

Towards High-Throughput Collective Communication via Harnessing PCIe and Dedicated Interconnects

Yuezheng Liu
Beihang University
Beijing, China
liuyuezheng@buaa.edu.cn

Menghao Zhang
Beihang University
Beijing, China
zhangmenghao@buaa.edu.cn

Xuebin Song
Beihang University
Beijing, China
soybean@buaa.edu.cn

Juner Shen
Beihang University
Beijing, China
shenje@buaa.edu.cn

Chunming Hu
Beihang University
Beijing, China
hucm@buaa.edu.cn

Xudong Liu
Beihang University
Beijing, China
liuxd@buaa.edu.cn

Abstract

Collective communication libraries (CCLs) for distributed LLM training focus mainly on dedicated high-speed interconnects and leave PCIe bandwidth under-exploited. For PCIe-only machines, they model collectives as GPU-to-GPU communication and create redundant traffic in the CPU-rooted PCIe hierarchy. For machines with dedicated interconnects, they either ignore PCIe or use a static multipath split that cannot react to runtime PCIe interference. This paper presents HyLink, a two-layer collective communication framework that unlocks PCIe bandwidth and aggregates it with dedicated interconnects. PCIe CCL makes this hierarchy explicit and uses host memory as an active routing point. It expresses topology-aware schedules in a CPU-driven DSL and executes them with an auto-pipelined runtime on GPU copy engines. Multipath Orchestrator splits each collective between the dedicated link and PCIe, and uses runtime feedback to rebalance the split under PCIe interference. In our evaluation across different PCIe-only topologies, PCIe CCL improves large-message collective bandwidth by 26.2% on average over NCCL and reduces Megatron-LM iteration time by 6.1%–6.6%. On the Ascend platform with dedicated interconnects, Multipath Orchestrator improves large-message collective bandwidth by 10.4% on average over HCCL and reduces Megatron-LM iteration time by 2.3%–4.6%.

CCS Concepts: • **Computer systems organization** → **Interconnection architectures**; • **Software and its engineering** → *Cooperating communicating processes*; • **Computing methodologies** → *Parallel algorithms*.

Keywords: Collective communication, PCIe, Multipath communication, Distributed LLM training

ACM Reference Format:

Yuezheng Liu, Menghao Zhang, Xuebin Song, Juner Shen, Chunming Hu, and Xudong Liu. 2026. Towards High-Throughput Collective Communication via Harnessing PCIe and Dedicated Interconnects. In *The 2026 ACM SIGOPS Annual Technical Conference (SIGOPS ATC 2026)*, November 15–18, 2026, Hong Kong, China. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3828523.3857136>

1 Introduction

The rapid growth of large language models (LLMs) has made distributed training a communication-heavy workload [38]. Modern training systems combine several forms of parallelism to train models that exceed the capacity of a single GPU [20]. These forms include Data Parallelism (DP), Pipeline Parallelism (PP), Tensor Parallelism (TP), and Expert Parallelism (EP). They repeatedly invoke collective communication primitives such as AllReduce, AllGather, ReduceScatter, and All-to-All. These collectives synchronize model states, exchange activations and gradients, and redistribute expert-routed tokens. Some of them move large tensors and sit on the critical path of each training iteration. As a result, communication has become one of the dominant bottlenecks in large-scale training. For example, Meta’s production measurements report that communication can account for up to 60% of total training time [33].

Many efforts have been made to reduce this bottleneck. Hardware vendors provide optimized collective communication libraries (CCLs). Examples include NCCL, RCCL, and HCCL [1, 9, 21]. Research systems also improve the communication stack by optimizing routing and synthesizing collective schedules [2, 3, 8, 19, 27, 35]. However, much of this progress targets GPU-to-GPU communication over dedicated high-speed interconnects such as NVLink, NVSwitch, xGMI [16], and HCCS [10]. While this focus is reasonable given the higher bandwidth of dedicated interconnects, it largely overlooks the PCIe fabric that every GPU node contains.



This work is licensed under a Creative Commons Attribution 4.0 International License.

SIGOPS ATC 2026, Hong Kong, China

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2818-1/2026/11

<https://doi.org/10.1145/3828523.3857136>

Fully exploiting PCIe bandwidth matters in two broad deployment classes. First, PCIe-only GPU servers remain common in industrial and academic environments, especially when deployment cost is a primary constraint. Cluster traces show that heterogeneous and legacy GPUs remain widely deployed in production ML platforms, many without a dedicated GPU interconnect [34]. For these servers, PCIe is the only path for collective communication, so its bandwidth directly determines collective performance. Second, high-end GPU nodes usually run collectives over dedicated interconnects. However, these nodes also contain a PCIe path that connects each GPU to the host CPU and I/O devices. The raw bandwidth of this path is not negligible. As Table 1 shows, per-GPU PCIe bandwidth can reach tens to more than one hundred GB/s on recent GPUs. This bandwidth can provide a meaningful supplement to the dedicated-link bandwidth. Thus, fully exploiting PCIe bandwidth can accelerate collectives in both PCIe-only and hybrid-link machines. However, current CCLs cannot fully exploit PCIe bandwidth because of two design limitations described below.

(1) PCIe topology mismatch. Existing CCLs still model PCIe collectives as communication between GPU pairs. This GPU-to-GPU abstraction is natural for dedicated link fabrics because they provide uniform connectivity between any pair of GPUs. However, PCIe connectivity between GPUs is non-uniform because of its hierarchical structure. In a typical multi-socket server, GPUs connect to PCIe switches or directly to CPU root complexes, and cross-NUMA interconnects such as Intel UPI [12] and AMD Infinity Fabric [25] link the root complexes on different sockets. On PCIe, the paths between different GPU pairs often share the same root complexes, cross-NUMA links, and host memory, which can cause redundant traffic in collectives. Since the GPU-to-GPU abstraction does not expose this sharing, current CCLs cannot detect or reduce the redundancy.

(2) PCIe aggregation gap. Existing CCLs also largely ignore the PCIe path on nodes with dedicated interconnects. In these nodes, a collective communication typically runs only over the dedicated link. The PCIe path remains idle even though it can provide supplemental bandwidth. Prior work such as Blink [31] shows the potential of combining paths. However, Blink exposes PCIe by disabling GPU peer access, and the resulting switching overhead can offset most of the benefit from using the PCIe path. It also uses a static split between paths and cannot react to runtime PCIe interference from background DMA, asynchronous checkpointing, log I/O, or co-located jobs. Once the split becomes imbalanced, multipath execution is gated by the slower path. FlexLink [29] adaptively balances traffic across NVLink, PCIe, and RDMA paths. However, its PCIe path follows a ring-based GPU-to-GPU schedule, which can retain the topology mismatch described above on hierarchical PCIe fabrics. Recent work, MPCCS [37], aggregates the dedicated link with the scale-out NIC fabric rather than the intra-node PCIe tree.

Its applicability is limited when scale-out bandwidth or NIC resources are insufficient.

To address these limitations, this paper presents HyLink, a two-layer collective communication framework with PCIe CCL and Multipath Orchestrator. HyLink unlocks PCIe bandwidth and aggregates it with dedicated interconnect bandwidth. Its two layers target the two limitations above. PCIe CCL makes the shared PCIe resources explicit in the collective abstraction instead of hiding them behind GPU pairs. It treats NUMA-local host memory as an active routing point and schedules multi-stage data movement across GPU and host buffers. Multipath Orchestrator controls hybrid execution at runtime by splitting each collective into two byte ranges. It runs one range over the dedicated link through the vendor CCL and the other over PCIe through PCIe CCL. The two paths execute concurrently within the same collective call and preserve the synchronization semantics of a single collective operation.

PCIe CCL addresses the topology mismatch through a CPU-driven Domain-Specific Language (DSL) and an auto-pipelined runtime. The DSL represents a collective as chunk-level dataflow. The dataflow covers GPU buffers and host buffers in specific NUMA domains. An algorithm can specify where a chunk is uploaded, how it is forwarded between NUMA-local host buffers, and which GPUs later download it. When GPUs share a PCIe switch, the algorithm can also copy chunks directly between them. The DSL serves as a route-level abstraction that lets designers adapt routes to different PCIe topologies without enumerating every small transfer. The runtime then lowers each chunk-level instruction into fine-grained subchunk transfers to pipeline dependent stages along each route. It issues these transfers to GPU copy engines, and CPU threads advance the pipeline through the event mechanism. Another benefit of this design is that GPU SMs stay out of the transport path, which reduces SM contention with training computation. Data-movement collectives such as AllGather run entirely on copy engines without any GPU kernel. Reduction collectives such as ReduceScatter and AllReduce first move peer chunks to the local GPU through copy engines and then launch short kernels only for local reduction.

Multipath Orchestrator addresses the hybrid-link problem by treating the dedicated interconnect and PCIe as two independent communication domains. Multipath Orchestrator intercepts the collective call and splits the tensor according to a runtime ratio. It launches the vendor CCL and PCIe CCL on independent streams and then joins their completion before releasing the user stream. Multipath Orchestrator records the execution time of both paths in each collective. A feedback-driven adaptive balancer then uses these measurements to update the split ratio for the next collective. If the PCIe path becomes slower due to interference, Multipath Orchestrator shifts traffic back to the dedicated link. When the PCIe path recovers, Multipath Orchestrator increases

the PCIe share again. This control loop keeps the two paths finishing at roughly the same time, since the slower path determines the completion time of the whole collective. For small messages, PCIe startup overhead can dominate the latency of the PCIe path. Multipath Orchestrator therefore falls back to the dedicated link for such messages.

HyLink is designed to work with existing training frameworks and the semantics of current collective communication interfaces. Because CPU threads and copy engines drive the transport path of PCIe CCL, it must handle two integration issues. (1) *Stream ordering*. A framework expects a collective to behave like any other operation on its stream. PCIe CCL inserts a stream write that marks when prior stream work has completed and a stream wait that holds later stream work until all of its transfers finish. (2) *Deadlock avoidance*. A framework may call a host-blocking CUDA API such as a synchronous device-to-host copy right after launching a collective. This call can hold the CUDA driver lock while waiting for the collective, and PCIe CCL needs the same lock to submit its remaining transfers. PCIe CCL breaks this circular wait by draining in-flight work outside the driver lock before such calls proceed.

We implement an open-source prototype of HyLink [11]. We evaluate HyLink with collective microbenchmarks and end-to-end Megatron-LM training. Across different PCIe-only topologies, PCIe CCL improves large-message collective bandwidth by 26.2% on average over NCCL and reduces end-to-end iteration time by 6.1%–6.6% in Megatron-LM training with intra-node TP. On the Ascend platform with dedicated interconnects, Multipath Orchestrator delivers 10.4% higher collective bandwidth on average than native HCCL in large-message microbenchmarks and reduces end-to-end iteration time by 2.3%–4.6% in Megatron-LM training. Mechanism studies further explain HyLink’s performance gains and robustness in PCIe-only and hybrid-link deployments. For the PCIe-only platform, NUMA-aware host staging reduces per-GPU D2H traffic and total cross-NUMA traffic by 33.3% and improves AllGather bandwidth by 9.8% over Ring. Subchunk pipelining improves AllGather bandwidth by 11.4% on average over the non-pipelined baseline. PCIe CCL’s copy-engine transport limits GEMM throughput degradation to at most 2.4% compared with 9.4% under NCCL during concurrent execution. For the hybrid-link platform, Multipath Orchestrator achieves 30.9% higher AllGather throughput under runtime PCIe interference than a fixed split tuned without interference.

This paper makes the following contributions:

- We identify PCIe bandwidth as an underexploited resource in both PCIe-only and hybrid-link GPU machines. We show that the GPU-to-GPU abstraction causes redundant traffic in PCIe collectives and that current CCLs lack efficient aggregation of PCIe and dedicated links.
- We design PCIe CCL, a PCIe-aware collective library that treats NUMA-local host memory as an active routing point. PCIe CCL expresses topology-specific routes through a route-level DSL and executes them through an automatically pipelined runtime based on copy engines.
- We design Multipath Orchestrator, a multipath orchestration layer that aggregates dedicated-link and PCIe bandwidth without modifying the vendor CCL. Multipath Orchestrator uses transparent task splitting, fork-join coordination, and adaptive load balancing with runtime feedback.
- We implement HyLink and show that it outperforms vendor CCLs on PCIe-only and hybrid-link platforms and remains robust under runtime PCIe interference.

2 Background and Motivation

2.1 Collectives in Distributed LLM Training

During distributed training, collective communication synchronizes tensors across GPUs. In AllReduce, each rank contributes one buffer and receives the element-wise reduction over all ranks. AllGather concatenates the buffers from all ranks and returns the full result to every rank. All-to-All exchanges distinct slices between every source-destination pair. Point-to-point send/recv transfers data between a specified pair of ranks. Modern collective communication libraries (CCLs) implement these primitives with algorithms such as Ring [4] and Tree [14] schedules.

Modern LLM training invokes these primitives in different parallelism strategies [20]. Data Parallelism (DP) mainly synchronizes gradients after local computation. The messages are large but can be scheduled as gradient buckets and overlap well with the remaining backward computation. Pipeline Parallelism (PP) transfers activations and gradients between adjacent stages, and its point-to-point communication can be partially hidden by micro-batch pipelining. Tensor Parallelism (TP) is typically confined within a node and inserts synchronization inside every transformer layer. Subsequent matrix multiplications depend on the collective results, leaving little room for overlap. Expert Parallelism (EP) redistributes tokens through bandwidth-bound All-to-All, which sits between token dispatch and expert computation and therefore directly stalls expert kernels. Because TP and EP communication is difficult to hide, these collectives place stringent bandwidth and latency demands on the interconnect and are commonly constrained to the intra-node scale-up domain. Measurements of Megatron-LM TP training on GPT-3 and Llama-2 models report that communication takes 22–47% of end-to-end iteration time, even on DGX-H100 clusters with NVLink/NVSwitch and InfiniBand [32]. These exposed communication costs motivate optimizing the intra-node collective tasks.

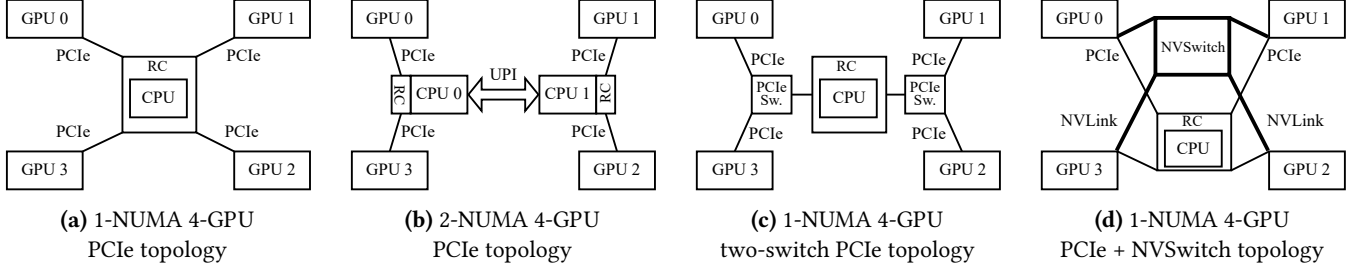


Figure 1. Representative physical topologies targeted by this work. Panels (a)–(c) are pure-PCIe deployments with different NUMA and PCIe-switch structures. Panel (d) is a hybrid deployment with both a dedicated NVSwitch fabric and a PCIe tree.

2.2 PCIe and Hybrid Interconnect Architectures

Dedicated GPU interconnects, such as NVIDIA NVLink and NVSwitch, AMD xGMI [16], and Huawei HCCS [10], directly connect GPUs through a full-mesh or switched fabric. The PCIe fabric that coexists with these dedicated links has a fundamentally different structure. PCIe in multi-GPU nodes forms a hierarchical topology centered around the CPU [28]. Each socket exposes a root complex to which GPUs and NICs attach, either directly or through an optional PCIe switch. Multiple sockets are joined by a cross-NUMA link such as Intel UPI [12] or AMD Infinity Fabric [25]. Across deployments, this hierarchical PCIe fabric is either the primary intra-node communication path or a supplementary path beside the dedicated fabric, which makes it an important optimization target that current CCLs do not fully exploit.

Two deployment classes motivate this work.

(1) **Cost-efficient nodes:** widely deployed on consumer-grade hardware and legacy machines that remain in active service, with no dedicated interconnect. Every GPU attaches to the root complex, and optional PCIe switches may connect GPUs with each other (Figure 1(a)–(c)). In this setting, PCIe directly determines intra-node collective performance. Existing GPU-centric CCLs [2, 3, 8, 15, 19, 21, 27] leave topology-level opportunities on PCIe unexploited (§2.3).

(2) **High-end nodes:** equipped with a dedicated interconnect fabric in addition to PCIe (Figure 1(d)). In this setting, PCIe provides a parallel data path alongside the dedicated link. As Table 1 shows, PCIe contributes a substantial fraction of the dedicated-link bandwidth across vendors and hardware generations. PCIe adds 14% on top of NVLink even on H100 SXM and reaches 32% on the export-tier H800 SXM.

2.3 Optimization Opportunities for Collectives

The hierarchical PCIe topology and the PCIe bandwidth that can supplement the dedicated interconnect on hybrid nodes give rise to two distinct optimization opportunities: topology-level CPU participation and link-level hybrid execution. We examine each opportunity below.

Topology-level: incorporating the CPU into the collective abstraction. Current CCLs such as NCCL [21] and MSCCL [8], collective synthesizers [2, 3, 19, 27], as well as

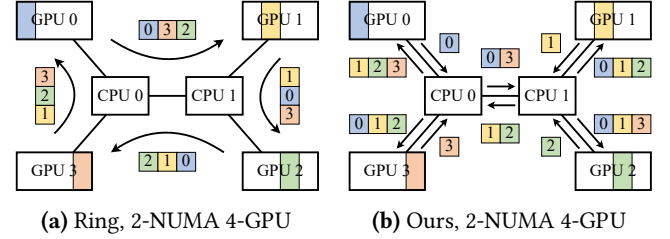


Figure 2. Optimization space on the PCIe topology for AllGather. Panel (a) illustrates the data-transfer pattern of a Ring AllGather on a 2-NUMA 4-GPU node. Panel (b) illustrates the corresponding host-staged variant. By making the CPU and host memory explicit participants, our design can reduce redundant transfers on the PCIe fabric and avoid unnecessary cross-NUMA traffic.

TCCL [15] for optimizing collective communication on PCIe GPU nodes, express collective communication as channels between GPU ranks. This GPU-to-GPU abstraction does not model the CPU or the PCIe hierarchy as explicit topology levels. On hierarchical PCIe topologies, this flat abstraction may map multiple logical GPU-to-GPU transfers onto the same shared PCIe resources, including root complexes and cross-NUMA links. The resulting schedules can create redundant PCIe and UPI traffic, as illustrated in Figure 2(a). A PCIe-aware CCL should instead make the CPU and host DRAM explicit participants in the collective data path. For example, AllGather can stage data in CPU memory to avoid unnecessary cross-NUMA transfers and reduce redundant data movement, as illustrated in Figure 2(b).

This topology-aware host-staging path addresses two inefficiencies in the PCIe topology. First, host DRAM is effectively half-duplex because device-to-host uploads and host-to-device downloads consume the same memory-bandwidth budget. Reducing either upload or download traffic can therefore improve performance when host DRAM is on the critical path. Second, topology-aware routing can reduce cross-NUMA traffic by avoiding repeated transfers of the same data over these links. Instead of repeatedly carrying the same chunk across UPI or Infinity Fabric as the Ring advances, the

Table 1. Per-GPU bi-directional bandwidth of the dedicated link (B_d) and PCIe (B_p) across modern GPUs and hardware generations. B_p/B_d is the per-GPU ratio of PCIe to dedicated-link bandwidth.

GPU	B_d	B_p	B_p/B_d
NVIDIA V100 32G SXM	300 GB/s	32 GB/s	10.7%
NVIDIA A100 80G SXM	600 GB/s	64 GB/s	10.7%
NVIDIA A800 40G SXM	400 GB/s	64 GB/s	16.0%
NVIDIA H100 SXM	900 GB/s	128 GB/s	14.2%
NVIDIA H800 SXM	400 GB/s	128 GB/s	32.0%
AMD Instinct MI300X	896 GB/s	128 GB/s	14.3%
Huawei Ascend 910B	392 GB/s	64 GB/s	16.3%

host-staged route keeps local distribution within a NUMA domain and only uses cross-NUMA links when the data must cross the socket boundary.

Link-level: unused PCIe bandwidth. Table 1 sets the theoretical ceiling on the PCIe bandwidth that hybrid execution can exploit. Across modern GPUs and hardware generations, the per-GPU PCIe bandwidth remains a substantial fraction of the dedicated-link bandwidth. To determine whether this bandwidth is actually available to hybrid execution, we identify intra-node collective windows in Megatron-LM training with Nsight Systems and sample PCIe throughput using `nvmlDeviceGetPcieThroughput`, which reports PCIe throughput over short intervals. In our training run without asynchronous checkpointing or co-located jobs, the PCIe bandwidth observed within intra-node collective windows stays below 1% of the PCIe peak, leaving most of the PCIe path idle while scale-up collective communication is active.

This headroom indicates that hybrid execution can use the idle PCIe path without delaying the training operations observed in this run. However, non-bottleneck asynchronous work and external background activity, such as asynchronous checkpointing, log I/O, and co-located tasks, can still introduce bursty PCIe traffic. A static traffic split cannot reduce PCIe-side work during such bursts, slowing the overall multipath completion time. Hybrid execution therefore requires dynamic traffic balancing.

Existing multipath collective systems demonstrate the value of using secondary communication paths, but their designs still have practical limitations. Blink [31] attempts to combine PCIe with the dedicated GPU interconnect. It uses GPU P2P transfers to move chunks along its transfer trees. Because the GPU driver routes P2P traffic over NVLink when peer access is enabled, Blink exposes the PCIe path by disabling GPU peer access. This design has two key limitations. First, enabling and disabling peer access adds switching cost that grows with the number of GPUs, reducing the benefit of hybrid execution. In Blink’s hybrid NVLink+PCIe experiment, PCIe adds only 2–5 GB/s over NVLink-only collectives, far below the tens of GB/s available on a PCIe Gen3 or Gen4

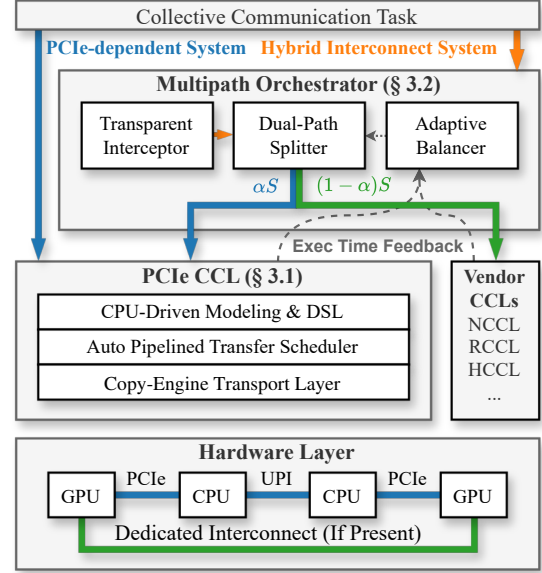


Figure 3. HyLink architecture overview.

x16 path. Second, Blink uses a static split that is fixed after measurements in the initial calls, but does not account for runtime PCIe bursts.

Recent work MPCCS [37] supplements the dedicated link with the scale-out NIC fabric rather than the intra-node PCIe tree. It targets GPU-cloud servers where the available scale-out bandwidth per GPU can cover the GPU’s PCIe bandwidth and scale-out networks are often underutilized. In contrast, our work uses the CPU and host memory as the downstream PCIe consumer and does not depend on scale-out network resources. The two approaches are complementary across deployments: host-memory paths can help when NIC count or scale-out bandwidth is limited, while NIC paths can help when host-side PCIe resources are the bottleneck.

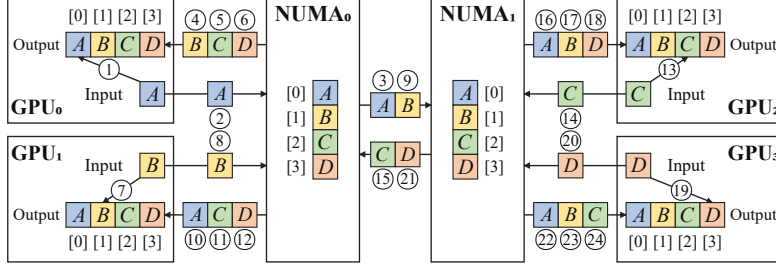
Summary and design requirements. The observations above point to two design requirements. First, a PCIe-aware CCL must expose the physical PCIe hierarchy rather than hide it behind a flat GPU-to-GPU abstraction. It should model root complexes, NUMA domains, and cross-NUMA links, and make CPU memory a schedulable staging point for collective data movement. Second, hybrid collective execution must use the PCIe fabric effectively and robustly alongside the dedicated interconnect, which requires low-overhead multipath coordination and runtime traffic balancing.

3 Design

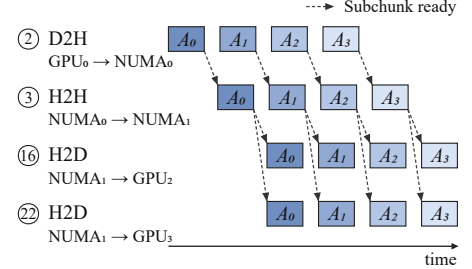
To address the two fundamental issues mentioned above, we propose HyLink. HyLink unlocks the underutilized PCIe bandwidth for collective communication and complements dedicated interconnects with PCIe bandwidth to achieve hybrid-link acceleration.

GPU ₀ NUMA ₀	Input: Chunk A	GPU ₁ NUMA ₀	Input: Chunk B	GPU ₂ NUMA ₁	Input: Chunk C	GPU ₃ NUMA ₁	Input: Chunk D
① D2DCopy (Input, Output[0])		⑦ D2DCopy (Input, Output[1])		⑬ D2DCopy (Input, Output[2])		⑲ D2DCopy (Input, Output[3])	
② D2H (Input, NUMA ₀ [0])		⑧ D2H (Input, NUMA ₀ [1])		⑭ D2H (Input, NUMA ₁ [2])		⑳ D2H (Input, NUMA ₁ [3])	
③ H2H (NUMA ₀ [0], NUMA ₁ [0])		⑨ H2H (NUMA ₀ [1], NUMA ₁ [1])		⑮ H2H (NUMA ₁ [2], NUMA ₀ [2])		㉑ H2H (NUMA ₁ [3], NUMA ₀ [3])	
④ H2D (NUMA ₀ [1], Output[1])		⑩ H2D (NUMA ₀ [0], Output[0])		⑯ H2D (NUMA ₁ [0], Output[0])		㉒ H2D (NUMA ₁ [0], Output[0])	
⑤ H2D (NUMA ₀ [2], Output[2])		⑪ H2D (NUMA ₀ [2], Output[2])		⑰ H2D (NUMA ₁ [1], Output[1])		㉓ H2D (NUMA ₁ [1], Output[1])	
⑥ H2D (NUMA ₀ [3], Output[3])		⑫ H2D (NUMA ₀ [3], Output[3])		⑱ H2D (NUMA ₁ [3], Output[3])		㉔ H2D (NUMA ₁ [2], Output[2])	

(a) Per-GPU DSL for AllGather (2 NUMA nodes, 4 GPUs).



(b) Full topology and data movement.



(c) Automatic pipelining of chunk A (4 subchunks).

Figure 4. Overview of PCIe CCL DSL-based AllGather on four GPUs across two NUMA nodes.

HyLink employs a two-layer co-design architecture:

- **PCIe CCL:** a collective communication library that breaks the flat GPU-to-GPU abstraction and accelerates PCIe collective communication through hierarchical modeling.
- **Multipath Orchestrator:** a scheduling framework that performs runtime feedback-driven dynamic load balancing between dedicated interconnects and PCIe.

The end-to-end workflow of HyLink operates as follows (Figure 3). In PCIe-only deployments, HyLink uses PCIe CCL to execute collective calls over the PCIe hierarchy. In hybrid deployments, HyLink uses Multipath Orchestrator to split each collective into two parallel sub-tasks based on a runtime splitting ratio. One sub-task is executed on the dedicated link path by invoking vendor CCLs, while the other is executed on the PCIe path via PCIe CCL. These two paths execute concurrently on independent device streams and perform a final synchronization upon completion.

3.1 PCIe CCL

To address the logical-to-physical topology mismatch in existing CCLs, PCIe CCL combines a topology-aware algorithm expression with an efficient execution runtime for PCIe collectives. It models a collective as multi-stage dataflow in which logical chunks move between GPU buffers and NUMA-aware host buffers, allowing host memory to serve as an explicit routing point in the PCIe hierarchy. The DSL specifies where chunks move, while the runtime executes this dataflow as a fine-grained transfer pipeline and orders transfers according to the corresponding data dependencies. The transfer pipeline is driven by copy engines and CPU threads

Table 2. Syntax of a PCIe CCL DSL instruction.

Instruction ::= Instr(rank, op, src, dst)
op ::= D2H H2D H2H D2DCopy D2DReduce
src, dst ::= GPU(rank, space, chunk) Host(numa, chunk)
space ::= input output scratch

to saturate PCIe transfer bandwidth while minimizing SM usage during collective communication.

Topology-Aware Collective DSL. Vendor CCLs commonly instantiate collective schedules from a few predefined algorithm templates. A template such as Ring [4] is easy to extend because ranks can always be connected into a cycle of communication channels regardless of scale. Such topology-agnostic templates fit poorly with diverse and hierarchical PCIe topologies. PCIe collectives therefore need an algorithm representation that can be tailored to each topology and that explicitly exposes the PCIe hierarchy and host memory. MSCCLang [8] shows that a collective DSL can avoid restricting algorithms to a small set of predefined templates by letting users describe chunk-level schedules. This DSL idea is useful for topology adaptation, but MSCCLang is still designed around GPU channels and primarily treats the topology as GPU-to-GPU edges. Its algorithmic abstraction does not model host memory as a first-class routing location, so it cannot naturally express PCIe-specific patterns such as placing one copy in a NUMA-local host buffer and letting multiple GPUs consume it from that buffer.

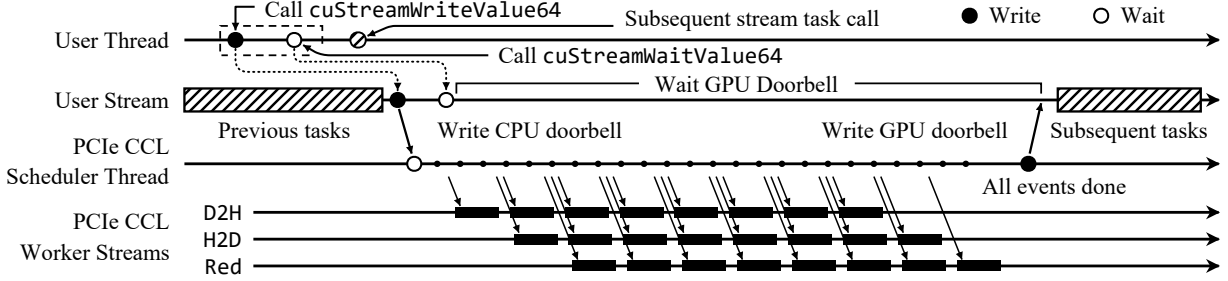


Figure 5. PCIe CCL stream-ordering synchronization.

PCIe CCL therefore adopts a human-in-the-loop DSL for PCIe collectives. Similar to TACCL’s communication-sketch philosophy [27], the algorithm designer specifies only the high-level route structure, namely where data is staged in host memory and which GPUs each NUMA domain delivers it to. Table 2 defines the instruction syntax used by the DSL. A collective algorithm is represented as a set of per-rank dataflow instructions. Each instruction moves one logical chunk between a source location and a destination location, which can be either GPU buffers or NUMA-local host buffers. These instructions expose the routing choices that matter for PCIe. D2H copies a chunk from a GPU to a specific NUMA domain, and H2D copies a chunk from a NUMA domain to a GPU. H2H relays a chunk between NUMA domains. D2DCopy copies a chunk either within one GPU or directly between two GPUs that share a PCIe switch with peer access. D2DReduce reduces one chunk into another on the same GPU. Designers can use these instructions to implement all common collective communication primitives as DSL programs.

Figure 4(a) illustrates this syntax with a 4-GPU AllGather across two NUMA nodes. Each rank contributes one chunk and receives all four chunks in its output buffer. The input chunk is written to the corresponding output position with D2DCopy, while the same chunk is uploaded to a NUMA-local host buffer through D2H and forwarded to the other NUMA node through H2H. The peer output chunks are then fetched from local host memory with H2D. Figure 4(b) maps these instructions onto the PCIe topology and shows the explicit host-buffer route.

Automatic Dataflow Pipelining. PCIe CCL intentionally keeps the DSL at chunk granularity so that algorithms remain readable and easy to adapt to different PCIe topologies. Executing one whole chunk at a time leaves downstream links idle until the upstream transfer finishes, while encoding subchunk-level steps directly in the DSL makes algorithms difficult to maintain. PCIe CCL therefore pipelines transfers automatically. The runtime splits each logical chunk into subchunks and lets a dependent transfer start as soon as the subchunk it needs is ready. The subchunk size is tuned for

different message sizes to balance pipeline overlap against per-transfer overhead.

The host-buffer route also determines the dependencies between transfers. Each H2H consumes a chunk produced by D2H, while each H2D waits for the D2H or H2H that produces its source chunk. The local D2DCopy instructions only depend on the corresponding rank’s input chunk and can proceed independently of the host-memory relay.

Figure 4(c) shows how these dependencies become synchronization signals for chunk A. The runtime splits A into fixed-size subchunks and keeps one monotonic progress counter in shared memory for each chunk slot in every buffer. When the D2H on GPU₀ finishes a subchunk, it advances the counter of NUMA₀[0]. The H2H checks this counter before relaying the same subchunk to NUMA₁[0] and then advances the counter of NUMA₁[0]. The H2D transfers on GPU₂ and GPU₃ check the counter of NUMA₁[0] before copying the subchunk into their output buffers.

Copy-Engine Transport Layer. Traditional GPU CCLs commonly use resident CUDA kernels to move data, perform reductions, and control communication pipelines [8, 21]. This design is effective for GPU-channel collectives, but these long-lived communication kernels rely on SM resources to make progress. When a collective overlaps with compute-intensive application kernels, SM contention can slow the communication path itself and reduce the benefit of overlapping communication with computation in modern training workloads (evaluated in §4.3).

Our insight is that PCIe data movement does not require SM-resident transport workers. GPU copy engines naturally handle bulk transfers that involve a GPU and can saturate PCIe links at the subchunk size. PCIe CCL therefore issues D2H, H2D, and D2DCopy subchunk transfers on separate streams using `cudaMemcpyAsync`, including peer copies between GPUs under the same PCIe switch. After each transfer is submitted, PCIe CCL records a CUDA event and polls these events to advance pipeline progress as transfers complete. H2H relays do not involve the GPU, so CPU-side runtime threads perform them with multi-threaded vectorized `memcpy` and advance the destination counter after each subchunk. For pure data-movement collectives such as AllGather,

this design completes the PCIe path without occupying GPU SMs. For reduction collectives such as ReduceScatter and AllReduce, PCIe CCL launches a short-lived reduction kernel for each local D2DReduce once its inputs are ready, so SM usage is limited to the local reduction work rather than the transport pipeline.

A collective must also preserve user-stream ordering. It must not read the input before earlier work in the stream has produced it, and later work in the same stream must not consume the output before the collective completes. Current CCLs with resident communication kernels satisfy this requirement naturally because the kernels are enqueued as stream work. In contrast, without such a resident kernel, PCIe CCL preserves the same ordering by using `cuStreamWriteValue64` and `cuStreamWaitValue64` on host-mapped variables, as shown in Figure 5. The user stream first writes a ready value, which makes the CPU-side runtime issue transfers only after prior user-stream work has completed. The user stream then waits on a completion value. After the CPU-side runtime and copy engines finish the collective transfers, the CPU updates the host-mapped completion value and releases the user stream with the same ordering semantics as an enqueued kernel.

This copy-engine-based execution model also needs to co-exist with host-blocking CUDA API calls issued by training frameworks without creating deadlocks. After submitting an asynchronous collective, a training framework may issue a host-blocking CUDA call that is ordered after the collective. A pageable device-to-host copy on the same stream starts executing only after the collective completes, and a device-synchronizing allocation waits for all outstanding work on the device, including the collective. The calling thread stays inside the CUDA driver during this wait and holds the driver lock. However, the collective completes only after the PCIe CCL CPU-side runtime submits its remaining copy-engine transfers and publishes the completion value that releases the stream wait, and both steps require the same driver lock. The framework thread thus waits for the collective while the collective waits for a lock held by the framework thread. PCIe CCL breaks this circular wait by intercepting host-blocking CUDA calls in the compatibility layer. Before entering the real CUDA call, the framework thread first drains any in-flight PCIe CCL collective through a CPU-side condition variable. This wait does not enter the CUDA driver, so PCIe CCL can continue submitting transfers and complete the collective. The guard is applied only to synchronous CUDA API calls that can block while waiting for GPU work, while stream-ordered asynchronous calls remain unchanged. Because this guard only moves an existing host-side wait outside the CUDA driver, it changes neither the semantics of the CUDA call nor the ordering of GPU work, and therefore affects neither correctness nor training performance.

Algorithm 1 Multipath Adaptive Control for α

Require: Data size S , threshold $S_{\min}$, margin $\gamma \in (0, 1)$, damping factor $\lambda \in (0, 1)$ satisfying $\lambda(1 + \gamma) > 1$

```

1: if  $S < S_{\min}$  then                                ▶ Small-message fallback
2:   Use dedicated link only; return
3: end if
4:  $r \leftarrow t_d/t_p$ 
5: if  $r \leq 1 - \gamma$  then                                ▶ PCIe is bottleneck
6:    $\alpha \leftarrow \alpha/2$ 
7: else if  $r \geq 1 + \gamma$  then                            ▶ Dedicated link is bottleneck
8:    $\alpha \leftarrow \alpha \cdot r \cdot \lambda$ 
9: end if
10:  $\alpha \leftarrow \text{clamp}(\alpha, 0, 1)$ 

```

3.2 Multipath Orchestrator

Using idle PCIe bandwidth in hybrid nodes requires distributing collective data across the PCIe path and the dedicated link path while coordinating the two paths as one collective operation. Multipath Orchestrator exposes a single collective interface and runs the vendor CCL and PCIe CCL as two independent communication domains.

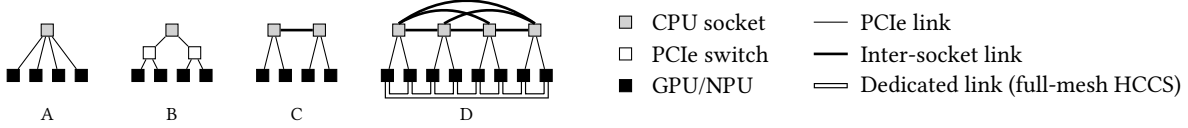
Multipath Task Splitting. In hybrid interconnect scenarios, Multipath Orchestrator splits each collective call into two parallel sub-tasks by partitioning the input tensor into two disjoint byte ranges. The *dedicated link path* invokes vendor CCLs, and the *PCIe path* invokes PCIe CCL. Each path executes the original collective semantics on its assigned byte range using its own optimized library. Multipath Orchestrator thus only decides how many bytes each path carries, while each library decides how to move them. The vendor CCL and PCIe CCL therefore keep their own algorithms, transport mechanisms, and stream semantics.

Multipath Orchestrator maintains a shared parameter table that maps collective communication metadata, including primitive type, data-size bucket, data type, and communicator size, to the current split ratio $\alpha \in [0, 1]$. All ranks use the same ratio to ensure global consistency. For each collective call with a total data size S , the splitter assigns αS to the PCIe path and $(1 - \alpha)S$ to the dedicated path. After both paths complete, Multipath Orchestrator measures their execution times t_d (dedicated) and t_p (PCIe) using the begin/end device events. Based on these measurements, the adaptive balancer updates α in the parameter table for subsequent calls, aiming to equalize t_d and t_p and minimize the overall latency $\max(t_d, t_p)$.

We adopt a proportional control strategy with a dead zone. Let $r = t_d/t_p$ denote the time ratio between the two paths. When r falls within a narrow band around unity ($1 - \gamma < r < 1 + \gamma$), the two paths are considered balanced and α is kept unchanged. When the PCIe path is the bottleneck ($r \leq 1 - \gamma$), α is halved to shift traffic back to the dedicated link. When the dedicated path is the bottleneck ($r \geq 1 + \gamma$), α is scaled

Table 3. Evaluation platforms and topology summary.

No.	GPUs	CPU (# NUMA)	Memory	PCIe configuration	PCIe P2P	Dedicated link
A	4× RTX 3090	1× AMD EPYC 7452	192 GB	Gen4 x8, No PCIe switch	Not supported	N/A
B	4× V100 PCIe	1× Intel Xeon E5-2680 v4	512 GB	Gen3 x16, Pairwise PCIe switches	Within NUMA	N/A
C	4× V100 PCIe	2× Intel Xeon E5-2650 v4	128 GB	Gen3 x16, No PCIe switch	Within NUMA	N/A
D	8× Ascend 910B	4× Kunpeng 920	2 TB	Gen4 x16, No PCIe switch	N/A	Full-mesh HCCS



proportionally to the observed imbalance with a damping factor $\lambda < 1$ to prevent overshooting. We choose a fixed $\lambda \in (0, 1)$ satisfying $\lambda(1 + \gamma) > 1$, ensuring $r\lambda > 1$ so that α can increase when the dedicated path is the bottleneck.

In practice, small-message collectives are dominated by startup latency. Splitting further reduces the message size on each path, increasing the share of startup overhead in communication time and potentially lowering the vendor CCL’s effective bandwidth. Therefore, we fall back to the dedicated path when the message size is small. For larger messages, startup overhead is amortized and splitting can improve performance by using the bandwidth of both paths. The complete adaptation algorithm is given in Algorithm 1. **Dual-Domain Coordination.** Multipath collective communication needs a coordination layer above individual communication libraries. In HyLink, the two domains are a vendor CCL over the dedicated interconnect and PCIe CCL over the PCIe hierarchy. Multipath Orchestrator coordinates their invocation at the collective-call level rather than modifying the internal transfer schedules of either library.

The dual paths are coordinated through an event-based fork-join model. Upon intercepting a collective call, Multipath Orchestrator records a synchronization event on the user stream. Both the dedicated link stream and the PCIe stream wait for this event before starting, ensuring input data readiness. The two paths then execute concurrently on independent streams, recording begin/end timing events for performance measurement. Finally, the user stream waits for the end events of both paths before proceeding. These events only express stream ordering and completion timing, leaving data movement to the underlying communication domains and adding little overhead to bandwidth-bound collectives.

4 Implementation and Evaluation

Our evaluation answers two questions: (1) how much does HyLink improve collective communication and end-to-end training, and (2) where do these gains come from? After describing the implementation and testbeds (§4.1), we answer the first question in §4.2 on both PCIe-only and hybrid-link platforms, and the second in §4.3 by validating routing,

pipelining, SM usage, and adaptive control under runtime PCIe interference.

4.1 Implementation and Testbeds

Implementation. We implement PCIe CCL and Multipath Orchestrator as two shared C++/CUDA libraries. The two layers are decoupled by design: PCIe CCL can be used independently in pure PCIe environments, while Multipath Orchestrator orchestrates dual-path parallelism in hybrid interconnect scenarios. The source code is publicly available [11].

Testbeds. Table 3 summarizes our evaluation platforms. We evaluate HyLink on both PCIe-only machines and a machine equipped with dedicated interconnects. Our PCIe-only platforms cover three topologies: all GPUs connected directly to a single CPU without PCIe P2P support (Platform A), GPU pairs sharing PCIe switches within one NUMA domain (Platform B), and GPUs attached to two CPU sockets in separate NUMA domains (Platform C). Our hybrid-link platform (Platform D) is an 8-NPU Ascend 910B server with a PCIe interconnect across four NUMA domains and a full-mesh HCCS dedicated interconnect. For the vendor CCL baselines, we use NCCL v2.30.3 with its built-in autotuning on Platforms A–C and HCCL from CANN 9.0.0 on Platform D.

4.2 Performance Acceleration

PCIe CCL Collective Microbenchmarks. We first evaluate whether PCIe CCL improves collective communication on PCIe-only platforms. We benchmark AllGather, ReduceScatter, and AllReduce across different message sizes, where each point reports the post-warmup average bus bandwidth across repeated iterations, and compare PCIe CCL with NCCL [21], synthesized MSCCL+SCCL schedules [2, 8], and TCCL [15]. Figure 6 reports the collective performance of PCIe CCL across Platforms A–C. The PCIe CCL schedules are hand-tuned DSL programs specialized to each PCIe topology. For messages of 64 MB and larger, PCIe CCL improves bandwidth by 26.2% over NCCL and 12.0% over TCCL on average across the three collective primitives. AllGather shows

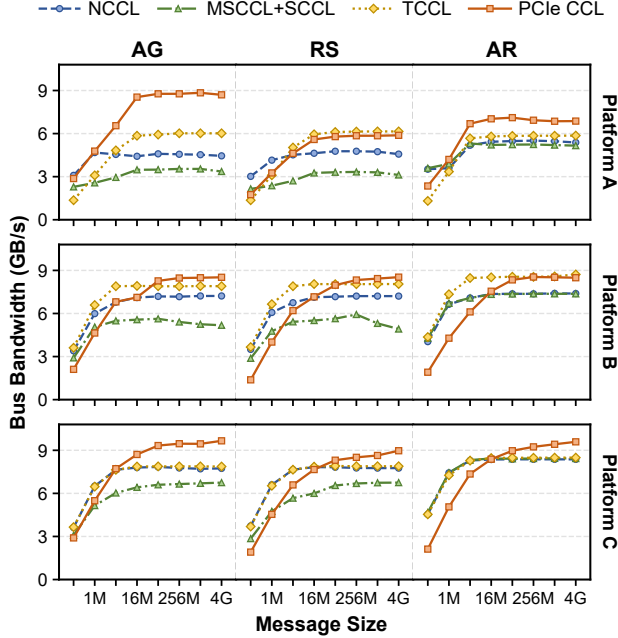


Figure 6. PCIe CCL collective microbenchmarks on PCIe-only machines with different topologies.

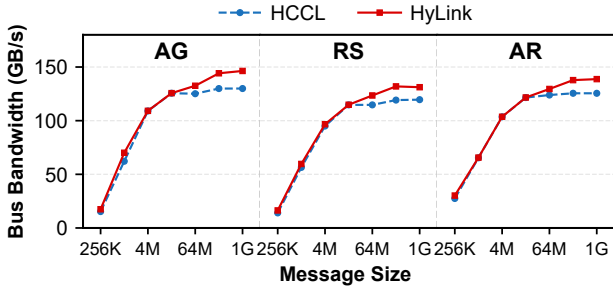


Figure 7. HyLink multipath collective bandwidth on 8 NPUs of Platform D.

the largest gains, with average improvements of 44.2% over NCCL and 24.4% over TCCL.

Multipath Orchestrator Microbenchmarks. We next evaluate Multipath Orchestrator, which runs each collective over the dedicated-link and PCIe paths in parallel, on all 8 NPUs of Platform D. We benchmark the same three collectives under the same measurement methodology, and compare HyLink with HCCL over HCCS interconnects. Figure 7 shows that HyLink improves bandwidth by 6.2% on average, and by 10.4% on average for messages of 128 MB and larger.

End-to-End Training. Figure 8 reports training iteration time normalized to NCCL on Platforms A–C and to HCCL on Platform D.

(1) *PCIe-only platforms.* We train a 3.05B GPT-like model with Megatron-LM [20] using TP=4, and compare NCCL with

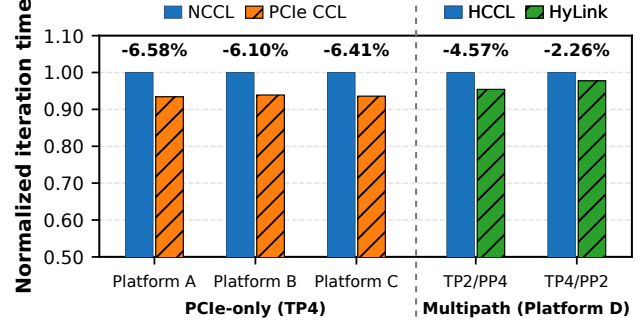


Figure 8. Normalized Megatron training iteration time on Platforms A–D.

PCIe CCL for the intra-node TP collectives on Platforms A–C. By accelerating collective communication across different PCIe topologies, PCIe CCL reduces iteration time by 6.1%–6.6% compared to NCCL.

(2) *PCIe+Dedicated platform.* We train a 7.00B GPT-like model with the same framework using TP=2/PP=4 and TP=4/PP=2, and compare HCCL with HyLink for the intra-node TP collectives on Platform D. By offloading part of each TP collective from HCCS to PCIe, HyLink reduces iteration time by 2.3%–4.6% compared to HCCL.

4.3 Mechanism and In-Depth Validation

Topology-Aware Routing Ablation. To isolate the value of NUMA-aware routing, we implement Ring AllGather using the PCIe CCL DSL and compare it with our NUMA-aware host-staging algorithm, both running on the same PCIe CCL runtime on Platform C. Our host-staging algorithm lets GPUs in each NUMA domain read the same input chunk from local host memory. Table 4 shows that sharing these local copies reduces both per-GPU D2H traffic and total cross-NUMA traffic by 33.3% compared to Ring. The measured bandwidth improves by 9.8% on average compared to Ring.

Runtime Pipelining Ablation. PCIe CCL exposes collective algorithms at chunk granularity, while the runtime breaks each chunk into subchunks and overlaps dependent D2H and H2D stages. Figure 9 shows that pipelining improves AllGather bandwidth on Platform C with a suitable subchunk size. Pipelining with the best measured subchunk size for each message size improves bandwidth by 11.4% on average over the non-pipelined baseline.

Compute Interference. Collectives that move data with SM kernels contend with computation for SMs, and PCIe CCL largely avoids this contention. On Platform C, we measure the bandwidth of collectives and the throughput of a GEMM when each runs alone and when both run concurrently on each GPU. Figure 10(a) shows that the GEMM reduces NCCL bandwidth by 13.6% on average but reduces PCIe CCL bandwidth by only 3.1%. Figure 10(b) shows that NCCL in turn

Table 4. AllGather algorithm ablation on Platform C.

AG output size	Ring bus BW	Host Staging bus BW	Ring per-GPU D2H/H2D	Ring total cross-NUMA	Host Staging per-GPU D2H/H2D	Host Staging total cross-NUMA
16 MB	8.07 GB/s	8.72 GB/s	12 / 12 MB	24 MB	8 / 12 MB	16 MB
64 MB	8.32 GB/s	9.35 GB/s	48 / 48 MB	96 MB	32 / 48 MB	64 MB
256 MB	8.55 GB/s	9.48 GB/s	192 / 192 MB	384 MB	128 / 192 MB	256 MB
1 GB	8.80 GB/s	9.48 GB/s	768 / 768 MB	1.5 GB	512 / 768 MB	1 GB

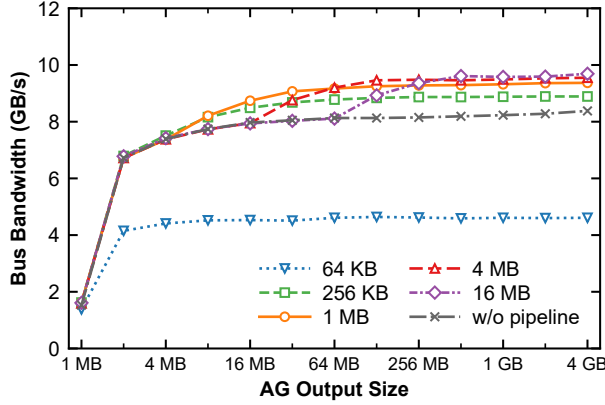


Figure 9. Runtime pipelining ablation of PCIe CCL.

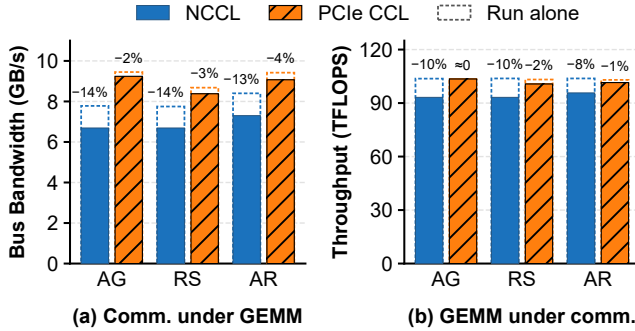


Figure 10. Two-way interference between collectives and GEMM on Platform C. (a) Bus bandwidth with the GEMM. (b) GEMM throughput with each collective.

reduces GEMM throughput by 9.4% on average but PCIe CCL reduces it by at most 2.4%.

Adaptive Split Under Runtime PCIe Interference. We evaluate the resilience of Multipath Orchestrator’s adaptive splitting strategy on Platform D. To emulate runtime PCIe interference, we inject periodic bursts of background PCIe traffic using `ac1rtMemcpyAsync` while the collective benchmark is running. The baseline uses a fixed splitting ratio α , similar to Blink [31], which is pre-tuned to be optimal without interference. In contrast, Multipath Orchestrator adjusts α at runtime based on the observed path progress.

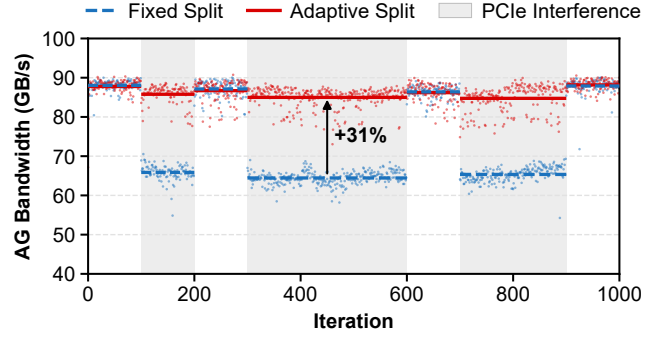


Figure 11. AllGather bandwidth on Platform D under periodic PCIe interference.

Figure 11 shows that the adaptive balancer achieves 30.9% higher throughput than the fixed-ratio baseline under sustained interference.

5 Discussion

Applicability to Multi-Node Training. (1) *PCIe-only machines.* PCIe CCL currently supports intra-node collectives. Its DSL abstraction could also be extended to cross-node data movement. (2) *Machines with dedicated interconnects.* Multipath Orchestrator aggregates bandwidth from the dedicated-link and PCIe paths. For inter-node collectives, NIC DMA traffic shares PCIe bandwidth with the host-memory path, limiting the benefit of dual-path splitting. In multi-node training, Multipath Orchestrator can still accelerate intra-node TP or EP collectives as they do not overlap with inter-node communication. HyLink can check communicator scope at initialization and enable PCIe-path acceleration only for intra-node communicators.

Applicability to Inference. LLM inference also relies on intra-node TP and EP, which produce large collective messages that are well-suited for HyLink. We serve Qwen3-32B with TP=4/PP=2 on Platform D using vLLM Ascend [30]. Compared to HCCL, HyLink reduces end-to-end batch latency by 5.4%. These results demonstrate that HyLink’s high-bandwidth capabilities are not limited to training workloads;

they are equally critical for alleviating communication bottlenecks in LLM inference tasks, thereby expanding the applicability of high-performance collective communication primitives to diverse LLM serving scenarios.

Coexistence with CPU Offloading. Our measurement in §2.3 was conducted without CPU memory offloading mechanisms and showed PCIe utilization below 1% of peak bandwidth during intra-node collective windows. However, enabling offloading can introduce PCIe contention when its transfers overlap with collectives. This contention may slow offloading transfers and degrade end-to-end performance if those transfers lie on the critical path. The impact of this limitation therefore depends on the workload and requires further evaluation.

Host-Side Data Movement. PCIe CCL supports two-hop and three-hop routes across NUMA nodes. In two-NUMA AllGather, two-hop routing (like Host Staging in Table 4) uploads the same source chunk separately to both NUMA-local host buffers for local downloads by peer GPUs. Three-hop routing (Figure 4) replaces the second upload with H2H forwarding. Since uploads and downloads share the bandwidth of half-duplex DRAM channels, reducing uploads can ease contention even when download volume is unchanged. However, H2H places additional pressure on host memory because DMA can bypass CPU caches and require the CPU to reread the staged data from DRAM before writing another host copy. The tradeoff between reduced PCIe uploads and additional memory accesses therefore needs to be evaluated experimentally on the target machine.

Data Layout in Multipath Orchestrator. Multipath Orchestrator splits a collective at the chunk level. For each logical chunk, the dedicated-link path handles the leading portion and the PCIe path handles the remaining portion through PCIe CCL’s strided collective interface. This mapping works naturally for element-wise collectives such as AllReduce, where both portions correspond to fixed output offsets. AllGather and ReduceScatter expose the main compatibility issue because their standard CCL interfaces write compact outputs for each path rather than leaving holes for the other path’s portion. PCIe CCL can write its strided portion directly, but the vendor CCL output may need one device-side copy to merge the compact result into the final layout. Backends with an AllGatherV-style or ReduceScatterV-style primitive, or direct placement into strided output offsets, can avoid this copy by writing each subrange to its final position. Multipath Orchestrator can adopt such interfaces, including future NCCL variable-size collective interfaces, without changing its higher-level split and fork-join design.

6 Related Work

Collective Communication Algorithms. Extensive research has been devoted to optimizing collective communication algorithms. Classic designs such as Ring [4] and Double Binary Tree [14] trade off between bandwidth utilization and latency. For heterogeneous interconnects, BlueConnect and Themis [7, 23] decompose AllReduce along hierarchical topology levels to balance load across links. Other work targets specific physical topologies such as torus, mesh, and reconfigurable optical networks [17, 26, 33]. Since hand-crafted algorithms struggle to generalize across diverse topologies, automated synthesis has attracted growing attention. Blink [31] formulates bandwidth-optimal collectives as maximum-weight spanning tree packing. SCCL [2] first performs joint latency-bandwidth Pareto optimization via SMT solving but faces exponential blowup at larger scales. Subsequent work [3, 19, 27, 35] improves scalability through search space pruning, alternative mathematical formulations, and topology symmetry exploitation. HyLink is orthogonal to these algorithms and could adopt them.

Collective Communication Libraries. The systems community has built production-grade libraries integrating algorithm selection, tuning, and transport optimization. Vendor CCLs such as NCCL [21], RCCL [1], and HCCL [9] bundle predefined algorithms with heuristic selection mechanisms. AutoCCL [36] applies coordinate descent for automated parameter tuning. TCCL [15] profiles NUMA-aware PCIe costs to find optimal ring orderings for PCIe clusters. MSCCLang [8] provides a chunk-oriented DSL and low-level asynchronous primitives for expressing and executing collective algorithms on GPUs, and ResCCL [18] introduces primitive-level scheduling to maximize link utilization across micro-batches. At the transport level, recent work [24, 39] explores multi-NIC bandwidth aggregation and flexible RDMA transport. However, these libraries and frameworks model the topology as flat GPU-to-GPU edges; HyLink breaks this abstraction by natively modeling the PCIe hierarchy with the CPU as an active participant.

Multipath Collective Communication. Several recent works split collective traffic across multiple interconnects to aggregate their bandwidth. Blink [31] combines NVLink and PCIe, but incurs overhead from toggling GPU peer access and uses a fixed traffic split that cannot adapt to runtime PCIe interference. MPCCS [37] combines scale-up and scale-out networks, targeting servers with a dedicated NIC per GPU and spare scale-out bandwidth. FlexLink [29] adaptively combines NVLink, PCIe, and RDMA, but its PCIe ring can incur redundant transfers on hierarchical PCIe topologies. HyLink exploits available PCIe bandwidth across diverse PCIe topologies and adapts to runtime interference without requiring sufficient scale-out bandwidth.

Communication-Computation Co-Optimization. Another line of work hides communication latency through

compute-communication overlap and fusion. CoCoNet [13] provides a unified DSL for semantics-preserving fusion and overlap transformations. Centauri [5] exposes finer-grained overlap opportunities by partitioning collective communication according to primitive type, hardware topology, and workload structure. Concerto [6] formulates communication scheduling as a resource-constrained scheduling problem and uses automatic decomposition to create overlap opportunities for critical communication. Chimera [22] focuses on hybrid parallel LLMs and reduces redundant data movement by reordering operators and fusing adjacent communication operators. These techniques are orthogonal to HyLink, which optimizes the collective primitive itself.

7 Conclusion

This paper presents HyLink, a two-layer collective communication framework for exploiting PCIe bandwidth in distributed LLM training. HyLink addresses two limitations in current CCL designs. PCIe CCL tackles the PCIe topology mismatch with a CPU-driven DSL and a pipelined copy-engine runtime for PCIe-aware routes. Multipath Orchestrator tackles the PCIe aggregation gap by splitting collectives across PCIe and the dedicated interconnect with runtime feedback. Our evaluation shows that HyLink makes effective use of PCIe bandwidth and delivers practical gains across PCIe-only and hybrid-link configurations.

Acknowledgments

We thank our shepherd and the anonymous reviewers for their valuable feedback. This work was supported in part by the National Key Research and Development Program (No. 2024YFB4505604), the National Natural Science Foundation of China (No. 62402025), the Fundamental Research Funds for the Central Universities and Huawei-BUAA Joint Lab. Menghao Zhang is the corresponding author.

References

- [1] AMD. 2018. *ROCm Collective Communication Library (RCCL)*. <https://github.com/ROCm/rccl>
- [2] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) (PPoPP '21). Association for Computing Machinery, New York, NY, USA, 62–75. <https://doi.org/10.1145/3437801.3441620>
- [3] Jiamin Cao, Shangfeng Shi, Jiaqi Gao, Weisen Liu, Yifan Yang, Yichi Xu, Zhilong Zheng, Yu Guan, Kun Qian, Ying Liu, Mingwei Xu, Tianshu Wang, Ning Wang, Jianbo Dong, Binzhang Fu, Dennis Cai, and Ennan Zhai. 2025. SyCCL: Exploiting Symmetry for Efficient Collective Communication Scheduling. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) (SIGCOMM '25). Association for Computing Machinery, New York, NY, USA, 645–662. <https://doi.org/10.1145/3718958.3750499>
- [4] Ernie Chan, Robert van de Geijn, William Gropp, and Rajeev Thakur. 2006. 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* (New York, New York, USA) (PPoPP '06). Association for Computing Machinery, New York, NY, USA, 2–11. <https://doi.org/10.1145/1122971.1122975>
- [5] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) (ASPLOS '24). Association for Computing Machinery, New York, NY, USA, 178–191. <https://doi.org/10.1145/3620666.3651379>
- [6] Shenggan Cheng, Shengjie Lin, Lansong Diao, Hao Wu, Siyu Wang, Chang Si, Ziming Liu, Xuanlei Zhao, Jiangsu Du, Wei Lin, and Yang You. 2025. Concerto: Automatic Communication Optimization and Scheduling for Large-Scale Deep Learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) (ASPLOS '25). Association for Computing Machinery, New York, NY, USA, 198–213. <https://doi.org/10.1145/3669940.3707223>
- [7] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. 2019. BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy. In *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 241–251. https://proceedings.mlsys.org/paper_files/paper/2019/file/0c8abcf158ed12d0dd94480681186fda-Paper.pdf
- [8] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. 2023. MSCCLang: Microsoft Collective Communication Language. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) (ASPLOS '23). Association for Computing Machinery, New York, NY, USA, 502–514. <https://doi.org/10.1145/3575693.3575724>
- [9] Huawei. 2024. *Huawei Collective Communication Library (HCCL)*. <https://gitee.com/ascend/cann-hccl>
- [10] Huawei Technologies Co., Ltd. 2024. *Atlas 800 Training Server Maintenance and Service Guide (Model 9000, Air Cooling) 19 – Technical Specifications*. Huawei Enterprise Support. <https://support.huawei.com/enterprise/en/doc/EDOC1100141951/43611d26/technical-specifications> Accessed: 2026-03-02.
- [11] HyLink Authors. 2026. *Source Code for HyLink*. <https://github.com/Networked-System-and-Security-Group/HyLink>
- [12] Intel Corporation. 2022. *Intel® Xeon® Processor Scalable Family Technical Overview*. Technical Overview 673025. Intel Corporation. <https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html> Accessed: 2024-05-20.
- [13] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. 2022. 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* (Lausanne, Switzerland) (ASPLOS '22). Association for Computing Machinery, New York, NY, USA, 402–416. <https://doi.org/10.1145/3503222.3507778>
- [14] Sylvain Jeagey. 2019. *Massively Scale Your Deep Learning Training with NCCL 2.4*. NVIDIA Corporation. <https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/>
- [15] Heehoon Kim, Junyeol Ryu, and Jaemin Lee. 2024. TCCL: Discovering Better Communication Paths for PCIe GPU Clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for*

- Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) (ASPLOS '24). Association for Computing Machinery, New York, NY, USA, 999–1015. <https://doi.org/10.1145/3620666.3651362>
- [16] Jayacharan Kolla, Pedram Alizadeh, and Gilbert Lee. 2025. Understanding RCCL Bandwidth and xGMI Performance on AMD Instinct MI300X. AMD ROCm Blogs. <https://rocm.blogs.amd.com/software-tools-optimization/mi300x-rccl-xgmi/README.html> Accessed: 2026-02-25.
- [17] Dongkyun Lim and John Kim. 2025. TidalMesh: Topology-Driven AllReduce Collective Communication for Mesh Topology. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE Computer Society, Los Alamitos, CA, USA, 1526–1540. <https://doi.org/10.1109/HPCA61900.2025.00114>
- [18] Tongrui Liu, Chenyang Hei, Fuliang Li, Chengxi Gao, Jiamin Cao, Tian-shu Wang, Ennan Zhai, and Xingwei Wang. 2025. ResCCL: Resource-Efficient Scheduling for Collective Communication. In *Proceedings of the ACM SIGCOMM 2025 Conference* (São Francisco Convent, Coimbra, Portugal) (SIGCOMM '25). Association for Computing Machinery, New York, NY, USA, 55–70. <https://doi.org/10.1145/3718958.3750514>
- [19] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. 2024. Rethinking Machine Learning Collective Communication as a Multi-Commodity Flow Problem. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) (ACM SIGCOMM '24). Association for Computing Machinery, New York, NY, USA, 16–37. <https://doi.org/10.1145/3651890.3672249>
- [20] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prithvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. 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* (St. Louis, Missouri) (SC '21). Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. <https://doi.org/10.1145/3458817.3476209>
- [21] NVIDIA. 2015. *NVIDIA Collective Communication Library (NCCL)*. <https://github.com/NVIDIA/nccl> Accessed: 2024-05-21.
- [22] Le Qin, Junwei Cui, Weilin Cai, and Jiayi Huang. 2025. Chimera: Communication Fusion for Hybrid Parallelism in Large Language Models. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 498–513. <https://doi.org/10.1145/3695053.3731025>
- [23] Saeed Rashidi, William Won, Sudarshan Srinivasan, Srinivas Sridharan, and Tushar Krishna. 2022. 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)*. Association for Computing Machinery, New York, NY, USA, 581–596. <https://doi.org/10.1145/3470496.3527382>
- [24] Zhenghang Ren, Yuxuan Li, Zilong Wang, Xinyang Huang, Wenxue Li, Kaiqiang Xu, Xudong Liao, Yijun Sun, Bowen Liu, Han Tian, Junxue Zhang, Mingfei Wang, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai Chen. 2025. Enabling efficient GPU communication over multiple NICs with FuseLink. In *Proceedings of the 19th USENIX Conference on Operating Systems Design and Implementation* (Boston, MA, USA) (OSDI '25). USENIX Association, USA, Article 6, 18 pages. <https://dl.acm.org/doi/10.5555/3767901.3767907>
- [25] Gabin Schieffer, Ruimin Shi, Stefano Markidis, Andreas Herten, Jennifer Faj, and Ivy Peng. 2024. Understanding Data Movement in AMD Multi-GPU Systems with Infinity Fabric. In *Proceedings of the SC '24 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis* (Atlanta, GA, USA) (SC-W '24). IEEE Press, 567–576. <https://doi.org/10.1109/SCW63240.2024.00079>
- [26] Daniele De Sensi, Tommaso Bonato, David Saam, and Torsten Hoefler. 2024. Swing: Short-cutting Rings for Higher Bandwidth Allreduce. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1445–1462. <https://www.usenix.org/conference/nsdi24/presentation/de-sensi>
- [27] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 593–612. <https://www.usenix.org/conference/nsdi23/presentation/shah>
- [28] Debendra Das Sharma. 2024. PCI-Express: Evolution of a Ubiquitous Load-Store Interconnect Over Two Decades and the Path Forward for the Next Two Decades. *IEEE Circuits and Systems Magazine* 24, 2 (2024), 47–61. <https://doi.org/10.1109/MCAS.2024.3373556>
- [29] Ao Shen, Rui Zhang, and Junping Zhao. 2025. FlexLink: Boosting your NVLink Bandwidth by 27% without accuracy concern. arXiv:2510.15882 <https://arxiv.org/abs/2510.15882>
- [30] vLLM Ascend Contributors. 2026. vLLM Ascend: Community Maintained Hardware Plugin for vLLM on Ascend. <https://github.com/vllm-project/vllm-ascend>.
- [31] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, and Ion Stoica. 2020. Blink: Fast and Generic Collectives for Distributed ML. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 172–186. https://proceedings.mlsys.org/paper_files/paper/2020/file/cd3a9a55f7f3723133fa4a13628cdf03-Paper.pdf
- [32] Guanhua Wang, Chengming Zhang, Zheyu Shen, Ang Li, and Olatunji Ruwase. 2024. Domino: Eliminating Communication in LLM Training via Generic Tensor Slicing and Overlapping. arXiv:2409.15241 [cs.DC] <https://arxiv.org/abs/2409.15241>
- [33] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Many Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 739–767. <https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang>
- [34] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 945–960. <https://www.usenix.org/conference/nsdi22/presentation/weng>
- [35] William Won, Midhilesh Elavazhagan, Sudarshan Srinivasan, Swati Gupta, and Tushar Krishna. 2024. TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 856–870. <https://doi.org/10.1109/MICRO61859.2024.00068>
- [36] Guanbin Xu, Zhihao Le, Yinhe Chen, Zhiqi Lin, Zewen Jin, Youshan Miao, and Cheng Li. 2025. AutoCCL: Automated Collective Communication Tuning for Accelerating Distributed and Parallel DNN Training. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. USENIX Association, Philadelphia, PA, 667–683. <https://www.usenix.org/conference/nsdi25/presentation/xu-guanbin>
- [37] Yuchen Xu, Jianglong Nie, Baojia Li, Mingzhuo Chen, Hao Lu, Guanyu Qu, Zhenchuan Liu, Shuangshuang Yin, Xiaojie Huang, Chunzhi He, Yinben Xia, Quan Wen, Xiang Li, Zekun He, Yachen Wang, Xian-neng Zou, Congcong Miao, and Wenfei Wu. 2026. Multipath Collective Communication Beyond Scale-up Networks in GPU Clouds. In *Proceedings of the 21st European Conference on Computer Systems* (McEwan Hall/The University of Edinburgh, Edinburgh, Scotland UK)

- (EuroSys '26). Association for Computing Machinery, New York, NY, USA, 564–585. <https://doi.org/10.1145/3767295.3769330>
- [38] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] <https://arxiv.org/abs/2303.18223>
- [39] Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Jiaqi Gao, Yilong Zhao, Yongji Wu, Kaichao You, Fengyuan Ren, Zhiying Xu, Costin Raiciu, and Ion Stoica. 2025. An Extensible Software Transport Layer for GPU Networking. arXiv:2504.17307 [cs.NI] <https://arxiv.org/abs/2504.17307>