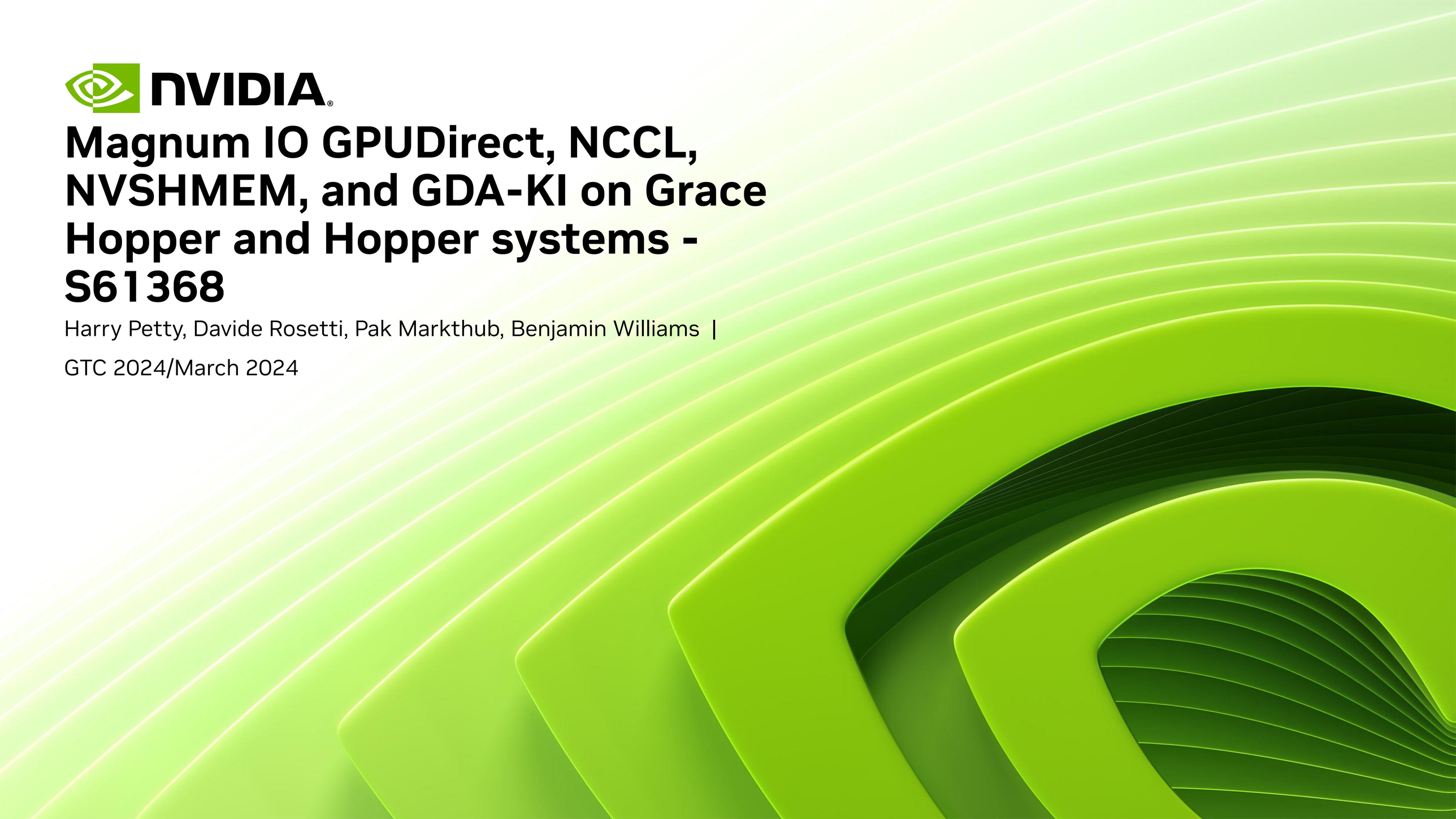


Magnum IO GPUDirect, NCCL, NVSHMEM, and GDA-KI on Grace Hopper and Hopper systems - S61368

Harry Petty, Davide Rosetti, Pak Markthub, Benjamin Williams |

GTC 2024/March 2024

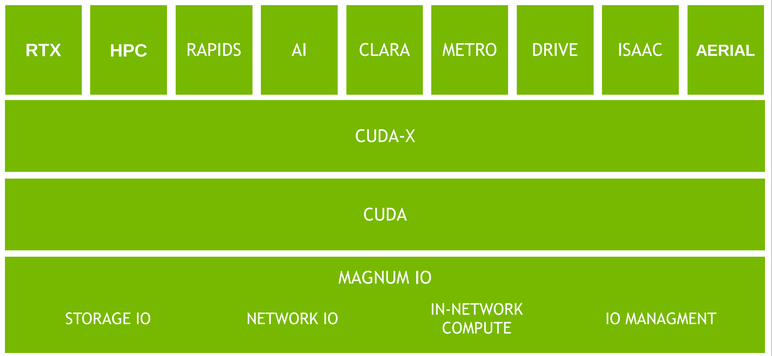
Agenda

- Introduction - Magnum IO Technologies
- Background on GPUDirect
- GPUDirect Async
- NVSHMEM IBGDA
- NCCL IBGDA
- Summary

Magnum IO™

For multi-GPU multi-node accelerated data center IO

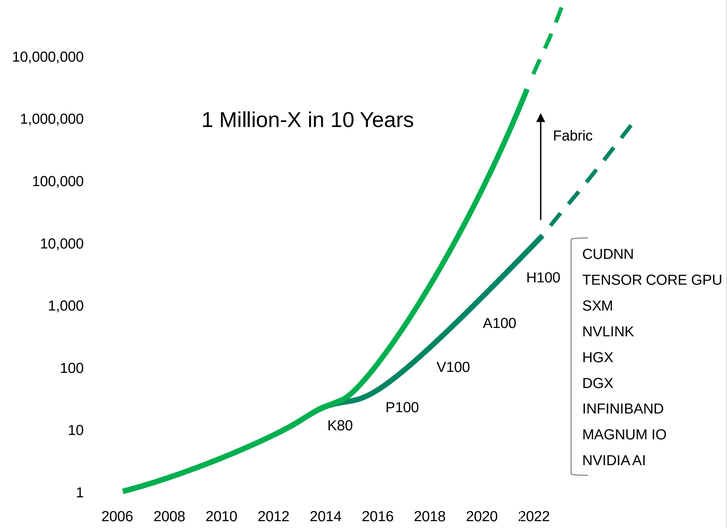
Magnum IO is the architecture for parallel, intelligent data center IO.



- Magnum IO software acceleration libraries and APIs accelerate GPU computing.
- Full stack accelerated computing with Magnum IO optimizes use of NCCL, SHARP, and available interconnects like NVLink and InfiniBand.

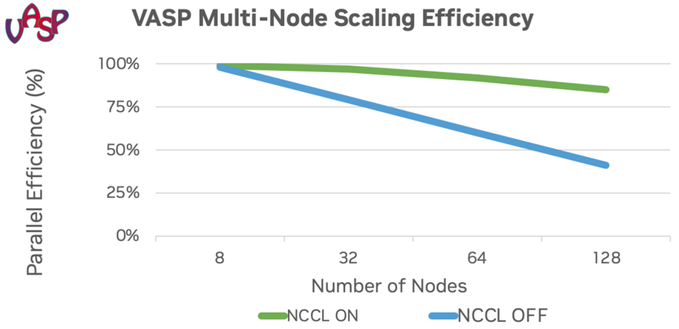
AI Performance from Fabrics

Accelerated by Magnum IO

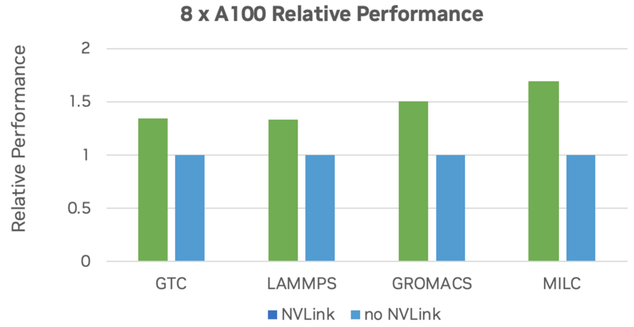


- Magnum IO enables large models to use multiple GPUs, lots of GPU memory, and achieve extreme IO speeds.
- GPUDirect and GPU initiated communications lower latency and deliver higher throughput.

Reduce Communication Bottlenecks So Applications Scale Efficiently



NVLink with Magnum IO Shows 30% Boost in Application Performance



Relative performance chart: Systems with 8 x A100 GPUs benchmarked with NVLink versus no NVLink (PCIe) show over 30% increase in relative application performance
VASP Scaling: Large Materials Science Simulations Using VASP and Magnum IO | Scale Efficiently from 8 to 128 servers (64-1024 GPUs)

Faster Applications with Accelerated GPU Communications

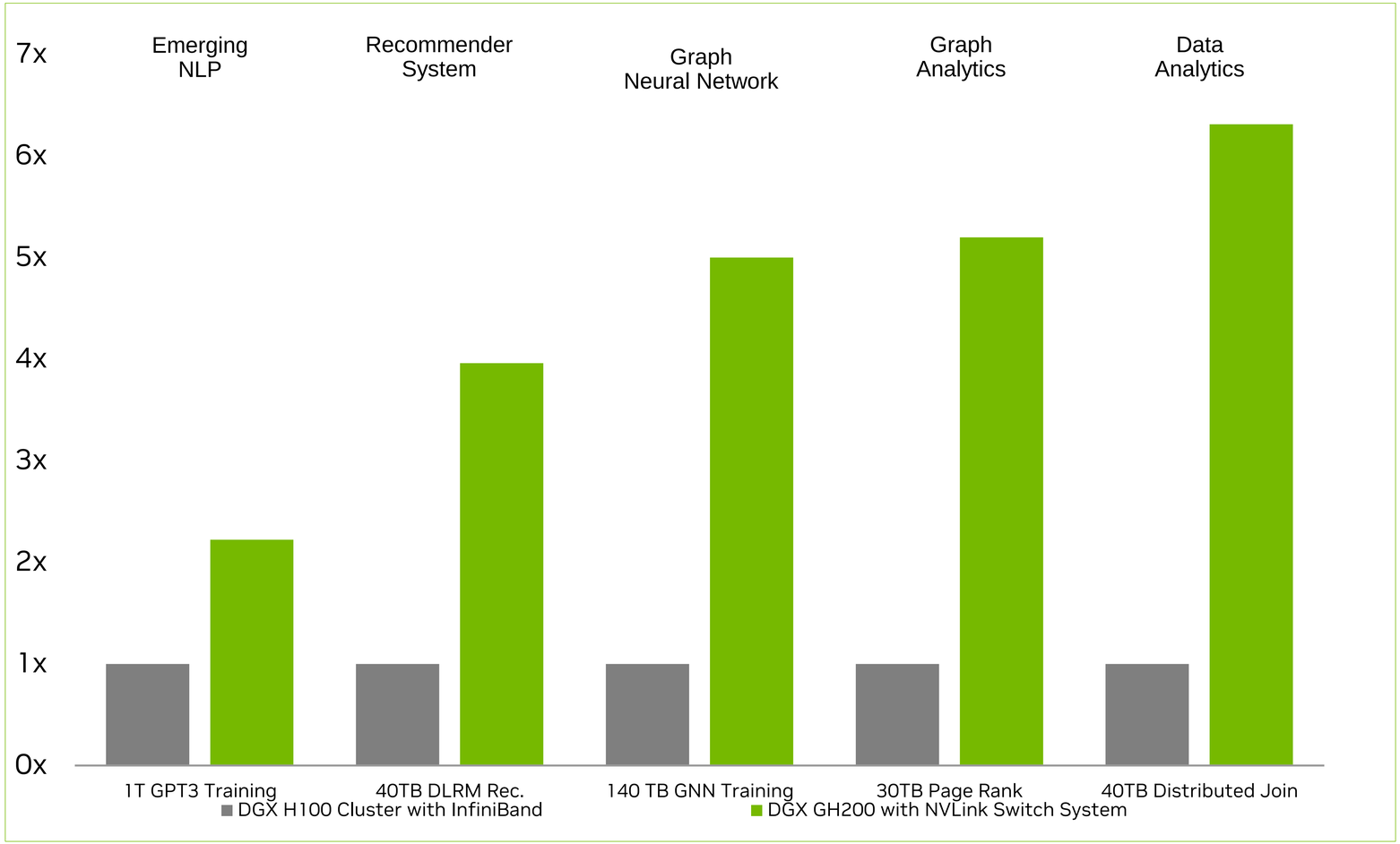
4th GEN NVLINK
900 GB/s from 18x25GB/sec bi-directional ports
GPU-2-GPU connectivity across nodes

3rd GEN NVSWITCH
All-to-all NVLink switching for 8-256 GPUs
Accelerate collectives - multicast and SHARP

NVLINK SWITCH
128 port cross-connect based on NVSwitch

DGX GH200 SYSTEM
128 TB/s bi-section bandwidth
256 Grace Hopper Superchips | 36 NVLink switches

KEY SOFTWARE SUPPORT
Acceleration libraries - CUDA®, CUDA-XTM, Magnum IO™
Multi GPU, multi-node enabled applications
such as Triton Inference Server or VASP

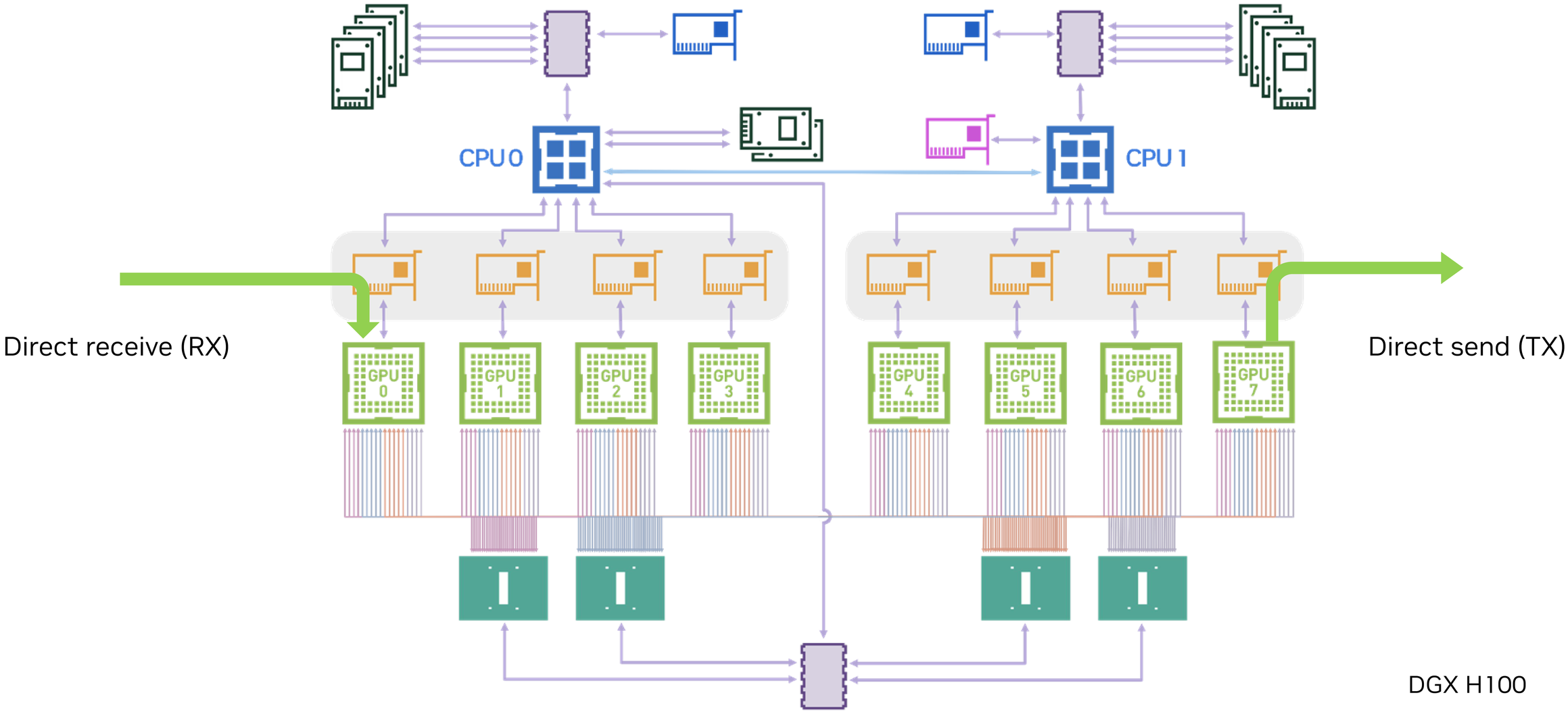


Source: NVIDIA internal projections
1T GPT3 Training: 32 GPU; 40TB DLRM Rec: 128 GPU; 140 TB GNN Training: 256 GPU; 30TB Page Rank: 128 GPU; 40TB Distributed Join: 128 GPU

Background on GPUDirect

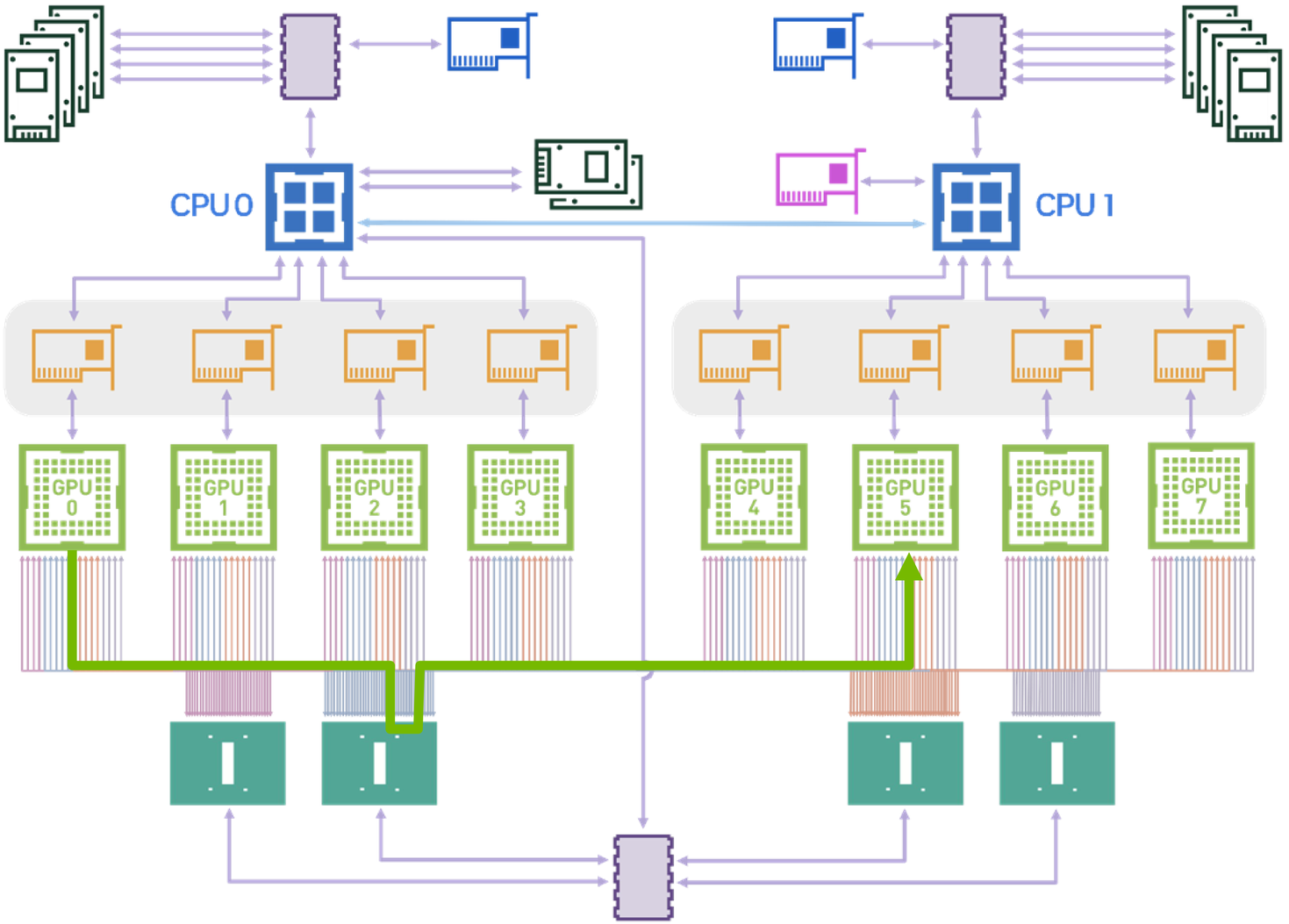
GPUDirect flows

GPUDirect RDMA



GPUDirect flows

GPUDirect P2P



e.g. SM store to peer over NVSwitch

DGX H100

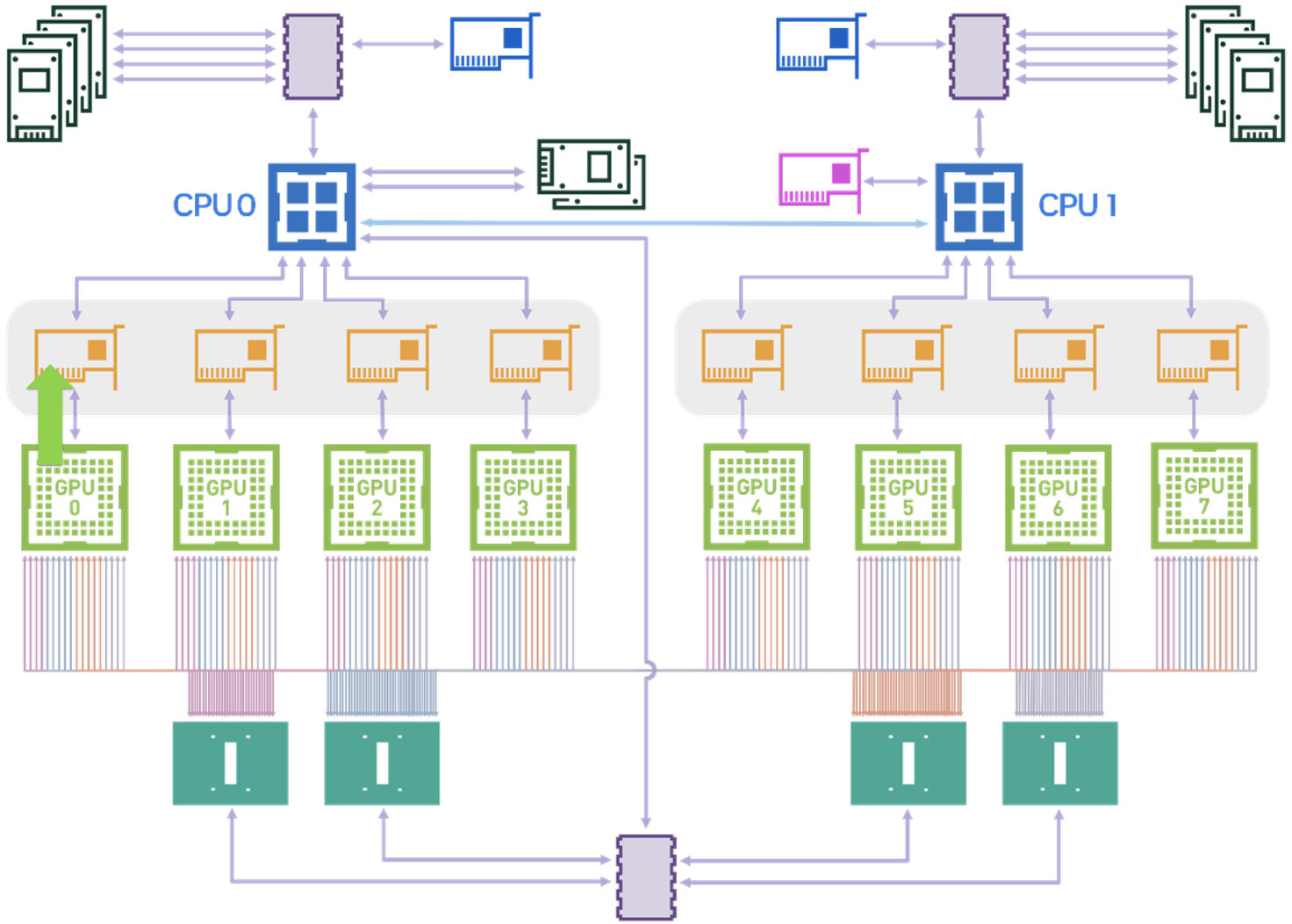
ConnectX-7 ConnectX-7 Network Module NVMe PCIe Switches NVSwitch PCIe 100 GbE CPU communication

NVIDIA

GPUDirect flows

GPUDirect Async

control plane offloading
e.g. GPU triggers NIC work requests



DGX H100

ConnectX-7 ConnectX-7 Network Module NVMe PCIe Switches NVSwitch PCIe 100 GbE CPU communication

NVIDIA

GPU HW support matrix

	GeForce	RTX	Data Center[3]
GPUDirect P2P with NVSwitch	N/A	N/A	YES[2]
GPUDirect P2P over PCIe	NO	YES	YES
GPUDirect P2P over NVLink	N/A	N/A	YES[2]
GPUDirect P2P over NVLink bridge	N/A[1]	N/A	YES
GPUDirect RDMA	NO	YES	YES
GPUDirect Async	YES	YES	YES

[1] except for GeForce RTX 3090 and other Ampere-class PCIe cards, with NVLink bridge
[2] may require SXM cards
[3] NVLink not available on Ada-based card

Fabric support matrix

Platform	x86, Arm SBSA	GH200	Tegra+iGPU
GPUDirect P2P	PCIe NVLink NVLink bridge NVSwitch	NVLink NVSwitch	N/A
GPUDirect RDMA	PCIe	NVLink-C2C	PCIe[1]
GPUDirect Async	PCIe	NVLink-C2C	N/A

[1] when using the host pinned memory allocator (cuMemHostAlloc)

OS support matrix

	Linux	Windows TCC	Windows WDDM
GPUDirect P2P with NVSwitch	YES	NO	NO
GPUDirect P2P over PCIe	YES	YES	YES[1]
GPUDirect P2P over NVLink	YES	YES	YES[1]
GPUDirect P2P over NVLink bridge	YES	YES	YES[1]
GPUDirect RDMA	YES	NO	YES[2]
GPUDirect Async	YES	NO	NO

[1] Linked Display Adapter (LDA) mode
[2] Requires NVIDIA RiverMax SDK

GPUDirect virtualization support

Technology	Host: Linux KVM Guest: Linux		Host: VMware ESXi[2] Guest: Linux	
	passthrough	vGPU[3]	passthrough	vGPU[3]
GPUDirect P2P over PCIe	YES	NO	YES	NO
GPUDirect P2P over NVLINK	YES	YES[1]	YES	YES[1]
GPUDirect RDMA	YES	YES	YES	YES
GPUDirect Async	YES	NO	YES	NO

[1] Only on 1:1 vGPUs
[2] VMware ESXi 7.0 HV or later
[3] https://docs.nvidia.com/grid, https://docs.nvidia.com/ai-enterprise

(GPUDirect) RDMA support matrix

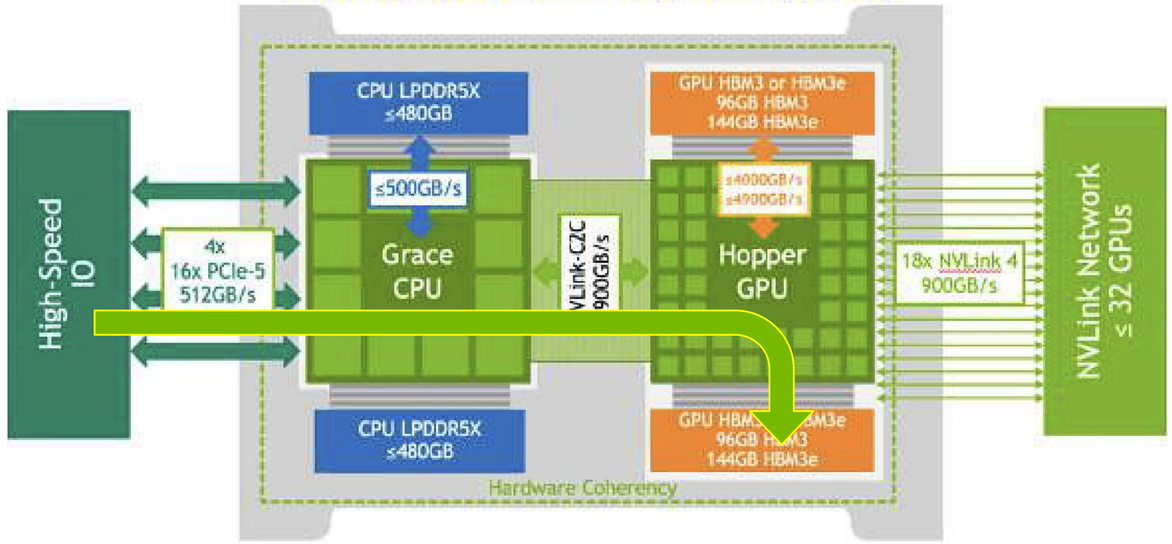
Memory allocator	X86 Arm SBSA	GH200	Tegra+iGPU
cudaHostAlloc	N/A	N/A	YES
cudaMalloc	YES[6]	YES[6]	NO
cuMemCreate	YES[4]	YES[4]	NO
cudaMallocAsync	YES[5]	YES[5]	NO
Unified Memory managed allocator	NO	YES[1]	YES[2]
Unified Memory system allocator	N/A	YES[1,3]	N/A

[1] requires PCIe device with *on-demand paging* support
[2] on iGPU, managed memory is mapped onto pinned host memory
[3] system memory does not support automatic migration
[4] explicit opt-in flag in CUmemAllocationProp for nv-p2p
[5] need CUDA 12.4+, limited to dma-buf
[6] supports both nv-p2p and dma-buf

GPUDirect RDMA updates

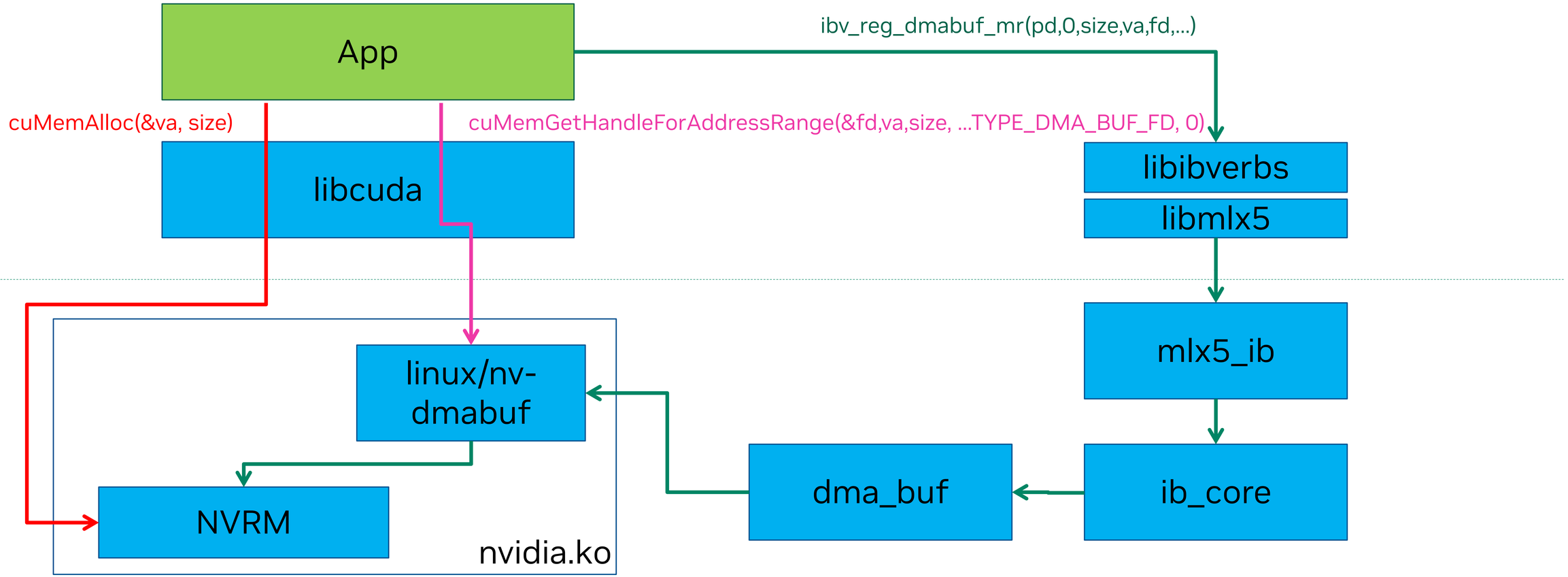
GPUDirect RDMA on GH200

- Grace PCIe Root-Complex
  - Supports four PCIe x16 Gen5 ports
  - Can bridge between PCIe and GPU memory via the NVLink C2C interconnect
  - For best performance
    - Prefer PCIe Relaxed Ordering
    - Maximize latency hiding, increase reorder buffers
    - Max Payload Size >= 128B
    - Max Read Request Size >= 512B
    - Take advantage of 10-bit tag support
- For Unified Memory, i.e. System Allocated Memory and CUDA Managed Memory
  - PCIe access works as expected, e.g. does not trigger automatic page migration, supports Atomics, etc.
  - Use device with page fault support, e.g. On Demand Paging
- On multi-socket platforms
  - Take advantage of NUMA architecture, i.e. affinity between PCIe devices and NUMA nodes
  - Use Extended GPU Memory (EGM) for GPU access to CPU memory



Upstream Linux kernel APIs for GPUDirect RDMA

NIC memory registration based on DMA-BUF



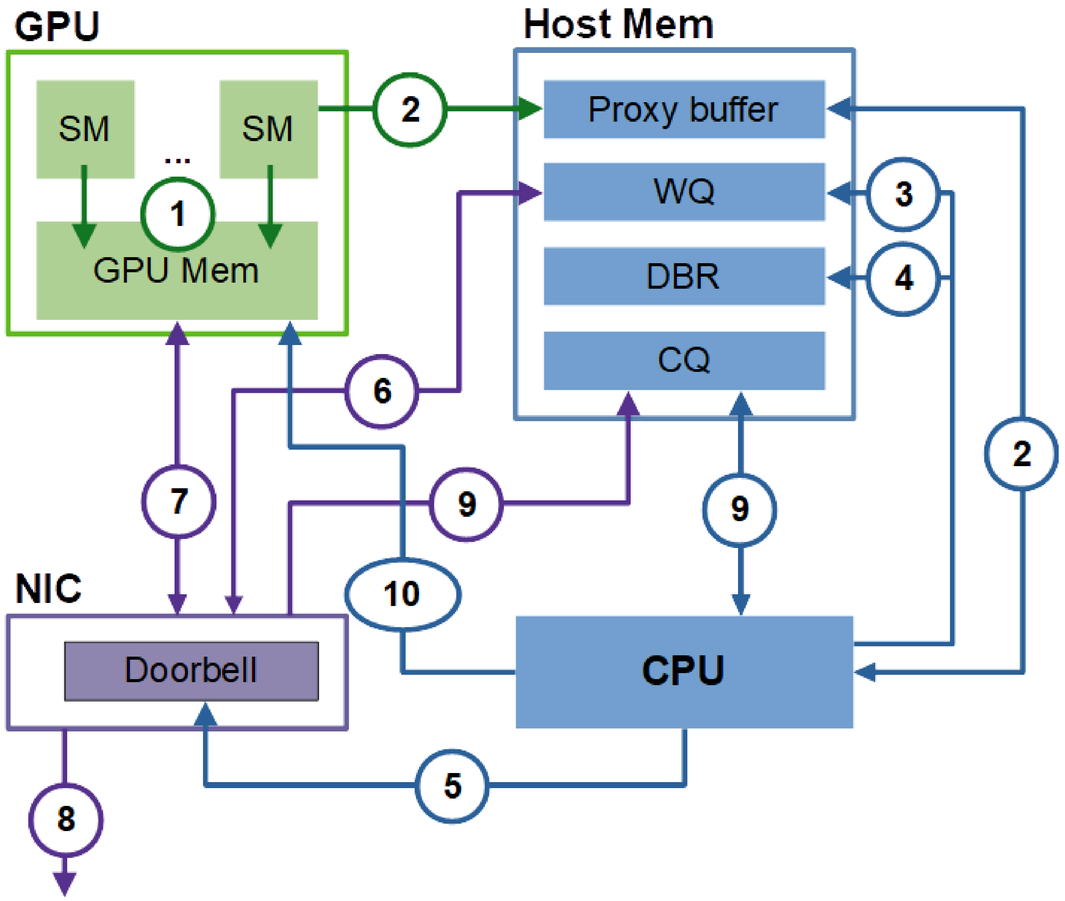
- Require kernel 5.12+ or backports, open GPU driver 515+, CUDA 12.7+
- Supported on data center and RTX GPUs
- Pure upstream experience, no MOFED, no nvidia-peermem or nv_peer_mem
- Current limitations: no sharing of BAR1 space among dma-bufs or other APIs (GDRCopy), always map whole dma-buf even when registering smaller portion

GPUDirect Async

For NVIDIA NIC

How CPU Proxy works

Control path through CPU



1. GPU generates data in GPU memory.
2. GPU notifies CPU. CPU busy-waits for the notification.
3. CPU creates WQE.
4. CPU updates DBR.
5. CPU rings the NIC DB.

Post-send (ibv_post_send)

6. NIC reads WQ.
7. NIC reads data.
8. NIC sends the data out.

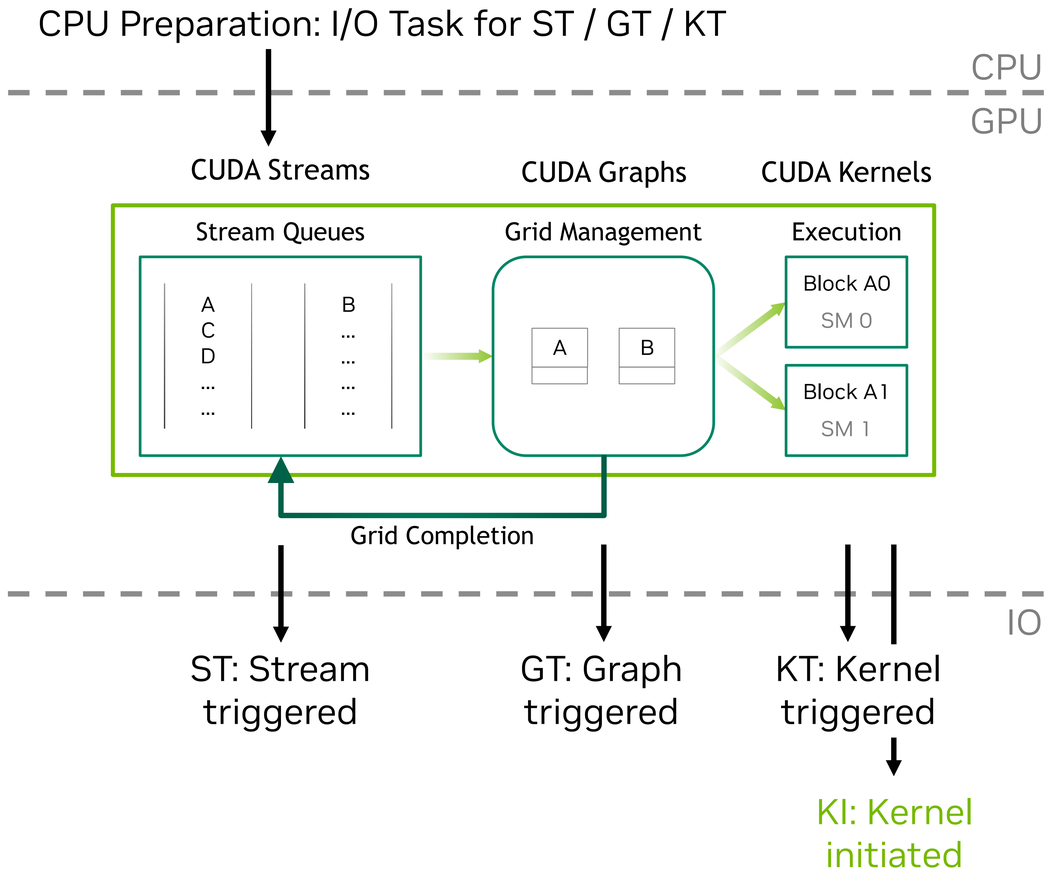
GPUDirect RDMA

9. NIC writes CQ. CPU is polling the CQ.
10. CPU notifies GPU.

CPU is still in the communication critical path!

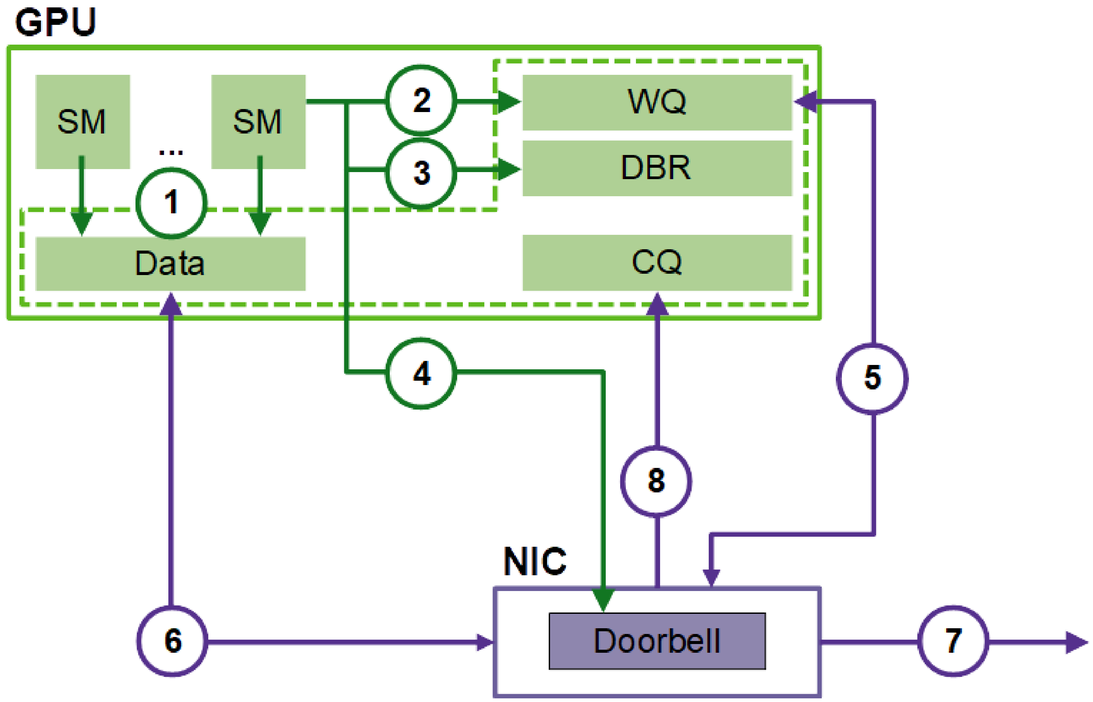
GPUDirect Async Family

- GPUDirect Async (GDA) moves the control path to GPU
- Integrate I/O control with CUDA work scheduling
- I/O control path involves request preparation and triggering
  - *Preparation* involves creating a work request that can be made available to the I/O device (e.g. enqueueing a WQE on a QP)
  - *Triggering* involves handoff of a prepared work request to the I/O device (e.g. ringing a doorbell)
- GPUDirect Async has two flavors
  - CPU prepared, GPU triggered
    - Integrates with stream, graph, kernel
  - GPU initiated (i.e. prepared and triggered)
    - Integrates with CUDA kernels



GDA-KI: GPUDirect Async – Kernel Initiated

Overview



1. GPU generates data in GPU memory.
2. GPU creates WQE.
3. GPU updates DBR.
4. GPU rings the NIC DB.

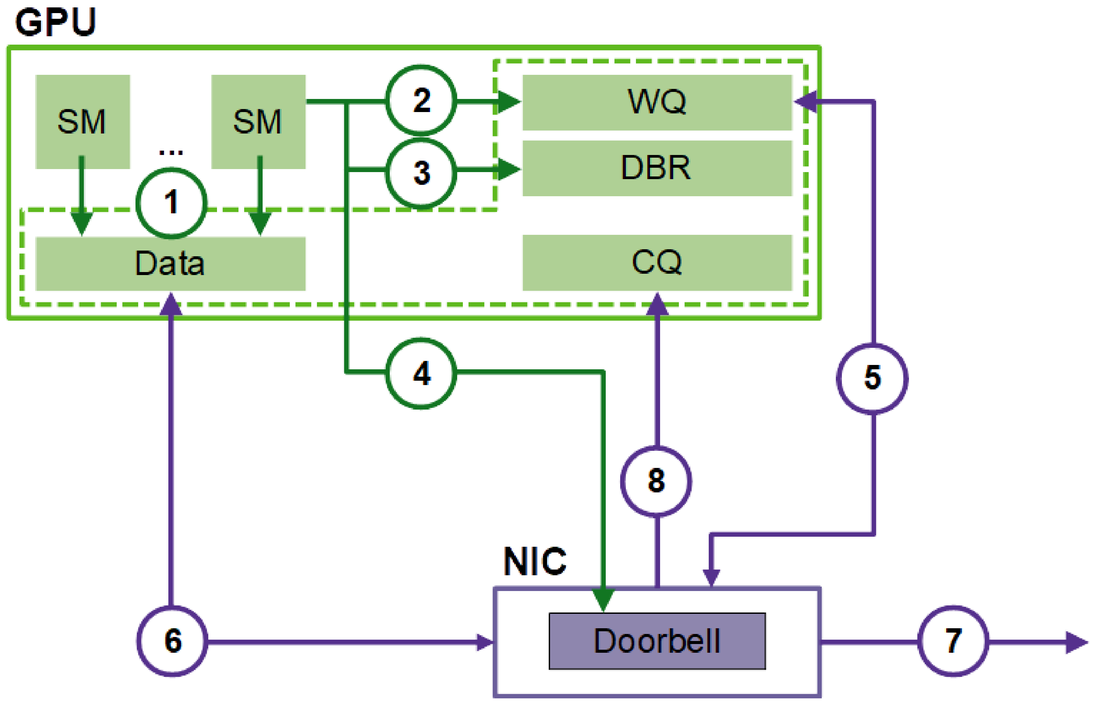
Post-send

5. NIC reads WQ.
6. NIC reads data.
7. NIC sends the data out.
8. NIC writes CQ. GPU polls CQ.

Moving the control path to GPU.
CPU is removed from the critical path.

GDA-KI: GPUDirect Async – Kernel Initiated

Post-send algorithm



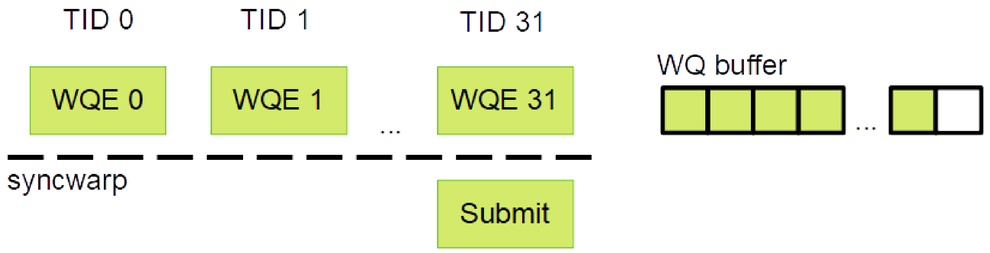
For (...) {
- A. Prepare Data ①
- B. Prepare WQE ②
- C. Membar
- D. Update DBR ③
- E. Membar
- F. Ring DB ④

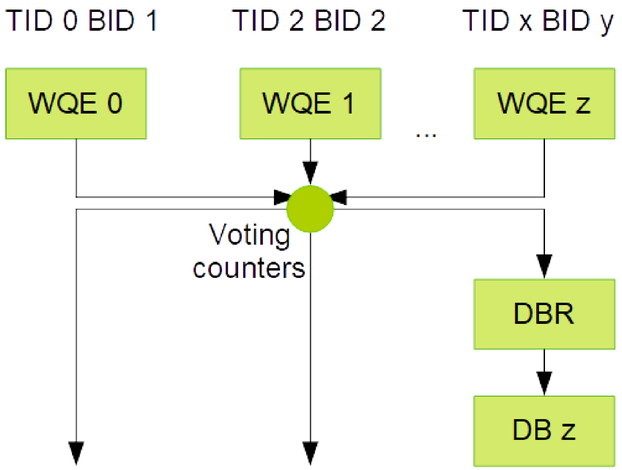
} **Post-send**

- Writing WQE, DBR, and DB is sequential. Membar is required to guarantee visibility ordering to the NIC.
- Membar is expensive on GPU.
- We can prepare multiple WQEs in parallel, sync, and one thread does item C – F.
- No dependency between QPs. Use GPU massive threads to handle multiple QPs concurrently.

GDA-KI: GPUDirect Async – Kernel Initiated

Parallelizing WQE creation with SW coalescing



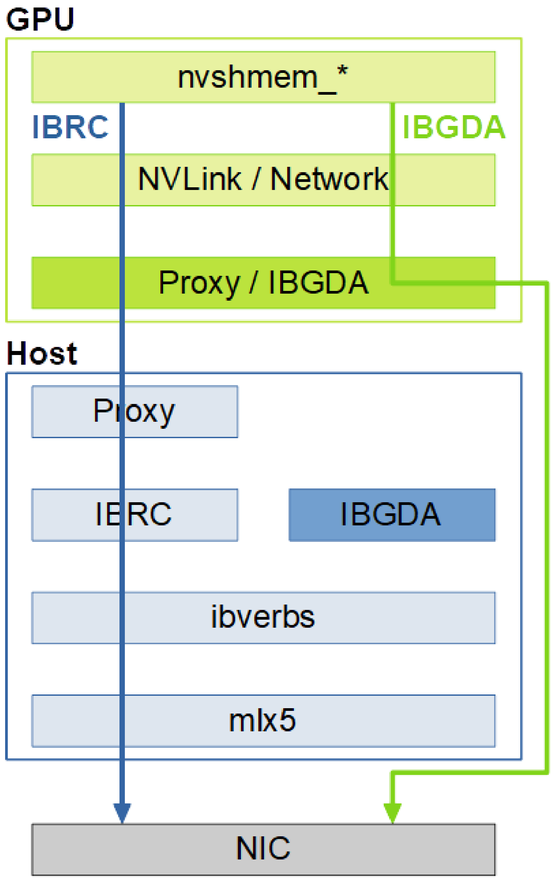


- GPU sends producer idx to the NIC by updating DBR and ringing DB. NIC processes all WQEs whose ID <= idx.
- Two coalescing levels: Warp and DB
- Threads in same warp create WQEs in parallel. One enters the DB coalescing function.
- Threads from different warps (or even CTAs) may share QP and want to submit WQEs around the same time. One will be selected to update DBR and write DB.
- One DB ringing (PCIe write) for n WQEs.

NVSHMEM IBGDA

IBGDA Transport

GDA-KI implementation in NVSHMEM and NCCL



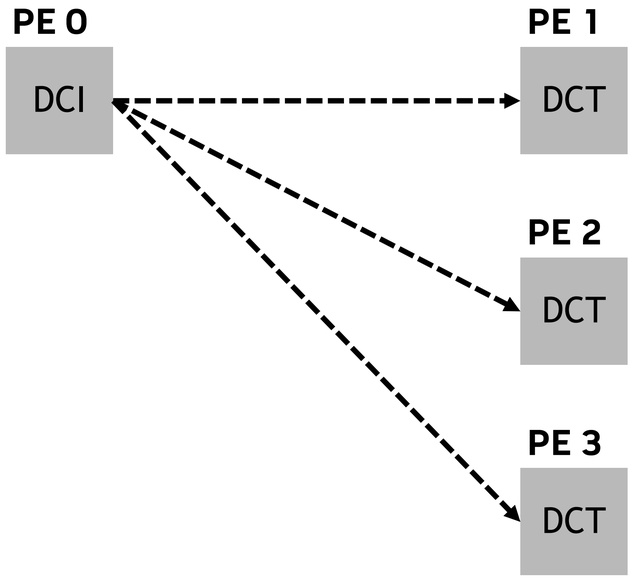
- Same NVSHMEM API for all transports.
- Use environment variable to select the active transport.
  - NVSHMEM_IB_ENABLE_IBGDA=1 ./application
- Optimization through environment variables.
  - NVSHMEM_IBGDA_NUM_DCI
  - NVSHMEM_IBGDA_NUM_RC_PER_PE
  - NVSHMEM_IBGDA_RC_MAP_BY
  - And many more …
  - https://docs.nvidia.com/nvshmem/api/gen/env.html

*IBRC are a CPU Proxy transport
*IBGDA is a GDA-KI transport

NVSHMEM IBGDA Performance Tuning

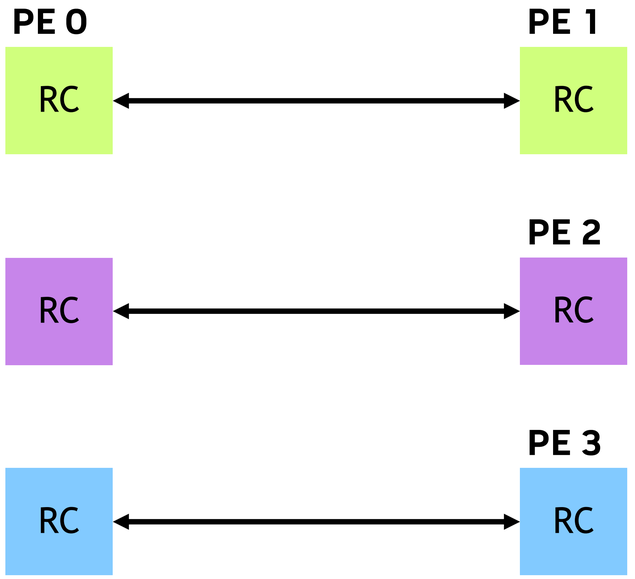
DC vs RC

DC Connection



IBGDA needs multiple QPs for high message rate

RC Connection



- A DCI can dynamically connect to any DCTs.
- One DCI and one DCT on each PE are enough for mesh.
- Submit WQEs to DCI. Each WQE tells DCT num it wants to connect.
- Switching DCT creates latency bubble due to connection establishment.

- RC connections are static. Create upfront.
- $O(P^2)$ for mesh. Imagine N QPs per peer on large cluster.
- Generally faster than DC. No latency bubble.

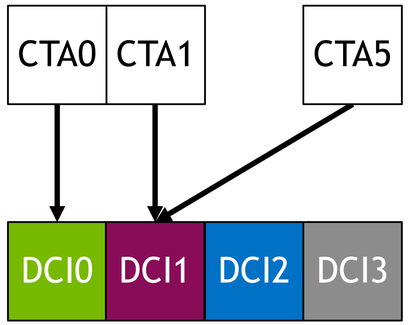
DC for large num PEs

RC for small num PEs

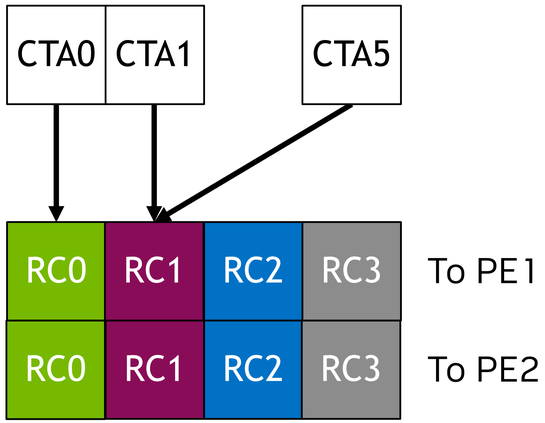
NVSHMEM IBGDA Performance Tuning

QP Mapping

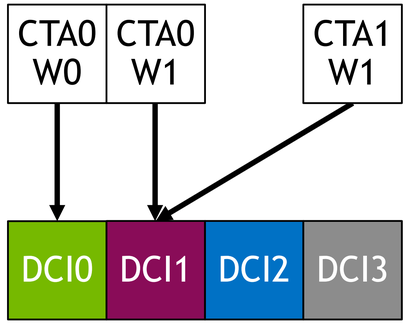
MAP_BY=cta



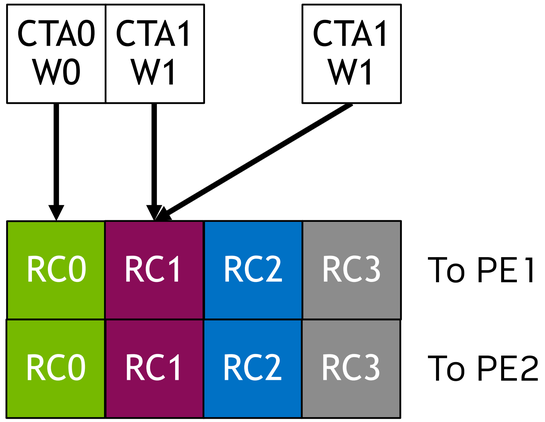
MAP_BY=cta



MAP_BY=warp



MAP_BY=warp



- Avoid frequently switching target PEs on the same DCI.
  - Latency bubble can severely impact performance.
- Avoid over subscribe one QP.
  - False dependency in the same QP can slow down comm.
- Possible mapping: cta, sm, warp, dct

NVSHMEM IBGDA Performance Tuning

Utilizing coalescing

- 3 NVSHMEM API scopes:
  - Thread: One thread to complete one op.
  - Warp: One warp to complete one op.
  - Block: One CTA to complete one op.
- Post-send is highly sequential. IBDA uses one thread to complete each op.
- All threads in the same warp calls same thread-scope API is better than warp-scope API.
  - Coalescing requires using same QP.

```
// Launch with my_kern<<<1,32>>>(dst_pe);
__global__ void my_kern(int dst_pe){
  nvshmem_putmem_nbi(&dst_addr[threadIdx.x],...,dst_pe);
}
```

Create 32 WQEs in parallel.
Thread #31 updates DBR and rings DB.

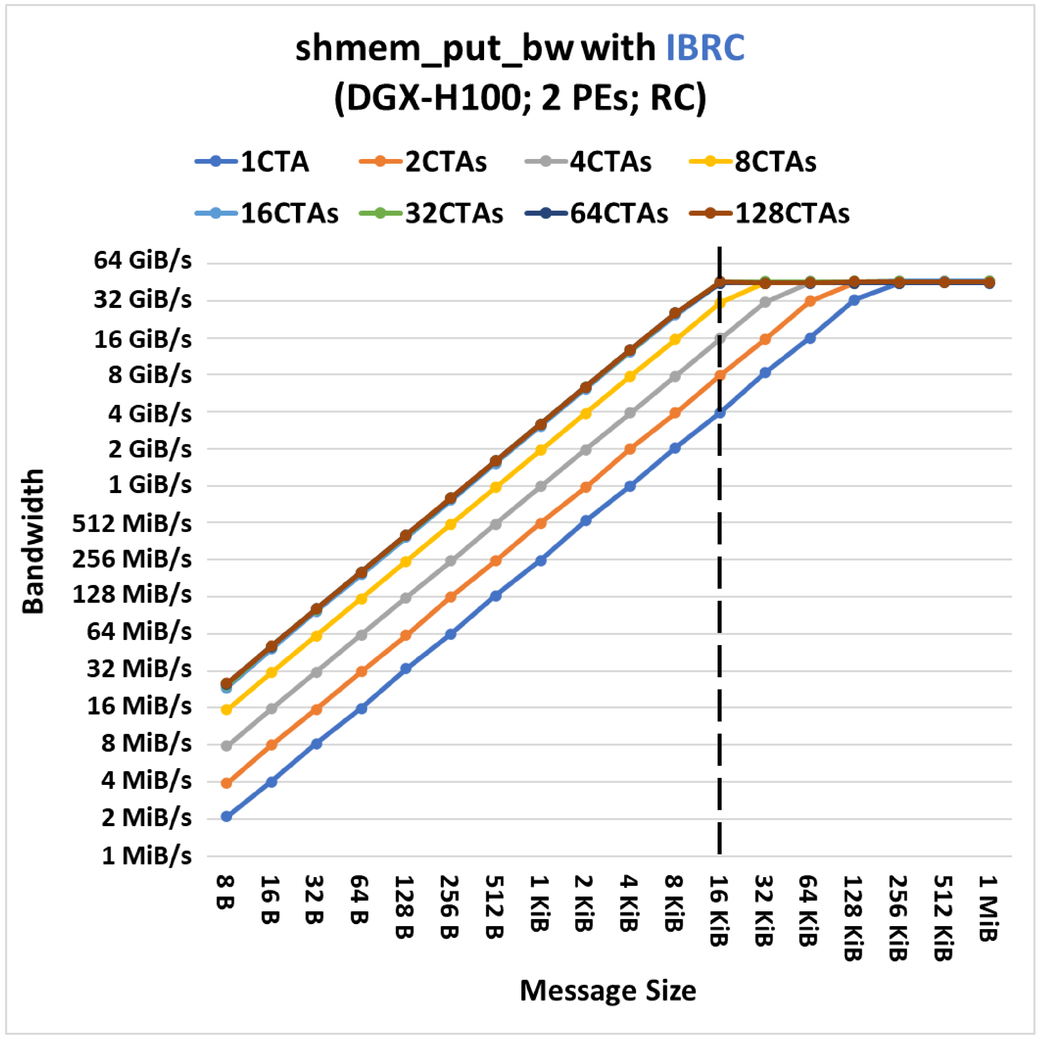
```
// Launch with my_kern<<<1,32>>>(dst_pe);
__global__ void my_kern(int dst_pe){
  for(i=0;i<32;++i)
    nvshmemx_putmem_nbi_warp(&dst_addr[i],...,dst_pe);
}
```

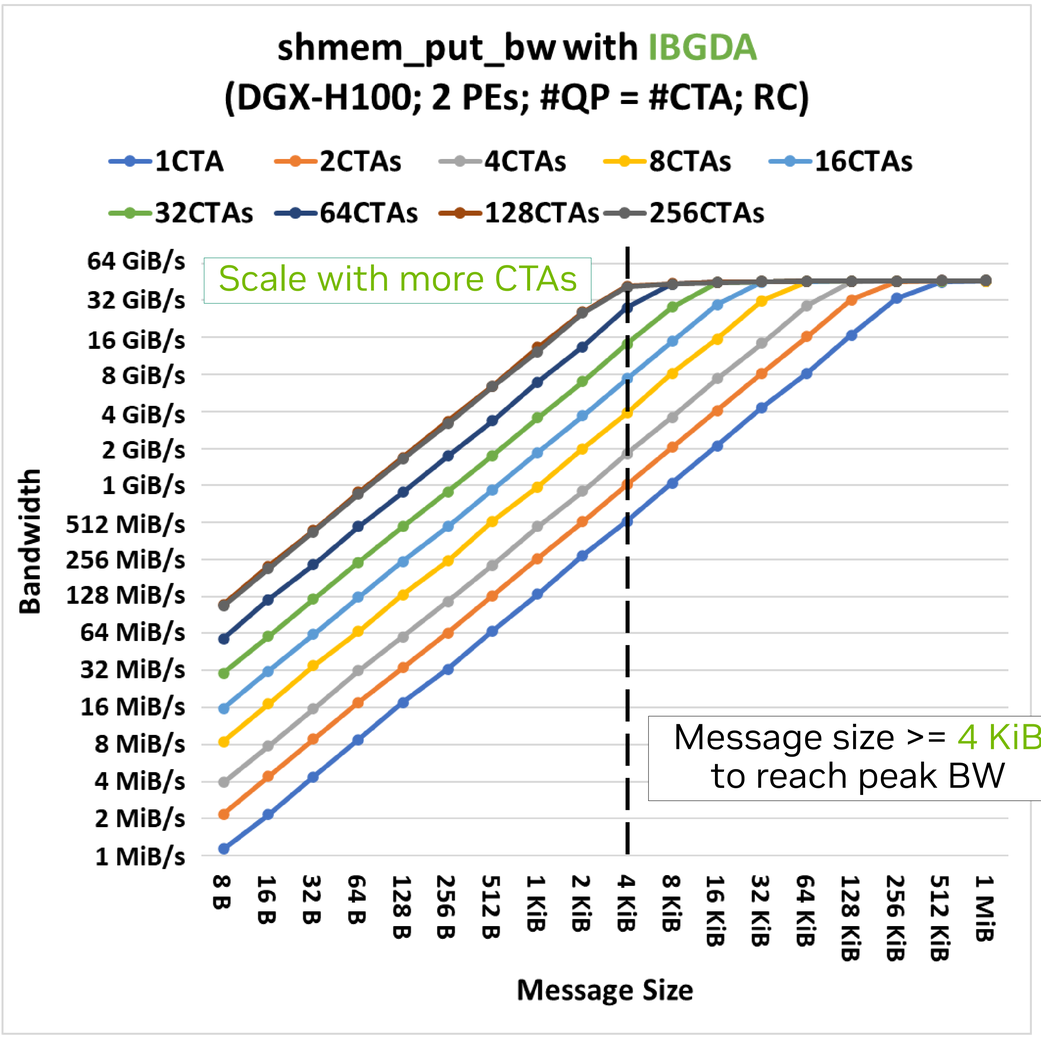
Create a WQE, update DBR, and ring DB 32 times.

NVIDIA.

NVSHMEM PUT

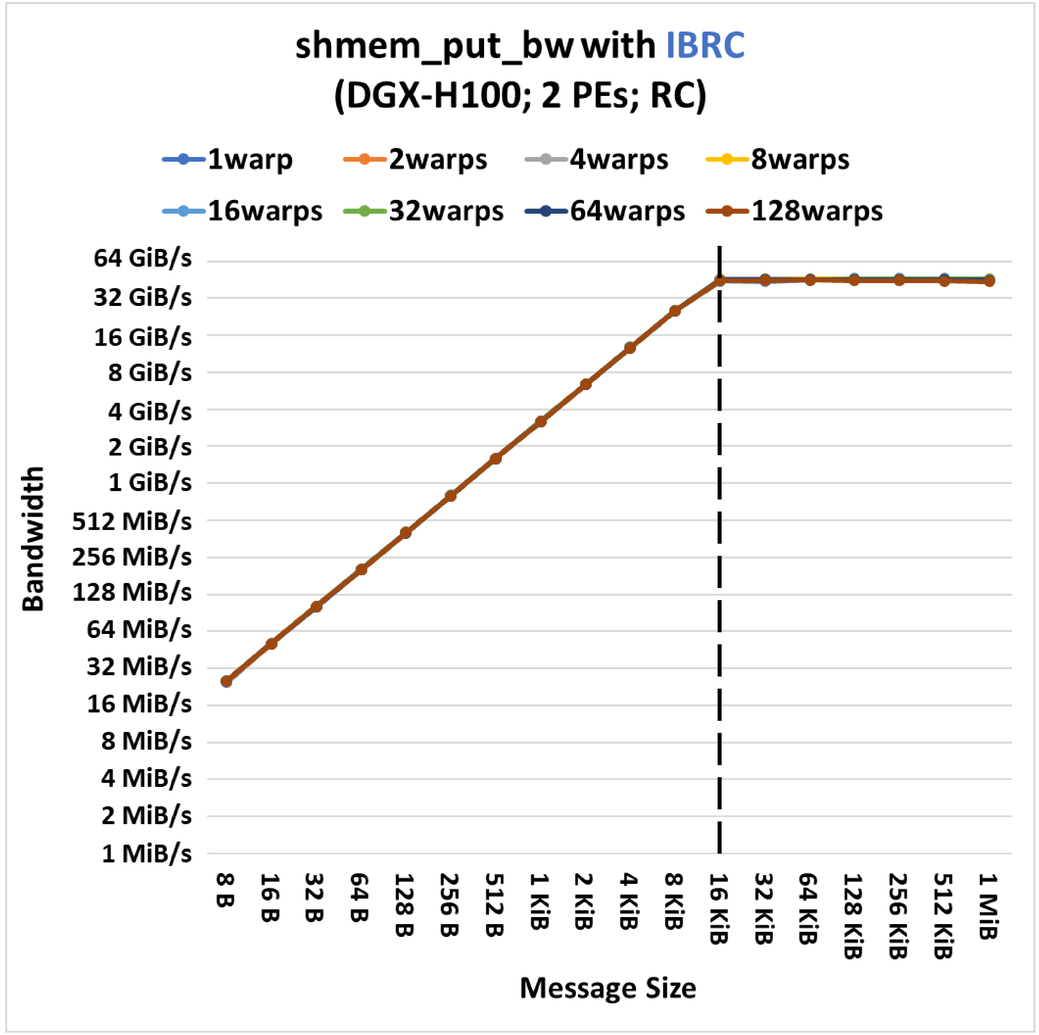
Block Scope

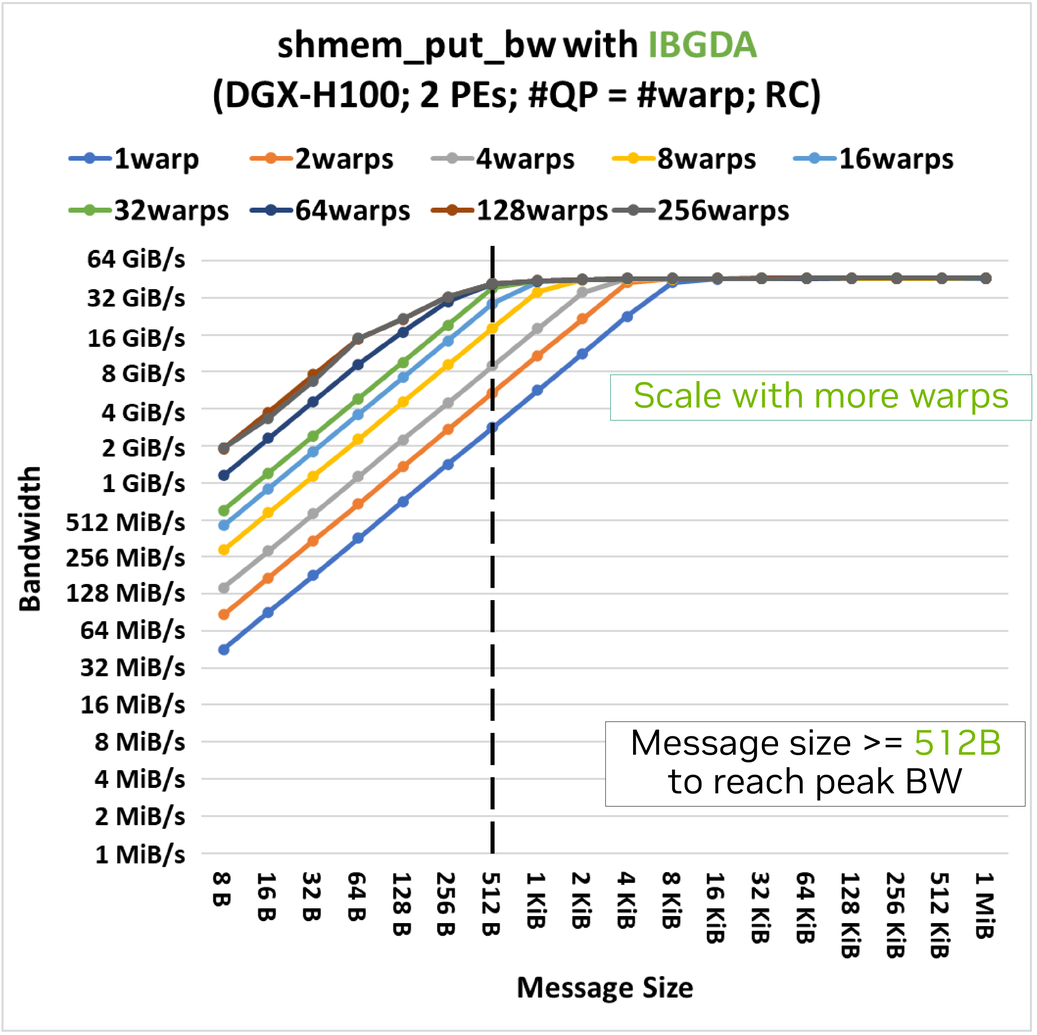




NVSHMEM PUT

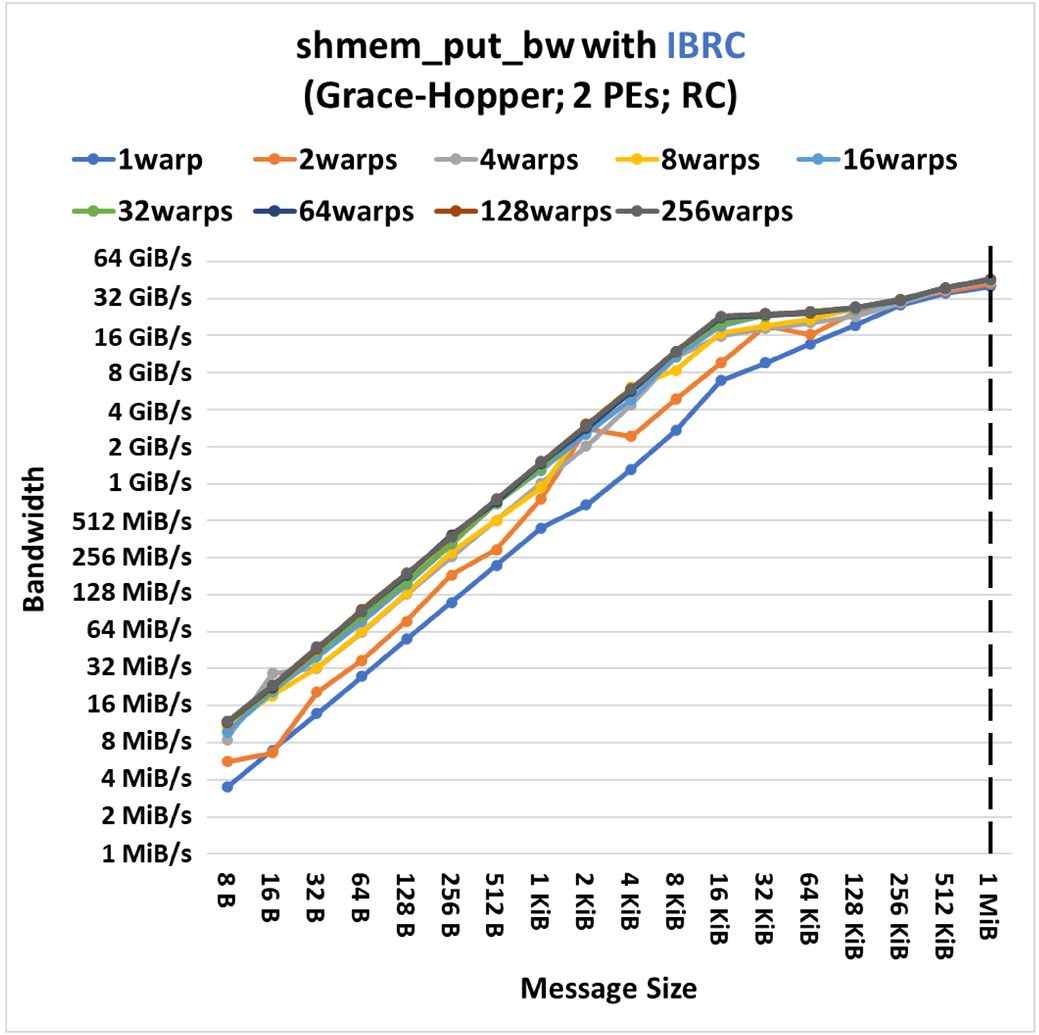
Thread Scope

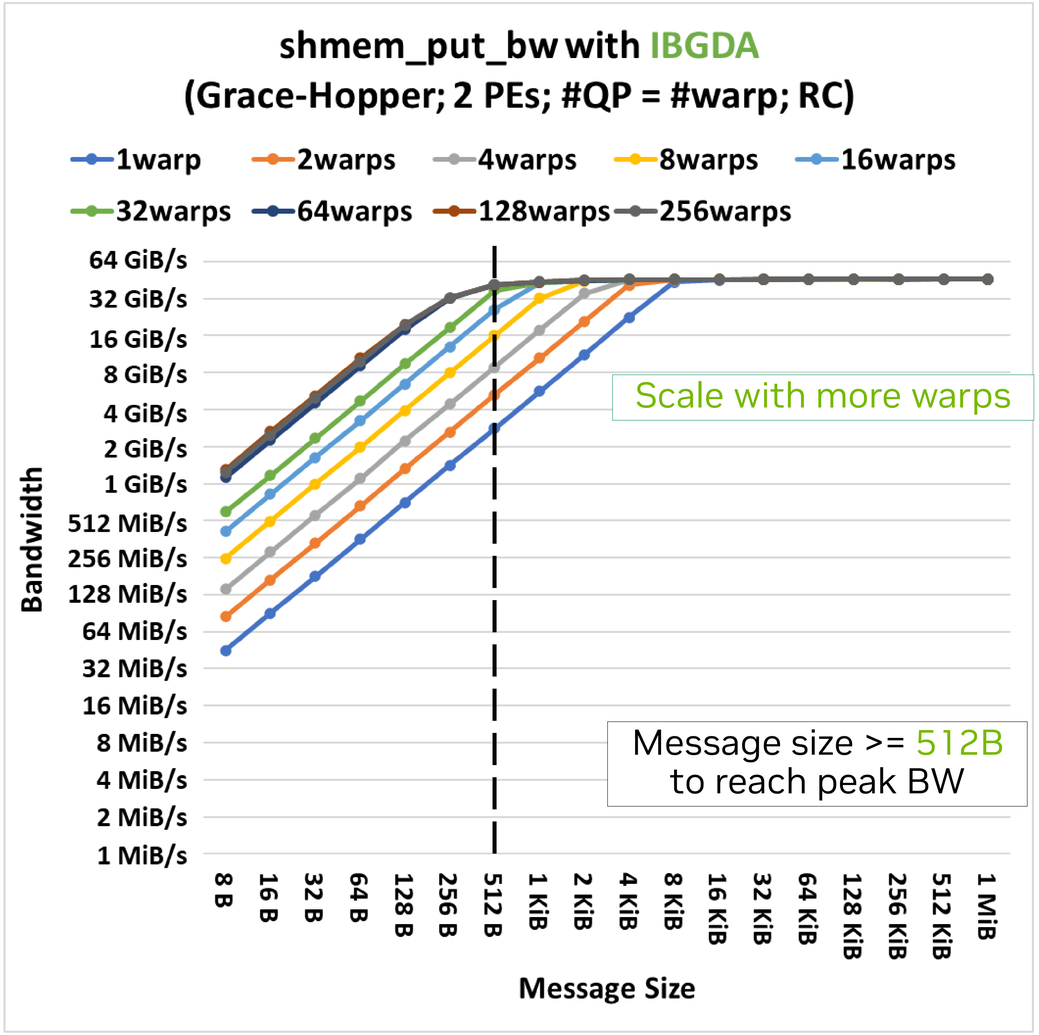




NVSHMEM PUT

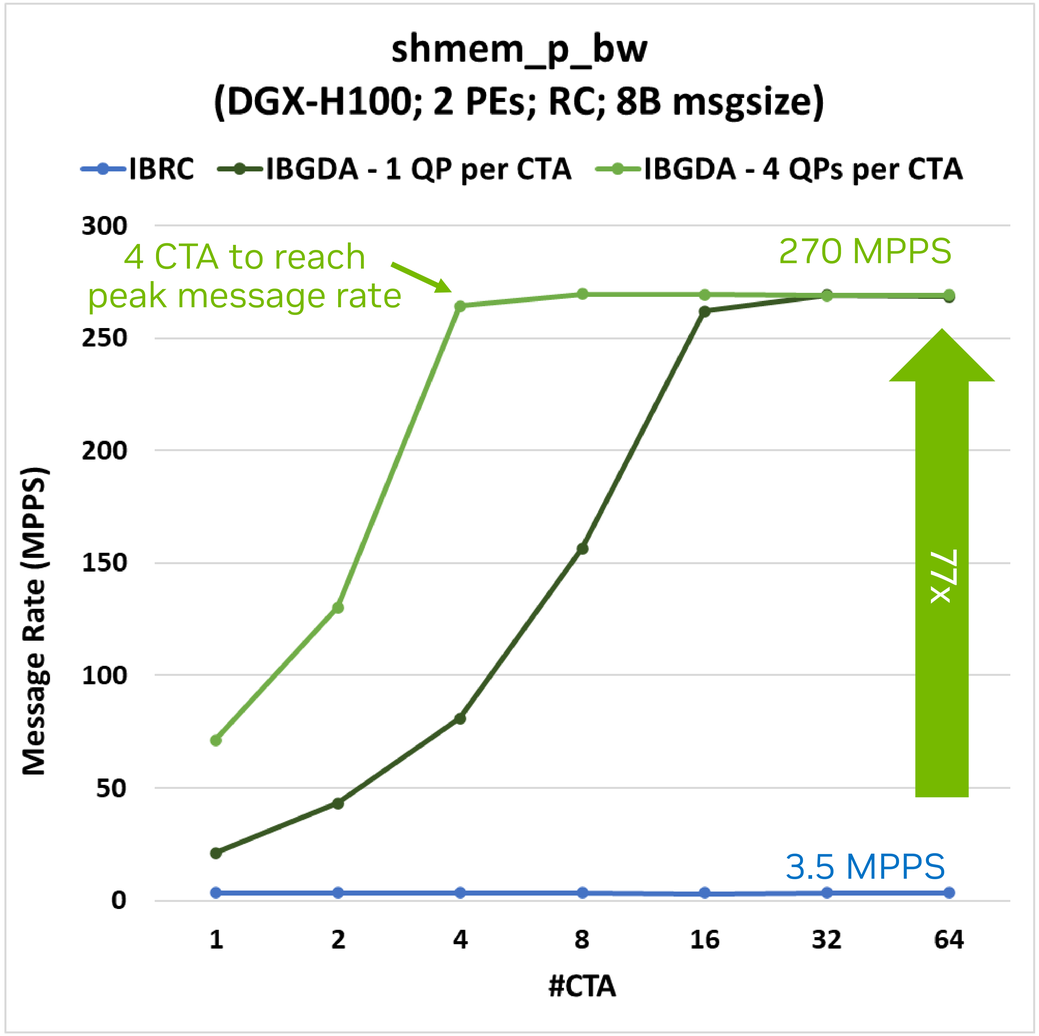
Thread Scope

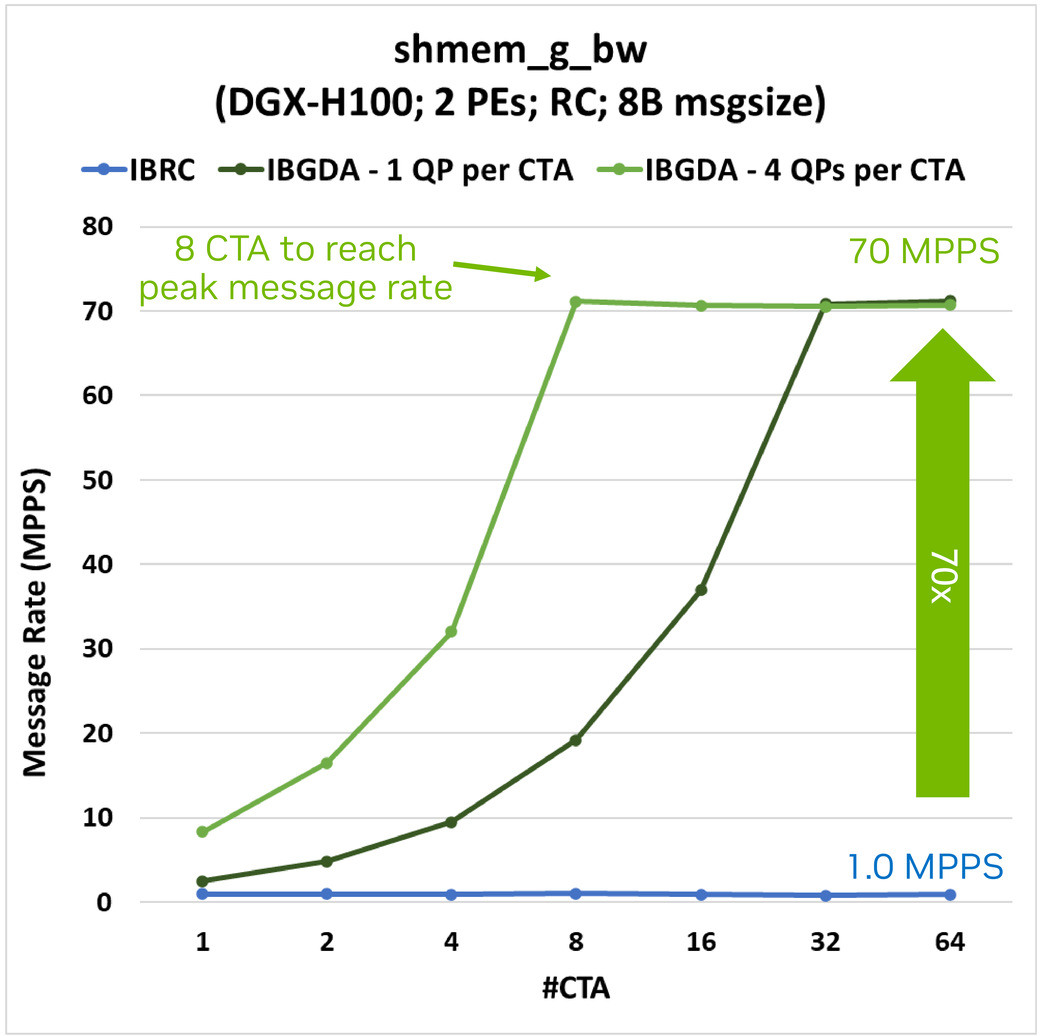




NVSHMEM P & G

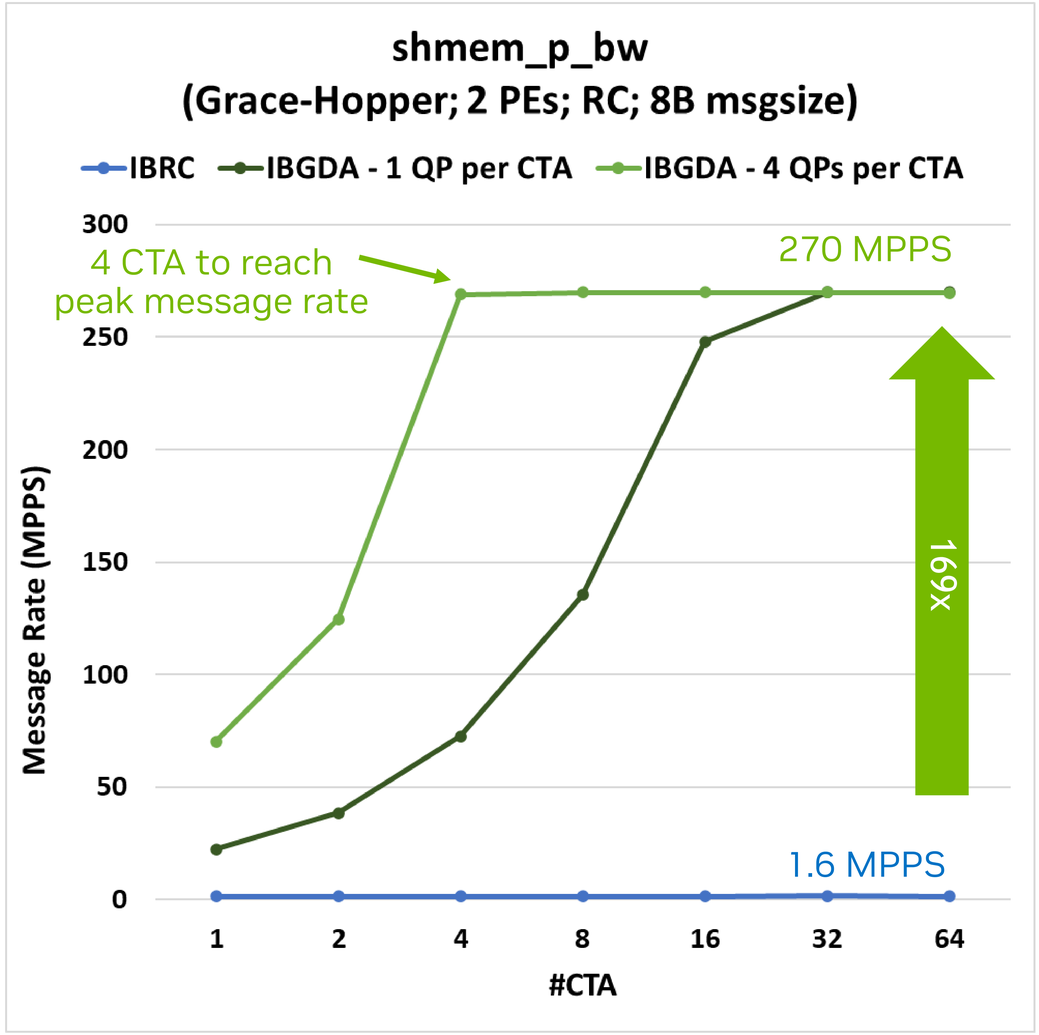
DGX-H100

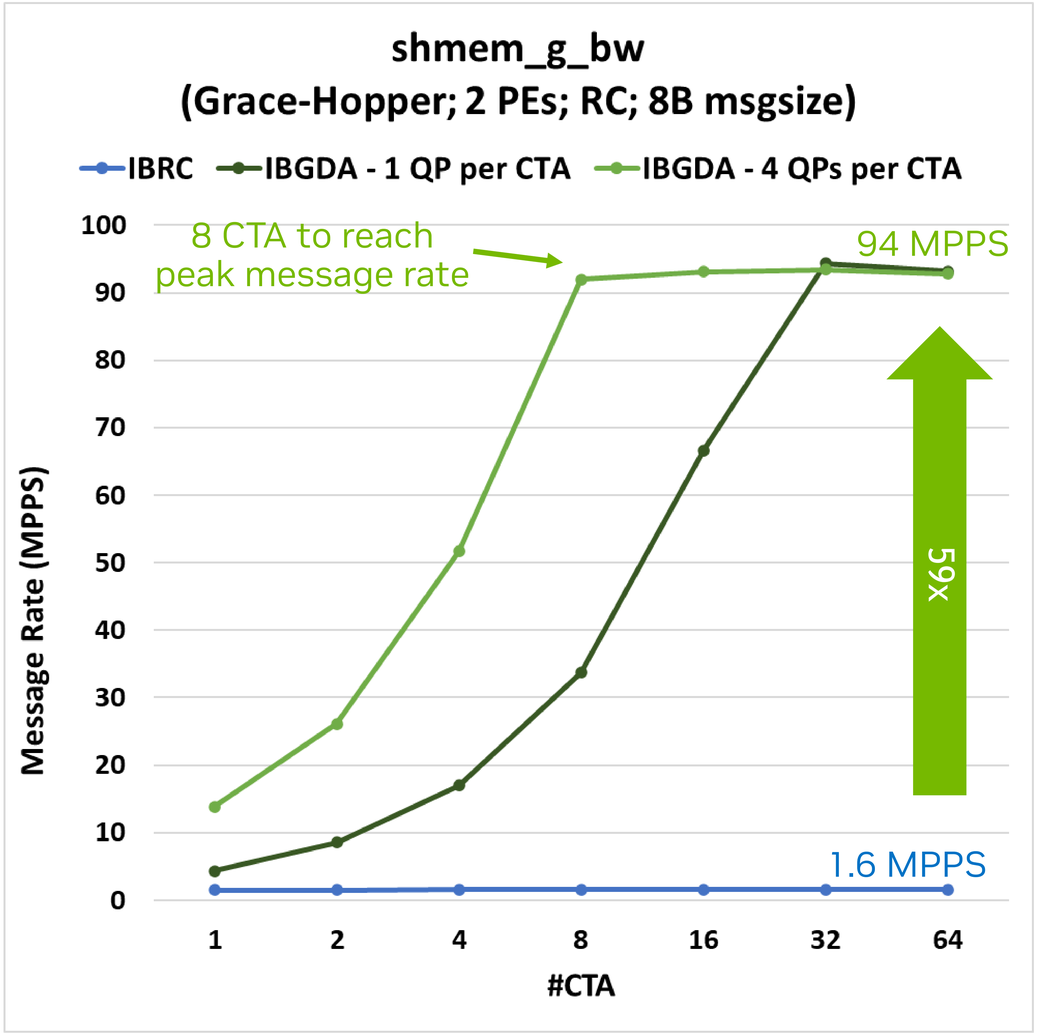




NVSHMEM P & G

Grace-Hopper



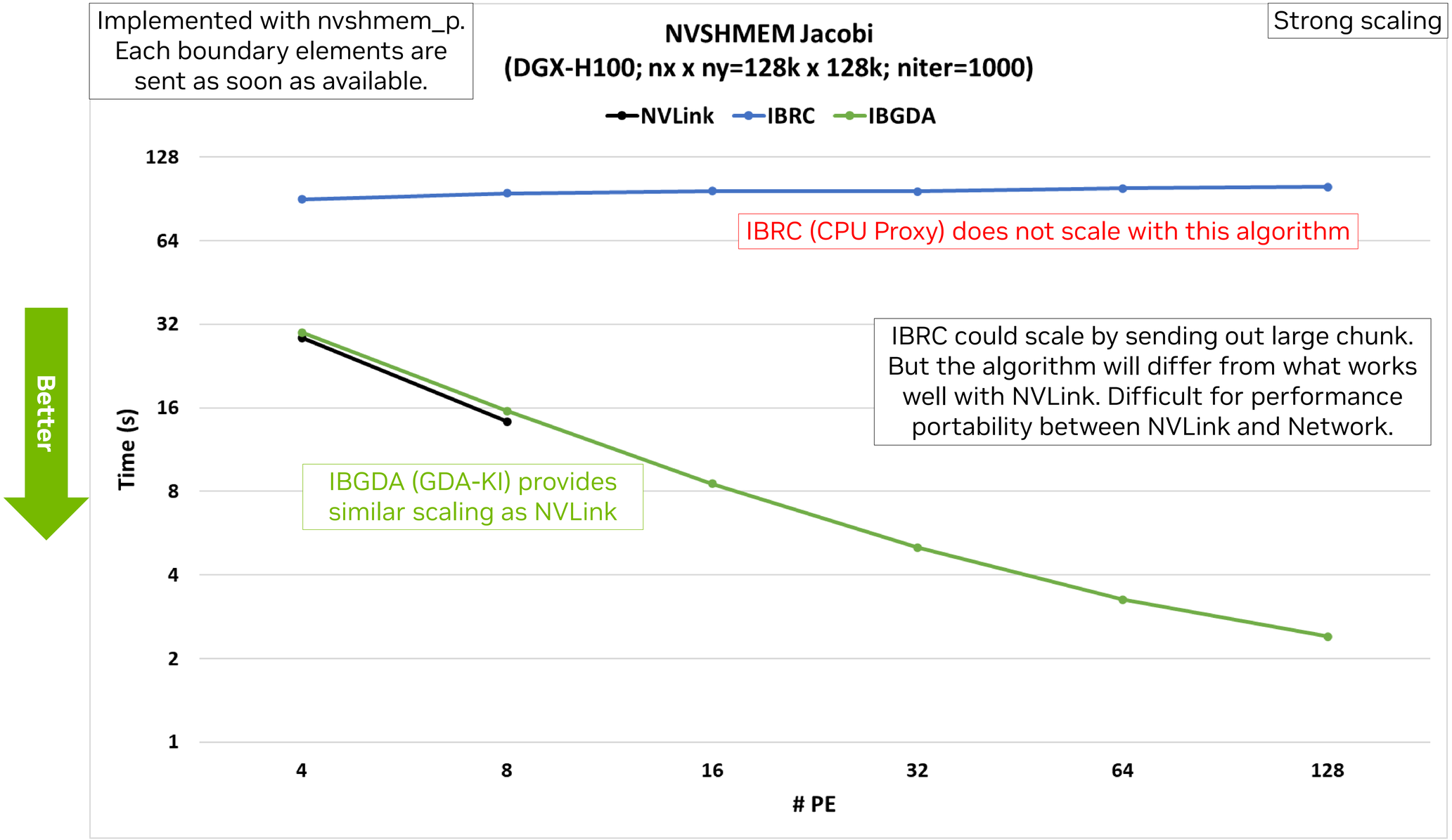


Case Study

Jacobi

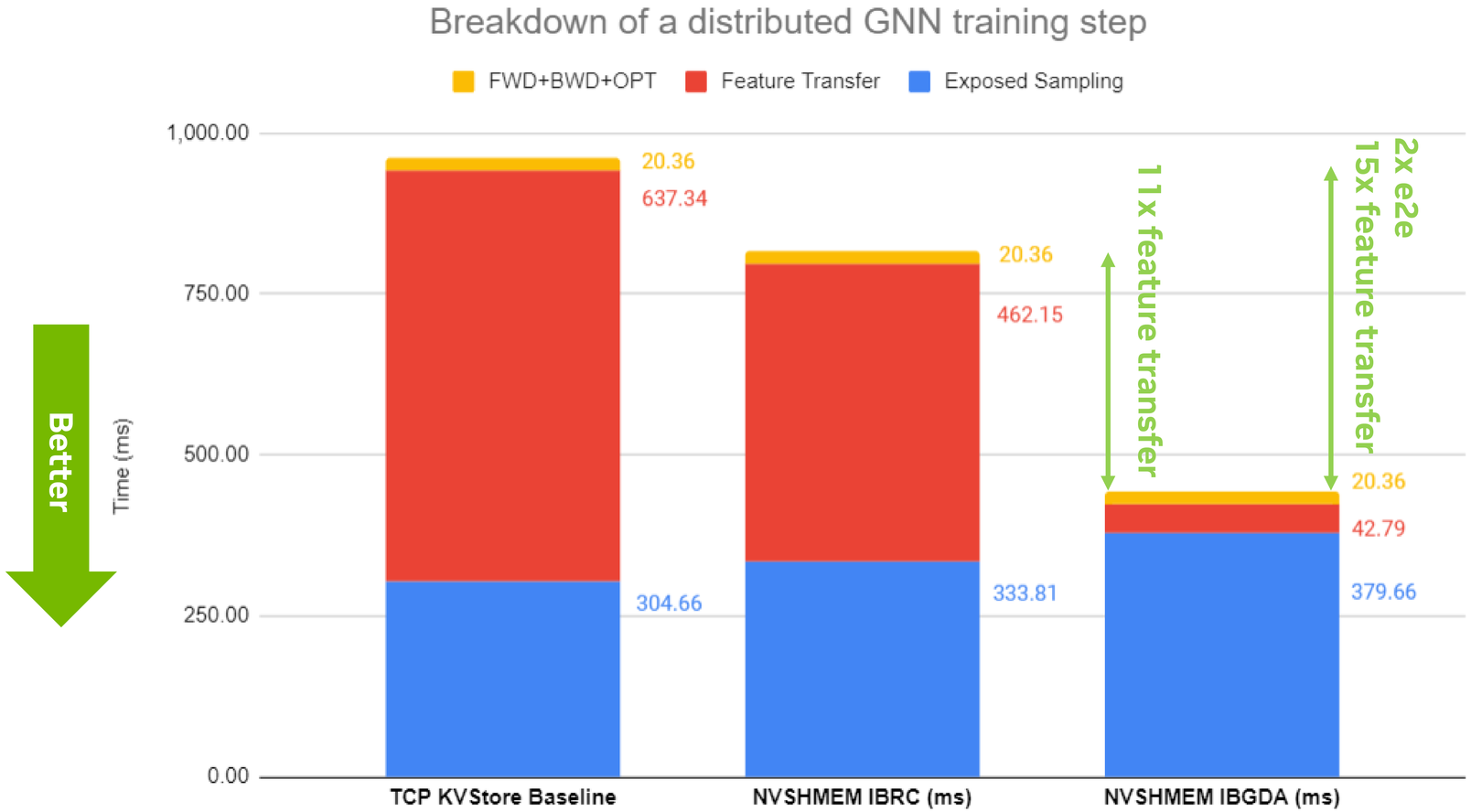
Implemented with nvshmem_p. Each boundary elements are sent as soon as available.

Strong scaling



Case Study

GNN



Details: mag240m link prediction training on 4x DGX A100 with 8x 200Gb/s IB each.
Feature (message) size is 1.5KB with 125K features fetched by each GPU from remote SYSMEMs per training step.

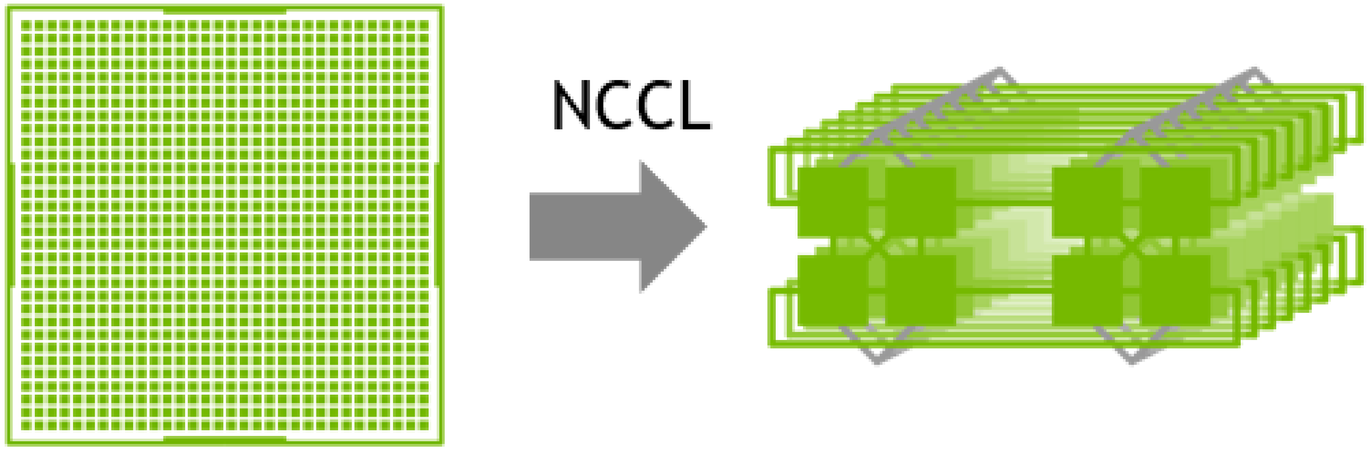
*Slide courtesy of Nicolas Castet.

NCCL IBGDA

NCCL IBGDA

Overview

- NCCL stands for **Nvidia Collectives Communications Library**
- NCCL implements communication primitives optimized for multi-GPU applications
- NCCL is used as a communications backend for all major DL frameworks
- IBGDA is currently an **experimental feature** in NCCL



NCCL IBGDA

Tradeoffs of using GDA-KI

- **Pros**
  - Much greater concurrency, thus much higher message rates
  - No need to write to sysmem to signal the proxy thread, reducing the overhead of a single operation
  - No taking of locks, context switching, or other possible overheads with a host OS
- **Cons**
  - Lose the overlap that comes with a proxy thread
  - CPU cores are individually fast and great for control flow
- **NCCL solution**
  - Offload **ncclSend** and **ncclRecv** using IBGDA
  - The target use case is **AlltoAll** communications patterns

NCCL AlltoAll

Overview

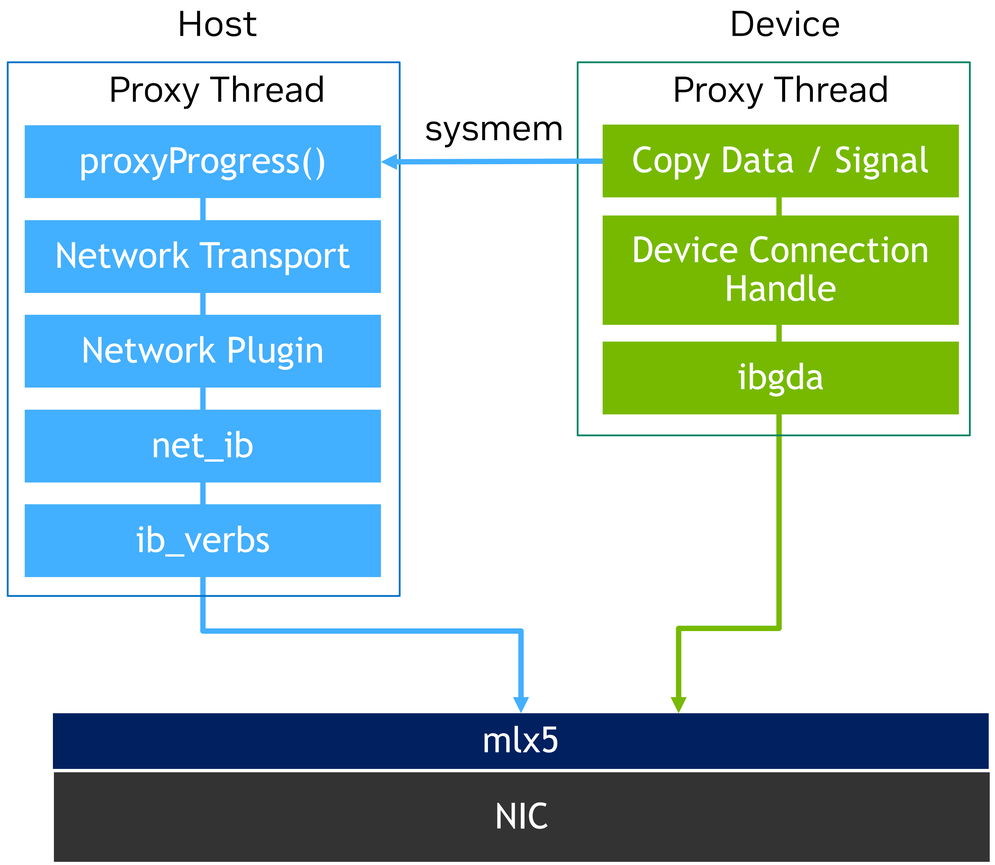
- AlltoAll requires N^2 roundtrips
- NCCL uses a single proxy thread per-communicator, which is responsible for all network operations
  - This is typically mapped one communicator per-GPU

```
NCCLCHECK(ncclGroupStart());
for (int r=0; r<nRanks; r++) {
  NCCLCHECK(ncclSend(((char*)sendbuff)+r*rankOffset, count, type, r, comm, stream));
  NCCLCHECK(ncclRecv(((char*)recvbuff)+r*rankOffset, count, type, r, comm, stream));
}
NCCLCHECK_COMM_WAIT(ncclGroupEnd(), comm);
```


NVIDIA.

NCCL Transports

IBGDA in NCCL

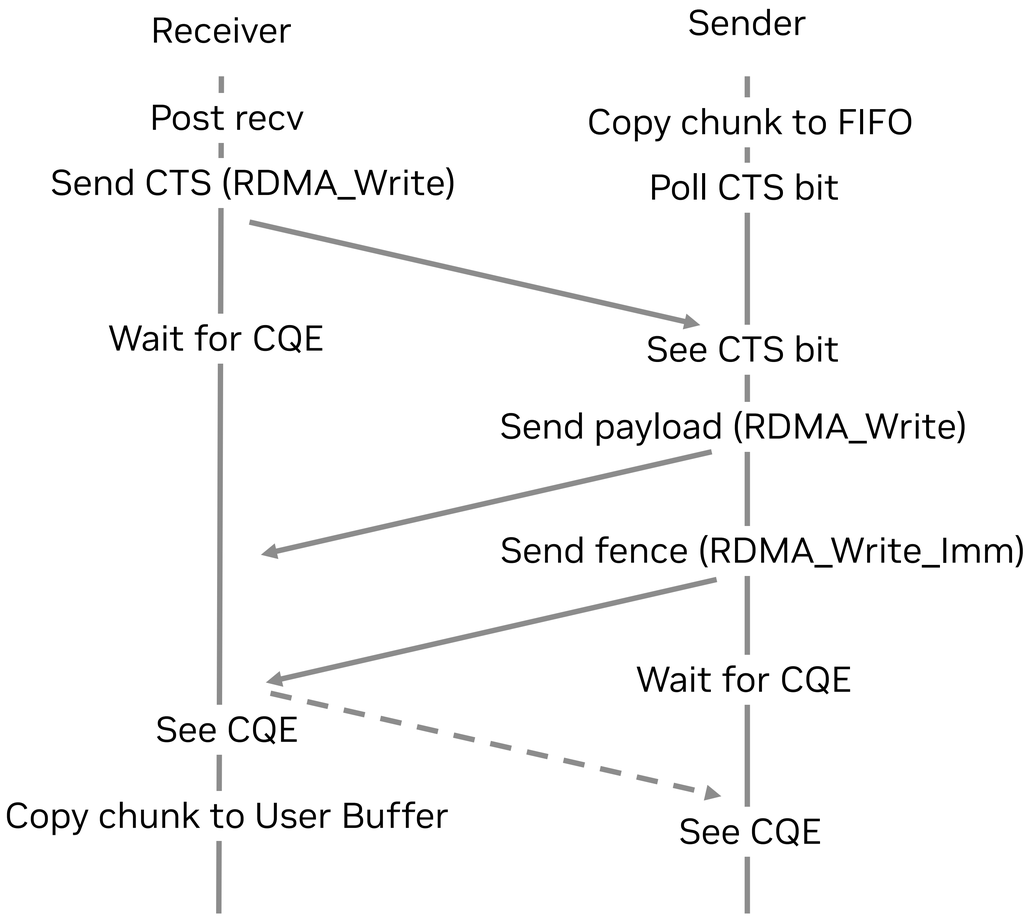


- NCCL core abstracts inter-GPU connections with the concept of a transport.
- The network transport require a **plugin** which implements networking primitives
- NCCL comes with two internal network plugins: **net_ib** and **socket**
- **IBGDA** in NCCL is implemented as an **external network plugin**

*Net IB are CPU Proxy transports
*IBGDA is an GDA-KI transport

NCCL Network Protocol

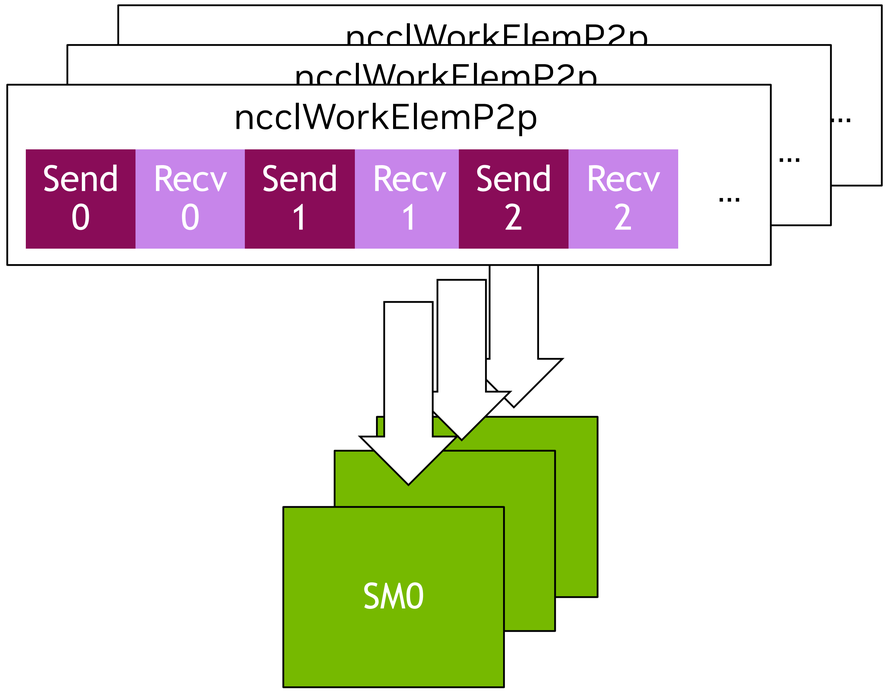
IBGDA in NCCL



- NCCL uses a **peer-to-peer** synchronization per-message
- The receiver arrives, posts a recv and sends CTS
- The sender copies their data chunk to the FIFO
- The sender sends a payload and fence operation to the NIC
- The receiver sees completion and copies the destination FIFO buffer to the user buffer

NCCL AlltoAll GPU Utilization

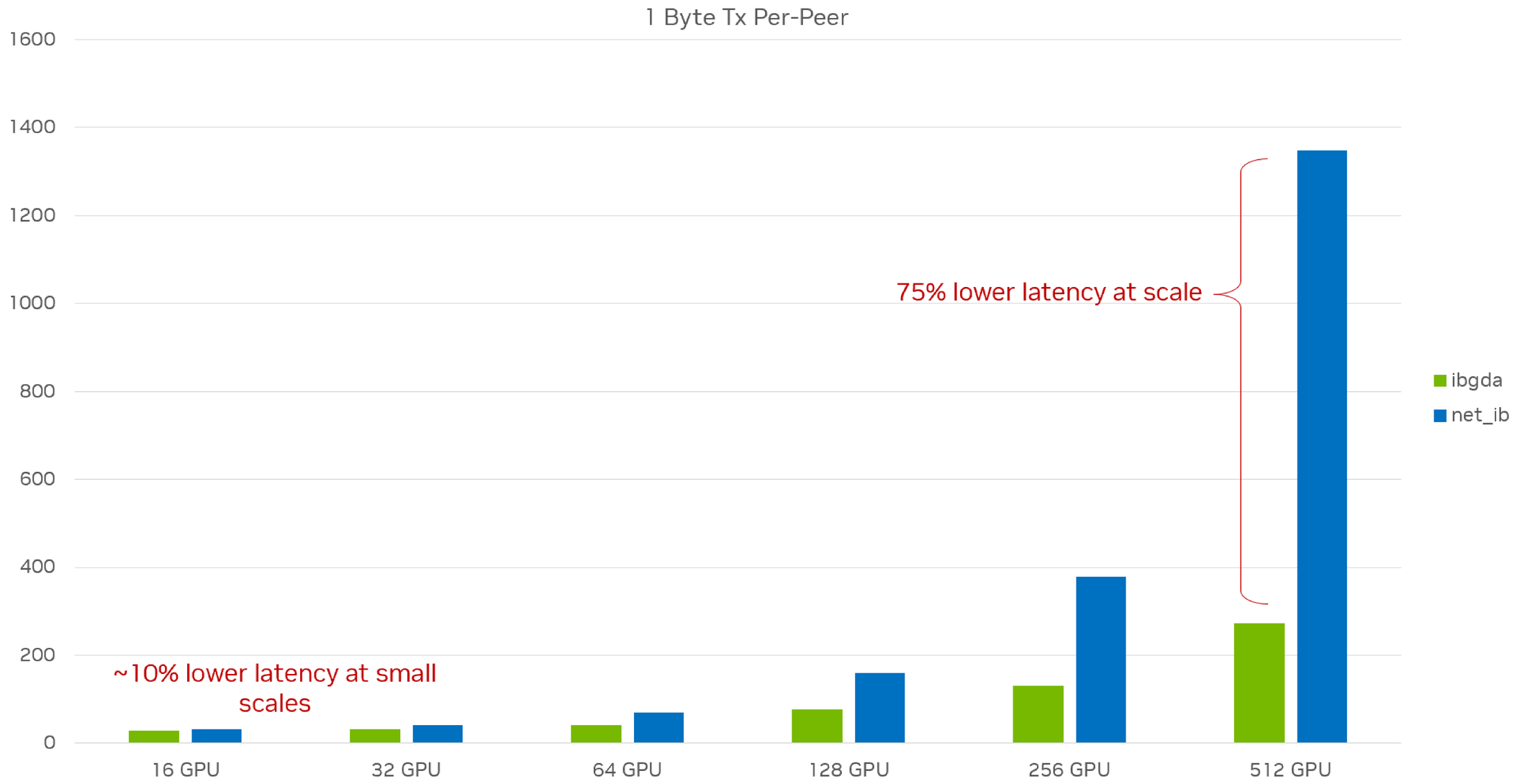
IBGDA in NCCL



- NCCL packs up to 8 send and 8 receive operations into a single **work element**
- One work element is executed by a single **channel**
- One channel contains its own network connections and FIFOs and is progressed by one **SM**
- Each channel can progress its own work in parallel
- NCCL by default uses up to 16 channels at once
- Thus, up to **128** send/recv pairs are executed in parallel
- For every send and recv, NCCL has a **control thread** which IBGDA uses to do network operations

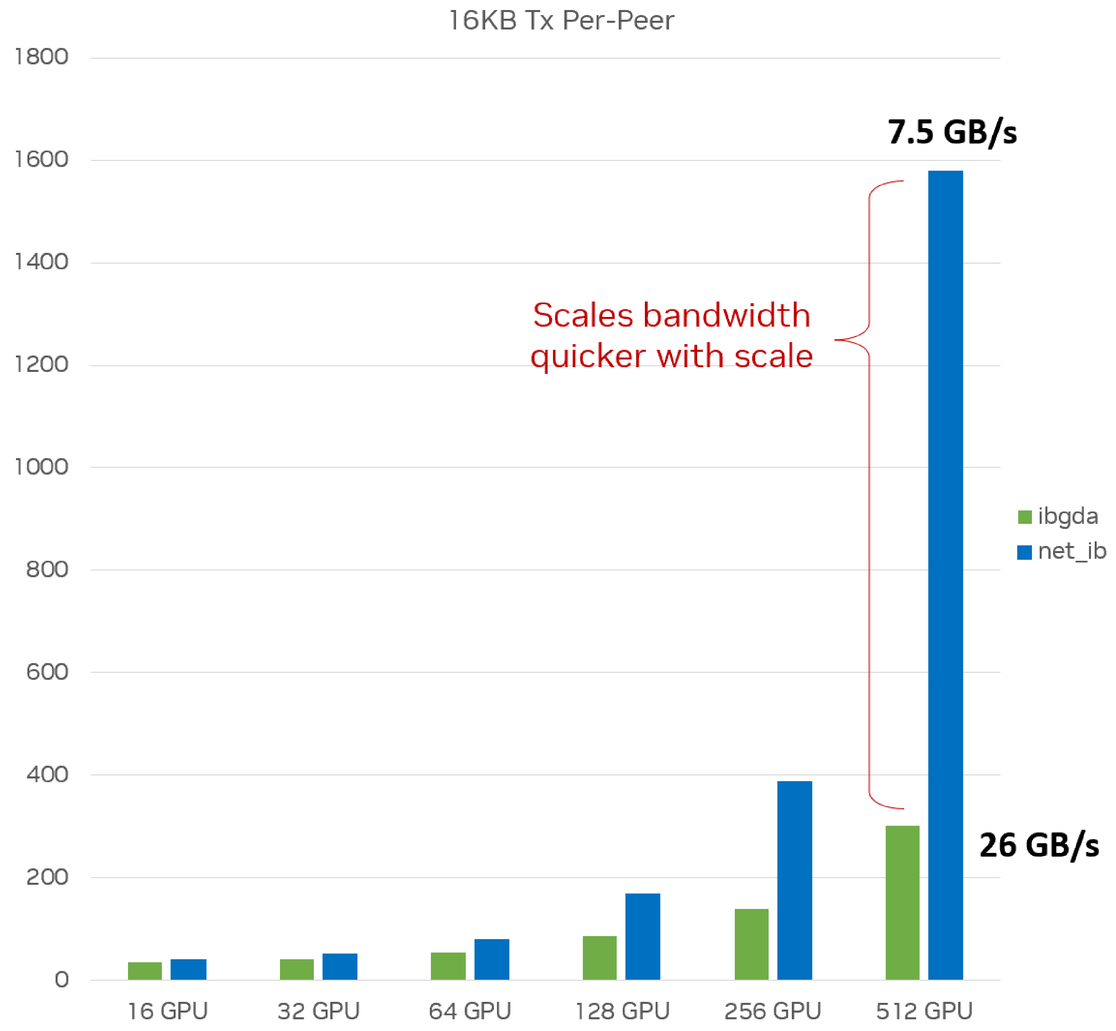
NCCL IBGDA Performance – Base Latency

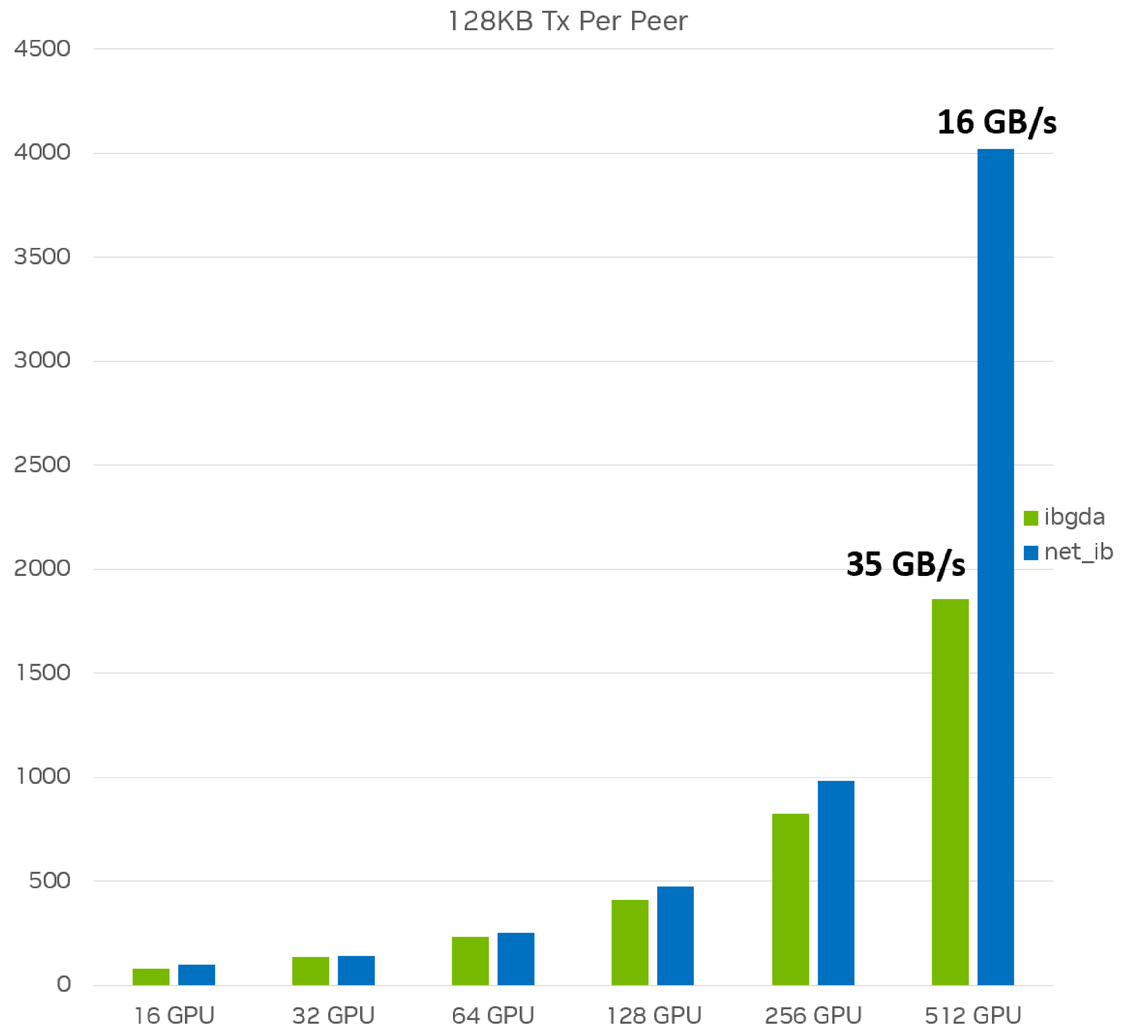
AlltoAll Latency on DGX-H100, **Smaller is Better**



NCCL IBGDA Performance – Mid Message Sizes

AlltoAll Latency on DGX-H100, **Smaller is Better**





NVIDIA.

Future Work

- Tuning for large message sizes
- Aggregation / Doorbell Coalescing
- Rail-optimized traffic
- Deeper pipelining
- User buffer registration

Closing Remarks

Summary

- Magnum IO is the architecture for parallel, intelligent data center IO
- Updates on GPUDirect technologies
  - support on various platforms, especially Grace-Hopper
  - GPUDirect RDMA with DMA-BUF
  - GPUDirect Async – Kernel Initiated (GDA-KI) moves the NIC control plan to GPU
- NVSHMEM IBGDA
  - An implementation of GDA-KI in NVSHMEM
  - Important tuning knobs for users
  - 512B message size to saturate BW, up to 175x higher message rate compared with IBRC, good strong scaling similar to NVLink
- NCCL IBGDA
  - An implementation of GDA-KI in NCCL
  - Up to 75% lower all-to-all latency compared with net_ib

Magnum IO @ GTC2024

S61339 - Multi GPU Programming Models for HPC and AI

S62129 - Training Deep Learning Models at Scale: How NCCL Enables Best Performance on AI Data Center Networks

P61487 - Toward Optimizing File IO on GPU Clusters

S62559 - Accelerating and Securing GPU Accesses to Large Datasets

CWE61229 - Inter-GPU Communication Techniques and Libraries for HPC and AI

P63163 Toward IOWN: Real-Time DNN Inference With CUDA Graphs and DOCA GPUNetIO