# BridgedRing: A Cost-Effective Hardware-Software Co-Design to Overcome the UPI Bottleneck in GPU Servers

WANFENG HUANG, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, China
YIHAO ZHAO, Peking University, Beijing, China
YAKUN ZHANG*, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, China and State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, China
JIANGHONG MA, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, China
YUNMING YE*, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, China

Distributed training on standard dual-socket GPU servers suffers from the *"UPI Wall"* — a bottleneck stemming from the interplay between the low physical bandwidth of the Ultra Path Interconnect (UPI) and the severe bidirectional contention induced by NCCL's ring-based collectives. Software-only optimizations cannot overcome this hardware constraint. We present BridgedRing, a hardware-software co-design that bypasses the UPI Wall using an off-the-shelf NVLink Bridge costing approximately $200 — less than 0.5% of the total cost of a typical 8-GPU server. Our topology-aware algorithm redirects all cross-NUMA traffic through this high-bandwidth bypass while preserving intra-NUMA communication over PCIe. Evaluated on 8×A6000 servers against NCCL and hierarchical algorithms: BridgedRing achieves up to 1.97× higher collective bandwidth and delivers 1.31× end-to-end throughput for GPT-13B training in `Megatron-LM` (TP=8, SP=8), 1.36× for GPT-6.7B, and 1.40× for Qwen3-1.7B in `PyTorch DDP`. Crucially, these gains require *no framework modifications*, offering immediate, cost-effective acceleration for the vast installed base of standard GPU servers.

CCS Concepts: • **Computer systems organization → Distributed architectures**; • **Hardware → Communication hardware, interfaces and storage**; • **Computing methodologies → Parallel computing methodologies**.

Additional Key Words and Phrases: distributed training, collective communication, Ring AllReduce, cross-NUMA communication, NVLink Bridge, GPU interconnects, hardware-software co-design

## 1 Introduction

The rapid advancement of Artificial Intelligence (AI), particularly in large-scale deep learning models such as Large Language Models (LLMs) and Generative AI, has driven an unprecedented demand for computational

New Paper, Not an Extension of a Conference Paper.
Authors' Contact Information: Wanfeng Huang, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, Guangdong, China; e-mail: huangwanfeng@stu.hit.edu.cn; Yihao Zhao, Peking University, Beijing, Beijing, China; e-mail: zhaoyh98@pku.edu.cn; Yakun Zhang, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, Guangdong, China and State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, Jiangsu, China; e-mail: zhangyk@hit.edu.cn; Jianghong Ma, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, Guangdong, China; e-mail: majianghong@hit.edu.cn; Yunming Ye, Shenzhen Key Laboratory of Internet Information Collaboration, Harbin Institute of Technology Shenzhen, Shenzhen, Guangdong, China; e-mail: yeyunming@hit.edu.cn.


ACM 1544-3973/2026/6-ART
https://doi.org/10.1145/3822179

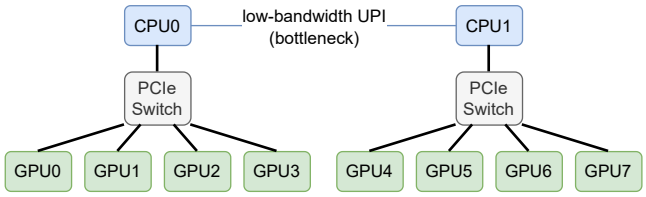



Fig. 1. Topology of a dual-socket, 8-GPU system (standard GPU server). The UPI link offers significantly less bandwidth for GPU communication compared to PCIe. Consequently, GPUs across different NUMA nodes (e.g., GPU 0 and GPU 4) experience lower communication bandwidth than GPUs within the same NUMA node (e.g., GPU 0 and GPU 1).

resources [12, 35]. These models comprise billions or even trillions of parameters, far exceeding the memory and processing capabilities of a single GPU. Consequently, distributed training has emerged as a standard paradigm, enabling the training of these large models by distributing the workload across multiple GPUs [1]. Collective communication operations – including AllReduce, ReduceScatter, and AllGather – are critical to synchronizing model parameters or gradients across the various parallel training strategies [15, 28]. As the number of GPUs and computational throughput continue to grow, the efficiency of these collective communications has become a dominant performance bottleneck, significantly limiting the scalability and training speed of modern AI workloads.

While high-end systems such as NVIDIA DGX servers [18] leverage proprietary NVLink and NVSwitch technologies to provide high-bandwidth, low-latency GPU interconnects, the vast majority of AI research and production workloads are deployed on commodity GPU servers. As illustrated in Figure 1, these systems typically adopt dual-socket Non-Uniform Memory Access (NUMA) architectures, where two CPUs are connected via a lower-bandwidth Ultra Path Interconnect (UPI) link [10]. GPUs are distributed across the PCIe root complexes of the two CPU sockets [6].

In such configurations, collective communication algorithms—particularly the widely used Ring AllReduce [8, 24]—face a fundamental limitation. Communication between GPUs attached to different CPU sockets must traverse the CPU root complexes and the cross-NUMA interconnect, incurring both additional latency and limited cross-socket bandwidth. Our empirical analysis shows that this architectural constraint leads to a *UPI bottleneck*, which we term the "*UPI Wall*." This bottleneck arises from two combined factors: (1) *Bandwidth asymmetry*: The UPI link provides lower peak throughput (e.g., 16–20 GB/s) than intra-NUMA PCIe or NVLink connections, imposing a hard upper bound on cross-socket data movement; (2) *Bidirectional contention*: Ring-based collectives induce simultaneous bidirectional traffic over the cross-NUMA link, leading to contention, protocol overheads, and reduced effective bandwidth. Together, these effects make the UPI link the dominant bottleneck for intra-node communication in commodity multi-GPU servers.

Consequently, a significant research gap exists: how to efficiently overcome the UPI bottleneck for collective communications in standard GPU servers without demanding costly and specialized hardware. On one hand, existing software-only topology-aware optimizations [32] attempt to mitigate this gap by carefully scheduling communication paths. While these approaches can yield notable speedups in multi-node settings by optimizing data placement and routing, their effectiveness is fundamentally constrained in cross-NUMA scenarios, where performance remains bottlenecked by the limited bandwidth of the UPI interconnect. Since such methods cannot introduce new physical high-bandwidth pathways, they are unable to eliminate the root cause of the cross-NUMA communication bottleneck. On the other hand, high-end solutions such as NVIDIA DGX systems, equipped with NVSwitch and full all-to-all NVLink connectivity, virtually eliminate this problem [18]. However, these platforms entail prohibitive costs and cater to specialized deployments, rendering them inaccessible to the vast majority of

users relying on standard multi-socket servers. This dichotomy underscores the urgent need for a cost-effective yet physically impactful solution that bridges the gap between software scheduling and hardware capability.

To address this critical gap, we propose BridgedRing, a novel and pragmatic hardware-software co-design that breaks the UPI bottleneck in standard GPU servers. The core idea of BridgedRing is to introduce a high-speed, direct physical bypass for cross-NUMA GPU traffic. First, on the hardware side, our approach requires a minimally invasive modification: the installation of a single, off-the-shelf NVLink Bridge [21] to connect a pair of GPUs, one from each NUMA node. This creates a dedicated, high-bandwidth data path that completely circumvents the congested CPU-UPI route. Second, on the software side, we propose a topology-aware collective communication algorithm that intelligently utilizes this new hardware resource. The algorithm identifies the NVLink Bridge and dynamically redirects cross-NUMA data chunks onto this high-speed bypass using a relay-style forwarding mechanism, while intra-NUMA traffic continues to flow efficiently over the existing PCIe and local interconnects. Finally, this co-design ensures that the right traffic is routed through the fastest path, maximizing communication performance with minimal hardware cost.

The contributions of this paper are as follows:

- We present the first systematic identification and quantification of the **UPI Wall** for ring-based collectives on standard dual-socket GPU servers. We explicitly decouple this bottleneck into **static raw bandwidth constraints** and **dynamic bidirectional contention effects**, revealing how their interplay causes severe performance degradation in AllReduce workloads.
- We propose BridgedRing, a low-cost hardware-software co-design that breaks the UPI Wall by leveraging a single cross-NUMA NVLink Bridge to create a high-speed bypass for cross-NUMA communication.
- We characterize the bridge GPUs through bandwidth-headroom analysis and fine-grained profiling, showing that our relay-based design maintains workload balance and introduces no additional stragglers.
- Through a fully functional prototype, we demonstrate that BridgedRing achieves up to **1.97**× higher communication bandwidth and **1.40**× end-to-end training throughput than NCCL's current implementation.
- We establish a new, cost-effective pathway to significantly enhance distributed training performance on the vast installed base of standard GPU servers, making high-performance AI training more accessible.

## 2 Background

### 2.1 Distributed Training and Collective Communication

Distributed training has become indispensable for large-scale deep learning models that exceed single-device capacity. Modern parallelization strategies—including Data Parallelism (DP) [14], Tensor Parallelism (TP) [28, 33], Pipeline Parallelism (PP) [9], and Fully Sharded Data Parallelism (FSDP) [36]—all rely on collective communication primitives to synchronize model states. Critical operations such as **AllReduce** (for gradient aggregation in DP) and **ReduceScatter**/**AllGather** (for parameter sharding in FSDP/TP) directly determine training scalability. In bandwidth-bound deep learning workloads, the performance of these collectives is dominated by interconnect efficiency, making them the primary bottleneck in distributed training systems.

### 2.2 Server Architectures for GPU Acceleration

Modern GPU servers broadly fall into two architectural paradigms: proprietary high-performance platforms with dense GPU fabrics, and standard dual-socket PCIe servers.

*2.2.1 Proprietary High-Performance Platforms.* Systems like NVIDIA DGX leverage custom SXM GPU modules interconnected via **NVLink** and **NVSwitch** technologies [7, 18]. These interconnections create a unified, all-to-all communication fabric that bypasses CPU bottlenecks entirely, providing consistent high-bandwidth connectivity

between all GPUs. While delivering superior performance, these specialized platforms represent a small fraction of deployed infrastructure due to their premium cost and ecosystem constraints.

*2.2.2 Standard Dual-Socket GPU Servers.* The majority of AI workloads execute on **standard dual-socket GPU servers**, where discrete GPUs are connected to CPUs via PCIe. As illustrated in Figure 1, these systems adopt a NUMA architecture in which GPUs are distributed across two CPU sockets. Consequently, GPU communication depends on NUMA locality: (1) *Intra-NUMA communication*—data transfers between GPUs attached to the same socket are routed through local PCIe switches, achieving high effective bandwidth; (2) *Cross-NUMA communication*—data exchanges between GPUs on different sockets traverse the corresponding PCIe root complexes and the cross-NUMA interconnect (e.g., UPI), incurring additional latency and reduced bandwidth.

This asymmetry leads to a substantial performance gap. Intra-NUMA bandwidth reaches 42.6 GB/s, whereas cross-NUMA bandwidth drops to 22.9 GB/s, corresponding to a 46.2% reduction (Table 3). This disparity identifies the cross-NUMA link as the primary bottleneck in multi-GPU communication on commodity servers.

## 2.3 Ring-Based Collective Algorithms

For bandwidth-bound workloads like distributed deep learning, ring-based algorithms dominate collective communication implementations [8]. Unlike latency-optimized tree algorithms [11] that excel with small messages, ring algorithms pipeline large data chunks through a logical device ring to maximize bandwidth utilization. NCCL and similar libraries default to ring algorithms for large-message collectives because (1) *Linear bandwidth scaling*: they achieve near-linear throughput growth with message size; (2) *Minimal transfer redundancy*: they avoid redundant data movement; (3) *Heterogeneous link adaptivity*: they efficiently tolerate unbalanced link bandwidths.

However, ring performance is critically dependent on the *slowest link* in the communication path. In standard dual-socket servers, this bottleneck is the UPI link during cross-NUMA transfers.

## 2.4 The UPI Wall Bottleneck

The combination of ring algorithm requirements and standard server topology creates what we term the **UPI Wall**: a performance barrier imposed by the limited bandwidth of the UPI interconnect. This bottleneck manifests through two compounding effects: (1) *Bandwidth asymmetry*: The UPI link provides significantly lower bandwidth (22.8 GB/s) than PCIe (42.6 GB/s); (2) *Bidirectional contention*: Ring algorithms require simultaneous bidirectional communication, causing severe congestion on the shared UPI link.

Software-only optimizations cannot overcome this physical constraint: they can reschedule traffic but cannot create new high-bandwidth paths. Existing approaches therefore still route cross-NUMA traffic through UPI, leaving performance untapped. Breaking the UPI Wall requires creating new high-bandwidth paths between NUMA domains, which our hardware-software co-design provides.

# 3 System Analysis and Motivation

This section proceeds in three steps. First, we characterize the role of collective communication in large-scale LLM training and quantify its contribution to end-to-end training time. Second, we analyze the UPI bottleneck through theoretical reasoning and empirical measurement. Third, we examine why hardware modification is necessary and why existing collective algorithms cannot effectively exploit such modifications. Together, these observations motivate the hardware-software co-design presented in Section 4.

## 3.1 Characterizing LLM Workloads

To ground our optimization target in realistic training workloads, we profile three representative LLM configurations: `GPT-6.7B (TP8)`, `GPT-6.7B (TP8, SP8)`, and `Qwen3-1.7B (DDP)`. We examine two aspects of these

traces. First, we quantify how much collective communication remains on the critical path despite computation-communication overlap. Second, we characterize the message-size distribution of those collectives to determine whether the dominant cost lies in the latency-bound or bandwidth-bound regime.

*3.1.1 Exposed Communication Fraction and Speedup Potential.* Even with computation-communication overlap enabled, hard synchronization barriers in both tensor and data parallelism leave a significant portion of communication time unmasked. We define two metrics: the *total communication fraction* ($f_{\text{comm}}$), which is the total time spent in communication kernels, and the *exposed communication fraction* ($f_{\text{exp}}$), which is the portion of communication time that lies on the critical path and cannot be overlapped with computation.

The exposed fraction $f_{\text{exp}}$ provides the theoretical upper bound of communication optimization: if all exposed communication could be eliminated, total iteration time would reduce to pure computation time, yielding a maximum speedup of $\frac{1}{1-f_{\text{exp}}}$. Table 1 summarizes our measurements:

Table 1. Communication overhead breakdown and theoretical speedup potential.

| **Workload** | **Total Comm. ($f_{\text{comm}}$)** | **Exposed Comm. ($f_{\text{exp}}$)** | **Max Speedup** |
|---|---|---|---|
| Qwen3-1.7B (DDP) | 83.3% | 65.0% | 2.86× |
| GPT-6.7B (TP8) | 63.2% | 50.9% | 2.04× |
| GPT-6.7B (TP8, SP8) | 79.0% | 54.9% | 2.22× |

Across all three workloads, the exposed communication fraction remains high, ranging from 50.9% to 65.0%. These values indicate that communication optimization can directly affect end-to-end training throughput rather than being fully hidden by computation.

*3.1.2 Message Size Distribution of Collective Operations.* To identify the dominant communication regime, we examine the size distribution of collective operations. Table 2 summarizes the per-iteration breakdown across different workloads. For GPT workloads, eight-step gradient accumulation satisfies global batch-size requirements under memory constraints. This practice increases the total number of collective operations per iteration.

Table 2. Communication trace analysis (per iteration).

| **Workload** | **Large Ops (>10 MiB)** | **Small Ops (<1 MiB)** | **Volume Share** |
|---|---|---|---|
| Qwen3-1.7B (DDP) | 113 (*AllReduce*) | 3 | 99.99% |
| GPT-6.7B (TP8) | 1,041 (*AllReduce*) | 59 | 99.98% |
| GPT-6.7B (TP8, SP8) | 2,601 (*AG/RS/AR*) | 60 | 99.85% |

The traces show that large collectives dominate LLM training communication. More than 94% of operations exceed 10 MiB, accounting for over 99.8% of total communication volume. Combined with the exposed communication fractions above, these results indicate that bandwidth-bound operations govern end-to-end training efficiency, while latency-bound small-message optimizations offer limited throughput gains.

## 3.2 The UPI Bottleneck in Dual-Socket GPU Servers

Standard dual-socket GPU servers exhibit a **dual-cluster topology** as illustrated in Figure 1, where GPUs connect to two separate CPU sockets via PCIe. We identify and quantify the performance limitation imposed by the UPI interconnect between CPU sockets.

Table 3. Measured GPU peer-to-peer memory copy bandwidth and bidirectional efficiency. All measurements were performed on an NVIDIA RTX A6000 system with dual Intel Xeon 4310 CPUs.

| Connection Type | Bidirectional (GB/s) | Unidirectional (GB/s) | Bi/2Uni (%) |
|---|---|---|---|
| Intra-NUMA | 42.62 ± 0.01 | 26.8 ± 0.00 | 79.43 ± 0.02 |
| Cross-NUMA | 22.88 ± 0.04 | 17.20 ± 0.06 | 66.51 ± 0.26 |
| NVLink | 89.80 ± 0.01 | 49.94 ± 0.01 | 89.90 ± 0.02 |

*3.2.1 Theoretical Analysis of Collective Performance.* Consider a dual-socket system with $N$ GPUs evenly distributed ($N/2$ per socket). We model this topology as a graph $G = (V, E)$ where vertices $V$ are partitioned into two sets $V_A$ and $V_B$ corresponding to the NUMA nodes. The UPI links form a **minimum cut** between $V_A$ and $V_B$ with bandwidth $B_{\text{UPI}}$, which is significantly lower than the intra-cluster PCIe bandwidth $B_{\text{PCIe}}$.

For the widely adopted Ring AllReduce algorithm, any ring spanning both clusters must contain exactly two cross-cluster links. For a message size $M$, the communication time $T$ consists of latency and bandwidth terms:

$$T = 2(N-1)\tau + \frac{2(N-1)}{N} \cdot \frac{M}{B_{\text{ring}}^{\text{eff}}}, \tag{1}$$

where ($\tau$) denotes the per-step latency and $B_{\text{ring}}^{\text{eff}}$ denotes the effective bandwidth of the bottleneck on the ring.

By the max-flow min-cut theorem, the maximum achievable bandwidth for cross-cluster collective communications is fundamentally bounded by the cross-NUMA UPI link capacity:

$$B_{\text{ring}}^{\text{eff}} \leq B_{\text{UPI}}^{\text{eff}}. \tag{2}$$

Critically, the standard Ring AllReduce creates simultaneous bidirectional traffic across the UPI link, resulting in severe contention. We term this phenomenon the **UPI Wall**, where the effective usable bandwidth is significantly reduced due to bidirectional interference:

$$B_{\text{UPI}}^{\text{eff}} = \eta \cdot B_{\text{UPI}}^{\text{phys}}, \quad \text{where } 0 < \eta < 1. \tag{3}$$

The efficiency factor $\eta$ captures protocol overheads, bidirectional interference, and resource contention effects.

This theoretical analysis reveals three fundamental insights: (1) The UPI bottleneck imposes a strict upper bound on cross-cluster collective performance; (2) Ring AllReduce exacerbates this bottleneck through bidirectional UPI contention; (3) Pure software optimizations cannot overcome this physical constraint.

*3.2.2 Empirical Validation of the UPI Wall.* We validate our theoretical analysis through microbenchmarks on a dual-socket server with two Intel Xeon 4310 CPUs (2×UPI @ 10.4 GT/s) and eight NVIDIA RTX A6000 [19] GPUs. Table 3 presents measured GPU peer-to-peer bandwidth using the nvbandwidth utility [17].

Our measurements reveal three critical observations: (1) Intra-cluster communication (within the same NUMA node) achieves 26.8 GB/s of unidirectional bandwidth by leveraging the local PCIe infrastructure; (2) Inter-cluster communication (across NUMA nodes) drops to 17.2 GB/s unidirectional, confirming the UPI bottleneck as the primary scalability limiter; (3) Bidirectional efficiency degrades significantly for cross-NUMA transfers (66.5%), compared to 79.4% for intra-NUMA and 89.9% for NVLink-connected GPUs.

Figure 2 demonstrates the impact on collective performance. For 256 MB messages, standard Ring AllReduce across all eight GPUs achieves only 13.0 GB/s effective bandwidth, substantially lower than both the physical UPI capacity and the measured cross-NUMA P2P bandwidth. This result confirms that bidirectional contention in Ring AllReduce severely degrades performance beyond the physical link limitations.

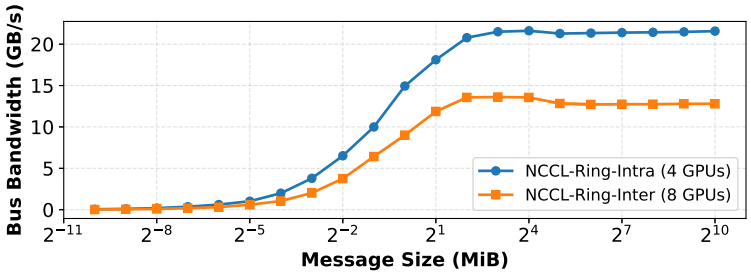



Fig. 2. AllReduce bandwidth comparison between 8-GPU (cross-NUMA) and 4-GPU (intra-NUMA) configurations. Due to UPI contention, the bidirectional bandwidth of the 8-GPU setup reaches only 66.51% of twice the unidirectional bandwidth.

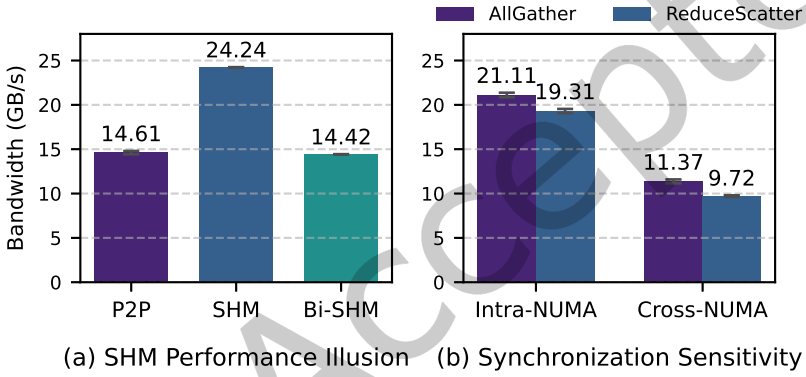



Fig. 3. Performance limitations of software-only optimizations. (a) Bidirectional bandwidth degradation with SHM across NUMA nodes. (b) Synchronization overhead in ReduceScatter operations compared to AllGather.



*3.2.3 Limitations of Software-Only Optimizations.* While modern collective libraries like NCCL and TCCL [13] employ topology-aware routing, they cannot overcome the fundamental UPI limitations. These libraries often prefer Shared Memory (SHM) over direct P2P transfers for cross-NUMA communication. Our measurements confirm SHM achieves 24.24 GB/s for unidirectional transfers, nearly double the 14.61 GB/s of raw P2P.

However, collective operations require **bidirectional**, **synchronous**, and **iterative** data exchanges. Under these conditions, two fundamental limitations emerge as shown in Figure 3: (1) *Bidirectional contention*: Even with SHM, bidirectional bandwidth collapses to 14.42 GB/s (measured via sendrecv_perf), far below the sum of unidirectional streams, due to UPI's limited full-duplex efficiency; (2) *Synchronization sensitivity*: ReduceScatter operations suffer a 13% performance penalty compared to AllGather under identical data volumes due to UPI-induced latency jitter and NUMA remote access delays.

These results confirm that software-only optimizations cannot overcome the fundamental bandwidth and latency limitations imposed by the physical UPI interconnect under realistic collective communication workloads.


(a) Standard ring path (NCCL): One cross-NUMA hop uses NVLink; the reverse hop traverses UPI.

(b) Ideal ring topology: Both cross-NUMA hops traverse the NVLink Bridge bidirectionally.

Fig. 4. Cross-NUMA communication paths in dual-socket GPU servers with a single NVLink Bridge.

## 3.3 Hardware Topology Modification: Necessity and Feasibility

*3.3.1 Theoretical Necessity of Physical Bypass.* Section 3.2 establishes that the UPI Wall represents a *physical* bottleneck that software-only optimizations cannot eliminate. The only viable solution is to introduce a high-bandwidth interconnect that bypasses the CPU-UPI-CPU path entirely.

An effective solution must satisfy three practical constraints: (1) Compatibility with existing PCIe GPU servers; (2) Low cost relative to the total system investment; (3) Minimal intrusion into firmware or OS stacks.

An off-the-shelf NVLink Bridge meets all these criteria, requiring no custom hardware or chassis modification. Modern server platforms such as the Dell DSS 8440 [5] and Supermicro SYS-420GP-TNR [29] feature adjacent PCIe x16 slots spanning across CPU sockets, with sufficient physical clearance and signal integrity to support cross-NUMA NVLink Bridge installation out of the box.

The cost of a 3-slot NVLink Bridge (e.g., RTX A6000 NVLink Bridge [22]) is approximately $200, representing less than 0.5% of a typical 8-GPU server's total cost ($40,000–$60,000). This modification raises the cross-NUMA bandwidth ceiling from $B_{\text{UPI}} \approx 23$ GB/s to $B_{\text{NVLink}} \approx 90$ GB/s, a 3.9× improvement in the max-flow bound.

*3.3.2 Empirical Assessment of Cross-NUMA NVLink.* We validate this hypothesis by installing an NVIDIA NVLink Bridge between GPU 3 (NUMA 0) and GPU 4 (NUMA 1). P2P bandwidth measurements confirm 89.8 GB/s bidirectional throughput across this new link.

However, standard NCCL Ring AllReduce achieves only a modest improvement from 13.0 GB/s to 15.2 GB/s. This limited gain occurs because only one direction of cross-NUMA traffic utilizes the NVLink Bridge; the reverse direction still traverses the UPI path as illustrated in Figure 4a.

This case study reveals a crucial insight: **the mere presence of a high-bandwidth bypass is insufficient without algorithmic coordination**. Although the hardware topology provides a high-bandwidth NVLink path between sockets, current collective algorithms utilize it in only one direction of the ring, leaving the reverse direction constrained by UPI.

## 3.4 Algorithmic Limitations in Exploiting Physical Bypasses

Despite the availability of high-bandwidth physical bypasses, existing collective algorithms fail to leverage them effectively due to two fundamental limitations:

*3.4.1 Topological Asymmetry Challenge.* In a dual-socket system with GPUs $\{0, 1, 2, 3\}$ on NUMA 0 and $\{4, 5, 6, 7\}$ on NUMA 1, constructing a Hamiltonian ring requires exactly two inter-cluster edges. However, an NVLink Bridge provides only a single interconnect between the two NUMA domains (e.g., between GPU 3 and GPU 4). As shown in Figure 4a, existing libraries such as NCCL and TCCL [13] can utilize this link for one cross-NUMA edge in the ring, while the other inter-cluster edge must still be routed through the CPU interconnect (e.g., UPI).

*3.4.2 GPU Relay Limitation.* Unlike network routers, GPUs lack autonomous forwarding capability. Even though GPU 4 has direct access to both GPU 7 (local PCIe) and GPU 3 (NVLink), NCCL cannot instruct GPU 4 to "relay"

GPU 7's data to GPU 3 without explicit algorithmic coordination. This limitation forces the second cross-NUMA communication step to remain bound to the UPI link.

*3.4.3 Key Insight.* Breaking the UPI Wall requires not just a physical bypass but a co-designed algorithm that actively orchestrates cross-NUMA reductions through the bridge GPUs. This insight directly motivates our BridgedRing design.

## 4 BridgedRing: Hardware-Software Co-Design

Building on the analysis in Section 3, we present BridgedRing, a minimal hardware-software co-design that transforms the cross-NUMA NVLink Bridge from a passive link into an active communication accelerator. Our approach preserves compatibility with existing collective interfaces while eliminating all UPI traversal.

### 4.1 Design Principles: Topology Reconstruction and Delegated Reduction

As demonstrated in Section 3.4, a physical bypass alone is insufficient to eliminate the cross-NUMA bottleneck. Although a cross-NUMA NVLink Bridge provides a high-speed path, standard collective libraries often retain logical dependencies on host-mediated UPI paths. BridgedRing addresses this mismatch through two coordinated design principles: (1) *Cross-NUMA Topology Reconstruction*: instead of using an NVLink Bridge for its conventional intra-NUMA peer-to-peer role, BridgedRing employs it as a dedicated cross-NUMA data path, raising the usable cross-NUMA bandwidth ceiling from 22.9 GB/s (UPI) to 89.8 GB/s (NVLink); and (2) *Delegated Reduction*: rather than confining cross-NUMA reduction to the destination GPU, BridgedRing delegates this operation to the bridge GPUs, enabling them to write reduced results directly into remote peer buffers. Consequently, cross-NUMA dependencies are served by the high-bandwidth bridge path rather than the congested host interconnect. The following subsections instantiate these principles in the hardware topology and collective protocol.

### 4.2 Hardware Topology Modification

We introduce a minimal, cost-effective hardware optimization for dual-socket PCIe GPU servers: installing an off-the-shelf NVLink Bridge between one GPU in each NUMA domain (e.g., GPU 3 on NUMA 0 and GPU 4 on NUMA 1). This bridge establishes a direct, bidirectional 90 GB/s link between the two sockets, enabling complete bypass of the CPU–UPI–CPU communication path.

This optimization requires no changes to the motherboard, CPU, or existing PCIe hierarchy. NVLink connectors are available on a wide range of modern GPUs, including NVIDIA A100/H100 datacenter accelerators and Ampere-based workstation GPUs (RTX A6000, RTX 3090). Many dual-socket server chassis provide sufficient physical clearance for the bridge. The total hardware cost is under $200 [22], making this modification highly practical.

The resulting topology is *asymmetric*: only the two *bridge GPUs* are connected across NUMA domains through the NVLink Bridge. This asymmetry necessitates a co-designed algorithm to fully exploit the topology.

### 4.3 BridgedRing Software

The core innovation of BridgedRing is treating bridge GPUs as *active reduction agents* rather than passive ring nodes. Instead of transmitting raw tensors across NUMA boundaries, each bridge GPU performs local reductions between data from its cluster and the remote bridge's buffer via NVLink, eliminating all UPI traffic.

*4.3.1 Logical Ring Reordering.* We reorder the logical ring as:

$$[\, g_L,\ \text{NUMA}_0 \text{ non-bridge GPUs},\ g_R,\ \text{NUMA}_1 \text{ non-bridge GPUs}\,],$$

where $g_L$ and $g_R$ are the NVLink-connected bridge GPUs. As shown in Figure 5, this placement creates two natural cross-NUMA boundaries at the bridge GPU transitions.


(a) Logical ring structure: GPUs ordered as $g_L$, NUMA 0 GPUs, $g_R$, and NUMA 1 GPUs, with all cross-NUMA traffic routed through the bridge GPUs.

(b) Physical implementation: Cross-NUMA traffic is fully offloaded to an NVLink bridge via delegated reduction, eliminating reliance on the UPI.

Fig. 5. BridgedRing communication topology. The logical ring (a) creates strategic cross-NUMA boundaries, while the physical implementation (b) confines all cross-NUMA traffic to the high-bandwidth NVLink path.

*4.3.2 Delegated Reduction Protocol.* During initialization, each GPU determines its `src` (data source) and `dst` (write target) for every communication round. Non-bridge GPUs follow standard ring assignment: `dst` is self, `src` is left neighbor in the logical ring.

Bridge GPUs implement a critical deviation: (1) Bridge GPU $g_L$ sets its `dst` to $g_R$'s buffer and its `src` to the left neighbor of $g_R$; (2) Bridge GPU $g_R$ sets its `dst` to $g_L$'s buffer and its `src` to the left neighbor of $g_L$.

As visualized in Figure 5b, this exchange delegates cross-NUMA reduction responsibilities to the bridge pair. Rather than transmitting data across the NUMA boundary, each bridge GPU performs a reduction with the accumulator residing in the peer bridge's memory buffer, eliminating UPI traversals for collective operations.

Algorithm 1 formalizes this protocol. The only modification from the standard ring is the neighbor assignment; the per-round loop remains unchanged. The Store operation on a bridge GPU automatically triggers an NVLink transfer because `dst` refers to a remote GPU's memory.

*4.3.3 UPI Elimination Guarantee.* Consider the two cross-NUMA dependencies induced by the logical ring, which correspond to the two cross-partition edges in the graph abstraction. In BridgedRing, these dependencies are resolved by assigning the reduction to the bridge GPUs. Specifically, each bridge GPU performs a local read from its intra-NUMA peer via PCIe and a remote read from the opposite NUMA domain via the NVLink Bridge, locally reduces the two operands, and writes the result back to the peer buffer over NVLink.

This execution pattern ensures that cross-NUMA data exchange is entirely confined to the NVLink Bridge, without traversing the CPU root complex or the cross-NUMA interconnect. As a result, BridgedRing provides three key properties: (1) *UPI-free communication*: all cross-NUMA data transfers are carried exclusively by the NVLink Bridge; (2) *Minimal specialization*: only the bridge GPUs participate in cross-NUMA coordination, while all other GPUs follow the standard ring schedule; (3) *Interface transparency*: the design preserves the standard AllReduce abstraction and requires no changes to application-level code.

## 4.4 Formal Correctness Analysis

In this section, we formally demonstrate the semantic equivalence between BridgedRing and the standard Ring AllReduce baseline. Our analysis is constrained to the **ReduceScatter** phase, given that the subsequent AllGather phase entails purely deterministic data propagation without any arithmetic state transitions.

*4.4.1 Notation and Definitions.* Let $\mathcal{R} = \{g_0, g_1, \ldots, g_{N-1}\}$ be a logical ring of $N$ GPUs, and let the data payload be partitioned into $N$ chunks, denoted by $c \in \{0, 1, \ldots, N-1\}$. Let $D_{i,c}^{(k)}$ denote the state of data chunk $c$ residing on GPU $g_i$ after step $k$. The initial state $D_{i,c}^{(0)}$ is the input chunk originating from $g_i$. In a standard Ring ReduceScatter,

**Algorithm 1** BridgedRing AllReduce algorithm.

**Require:** Rank $i$, total ranks $N$, bridge IDs $g_L$, $g_R$

// — Neighbor assignment phase
**if** $i = g_L$ **then**
    dst $\leftarrow g_R$ ▷ write results to remote bridge
    src $\leftarrow (g_R - 1 + N) \bmod N$ ▷ read from remote bridge's left neighbor
**else if** $i = g_R$ **then**
    dst $\leftarrow g_L$
    src $\leftarrow (g_L - 1 + N) \bmod N$
**else**
    dst $\leftarrow i$
    src $\leftarrow (i - 1 + N) \bmod N$
**end if**
// — ReduceScatter phase
**for** round $r = 0$ to $N - 2$ **do**
    $x \leftarrow$ Load(src) ▷ incoming data chunk
    $y \leftarrow$ Load(dst) ▷ local or remote accumulator
    $z \leftarrow$ Reduce$(x, y)$
    Store(dst, $z$) ▷ store to local or remote buffer
**end for**
// — AllGather phase
**for** round $r = 0$ to $N - 2$ **do**
    $x \leftarrow$ Load(src)
    Store(dst, $x$)
**end for**

the state transition at step $k$ is defined as:

$$D_{i,c}^{(k)} = D_{i,c}^{(k-1)} \oplus D_{\text{prev}(i),c}^{(k-1)}, \tag{4}$$

where $\oplus$ is the reduction operator and $\text{prev}(i)$ is the upstream neighbor of $g_i$ in the logical ring.

#### 4.4.2 *Semantic Equivalence of Delegated Reduction.*

Lemma 4.1 (Delegated Correctness). *The delegated reduction performed by a bridge GPU $g_b$ on behalf of its cross-NUMA peer $g_p$ is semantically equivalent to the standard local reduction at $g_p$.*

Proof. In BridgedRing, when the logical data flow crosses the NUMA boundary from $g_{\text{prev}(p)}$ to $g_p$, the bridge GPU $g_b$ executes the standard operation $D_{p,c}^{(k)} = D_{p,c}^{(k-1)} \oplus D_{\text{prev}(p),c}^{(k-1)}$ on behalf of $g_p$.

The delegated execution on $g_b$ follows this sequence: (1) *Fetch*: Load $D_{\text{prev}(p),c}^{(k-1)}$ from $g_{\text{prev}(p)}$; (2) *Fetch Peer*: Load $D_{p,c}^{(k-1)}$ from $g_p$; (3) *Compute*: $z = D_{p,c}^{(k-1)} \oplus D_{\text{prev}(p),c}^{(k-1)}$; (4) *Delegated Store*: Store $z$ into $g_p$, updating the $D_{p,c}^{(k)}$.

Since the delegated execution applies the operator $\oplus$ to the identical step $(k-1)$ operands, the computed $z$ is equivalent to a local reduction. Furthermore, under our system model that guarantees memory visibility prior to step $k+1$, this remote update preserves the logical state transition of the standard algorithm at $g_p$. □

#### 4.4.3 *Global Equivalence Proof.*

Theorem 4.2 (Global Correctness). *After $N-1$ steps of the ReduceScatter phase, each GPU $g_i$ in BridgedRing holds the fully reduced chunk $c = i$: $D_{i,i}^{(N-1)} = \bigoplus_{j=0}^{N-1} D_{j,i}^{(0)}$.*

Proof. We prove global correctness by induction on the step index $k$. *Base case ($k = 0$):* Both algorithms initialize with the identical input distribution $D_{i,c}^{(0)}$. *Inductive step:* Assume the invariant $D_{i,c}^{(k)} = D_{i,c,\text{standard}}^{(k)}$ holds for all GPUs $i$ and chunks $c$ at step $k$. For step $k+1$, two cases arise for each GPU $g_i$: (1) If the transition $(\text{prev}(i) \to i)$ is intra-NUMA, BridgedRing executes the standard local kernel and preserves the invariant. (2) If the transition is cross-NUMA, Lemma 4.1 ensures the delegated reduction produces a state $D_{i,c}^{(k+1)}$ equivalent to the standard local reduction result.

Thus, the invariant holds for $k+1$. By induction, the $(N-1)$-step ReduceScatter phase yields the same final state as the standard algorithm. The subsequent AllGather phase propagates these reduced chunks using pure data copies, ensuring final end-to-end consistency. □

## 4.5 Workload Balance and Bridge Efficiency Analysis

This section analyzes the workload distribution within BridgedRing to confirm that the **bridge GPUs** do not constitute a performance bottleneck. Guided by the Delegated Reduction protocol (Algorithm 1), we systematically demonstrate that the bridge pair introduces negligible overhead.

*4.5.1 Symmetry in PCIe Traffic Distribution.* The optimal performance of ring-based collectives necessitates uniform resource utilization. BridgedRing maintains strict topological symmetry across the **PCIe-PLX fabric** within each NUMA domain: (1) *PCIe Traffic Parity*: Every GPU in the logical ring, irrespective of its role as a bridge or non-bridge node, performs exactly **one remote `Load`** operation via the PCIe bus per round (reading from `src`). This traffic is localized within the PCIe switches (PLX), guaranteeing perfectly balanced utilization of the intra-NUMA fabric. (2) *Computational Parity*: Each GPU performs exactly **one vector reduction** per round, ensuring no disparate arithmetic burden is imposed on the bridge pair.

*4.5.2 Overlapping Overhead via SM-Direct Memory Access.* In contrast to DMA-driven transfers, BridgedRing employs **SM-direct memory access**, enabling the Streaming Multiprocessors (SMs) to issue memory instructions directly to remote VRAM across heterogeneous physical links. We leverage two dimensions of parallelism to mask bridge-related latency: (1) *Instruction-Level Parallelism (ILP)*: During the *ReduceScatter* phase, a bridge GPU fetches operands from dual sources: `Load`($src$) via PCIe and `Load`($dst$) via NVLink. Because these instructions target discrete physical interconnects, the SM issues them concurrently. The GPU memory subsystem manages these outstanding requests in parallel, effectively masking the NVLink load latency behind the PCIe load execution. (2) *Thread-Level Parallelism (TLP)*: We exploit the abundant TLP of GPU architectures to hide the latency of the delegated `Store` operation. When a thread warp stalls awaiting the NVLink `Store` (committing the reduced chunk $k$ to the peer's VRAM), the SM scheduler performs a zero-overhead context switch to alternate warps. These active warps proceed to issue `Load` and `Reduce` instructions for chunk $k+1$. This software pipelining guarantees that the supplementary NVLink store does not elongate the critical path of the communication kernel.

*4.5.3 Bandwidth Headroom Analysis.* Finally, we formally quantify the execution time of a bridge GPU relative to the baseline non-bridge path. Let $T_{\text{PCIe}} = \frac{S}{B_{\text{PCIe}}}$ denote the duration of the PCIe load, and $T_{\text{NVLink}} = \frac{2S}{B_{\text{NVLink}}}$ represent the cumulative duration of the dual NVLink operations (one load and one store) for a chunk of size $S$.

Given the **ILP-driven overlap** established above, the theoretical communication time for a bridge GPU ($T_{\text{bridge}}$) is bounded by:

$$T_{\text{bridge}} = \max(T_{\text{PCIe}}, T_{\text{NVLink}}) \tag{5}$$

On our evaluation platform (8×A6000), the measured NVLink Bridge bandwidth ($B_{\text{NVLink}} \approx 89.8$ GB/s) exceeds twice the effective PCIe Gen4 bandwidth ($B_{\text{PCIe}} \approx 42.6$ GB/s). Consequently:

$$T_{\text{NVLink}} = \frac{2S}{B_{\text{NVLink}}} < \frac{S}{B_{\text{PCIe}}} = T_{\text{PCIe}} \implies T_{\text{bridge}} = T_{\text{PCIe}} \tag{6}$$

This inequality formally demonstrates that a bridge GPU's total execution time is strictly bottlenecked by the identical PCIe constraint as non-bridge GPUs. Furthermore, datacenter-class GPUs such as the NVIDIA A100 provide a theoretical NVLink bandwidth of up to 600 GB/s [18]. With such massive bandwidth headroom, the bridge-induced hardware overhead becomes strictly negligible.

## 5 Implementation

The implementation of BridgedRing integrates a minimal hardware modification and a communication kernel. This realization comprises two primary components. First, we architect asymmetric memory channels within the MSCCL++ framework to orchestrate the delegated reduction protocol. Second, we modify the physical node topology by installing a commodity NVLink Bridge across NUMA domains. Together, these elements enable high-bandwidth cross-socket transfers while remaining a drop-in replacement for widely-used collective libraries.

### 5.1 Architecting Asymmetric Memory Channels

The main software challenge in BridgedRing is managing asymmetric data movement and synchronization on the bridge GPUs. Each bridge GPU concurrently reads from an intra-NUMA PCIe neighbor and performs bidirectional memory accesses to cross-NUMA NVLink peer memory. Building upon the `MemoryChannel` abstraction provided by MSCCL++, we extend it to support this delegated access pattern.

*5.1.1 Decoupling Logical Ranks from Physical Memory Ownership.* Conventional collective libraries (e.g., NCCL) assume that each rank writes exclusively to its local destination buffer. BridgedRing relaxes this assumption for the bridge pair. As defined in Algorithm 1, a bridge GPU reads partial results from its upstream intra-NUMA neighbor and stores the reduced result directly into the remote peer's buffer over NVLink.

We leverage two mechanisms to decouple logical ranks from physical memory ownership: (1) *Cross-NUMA IPC Setup:* During initialization, we establish a *Delegated Channel*. This channel imports the peer GPU's VRAM into the bridge GPU's virtual address space via CUDA IPC handles, exposing a valid remote pointer to the bridge kernel for NVLink-routed memory operations. (2) *Role-Specific Buffer Selection:* We implement role-dependent pointer selection inside the CUDA kernel. Instead of the standard `read_prev-reduce-store_local` cycle, the bridge GPU executes a `read_prev_and_peer-reduce-store_peer` sequence. This selection confines the cross-NUMA reduction traffic strictly to the NVLink bridge, bypassing the congested cross-NUMA interconnect.

*5.1.2 Pipelining and Asynchronous Progress.* To maximize the utilization of the heterogeneous PCIe-NVLink fabric, our execution kernel relies on two synergistic mechanisms: (1) micro-chunk pipelining to mask structural access latency, and (2) asynchronous peer signaling to prevent PCIe-side bottlenecks from stalling cross-NUMA writebacks.

**Micro-Chunk Pipelining.** We subdivide each large collective chunk into fine-grained *micro-chunks* (e.g., 256 KiB). For a given micro-chunk, the SM concurrently issues two independent `Load` requests via PCIe and NVLink, respectively. Across successive micro-chunks, the SM effectively hides the NVLink access latency. It achieves this by overlapping the NVLink `Store` of micro-chunk $k$ with the PCIe `Load` of micro-chunk $k + 1$ through rapid warp context switching.

**Asynchronous Peer Signaling.** We implement a non-blocking notification protocol utilizing MSCCL++ peer atomics. The bridge GPU signals delegated-store completion to its NVLink peer immediately upon remote write

visibility. This explicitly decouples the peer notification from any subsequent PCIe-side progress in the pipeline. Consequently, the system ensures independent forward progress across the NUMA boundary.

*5.1.3 Implementation Complexity and Portability.* Our implementation is notably concise, requiring approximately **1,200 lines** of CUDA/C++ code. The kernel effectively masks bridge-node memory latency by exploiting native warp-level parallelism. By offloading transport complexity to the MSCCL++ `MemoryChannel` abstraction, BridgedRing acts as a drop-in replacement for standard NCCL collectives. Moreover, it remains fully compatible with existing deep learning frameworks and the standard CUDA P2P memory model.

## 5.2 Minimal Hardware Realization

The physical layer of BridgedRing requires bridging two cross-NUMA GPUs (e.g., GPU 3 on NUMA 0 and GPU 4 on NUMA 1) with a standard NVLink Bridge. This *topology-reconstruction strategy* requires no modifications to the OS kernel, NVIDIA drivers, or firmware. CUDA automatically recognizes this bridge as a standard peer-to-peer link. The connection is reflected as a high-bandwidth path in the `nvidia-smi topo -m` matrix.

Our implementation includes an automatic topology detection suite. This suite validates the presence of the cross-NUMA link and verifies peer-to-peer accessibility. If the hardware bridge is absent, the system falls back to the standard Ring algorithm. This fallback ensures robust deployment across diverse GPU clusters.

# 6 Evaluation

Our evaluation validates the core thesis of this work: breaking the UPI bottleneck in asymmetric NUMA systems requires *hardware-software co-design*. In Section 6.2, we present microbenchmarks of collective operations to quantify the raw communication speedup enabled by our NVLink Bridge and algorithmic restructuring. In Section 6.3, we demonstrate real-world impact through end-to-end LLM training on GPT models of varying scales. Section 6.4 provides an ablation study that isolates the individual contributions of hardware and software components. In Section 6.5, we analyze why even state-of-the-art topology-aware schedulers cannot eliminate UPI traffic under standard collective semantics, establishing that neither hardware nor software alone suffices to overcome the UPI Wall. Finally, in Section 6.6, we demonstrate that delegated reduction overhead is fully covered by NVLink Bridge acceleration.

## 6.1 Experimental Setup

**Hardware Configuration.** All experiments run on a dual-socket server with two Intel Xeon Silver 4310 CPUs (12 cores each), 1 TB DDR4 memory, and eight NVIDIA RTX A6000 GPUs (48 GB memory each). Each CPU socket hosts four GPUs connected via a PCIe 4.0 switch (PLX PEX880xx), and the two sockets are linked by two 10.4 GT/s UPI interconnects. We evaluate two configurations: (1) *Baseline (Non-Bridge)*: Standard dual-socket PCIe server without hardware modification; (2) *Bridge*: Identical to Baseline, with a single NVLink Bridge installed between GPU 3 (socket 0) and GPU 4 (socket 1), providing 100 GB/s theoretical bidirectional bandwidth.

**Software Stack.** The system runs Debian 12 with Linux kernel 6.1 and Docker 20.10. Our experiments utilize two container environments from NVIDIA: (1) *Collective communication evaluation*: All communication benchmarks (NCCL 2.28.7, nccl-tests 2.17.1, TCCL commit git+351d06, and BridgedRing built on MSCCL++ commit `git+571fee`) execute within the same container environment `nvcr.io/nvidia/cuda:12.2.2-devel-ubuntu22.04`. This ensures fair comparison by eliminating container-induced performance variations; (2) *End-to-end training*: PyTorch 2.5 and Megatron 0.13.1 run in the optimized PyTorch container `nvcr.io/nvidia/pytorch:24.01-py3`, which includes pre-configured CUDA 12.3 and cuDNN 8.9 libraries for training workloads.

**Baselines and Competitors.** To rigorously evaluate our solution and isolate the benefits of hardware-software co-design, we compare BridgedRing against five baselines: (1) *NCCL (Auto)*: The default NCCL implementation with its internal auto-tuning mechanism; (2) *NCCL-Ring*: The standard ring algorithm enforced via

Table 4. Training setups for end-to-end evaluation. All models use GPT-2 vocabulary size (50257 tokens) and 2048 context window. Pipeline parallelism (PP) = 1 for all GPT-style models.

| Model | Parallelism Config | Microbatch Size | Optimizer |
|---|---|---|---|
| *Qwen3-1.7B* | DP=8 (PyTorch DDP) | 1 | Standard |
| *Qwen3-1.7B* | DP=8 (PyTorch DDP) | 2 | Standard |
| GPT-6.7B | TP=8, DP=1 | 1 | Standard |
| GPT-6.7B | TP=8, SP=8, DP=1 | 1 | Distributed |
| GPT-6.7B | TP=8, DP=1 | 2 | Standard |
| GPT-6.7B | TP=8, SP=8, DP=1 | 2 | Distributed |
| GPT-13B | TP=8, DP=1 | 1 | Standard |
| GPT-13B | TP=8, SP=8, DP=1 | 1 | Distributed |

`NCCL_ALGO=Ring`, representing the classic symmetric collective; (3) *NCCL-Intra*: NCCL-Ring executed strictly on four intra-NUMA GPUs, representing the theoretical optimal bandwidth ceiling without cross-NUMA penalties; (4) *Hierarchical-NCCL[31]*: We implement a topology-aware hierarchical baseline using native NCCL primitives, inspired by NUMA-aware communication strategies such as TCCL. The design follows a three-stage pipeline consisting of an intra-NUMA *ReduceScatter*, a cross-NUMA *AllReduce*, and an intra-NUMA *AllGather*. To overlap inter- and intra-NUMA communication, the pipeline employs a three-buffer scheme that enables concurrent execution across stages. This configuration approximates the performance limit of software-based hierarchical routing over the CPU interconnect (e.g., UPI), without introducing additional memory overheads; (5) *NCCL-Bridge*: NCCL-Ring executed with the NVLink Bridge installed, representing the performance limit of hardware-only modifications; (6) *BridgedRing (Ours)*: Our hardware-software co-designed solution that fully bypasses the UPI bottleneck via delegated reduction.

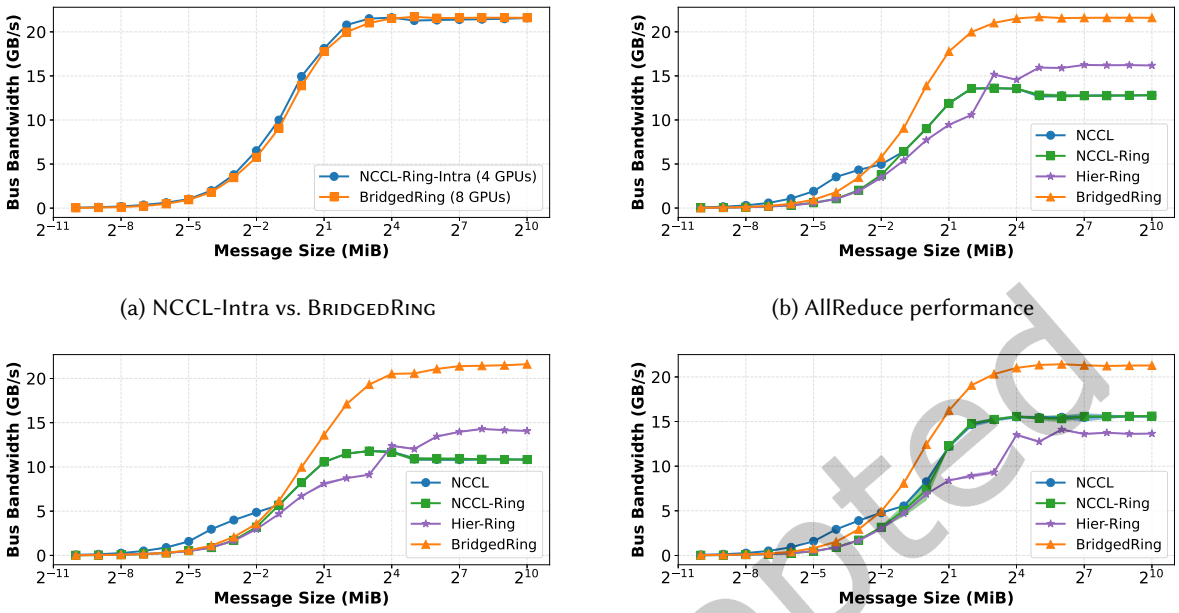
**Workloads.** We evaluate two workload categories: (1) Microbenchmarks: Standard `all_reduce_perf`, `reduce_scatter_perf`, and `all_gather_perf` from nccl-tests [16], with 20 warmup and 50 measurement iterations across message sizes from 1 KiB to 1 GiB; (2) End-to-end training: two BF16 workloads: (a) Qwen3-1.7B [34] with PyTorch DDP; (b) GPT-style models based on GPT-2 [23] (implemented in Megatron-LM [28]): GPT-6.7B (32 layers, 4096 hidden dim, 32 heads) and GPT-13B (40 layers, 5120 hidden dim, 40 heads), with GPT-2 vocabulary (50257 tokens) and 2048-token context (see Table 4).

## 6.2 Microbenchmark Results

Figure 6a compares the effective AllReduce bandwidth achieved by the standard NCCL-Ring on four intra-NUMA GPUs and by BridgedRing on eight cross-NUMA GPUs across varying message sizes. The results show that BridgedRing sustains nearly the same bus bandwidth as the intra-NUMA NCCL baseline, demonstrating that the NVLink Bridge effectively mitigates the UPI Wall even when the communication spans both NUMA domains.

Figure 6b-d compares the throughput of NCCL, NCCL-Ring, Hier-Ring, and BridgedRing across three collective communication primitives: AllReduce, ReduceScatter, and AllGather. The results exhibit distinct behaviors for small and large messages.

For small messages (below roughly 512 KiB), all implementations achieve comparable performance. This result is expected because small collectives are latency-bound. As quantified in Section 3.1, such small messages account




(a) NCCL-Intra vs. BridgedRing

(b) AllReduce performance

(c) ReduceScatter performance

(d) AllGather performance

Fig. 6. (a) AllReduce bus bandwidth comparison. BridgedRing achieves comparable bus bandwidth utilization to NCCL-Ring in a UPI-free topology, effectively breaking the UPI bottleneck. (b-d) Collective communication performance across message sizes for cross-NUMA GPU communication. BridgedRing achieves speedup of 1.67× (AllReduce), 1.97× (ReduceScatter), and 1.40× (AllGather) over NCCL-Ring, with ReduceScatter benefiting most due to its inherent sensitivity to UPI constraints.



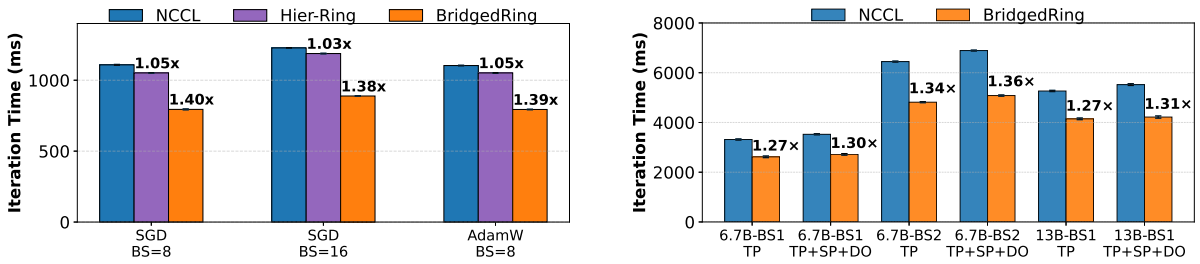
for less than 0.2% of the total data volume in LLM training; thus, the lack of speedup in this regime has a negligible impact on end-to-end performance. Our work primarily targets the bandwidth limitation of large messages.

For large messages, throughput becomes bandwidth-bound, and the performance is dictated by the slowest interconnect. In this regime, a noteworthy observation arises from the NCCL baseline performance: while both AllGather and ReduceScatter suffer from the UPI bottleneck, the performance of **ReduceScatter degrades more significantly than AllGather** in the cross-NUMA setting. This degradation is attributable to its higher **synchronization sensitivity**. ReduceScatter involves computation between data transfers, making its performance more susceptible to the increased latency and coordination overhead inherent in traversing the NUMA domain.

## 6.3 End-to-End Training Performance

Figure 7a shows the training performance of Qwen3-1.7B using PyTorch DDP. BridgedRing consistently reduces iteration time across batch sizes, achieving 29% lower average iteration time (1.40× speedup) compared to NCCL. The improvement is most pronounced at larger batch sizes where communication dominates computation.

The end-to-end speedup (1.40×) is lower than the microbenchmark peak AllReduce speedup (1.67×) because communication constitutes only a fraction of total iteration time. Based on the profiling in Section 3.1, the total communication fraction ($f_{\text{comm}}$) of Qwen3-1.7B is 83.3%. Following Amdahl's Law, if all communication kernels achieved this peak speedup, the theoretical end-to-end speedup would be $\frac{1}{(1-f_{\text{comm}})+f_{\text{comm}}/1.67} \approx 1.50\times$. Our measured result of 1.40× falls slightly below this upper bound. This gap arises because our optimization




(a) DDP speedup

(b) TP SP speedup

Fig. 7. (a) Qwen3-1.7B training time. BridgedRing reduces iteration time by 28.6% (1.40× speedup) compared to NCCL. (b) GPT model training time across six experimental configurations. BridgedRing yields up to 1.36× speedup over NCCL.

delivers relatively lower speedup on smaller collective operations, which moderately reduces the overall average communication gain across the entire training trace.

Figure 7b demonstrates end-to-end training performance across six configurations using GPT-2 vocabulary (50257 tokens) and 2048 context window. We exclude hierarchical collective algorithms from this baseline comparison. The Megatron-LM framework enforces a runtime constraint of `CUDA_DEVICE_MAX_CONNECTIONS = 1`, which forces existing hierarchical AllReduce implementations to degenerate into sequential execution. This severe serialization degrades interconnect bandwidth to only 9.66 GB/s—well below the native NCCL baseline of 12.77 GB/s. Consequently, this degraded scheme offers no meaningful comparative value for our evaluation. The end-to-end training performance results reveal two key insights:

**Microbatch size drives acceleration.** For pure tensor parallelism (TP=8), BridgedRing achieves up to 1.34× speedup over NCCL. Moreover, larger microbatch sizes consistently yield greater acceleration—1.34× for batch size 2 versus 1.27× for batch size 1—confirming that workloads with higher communication intensity benefit most from eliminating the UPI bottleneck.

**Communication-dominated workloads show amplified benefits.** When combining tensor parallelism with sequence parallelism and a distributed optimizer (TP+SP+DO), BridgedRing achieves a modestly higher speedup (1.31×) compared to pure tensor parallelism (1.27×) in GPT-13B. This slight improvement aligns with expectations: SP and DO increase cross-NUMA communication volume through AllGather and ReduceScatter operations, making performance more sensitive to UPI bottlenecks. Consequently, BridgedRing's elimination of UPI traffic yields proportionally greater gains as communication overhead becomes more dominant.

These results show that BridgedRing's benefits increase with communication intensity. It provides the largest gains for modern LLM training configurations, where high parallelism and optimization strategies increase cross-NUMA communication demand.

## 6.4 Ablation Study: Hardware vs. Software Contributions

Figure 8 isolates the impact of the NVLink Bridge hardware. Adding the bridge to standard NCCL (NCCL-Bridge) yields up to 1.67× speedup for ReduceScatter and 1.47× for AllReduce. This substantial gain occurs because: (1) NVLink Bridge provides a high-bandwidth path that bypasses UPI for one communication hop; (2) NVLink Bridge reduces UPI contention by eliminating bidirectional traffic on the interconnect; (3) NCCL's topology-aware scheduler automatically utilizes the highest-bandwidth path available.

However, Figure 9 reveals that hardware alone is insufficient. Even with the NVLink Bridge, NCCL-Bridge still requires one hop across the UPI interconnect due to the cyclic dependency of ring collectives (e.g., GPU 7 must

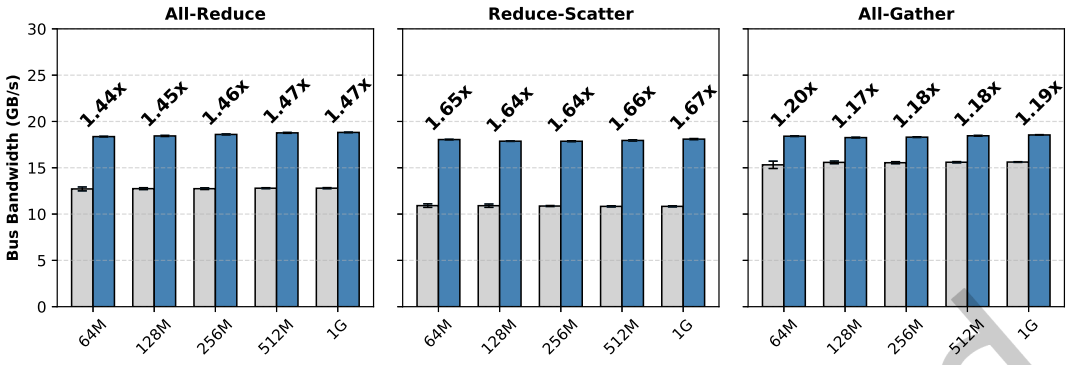



Fig. 8. Hardware-only contribution: NVLink Bridge with NCCL yields up to 1.67× speedup by eliminating UPI contention.

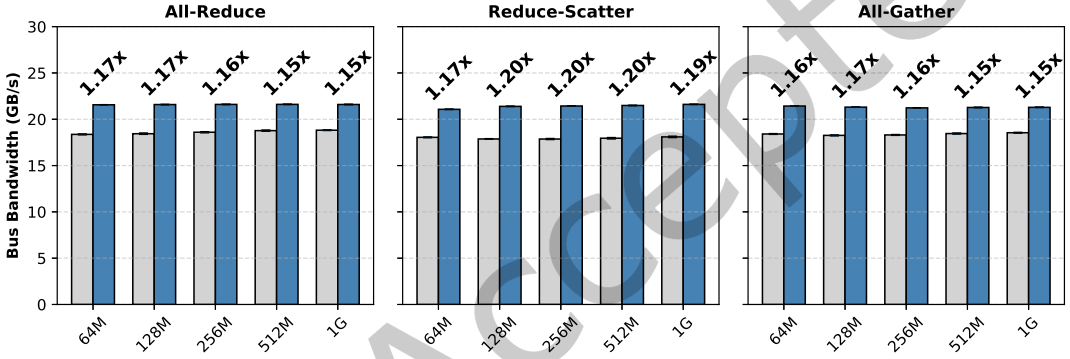



Fig. 9. Software-only contribution: BridgedRing provides up to 1.20× speedup over NCCL-Bridge by eliminating UPI traffic.

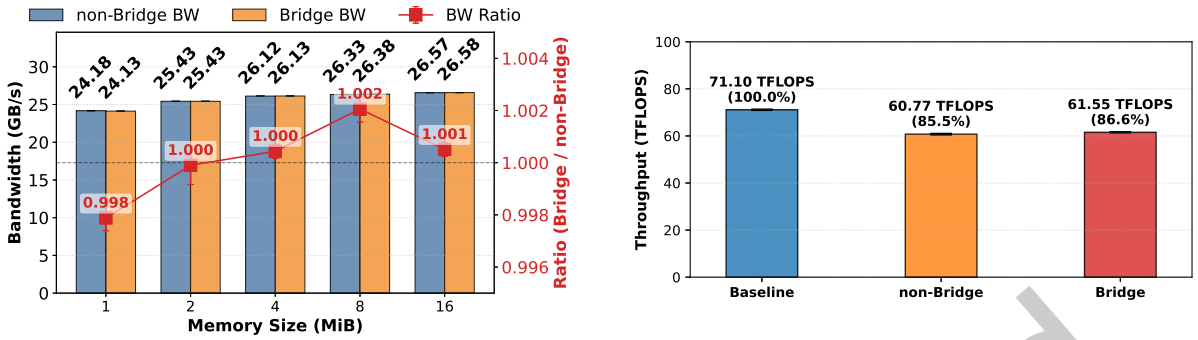
communicate with GPU 0). BridgedRing eliminates this limitation through algorithmic restructuring, providing an additional up to 1.20× speedup by performing cross-domain communication exclusively through the NVLink Bridge and eliminating all UPI-traversing traffic by algorithmic design.

The combined hardware-software approach achieves 1.97× speedup over the baseline NCCL-Ring implementation, demonstrating the necessity of co-design.

## 6.5 Limitations of Pure Software Approaches: TCCL Case Study

**TCCL Cannot Eliminate UPI Traffic on Standard Collective Semantics.** We compare against TCCL [13], a state-of-the-art topology-aware scheduler that uses exhaustive search to find optimal communication schedules. On a 4-GPU cross-NUMA configuration, TCCL achieves 16.64 GB/s for AllGather, which is still 20.5% slower than intra-NUMA execution (20.92 GB/s). Analysis reveals that TCCL's optimal schedule still requires one hop across the UPI boundary due to the cyclic dependency of ring collectives. This confirms a fundamental limitation: **no scheduling optimization can eliminate UPI traffic while preserving standard ring collective semantics**.

**Scalability Failure at Full System Scale.** When scaling to 8 GPUs, TCCL completes schedule generation but fails at runtime. Its execution engine attempts to allocate excessive shared memory for cross-NUMA communication, causing system instability despite 1 TB of host memory.




(a) Effective bandwidth

(b) GEMM throughput

Fig. 10. Microbenchmark results on an isolated 3-GPU setup. (a) compares bandwidth across physical paths; (b) shows GEMM throughput with and without concurrent communication.

**The Co-Design Advantage.** In contrast to the preceding two limitations, BridgedRing achieves 21.28 GB/s on 8-GPU cross-NUMA AllGather—matching the NCCL intra-NUMA performance (20.92 GB/s)—by restructuring the collective operation. This result demonstrates that closing the NUMA performance gap requires moving beyond monolithic ring semantics, a limitation that persists even in state-of-the-art topology-aware schedulers.

## 6.6 Bridge GPU Profiling and Straggler Analysis

To verify that the dual-interconnect responsibilities of bridge GPUs do not introduce measurable straggler overhead, we evaluate their execution behavior across two granularities. First, we analyze the end-to-end training timeline to assess system-level runtime skew. Second, we isolate the delegated-reduction path via a microbenchmark to quantify fine-grained interference.

*6.6.1 End-to-End Training Timeline.* To assess the end-to-end impact of the delegated reduction, we examine the GPU timeline alignment during a training workload. Specifically, we trace the training process of a Qwen3-1.7B model using PyTorch DDP. In this setup, gradient synchronization (AllReduce) occurs concurrently with the backward pass. We utilize `nsys profile` to compare the execution timelines of bridge and non-bridge GPUs.

The profiling results demonstrate high execution symmetry between the two node types. The backward-pass timeline differs only slightly between bridge and non-bridge GPUs (260 ms versus 258 ms), and we use this small gap only as evidence that no obvious bridge-specific tail emerges rather than as a claim of statistically significant per-GPU runtime difference. A granular inspection likewise reveals no visible "tail" latency extension on the bridge nodes. Together, these observations indicate that the delegated reduction mechanism does not introduce measurable runtime skew into the end-to-end training pipeline.

*6.6.2 Isolated Path Performance and Interference.* To accurately isolate the delegated-reduction path from global ring synchronization effects, we design a controlled 3-GPU microbenchmark. Specifically, GPU 0 and GPU 1 communicate exclusively via PCIe, whereas GPU 1 and GPU 2 connect via an NVLink Bridge. Consequently, this structural setup effectively replicates the local-to-remote connectivity of a bridge node within our 8-GPU logical ring. Using this isolated topology, we evaluate two specific fine-grained behaviors: path bandwidth isolation and compute interference.

*Path Isolation Benchmark.* We compare two reduction scenarios on the test node (GPU 1). First, in the native PCIe path, GPU 1 fetches a chunk from GPU 0 via PCIe and reduces it into local memory. This represents the

standard ring behavior of a non-bridge GPU. Second, in the bridge-mediated path, GPU 1 fetches a chunk from GPU 0 via PCIe and reduces it into the remote memory of GPU 2 via NVLink. This mirrors the delegated reduction logic defined in Algorithm 1.

As shown in Figure 10a, the effective bandwidth of the bridge-mediated path perfectly matches the native PCIe path. This empirical result validates our analytical model from Section 4.5. Because the NVLink headroom ($B_{\text{NVLink}} \approx 89.8$ GB/s) substantially exceeds the PCIe bandwidth ($B_{\text{PCIe}} \approx 42.6$ GB/s), the bridge GPU fully absorbs the additional memory traffic without extending the execution window of the standard PCIe load.

*Compute Interference Analysis.* To quantify the compute interference from active delegation, we measure the throughput of a `cuBLAS GEMM` workload on GPU 1 during concurrent reduction operations. Figure 10b presents the GEMM TFLOPS across three scenarios: (1) standalone execution, (2) concurrent with standard PCIe reduction, and (3) concurrent with bridge-mediated reduction.

The empirical results reveal two key insights regarding resource contention. First, concurrent communication inevitably consumes a fraction of shared GPU resources, causing both the standard PCIe scenario (85.5%) and the bridge-mediated scenario (86.6%) to degrade relative to the standalone baseline (100%). Second, however, the bridge-mediated execution (86.6%) performs no worse than the non-bridge baseline (85.5%). This observation confirms that the specific act of delegation imposes negligible supplementary overhead, ensuring that bridge nodes will not become compute stragglers during actual training.

## 7 Related Work

### 7.1 Topology-aware Collective Algorithms

Existing research has extensively explored software-only approaches to optimize collective communications through topology awareness [8, 13, 32]. These methods improve communication efficiency by carefully mapping communication patterns to underlying hardware architectures, such as NUMA domains and PCIe hierarchies.

A significant advancement in this domain is **collective algorithm synthesis**, exemplified by SCCL [2], TCCL [13], TACCL [25], and TACOS [32]. These frameworks automatically generate optimized communication schedules for given physical topologies by formulating collective planning as a constrained optimization problem. They can reduce cross-NUMA traffic and balance load across PCIe links.

However, these sophisticated algorithms, while representing the state-of-the-art in software optimization, are constrained by the physical bandwidth of underlying interconnects, particularly the UPI link between CPU sockets. As we demonstrate in Section 6.5, even the best schedule cannot eliminate UPI hops under standard ring semantics. Consequently, they remain bound by physical constraints and unable to address root causes like the *UPI Wall*. This limitation motivates our co-design approach: rather than searching for better schedules on a bottlenecked topology, we augment the topology itself with a minimal, cost-effective hardware modification.

### 7.2 Dedicated Hardware Interconnects

In contrast to software approaches, dedicated hardware solutions provide significant performance improvements through high-bandwidth, low-latency interconnects. Prominent examples include NVIDIA's SXM-based platforms like the DGX series, which integrate NVLink and NVSwitch technologies [18]. These technologies bypass CPU and PCIe hierarchies, eliminating bottlenecks like UPI and offering superior collective performance.

The critical limitation of these solutions is their **prohibitive cost** and **proprietary nature**, requiring custom server designs that are largely inaccessible for commercial off-the-shelf (COTS) PCIe-based GPU servers. While highly effective, they do not address the need for cost-effective improvements in standard server environments that constitute the majority of AI research and deployment infrastructure. BridgedRing bridges this gap: it introduces a targeted hardware augmentation—an NVLink Bridge between a pair of cross-NUMA GPUs—that delivers near-intra-NUMA performance without requiring full NVSwitch or SXM redesign.

### 7.3 Programmable Communication Frameworks

Programmable collective communication libraries, such as MSCCL [3, 4] and MSCCL++ [26], provide foundational frameworks for developing custom collective algorithms. These libraries offer flexible APIs and domain-specific languages that enable fine-grained control over data movement and computation placement.

This programmability is particularly valuable for exploring hardware-specific optimizations and adapting collective operations to diverse system architectures. We leverage such frameworks to implement BridgedRing as a topology-aware variant of Ring algorithms that redirects critical cross-NUMA traffic through the added NVLink path, demonstrating how programmable collectives can unlock the potential of modest hardware enhancements.

## 8 Discussion

### 8.1 The Worsening UPI Wall and Future Interconnects

The UPI bottleneck will exacerbate due to a fundamental scaling disparity. Our measurements show that from Intel Xeon Ice Lake (Gold 6330) to Emerald Rapids (Platinum 8558), the bidirectional UPI bandwidth increased from 14.35 GB/s to only 24.04 GB/s—a mere 1.7× improvement over two generations. In contrast, PCIe bandwidth doubles with each generation (e.g., 32 GB/s to 64 GB/s from Gen4 to Gen5), and modern GPU interconnects (e.g., NVLink on H100 [20]) deliver up to 600 GB/s. Consequently, this stagnant UPI scaling will increasingly penalize cross-socket communication, effectively nullifying the bandwidth gains of peripheral accelerators.

Looking beyond current multi-socket limits, emerging interconnect standards will reshape this landscape. However, it is necessary to distinguish between host-centric and accelerator-native fabrics when evaluating their impact on intra-node collectives. Compute Express Link (CXL) [27] leverages advanced PCIe physical layers to expand host-device and cross-NUMA bandwidth. Yet, CXL remains a host-managed hierarchy. Routing GPU collective traffic through CXL still consumes host root complex resources and competes with cache coherency protocols. In such environments, the physical bypass principle of BridgedRing remains architecturally valuable. Conversely, Ultra Accelerator Link (UALink) [30] aims to standardize all-to-all, direct accelerator connectivity, structurally mirroring high-end NVLink switch fabrics. In these future scale-up pods, the cross-NUMA host bottleneck is fundamentally eliminated. Consequently, the deployment of such fabrics will inevitably diminish the specific relevance of BridgedRing.

Nevertheless, adopting these next-generation fabrics often requires substantial infrastructure redesign and premium hardware investment. For the installed base of commodity dual-socket PCIe servers and near-term cost-sensitive deployments, BridgedRing offers an immediately deployable, approximately $200 co-design solution that bypasses the UPI Wall and improves training efficiency.

### 8.2 Potential Extensions

Our current prototype targets dual-socket NVIDIA servers. However, the core design relies entirely on generalized hardware-software primitives. These primitives include a physical P2P bypass and explicit remote memory access. This generalization provides a conceptual foundation for extending BridgedRing along two primary dimensions: (1) cross-vendor adaptability and (2) complex multi-socket topologies.

**Cross-Vendor Adaptability.** Our control logic is largely hardware-agnostic. It builds upon MSCCL++ [26] to access unified P2P communication abstractions across both CUDA and ROCm. Consequently, the BridgedRing architecture could theoretically be ported to AMD Instinct platforms. In such a hypothetical setup, Infinity Fabric links would provide the requisite cross-NUMA physical bypass. As we have not yet evaluated this empirically, we defer cross-vendor physical validation to future work.

**Complex Multi-Socket Topologies.** This design paradigm also offers a conceptual path to scale beyond dual-socket constraints. Consider a hypothetical quad-socket server as an example. The system could first compose

localized GPU rings within individual NUMA domains. It would then stitch adjacent domains together using explicit cross-socket bridge links (e.g., S0-S1-S2-S3).

### 8.3 Deployment Considerations

To assess the practical utility of BridgedRing, we analyze its integration into widely-used training infrastructures along two dimensions: structural compatibility and system-wide performance.

**Integration with Existing Stacks.** Many widely-used training systems for large-scale models adopt a hierarchical collective structure. These architectures separate communication into an intra-node phase and an inter-node phase. BridgedRing specifically targets the intra-node execution. This targeted substitution is feasible because the bridge-mediated reduction mirrors the standard PCIe traffic pattern. As established in Section 4.5, it introduces no supplementary load to the host PCIe fabric. Consequently, BridgedRing can be integrated as a drop-in intra-node schedule. It requires no modifications to the inter-node communication protocols.

**System-Wide Performance Bounds.** The end-to-end benefit of this substitution depends on the workload's critical path. The UPI Wall is a strictly local hardware bottleneck. Therefore, the intra-node bandwidth advantage of BridgedRing remains constant regardless of the total cluster scale. However, following Amdahl's Law, the aggregate impact on training speed will diminish as inter-node network latency consumes a larger share of the total iteration time. This occurs frequently in massive clusters spanning hundreds of nodes. The deployment value of BridgedRing is therefore highest in environments where intra-node collectives remain a prominent bottleneck within the global communication phase.

### 8.4 Interaction with Inter-Node Data Parallelism

Beyond intra-node optimization, BridgedRing's impact on inter-node parallelism also affects end-to-end performance in large-scale clusters. This impact depends primarily on the server's NIC attachment topology, which varies across deployments. We discuss two common configurations:

**NIC-on-CPU Topology**: When the NIC is attached to the CPU's PCIe root complex, inter-node traffic must traverse the GPU, the PLX switch, and the CPU root complex before reaching the NIC. In a standard NCCL Ring, cross-NUMA traffic follows a similar path and further traverses the cross-NUMA interconnect (e.g., UPI), resulting in contention on the shared GPU–PLX–CPU PCIe path. By redirecting cross-NUMA collective traffic to the NVLink Bridge, BridgedRing reduces the amount of collective traffic that would otherwise share this path. In such a topology, this reduction in contention leaves more PCIe bandwidth available for inter-node RDMA traffic, potentially improving end-to-end performance beyond the intra-node communication gain itself.

**NIC-on-PLX Topology**: In systems where the NIC and GPUs are attached to the same PLX switch, inter-node traffic is routed locally through that switch without traversing the CPU root complex or the cross-NUMA interconnect. In this topology, the GPU-to-NIC path is largely separate from the cross-socket path that BridgedRing optimizes for intra-node collectives. As a result, BridgedRing mainly changes intra-node communication, and its direct effect on GPU-to-NIC transfers is expected to be limited.

Overall, the interaction between BridgedRing and inter-node data parallelism depends on NIC placement. CPU-attached NICs may benefit indirectly from the reduction in shared PCIe/CPU-path contention, whereas PLX-attached NICs are more likely to see little direct effect.

## 9 Conclusion

In this paper, we introduced BridgedRing, a novel hardware-software co-design that overcomes the cross-NUMA performance barrier for collective communication on standard multi-socket servers. We identified the **UPI Wall** as a fundamental **physical limitation** that standard algorithms and software-only optimizations cannot fully overcome. To bypass this barrier, we first proposed a cost-effective hardware modification: adding an NVLink

Bridge to create a high-speed channel between a pair of cross-NUMA GPUs. Complementing this hardware modification, we further proposed **BridgedRing algorithm**, a topology-aware variant of Ring algorithm that intelligently redirects critical cross-NUMA traffic to this dedicated path.

Our comprehensive evaluation demonstrates BridgedRing's effectiveness: microbenchmarks show a up to **1.97×** speedup in collective communications bandwidth (effectively matching NCCL intra-NUMA performance), while end-to-end training achieves up to **1.40×** throughput improvement. As a practical, scalable, and cost-effective solution compatible with existing server architectures, BridgedRing provides a vital low-barrier upgrade path for commercial server ecosystems, enabling more efficient large-scale deep learning training without requiring specialized hardware.

## Acknowledgments

We sincerely thank the anonymous referees for their valuable comments and helpful suggestions. This work is supported by the Shenzhen Science and Technology Program under grant number (ZDCYKCX20250901092700001, SYSPG20241211173609009, KCXFZ20240903093006009), and the National Natural Science Foundation of China under grant number (62272130).

## References

[1] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)* 52, 4 (2019), 1–43.

[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*. ACM, 62–75.

[3] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. 2022. MSCCL: microsoft collective communication library. *arXiv preprint arXiv:2201.11840* (2022).

[4] 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*. 502–514.

[5] Dell Inc. 2019. *DSS 8440 Spec Sheet.* Specification Sheet H17992. Dell Technologies. https://www.delltechnologies.com/asset/en-us/products/servers/briefs-summaries/dellemc-dss8440-spec-sheet.pdf

[6] Frank Denneman. 2020. PCIe Device NUMA Node Locality. *FrankDenneman.nl Blog* (January 2020). https://frankdenneman.nl/2020/01/10/pcie-device-numa-node-locality/

[7] Exxact Corporation. 2024. SXM vs PCIe: GPUs Best for Training LLMs like GPT-4. https://www.exxactcorp.com/blog/deep-learning/sxm-vs-pcie-gpus-best-for-training-llms-like-gpt-4. Accessed on October 27, 2025.

[8] Zhiyi Hu, Siyuan Shen, Tommaso Bonato, Sylvain Jeaugey, Cedell Alexander, Eric Spada, James Dinan, Jeff Hammond, and Torsten Hoefler. 2025. Demystifying NCCL: An in-depth analysis of GPU communication protocols and algorithms. *arXiv preprint arXiv:2507.04786* (2025).

[9] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems* 32 (2019).

[10] Intel Corporation. 2017. *Intel Xeon Processor Scalable Family Technical Overview.* Technical Report. Intel Corporation. White Paper.

[11] Sylvain Jelger. 2019. Massively Scale Deep Learning Training with NCCL 2.4. (Apr 2019). https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/

[12] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[13] Heehoon Kim, Junyeol Ryu, and Jaejin 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*. 999–1015.

[14] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. 2020. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[15] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 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*. 1–15.

[16] NVIDIA. 2022. *nccl-tests: Tests for the NVIDIA Collective Communications Library (NCCL).* https://github.com/NVIDIA/nccl-tests
[17] NVIDIA. 2023. *nvbandwidth: A tool for bandwidth measurements on NVIDIA GPUs.* https://github.com/NVIDIA/nvbandwidth
[18] NVIDIA. 2020. *NVIDIA DGX A100 System Architecture.* Technical Report. NVIDIA Corporation. WP-10021-001_v1.1.
[19] NVIDIA. 2020. *NVIDIA RTX A6000 Datasheet.* Data Sheet. NVIDIA Corporation. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/proviz-print-nvidia-rtx-a6000-datasheet-us-nvidia-1454980-r9-web%20(1).pdf Document nvidia-1454980-r9.
[20] NVIDIA. 2024. *NVIDIA H100 Tensor Core GPU Datasheet.* Data Sheet. NVIDIA Corporation. https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306
[21] NVIDIA. 2025. *NVLink High-Speed GPU Interconnect.* NVIDIA Corporation. https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/
[22] PNY Technologies. 2025. *RTX A6000 NVLink Bridge 3-Slot.* https://www.amazon.com/dp/B0953WXQHX
[23] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).
[24] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).
[25] 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)*. 593–612.
[26] Aashaka Shah, Abhinav Jangda, Binyang Li, Caio Rocha, Changho Hwang, Jithin Jose, Madan Musuvathi, Olli Saarikivi, Peng Cheng, Qinghua Zhou, et al. 2025. MSCCL++: Rethinking GPU Communication Abstractions for Cutting-edge AI Applications. *arXiv preprint arXiv:2504.09014* (2025).
[27] Debendra Das Sharma. 2022. Compute Express Link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. *2022 IEEE Symposium on High-Performance Interconnects (HOTI)* (2022), 5–12. https://api.semanticscholar.org/CorpusID:252997701
[28] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).
[29] Supermicro. 2025. *GPU SuperServer SYS-420GP-TNR.* Super Micro Computer, Inc. https://www.supermicro.com/en/products/system/gpu/4u/sys-420gp-tnr
[30] UALink Consortium. 2024. Industry Leaders Form Ultra Accelerator Link (UALink) Promoter Group to Drive Data Center AI Connectivity. Official Press Release. https://ualinkconsortium.org/news/
[31] Yuichiro Ueno and Rio Yokota. 2019. Exhaustive Study of Hierarchical AllReduce Patterns for Large Messages Between GPUs. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 430–439. doi:10.1109/CCGRID.2019.00057
[32] 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)*. IEEE, 856–870.
[33] Qifan Xu and Yang You. 2023. An efficient 2d method for training super-large deep learning models. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 222–232.
[34] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).
[35] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).
[36] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3848–3860.

Received 22 November 2025; revised 28 April 2026; accepted 4 June 2026