



Demystifying NCCL: An In-depth Analysis of GPU Communication Protocols and Algorithms

ZHIYI HUM SIYUAN SHEN, TOMMASO BONATO, SYLVAIN JEAUGEY, CEDELL
ALEXANDER, ERIC SPADA, JAMES DINAN, JEFF HAMMOND, TORSTEN HOEFLE

PRESENTED BY YUSONG YAN – CSE 561 AUTUMN 2026 READING PRESENTATION

Contents

- ▶ High Level Ideas of this paper
- ▶ What is NCCL (NVIDIA Collective Communication Library)
- ▶ The three communication protocols in NCCL – trade-offs
 - ▶ Simple, LL (Low Latency), LL128 (Low Latency – 128 Bytes)
- ▶ How data transfer between GPUs in this system
- ▶ Collective Algorithms in NCCL
- ▶ Benchmark results analysis
- ▶ Open Question & Discussions

NCCL – High Level Idea

- ▶ When multiple Nvidia GPUs need to compute data together (collaboratively), need to design a protocol to arrange data movement between those GPUs
- ▶ Essential for achieving high performance in distributed AI and HPC workloads
 - ▶ You are using four GPUs to compute a large gradient (for example)
 - ▶ Each GPU computes a different portion of the result efficiently, but the data (stored in the GPU's memory – that's on the GPU board) can not be accessed directly by other GPUs
 - ▶ Need to implement a protocol to combine those gradients together, and let each individual GPU know the combined result – so called AllReduce
- ▶ Emphasizing low latency and high bandwidth

What is NCCL – The API

- ▶ Each GPU corresponds to a communicator object to invoke NCCL operations
- ▶ Initialization – `ncclCommInitAll/ncclCommInitRank` create communicators
- ▶ Termination – `ncclCommDestroy (safely)/ncclCommAbort (force)` destroy/terminate communicators
- ▶ Collective Communication (AllReduce = ReduceScatter + AllGather)
 - ▶ `ncclAllReduce` – Perform reduce (sum, prod, etc.) on datas of GPUs, send complete result to all GPUs
 - ▶ `ncclBroadcast` – Send data from a specific (root) GPU to all other GPUs
 - ▶ `ncclReduce` – Perform reduce on datas of GPUs, save only to specific (root) GPU
 - ▶ `ncclAllGather` – Collect data from GPUs, concatenate data based on Rank, send to all GPUs
 - ▶ `ncclReduceScatter` – Perform reduce, split the compute result, send to each GPU correspondingly
- ▶ Point-to-point Communication – `ncclSend/ncclRecv`
- ▶ Group Calls – bracket sequence of NCCL calls & delay execution until group ends via `ncclGroupStart` & `ncclGroupEnd`

What is NCCL – Execution Models

- ▶ One CPU process per GPU
 - ▶ Greater control over process placement
 - ▶ Improve data locality via NUMA memory
- ▶ One CPU thread per GPU
 - ▶ Efficient intra-thread memory sharing (between GPUs)
 - ▶ Allow direct cross rank memory access (GPU buffer) reduce copy-use overhead
- ▶ One CPU thread for multiple GPUs
 - ▶ Low CPU thread/process scheduling overhead
 - ▶ Deterministic execution – best for small scale prototyping

What is NCCL – Communication Channels

- ▶ A single SM (Streaming multiprocessor) might not be able to fully utilize the GPU and/or network resources, so NCCL provide multiple parallel communication channels
- ▶ Each channel is launched as a separate CUDA block, which will process a portion of data, and use the NCCL defined Ring or Tree topology for communication
 - ▶ Ring topology: each GPU identifies its immediate predecessor and successor to form a unidirectional communication ring
 - ▶ Tree topology: each GPU tracks its parent and child ranks, establishing a logical communication tree
- ▶ Using too much channels will cause the data chunk small for each individual communication channel, and thus reduce the utilization of the NIC (Network Interface Card)'s bandwidth, and thus might reduce the performance
 - ▶ NCCL has heuristic to reduce number of channel for small messages
- ▶ More Channel gives higher GPU parallelism, while larger chunks gives better network efficiency

The three communication protocols in NCCL – trade-offs

TABLE I
COMPARISON OF NCCL COMMUNICATION PROTOCOLS

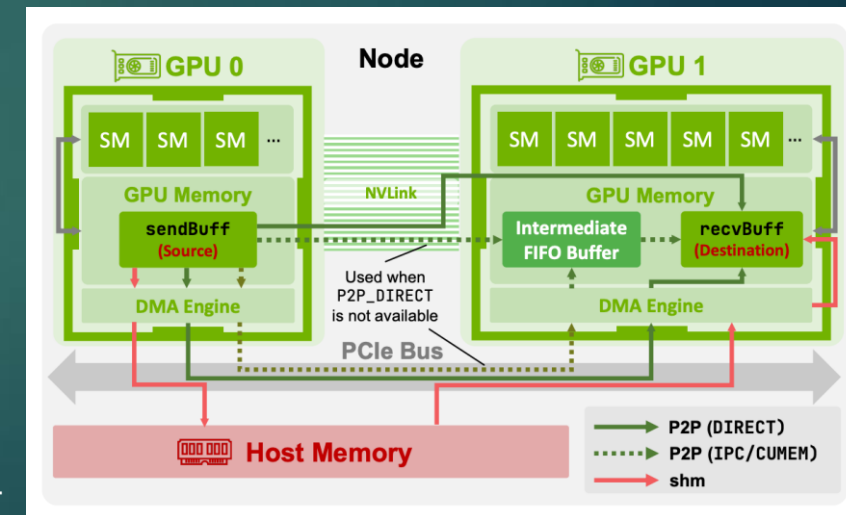
	Simple	LL	LL128
Design Goal	High bandwidth	Low latency	Low latency and high bandwidth
Synchronization Mechanism	Memory fences (high overhead)	Flag-based synchronization	Flag-based synchronization
Payload	Data chunks	4B data + 4B flag	120B data + 8B flag
Bandwidth Utilization	Near peak	25 ~ 50% of peak [11]	~ 95% of peak [11]
Latency Per-hop	$\sim 6\mu s$	$\sim 1\mu s$	$\sim 2\mu s$

How data transfer between GPUs in this system

- ▶ Distinct data transfer strategies & transport mechanism depending on scenario (same machine intra-node, or cross machine inter-node)
- ▶ Intra-node Data Transfer (source and dest GPU are on the same server/node)
 - ▶ Priority: NVLink Peer-to-peer (P2P) > PCIe GPUDirect P2P > Shared Memory (SHM)
 - ▶ NCCL P2P bypasses the need for IPC (Inter Process Communication) handles by employing direct GPU memory pointers within the same address space
 - ▶ NCCL transfer data directly between source and destination buffers rather than routing through an intermediate FIFO buffer
 - ▶ If P2P communication is not available, will fallback to SHM
 - ▶ If NVLink not available, and two GPUs are attached to different CPU socket, might fallback from PCIe GPUDirect P2P to SHM

TABLE II
NCCL COMMUNICATION CHARACTERISTICS AND TRANSPORTS

	Intra-Node		Inter-Node	
Transport	P2P	p2p.cc	NET	net_ib.cc
	SHM	shm.cc		net_socket.cc
	NVLS	nvls.cc	COLLNET	coll_net.cc
Physical Interconnect	NVLink		InfiniBand	
	PCIe		RoCE	
			TCP/IP (Socket)	
Optimizations	GPUDirect P2P		GPUDirect RDMA	
	P2P_DIRECT			

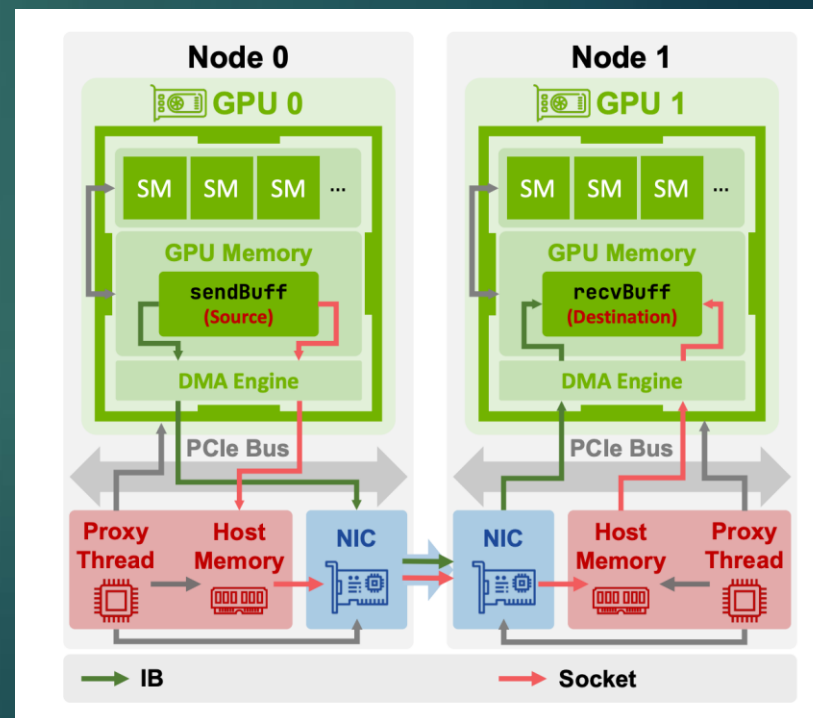


How data transfer between GPUs in this system

- ▶ Inter-node Data Transfer (source and dest GPU are NOT on the same server/node)
 - ▶ Use standard TCP Socket or High-Perf. InfiniBand Verbs
 - ▶ Socket-Based Communication (when no RDMA available)
 - ▶ Host (main) memory allocated for CUDA mapped memory
 - ▶ Sender first copy from GPU to host memory, then go to socket
 - ▶ Receiver get data from host memory, need to copy to GPU
 - ▶ Reliance on host memory create extra memory copy overhead
 - ▶ InfiniBand Verbs (IB Verbs) Transport (with RDMA available)
 - ▶ Enable direct data movement with minimal CPU intervention
 - ▶ Transfers staged though an intermediate buffer, location based on hardware support and configuration (host memory if NIC can't access the GPU memory)

TABLE II
NCCL COMMUNICATION CHARACTERISTICS AND TRANSPORTS

	Intra-Node		Inter-Node	
Transport	P2P	p2p.cc	NET	net_ib.cc
	SHM	shm.cc		net_socket.cc
	NVLS	nvls.cc	COLLNET	coll_net.cc
Physical Interconnect	NVLink		InfiniBand	
	PCIe		RoCE	
			TCP/IP (Socket)	
Optimizations	GPUDirect P2P P2P_DIRECT		GPUDirect RDMA	

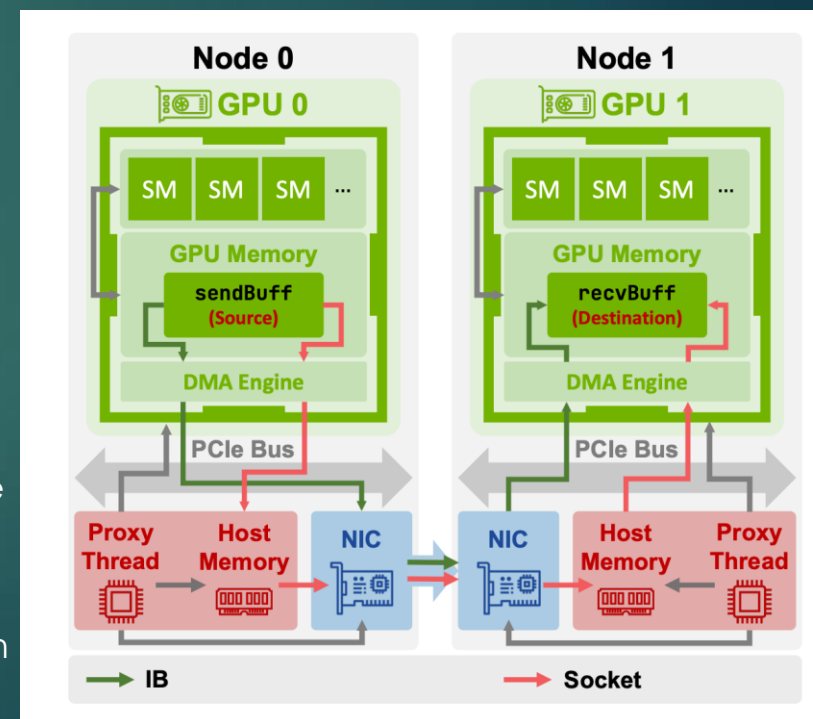


How data transfer between GPUs in this system

- ▶ Inter-node Data Transfer (source and dest GPU are NOT on the same server/node)
 - ▶ NCCL's InfiniBand Verbs Optimizations
 - ▶ GPUDirect RDMA Optimization
 - ▶ Host memory staging introduced additional wasty memory copy and occupies PCIe bandwidth
 - ▶ This optimization enables NIC to access GPU memory directly
 - ▶ Per-peer Multi-channel Connections
 - ▶ Improve bandwidth utilization and reduce connection by default creating 2 logical channels per remote GPU-NIC communication with round-robin scheduling
 - ▶ QP (Queue Pair) Layout
 - ▶ Separate CTS control traffic with high bandwidth data transfer to reduce head-of-line blocking
 - ▶ Local Flush with Loop-back RDMA_READ
 - ▶ Ensures the data written to GPU memory by the NIC is completely written and visible to the GPU before GPU is using the data – reduce waiting

TABLE II
NCCL COMMUNICATION CHARACTERISTICS AND TRANSPORTS

	Intra-Node		Inter-Node	
Transport	P2P	p2p.cc	NET	net_ib.cc
	SHM	shm.cc		net_socket.cc
	NVLS	nvls.cc	COLLNET	coll_net.cc
Physical Interconnect	NVLink PCIe		InfiniBand RoCE TCP/IP (Socket)	
Optimizations	GPUDirect P2P P2P_DIRECT		GPUDirect RDMA	



Collective Algorithms in NCCL

- ▶ Communication Primitives

- ▶ Including send, recv, recvReduceSend, and recvReduceCopySend, represent distinct data movement or computation pattern
- ▶ Those basic communication primitives together forms complicated collective algorithm

- ▶ Iterative Execution

- ▶ Input -> Channels -> Loops -> Chunks -> Element
- ▶ Each channel is in charge of its own disjoint data region, and sub-divide to loops, chunks and operate data iteratively
- ▶ Buffer slots are use to pipeline the process, so different data chunks are in different stages (compute, send, recv)

- ▶ CUDA Hardware mapping

- ▶ Grid -> Block -> Channel -> Warp -> Threads
- ▶ All threads in a warp perform the same sequence of operations on different data to utilize the SMT property of GPU
- ▶ Maintain multiple concurrent data movement pipelines simultaneously -> high bandwidth utilization

Collective Algorithms in NCCL

► Ring AllReduce (k participating GPUs)

- Step 0: each GPU send one segment of its local data to its neighbor
- Next k-2 steps: each GPU repeatedly executes a `recvReduceSend` – receives a data segment from preceding neighbor, perform reduction and forward to the subsequent GPU
- Next k-2 steps: each GPU executes `recvCopySend` – receives reduced segment, copy to output buffer, forward to subsequent GPU
- Step 2k-2: each GPU perform `recv` to finish the operation

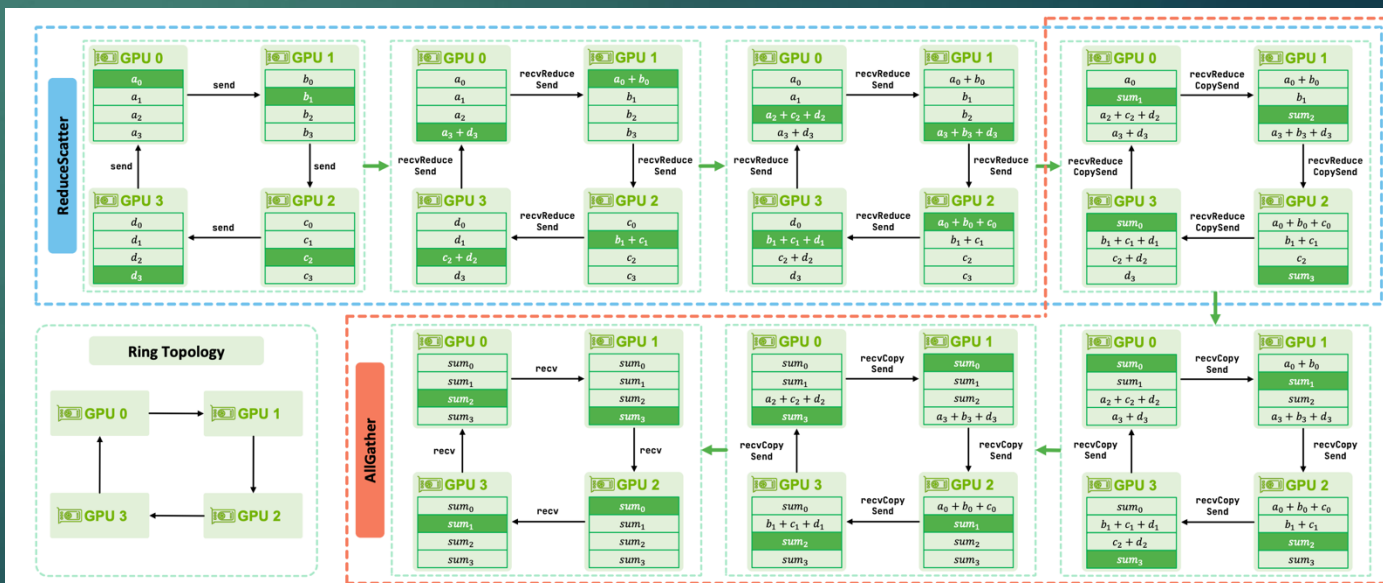


Fig. 4. Illustration of the Ring AllReduce algorithm in NCCL across 4 GPUs connected in a ring topology, highlighting the sequence of GPU communication primitives within a single loop iteration.

Collective Algorithms in NCCL

► Tree AllReduce

► Reduce

- Leaf GPUs send local data upward, Middle GPUs receive data from child node(s) and reduce via the `recvReduceSend` primitive to pass result upwards, Root GPU perform `recvReduceCopySend`, complete the reduction and copy result to user provided buffer

► Broadcast – `recvCopySend` and `recv` to propagate the result back to all nodes

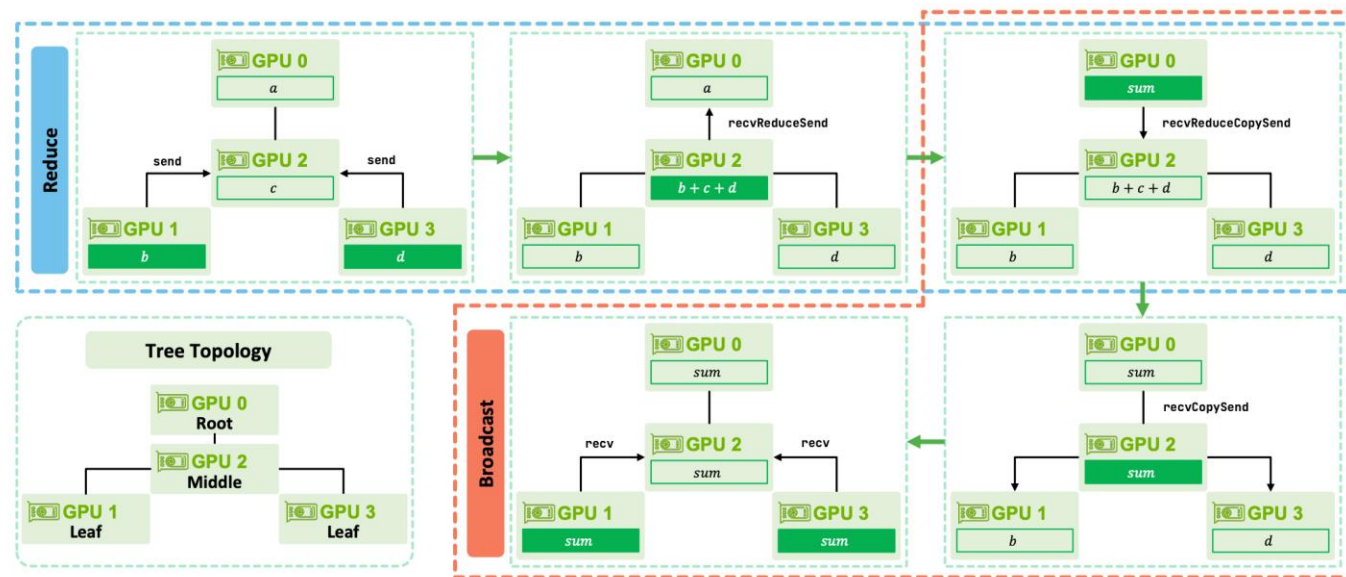


Fig. 5. Illustration of the Tree AllReduce algorithm in NCCL across 4 GPUs connected in a tree topology, highlighting the sequence of GPU communication primitives within a single loop iteration.

Benchmark results analysis

► Benchmark Takeaways:

- For small messages, LL/LL128 (low latency) protocol performs better
- For large inter node messages, simple works much better
- For intra-node messages, LL128 works well on general cases (ranging from small message to large messages)
- Tree Algorithm is better for small AllReduce problems, while Ring Algorithm is better for large AllReduce problems

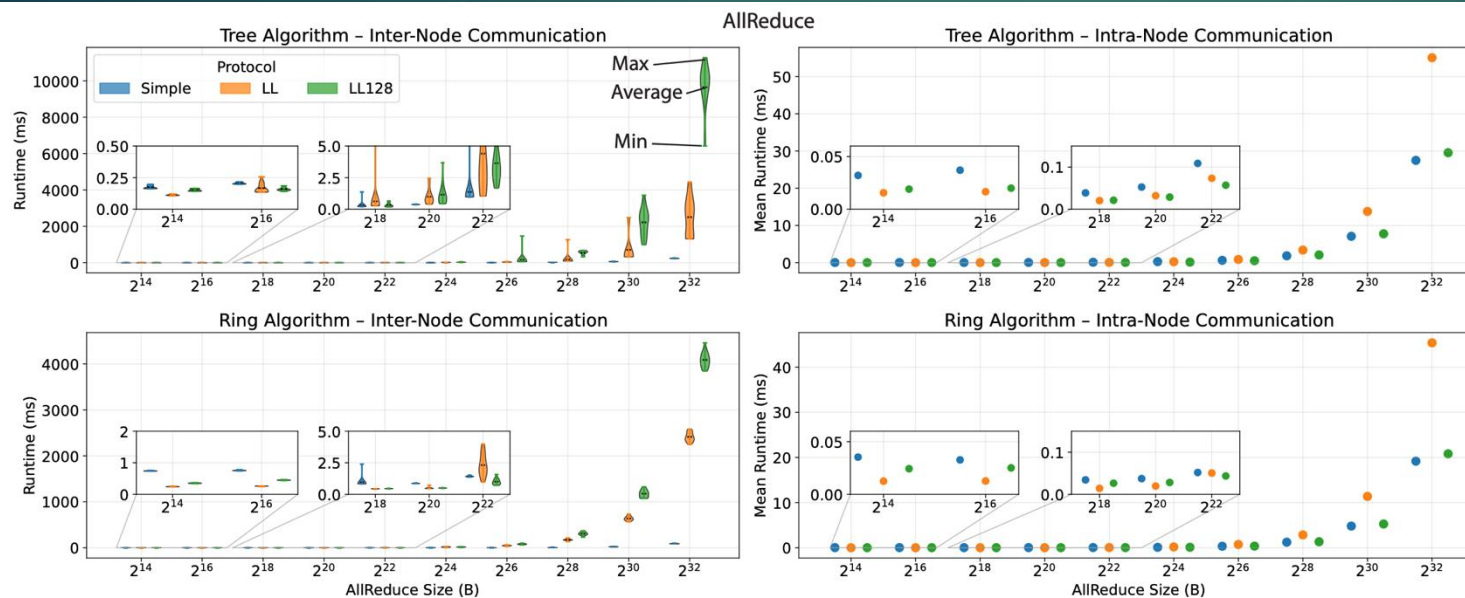


Fig. 6. Runtime comparison of protocols for Ring and Tree AllReduce when running inter- and intra-node. Each data point consists of 20 runs with a warm-up phase. For intra-node communication we report only the median value for readability as the variance is very low.

Open Question & Discussions

- ▶ To facilitate GPU communicate with other GPUs better, shall we directly put a NIC (optical or ethernet) inside the GPU card, where the GPU core/controller have directly access to the NIC?
 - ▶ Will that improve inter-node GPU communication quality (bandwidth & latency)?
 - ▶ Will that also help improve intra-node GPU communication when the two GPUs are not attached to the same CPU socket?
- ▶ Does NCCL optimizations (bypassing operating system abstractions) creates security concerns? What should user do to protect their datacenters from attackers?
 - ▶ When a GPU can directly access the memory of other GPUs, if that GPU is attacked then the attacker can read/write other GPUs' memory. Should we adapt those cross-layer optimizations that might create security vulnerabilities?
- ▶ If NCCL can access some performance metrics, shall we try to let the NCCL protocol learn how to configure itself? Or shall we implement some hard-coded rules?
- ▶ How do we efficiently prototype/performance-model this kind of large-scale system?