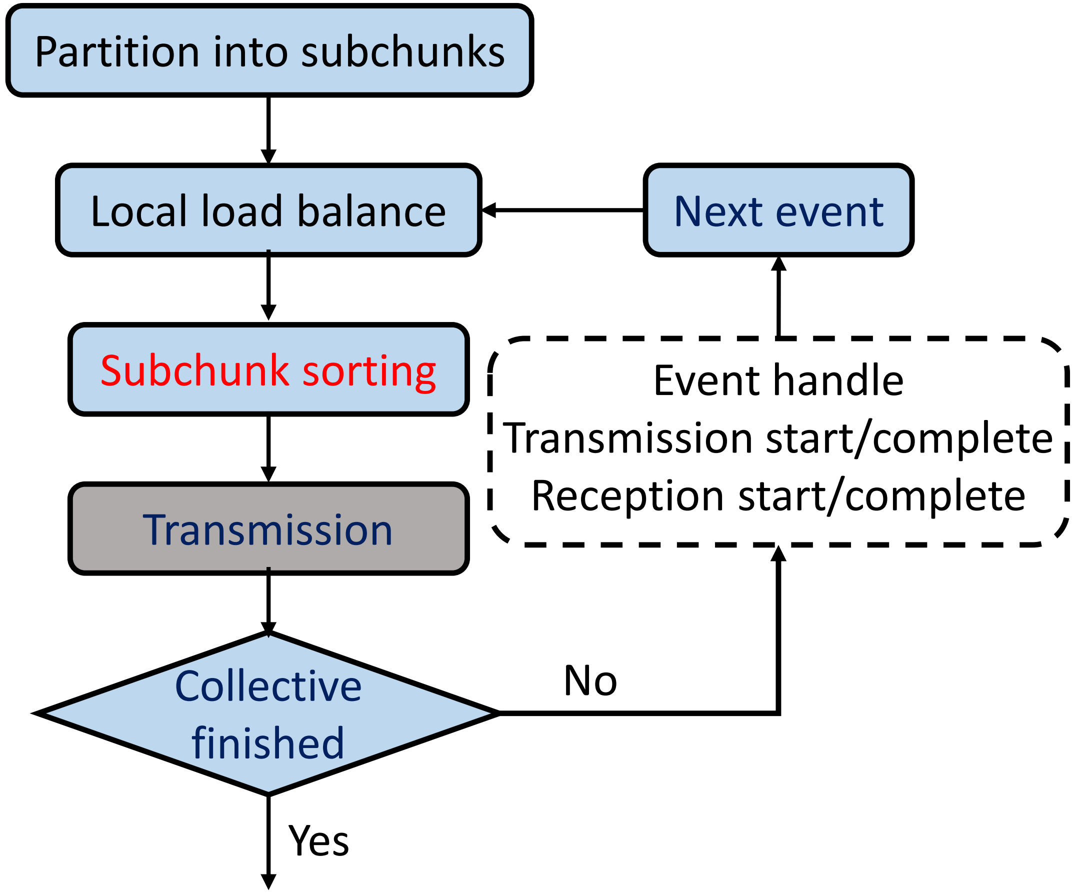
2. Sorting — Resolve Duplicates Locally

➢ For each duplicate subchunk, choose the (sender, receiver) pair with **lowest estimated delay**, and **remove it from other queues**

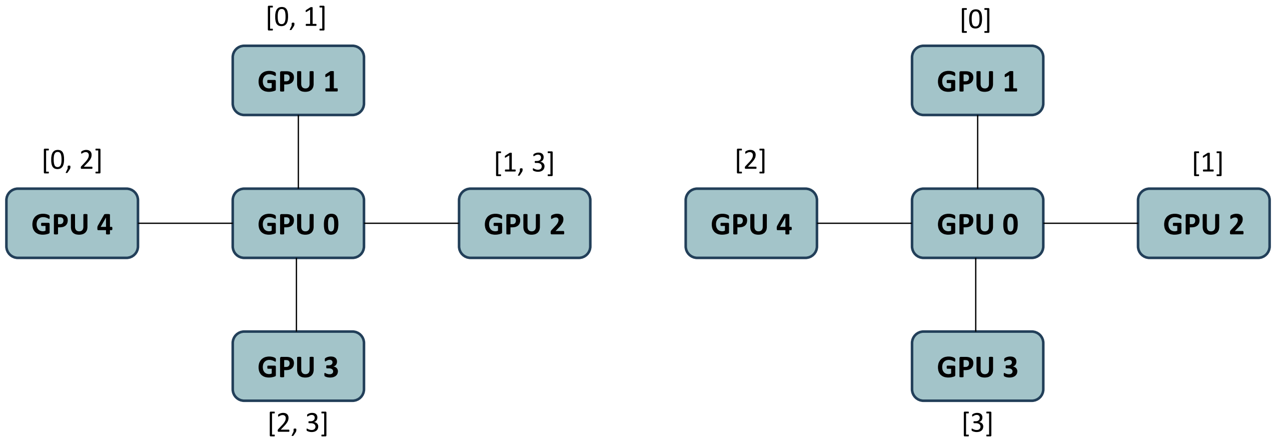
3. After Sorting — Unique Assignments

➢ Each subchunk now appears in only one queue

Flow Chart of the Proposed Algorithm



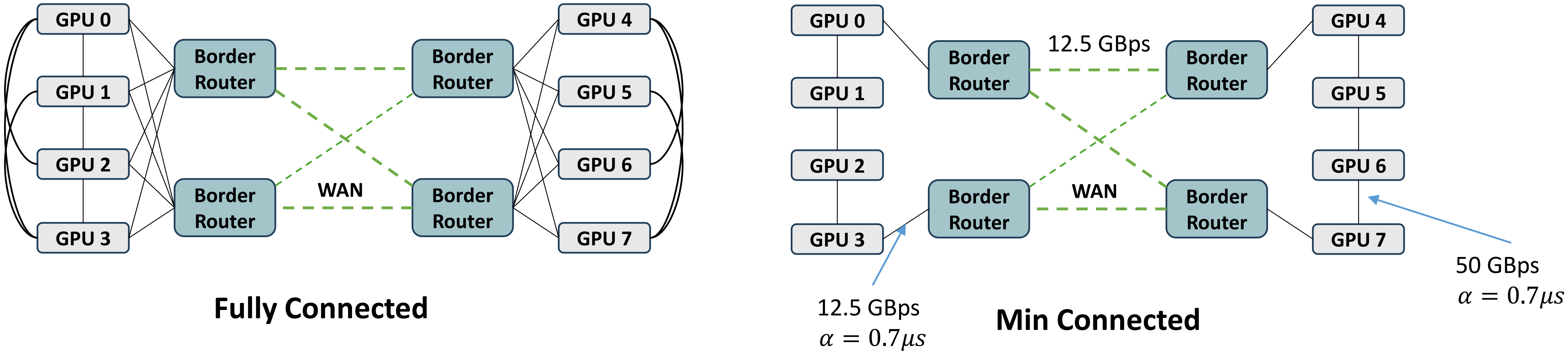
GPU to GPU



(a) Before subchunk sorting (b) After subchunk sorting

- For GPU to GPU, Subchunk sorting only performs the function of preventing redundant sending (Sending several same subchunks to the same receiver)

Numerical Result: Topology



1. Test topologies with different internal connectivities within the data center (**connectivity = number of edges in the current topology / number of edges in a fully connected graph**): randomly vary the number of internal edges in each DC, ensuring that each DC remains a connected graph
2. Vary the propagation delay of the links in the WAN

Numerical Result: Scalability Analysis

POLITECNICO MILANO 1863

Table 1: Execution Time of Different Algorithm with 150km WAN Delay (Connectivity = 0.5)

Unit: s	SCALE-CCL				TE-CCL				Shortest Path				NCCL			
Num. Subchunks	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4	8
1MB	0.04	0.08	0.22	0.75	4.61	35.24	1348.71	x	0.08	0.18	0.39	0.95	0.054	0.104	0.233	0.575
4MB	0.04	0.09	0.22	0.76	2.34	23.45	776.06	x	0.08	0.17	0.38	0.96	0.052	0.107	0.240	0.610
16MB	0.04	0.09	0.29	0.77	1.96	22.75	705.71	x	0.08	0.17	0.36	0.81	0.052	0.104	0.230	0.580
64MB	0.05	0.09	0.23	0.75	1.82	22.95	676.27	x	0.08	0.17	0.34	0.79	0.052	0.101	0.221	0.546
256MB	0.04	0.09	0.22	0.76	2.01	17.22	707.71	x	0.08	0.17	0.32	0.77	0.052	0.100	0.217	0.537

SCALE-CCL

- Scheduling time stays below **1 s** in all cases (e.g., 1 MB: 0.04 → 0.75 s)
- Grows **smoothly and near-linearly** with the number of subchunks

TE-CCL

- At 1 subchunk, takes **2–5 s** depending on chunk size
- At 4 subchunks jumps to **600–1300 s**, and fails entirely at 8 subchunks (more than 1 hour)

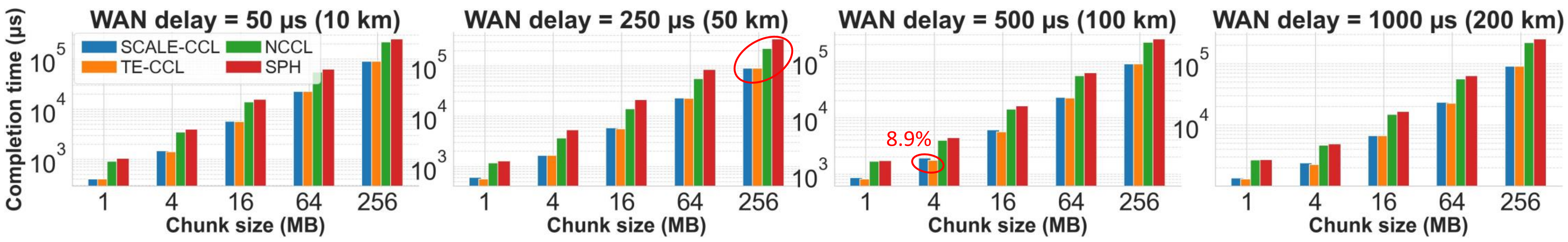
Shortest Path (SPH)

- Runs in the **0.3–1.0 s** range across all settings
- Consistently **slower than SCALE-CCL** (e.g., 1 MB / 8 subchunks: 0.95 s vs. 0.75 s)

NCCL

- Largely insensitive to **chunk size** (1–256 MB shows small variation)
- Increases with subchunks: from **≈0.05 s** (1 subchunk) to **≈0.55–0.61 s** (8 subchunks)

Numerical Result: Different WAN Delay



Completion Time vs. chunk size under different WAN delay with 0.5 connectivity and 1 Subchunk per GPU

SCALE-CCL

- Stays within **10%** of TE-CCL across all WAN delays and chunk sizes (max gap ≈ **8.9%**, avg ≈ **3%**)
- At 250 μs and 256 MB: **90,252 μs**, only **0.3%** slower than TE-CCL (90,002 μs)

TE-CCL

- Achieves the **lowest completion time**, but only slightly better than SCALE-CCL (sub-10% gap everywhere)
- Completion time increases with WAN delay similarly to SCALE-CCL, but requires much higher scheduling overhead (from Table 1)

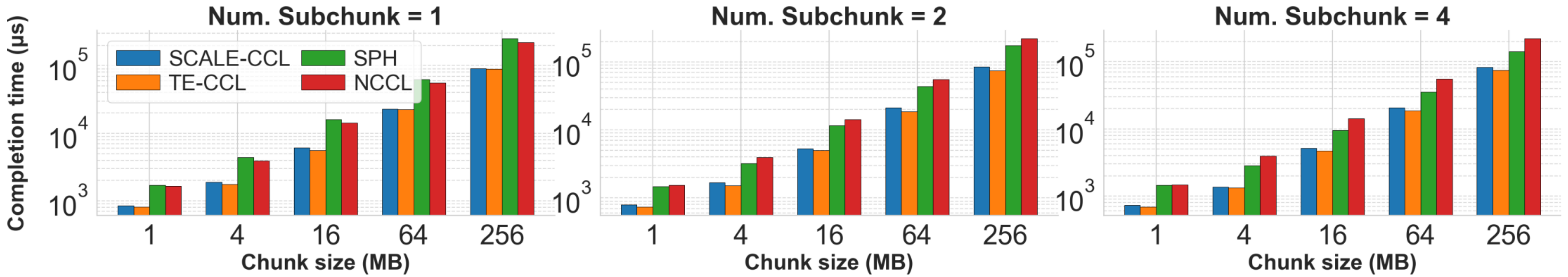
Shortest Path (SPH)

- Significantly slower than WAN-aware schemes
- Curves grow more steeply with WAN delay, showing poor robustness to higher propagation latency

NCCL

- Also degrades under WAN delay
- Lacks WAN-aware path selection, so its completion time tracks propagation delay more closely

Numerical Result: Different Subchunks



Completion Time vs. chunk size under different number of Subchunks with 100 km WAN Delay and 0.5 connectivity

SCALE-CCL

- Within **10–15%** of TE-CCL across all chunk sizes and subchunks
- More subchunks reduce completion time (e.g., at 16 MB & 2 subchunks: **5313 μs**)

TE-CCL

- Best completion time in all settings
- Also benefits from subchunking but with high synthesis cost (per Table 1)

SPH

- Much slower than SCALE-CCL (e.g., 16 MB & 2 subchunks: **11,566 μs**)
- Improves with more subchunks but still far behind WAN-aware methods

NCCL

- At 16 MB & 2 subchunks: **14,251 μs** (≈**2.7×** slower than SCALE-CCL)
- Weak sensitivity to subchunks for medium/large chunks due to fixed ring structure

Conclusion

- We presented **SCALE-CCL**, a WAN-aware collective communication library that synthesizes AllGather schedules across geo-distributed datacenters
- By combining **subchunking**, **local queue–based decisions**, and a **lightweight event-driven scheduler**, SCALE-CCL reacts to WAN variability while keeping schedule generation extremely fast
- Experiments show that SCALE-CCL maintains **near-optimal completion time** while scaling to large topologies and high subchunk counts, making WAN-aware collective scheduling practical for modern cross-DC training