Multi-GPU Programming in NCCL and NVSHMEM

Jeff Hammond
Distinguished Engineer
GPU Communication Software

GPU MODE *Lecture 67: NCCL and NVSHMEM*
https://youtu.be/zxGVvMN6WaM

40 million

2 billion

40 million FP64 op/s…

2 billion FP16 op/s…

NVIDIA

40 million FP64 op/s…

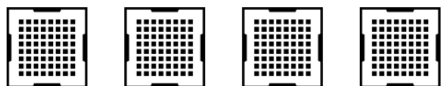
2 billion FP16 op/s…

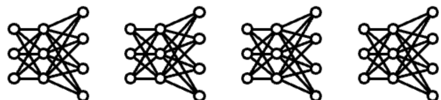
…in 1 microsecond

Overview of NCCL and NVSHMEM

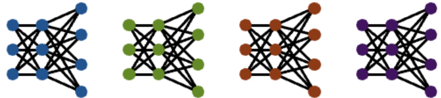
Overview of NCCL

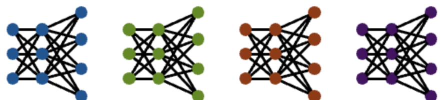
PCI Server





	Data Parallelism / FSDP	All-reduce, all-gather, reduce-scatter
	Tensor Parallelism	All-reduce, all-gather, reduce-scatter
	Pipeline Parallelism	Send / receive
	Expert Parallelism	All-to-all





DGX/HGX

Large Systems

"NCCL: The Inter-GPU Communication Library Powering Multi-GPU AI", Sylvain Jeaugey. GTC25-S72583: https://www.nvidia.com/en-us/on-demand/session/gtc25-s72583/

NCCL Ecosystem

Enabling breakthrough AI research at unprecedented scales

CLOUD & INFRASTRUCTURE

ORACLE

AI INNOVATORS & RESEARCH

ANTHROP\C

NCCL Host API Examples

// host APIs
ncclResult_t **ncclAllReduce**(const void* sendbuff, void* recvbuff, size_t count, ncclDataType_t datatype, ncclRedOp_t op, ncclComm_t comm, cudaStream_t stream);

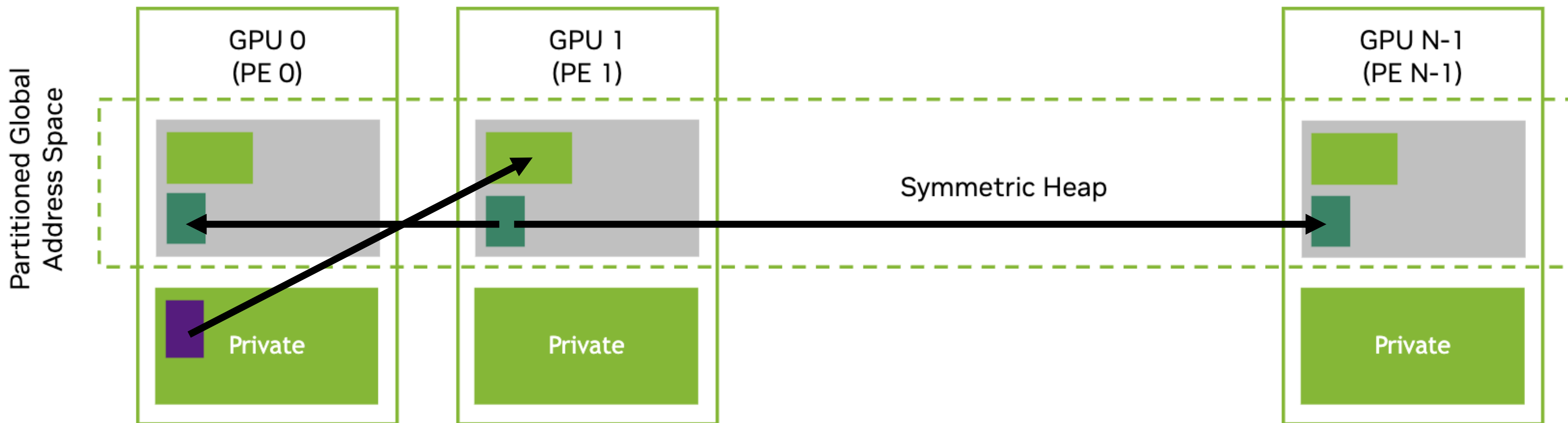
ncclResult_t **ncclGroupStart**();

ncclResult_t **ncclSend**(const void* sendbuff, size_t count, ncclDataType_t datatype, int peer, ncclComm_t comm, cudaStream_t stream);

ncclResult_t **ncclRecv**(void* recvbuff, size_t count, ncclDataType_t datatype, int peer, ncclComm_t comm, cudaStream_t stream);

ncclResult_t **ncclGroupEnd**();

Overview of NVSHMEM



"NVSHMEM: GPU-Integrated Communication for NVIDIA GPU Clusters", Akhil Langer and Jim Dinan.
GTC21-S32515: https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s32515/

NVSHMEM AI Ecosystem

C Host and Device APIs	NVSHMEM4Py

Use-cases

Custom Communication Kernels (EP in MoE)
Fused compute-comm kernel
Zero-SM and low latency collectives
One-sided pt-to-pt comms

Organizations

deepseek

perplexity

ByteDance

NVIDIA

Meta

Google

Libraries/Frameworks

OpenXLA

FlashInfer

PyTorch

Google: JAX - Pallas & MosaicGPU
Perplexity: PPLX kernels
Deepseek: DeepEP library
Meta: PyTorch SymmetricMemory
ByteDance: Triton Distributed, TileLink, FLUX
Nvidia: NeMO, cuBLASMp, cuFFTMp, nvmath-python, Numba-CUDA (NVSHMEM 3.5)

NVSHMEM API Examples

// host APIs
void nvshmem_putmem(void *dest, const void * src, size_t nb, int pe)
void nvshmem**x**_putmem_**on_stream**(void *dest, const void * src, size_t nb,
int pe, cudaStream_t stream)

// **device APIs**
void nvshmem_putmem(void *dest, const void *src, size_t nb, int pe)
void nvshmem**x**_putmem_**block**(void *dest, const void * src, size_t nb, int pe)
void nvshmem**x**_putmem_**warp**(void *dest, const void * src, size_t nb, int pe)

// host- and device-initiated collectives...
// (cooperative group launch required for synchronizing operations)

NVIDIA

NCCL *Device* API Examples

```
// host APIs
ncclResult_t ncclCommWindowRegister(comm, buffer, size, &win,
                                    NCCL_WIN_COLL_SYMMETRIC);
ncclResult_t ncclDevCommCreate(comm, &reqs, &devComm);
ncclResult_t ncclGetLsaDevicePointer(ncclWindow_t window, size_t offset,
                                     int lsaRank, void **outPtr);

// device APIs
void * ncclGetLsaPointer(win, offset, peer);
void * ncclGetLsaMultimemPointer(win, offset, devComm);

// see also GIN (read the docs)
ncclGin::put(..)
ncclGin::flush(..)
```

NCCL symmetric memory was introduced in 2.28, GPU-initiated networking (GIN) in 2.28.7 and host-initiated one-sided in 2.29.

A Brief History of Communication Libraries

The Two Paradigms of HPC Communication

- Two-sided aka message passing, as exemplified by MPI-1.
  - The MPI Forum started in 1993, to standardize what was then a large set of similar but incompatible messaging libraries.
  - MPI Send-Recv can be implemented using e.g. POSIX sockets and thus are considered universally portable.
  - MPI was designed when CPUs were (often) much faster than networks…
  - Two-sided communication combines synchronization and data movement.
- One-sided aka remote memory access (RMA), as exemplified by SHMEM.
  - SHMEM was created for the Cray T3D, which was a revolutionary MPP system that supported direct access to remote memory.
  - (Open)SHMEM's design assumes an SMP or RDMA network and discourages implementations that don't support asynchronous progress in hardware.
  - One-sided communication decouples synchronization and data movement.

Brief MPI history: https://www.hpcwire.com/2017/05/01/mpi-25-years-old/
Original SHMEM paper: https://cug.org/5-publications/proceedings_attendee_lists/1997CD/S95PROC/303_308.PDF

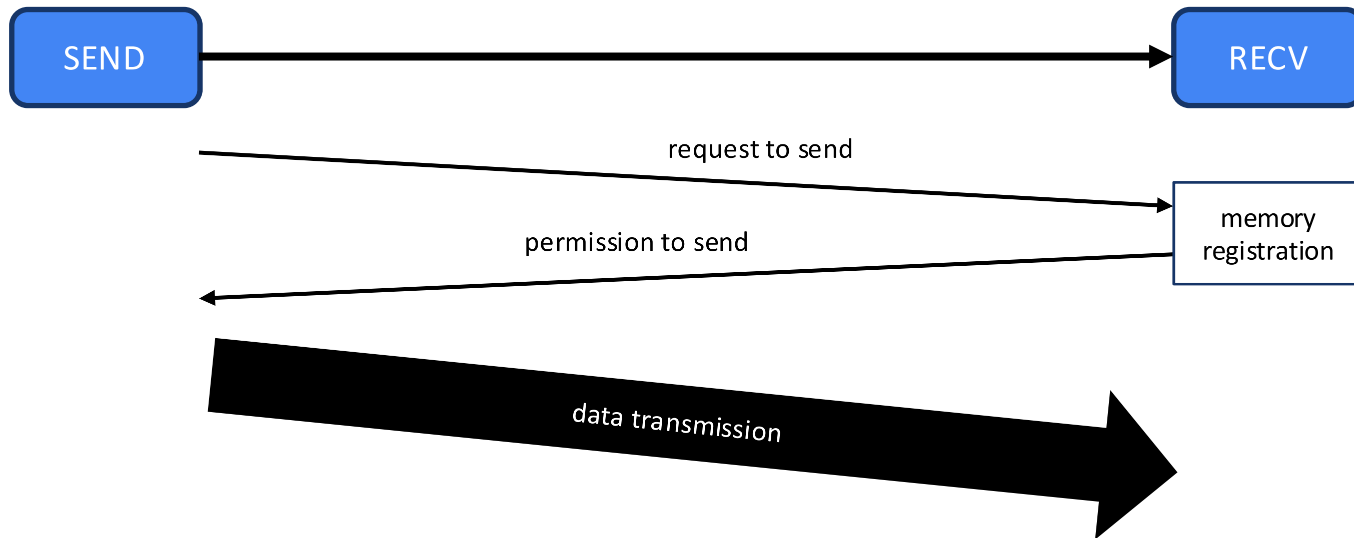
Two-sided communication

Sender knows:
Input buffer address
Input buffer size
Input data type
Message tag
Receiver ID

The matching protocol allows the input buffer to be written into the output buffer.

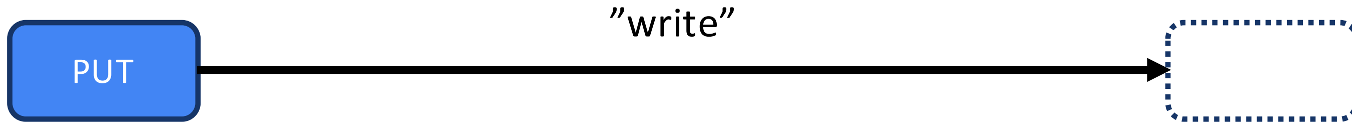
Receiver knows:
Output buffer address
Output buffer size
Output data type
Message tag
Sender ID

Two-sided communication



OSU paper on MPI Send-Recv on IB with RDMA:
https://mvapich.cse.ohio-state.edu/static/media/publications/abstract/surs-ppopp06.pdf

One-sided communication



Initiator knows:
Input buffer address
Output buffer address/offset
Buffer size
Data type
Target ID

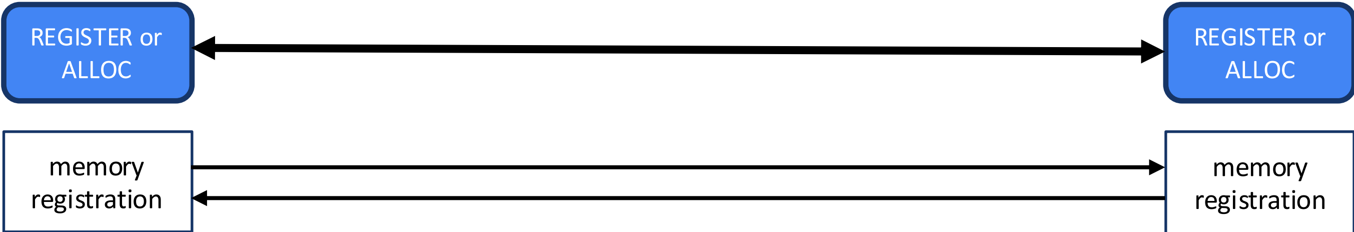
Before the PUT call is made, memory for the target buffers is registered with both sides.

One-sided communication

Before the GET call is made, memory for the target buffers is registered with both sides.

Initiator knows:
Input buffer address/offset
Output buffer address
Buffer size
Data type
Target ID

One-sided communication



Why does this matter?

- If the synchronizing steps do not line up, processors may idle while waiting.
- Synchronization between host and device should be minimized.
- Synchronizing GPU blocks/threads should also be avoided.
- Data transmission is much easier to offload to specialized hardware (RDMA interconnects like IB, GPU copy engines aka CE) than message matching logic.

Most important AI/HPC communication patterns are persistent, so amortizing away setup costs is beneficial.

MPI vs NCCL

The most obvious difference is stream support, but there are more differences:

- MPI allows underflow: send_count < recv_count.
- MPI datatypes allow arbitrary, nested, heterogeneous, noncontiguous data.
- MPI supports tags, which distinguish messages within a communicator.
- MPI supports “wildcards” (e.g. ANY_SOURCE), which make message matching hard.
- MPI supports multiple ranks per GPU.

NCCL supports none of these features. MPI 4.0 allows some of them to be disabled, but as they are on by default, little to no effort has gone into optimizing for these scenarios.

NCCL also supports multiple ranks (i.e. GPUs) per process, which some AI workloads require.

Mitigating MPI Message Matching Misery. ISC 2016. https://doi.org/10.1007/978-3-319-41321-1_15

GPU Communication

DGX-H100

Spec	Per H100 GPU	System Total
FP64 Performance	34 TFLOPS	272 TFLOPS
FP64 Tensor Core	67 TFLOPS	536 TFLOPS
Memory Bandwidth	3.35 TB/s	26.8 TB/s
GPU Memory	80 GB HBM3	640 GB
NVLink Bandwidth	900 GB/s bidirectional	3.6 TB/s bisection

CPU CPU

CX7 HCA/NIC w/ PCIe Switch | CX7 HCA/NIC w/ PCIe Switch | CX7 HCA/NIC w/ PCIe Switch | CX7 HCA/NIC w/ PCIe Switch | CX7 HCA/NIC w/ PCIe Switch | CX7 HCA/NIC w/ PCIe Switch | CX7 HCA/NIC w/ PCIe Switch | CX7 HCA/NIC w/ PCIe Switch

H100 H100 H100 H100 H100 H100 H100 H100

NVSwitch NVSwitch NVSwitch NVSwitch

OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP OSFP

4 NVLinks 5 NVLinks 1 400Gb

DGX H100

See NVLink @ https://hc34.hotchips.org/

Comparing runtime systems with exascale ambitions using the Parallel Research Kernels

R. F. Van der Wijngaart[1], A. Kayi[1], J. R. Hammond[1], G. Jost[1], T. St. John[1], S. Sridharan[1], T. G. Mattson[1], J. Abercrombie[2], and J. Nelson[2]

[1] Intel Corporation, Hillsboro, Oregon, USA.
[2] University of Washington, Seattle, WA, USA.

MB/s
1000000
800000
600000
400000
200000
0

MPI1
MPIOPENMP
MPIRMAfence
MPIRMAflush
MPISHM12
MPISHM24
SHMEM
BUPC
CRAYUPC
CHARM++1
GRAPPA

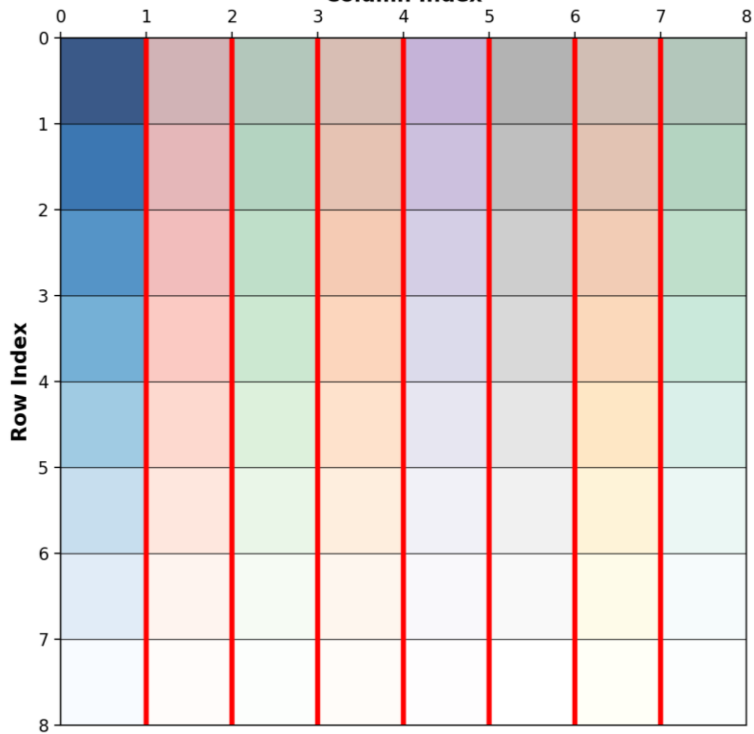
24 48 96 192 384 768 1536 3072 6144 12288
Cores

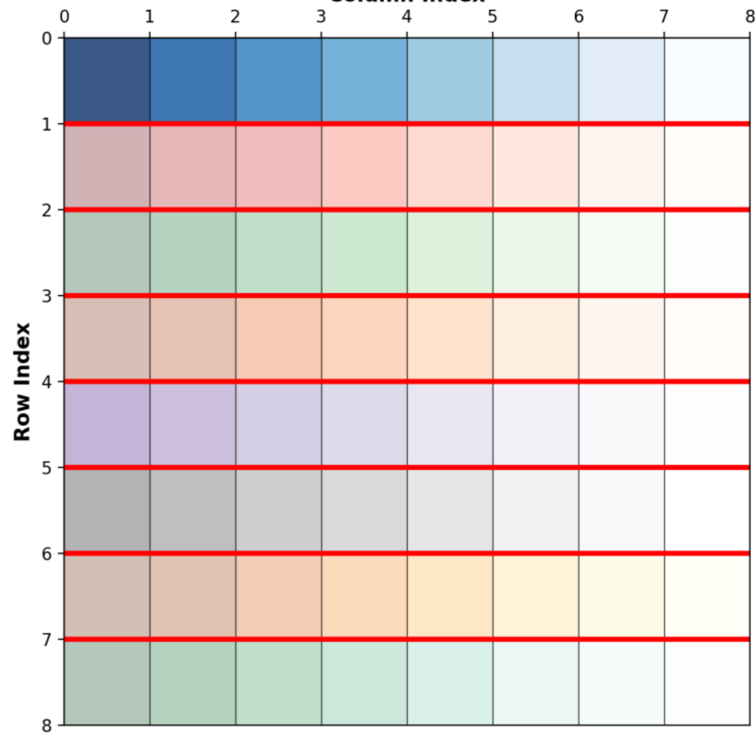
https://link.springer.com/book/10.1007/978-3-319-41321-1

https://github.com/ParRes/Kernels/

https://youtu.be/HTbjM5GDIRM

Matrix Transpose Communication - Processing Rank 0





Column blocks are mapped onto GPUs 0..7

NVSHMEM implementations of transpose

Alltoall
Block exchange (**A2A** comm)
Transpose all blocks
Increment input matrix

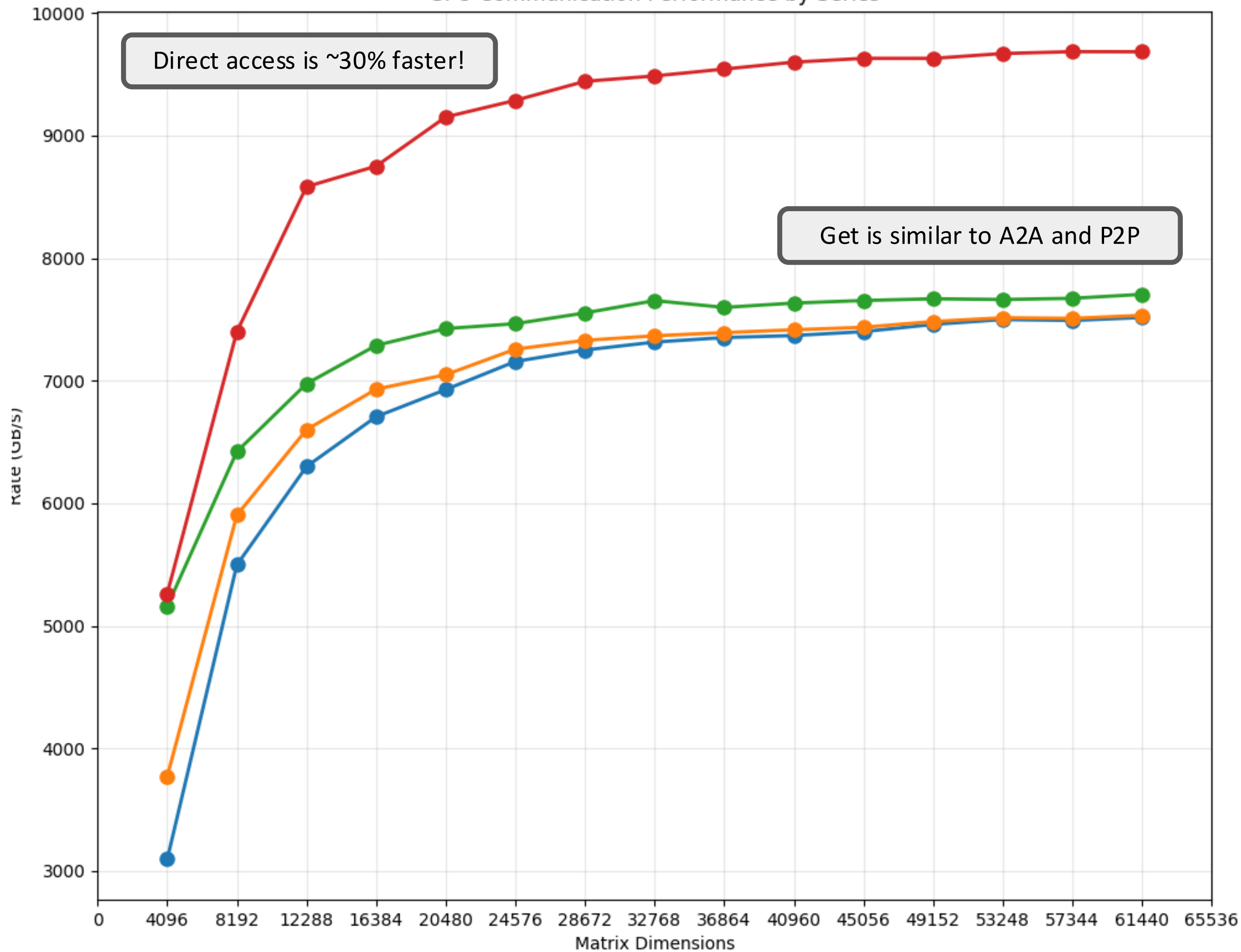
```
# Algorithm
for (i,j) in matrix:
  B(i,j) += A(j,i)
  A(j,i) += 1.0
```

One-sided – explicit comm
For all blocks
  Fetch a block (**Get**)
  Transpose the block

Barrier
Increment input matrix

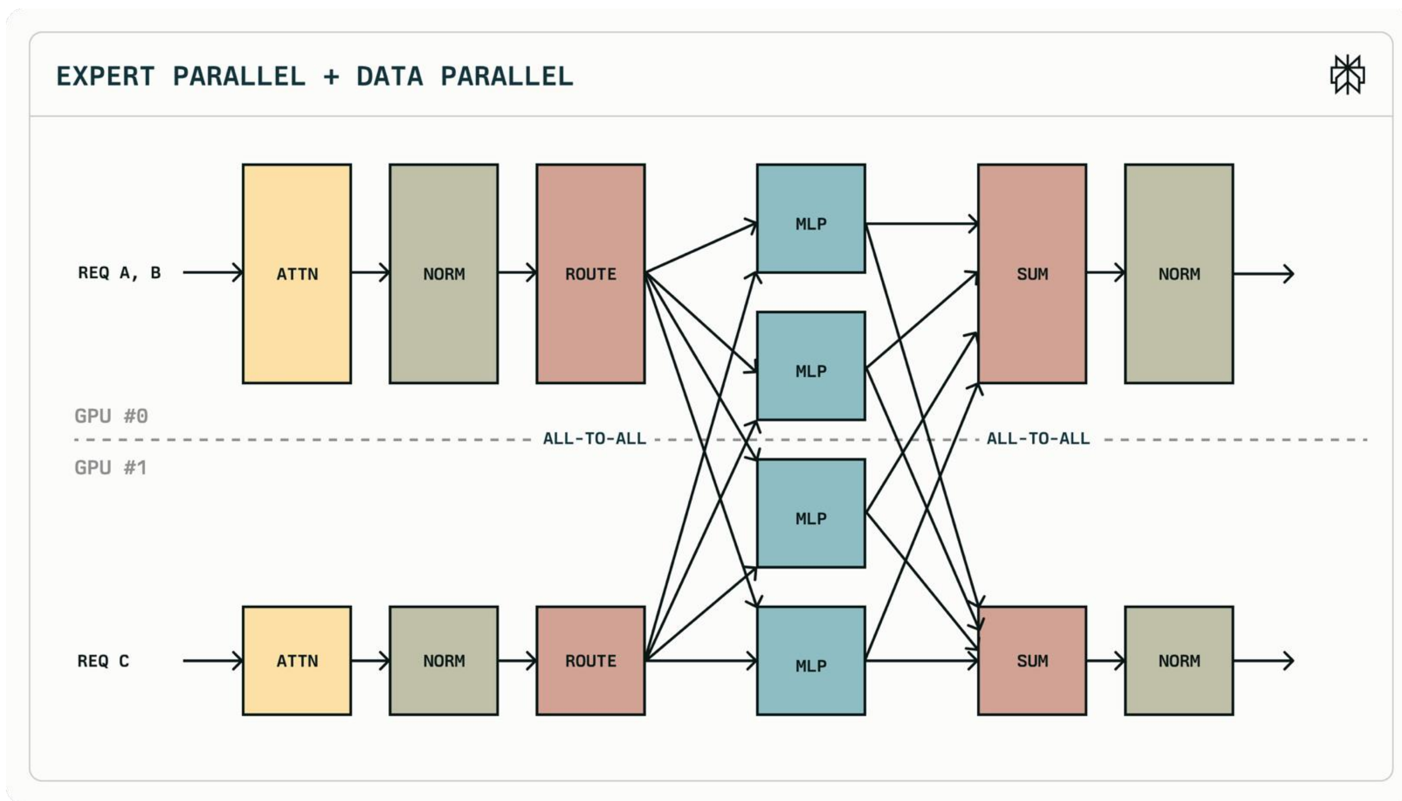
One-sided – implicit comm
For all blocks
  Acquire remote pointer
  Transpose the block w/ pointer

Barrier
Increment input matrix


GPU Communication Performance by Series
Direct access is ~30% faster!
Get is similar to A2A and P2P
Rate (GB/s)
Matrix Dimensions
10000
9000
8000
7000
6000
5000
4000
3000
0
4096
8192
12288
16384
20480
24576
28672
32768
36864
40960
45056
49152
53248
57344
61440
65536
NVSHMEM a2a v2 device=false
NVSHMEM a2a bulk device=false
NVSHMEM get v2 device=true
NVSHMEM ptr v2 device=true

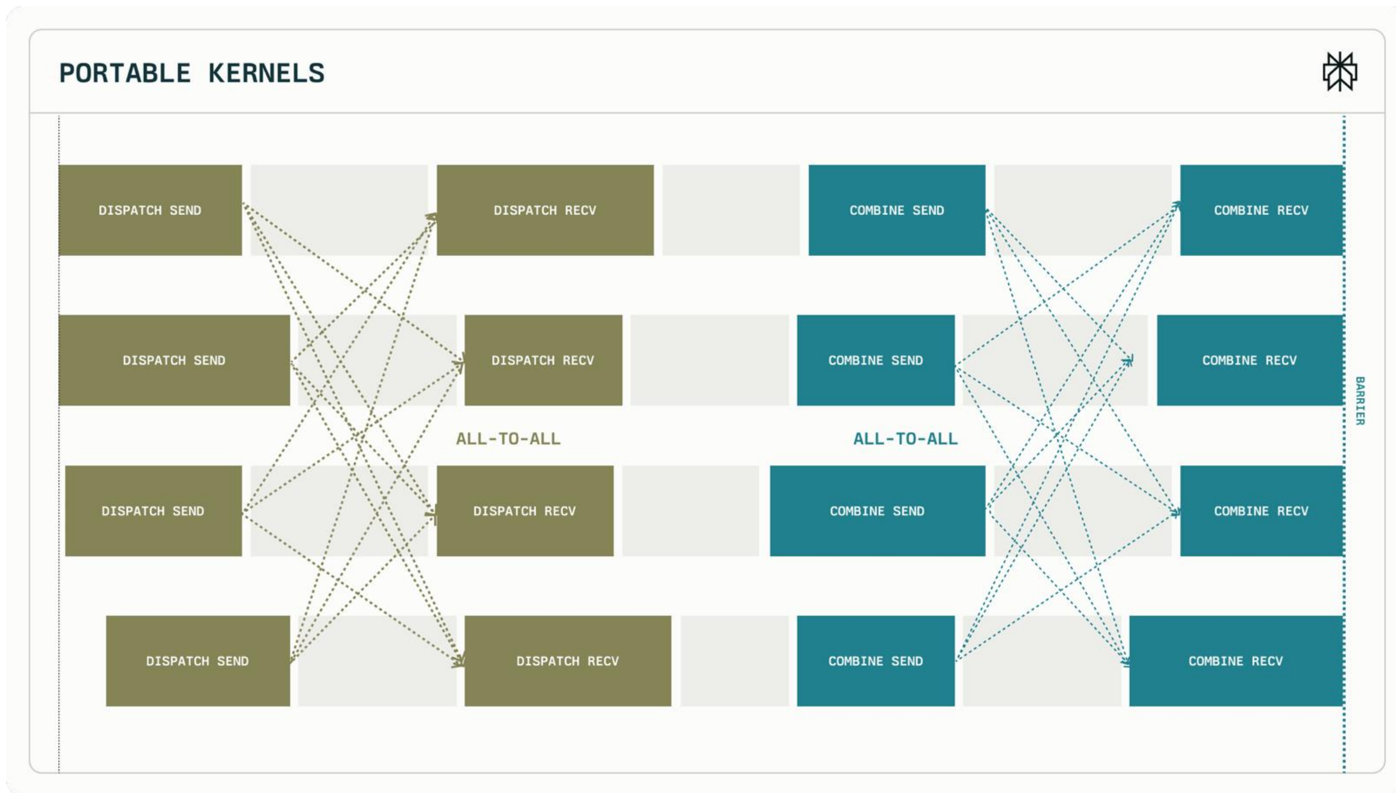
AI Inference

Mixture-of-Experts



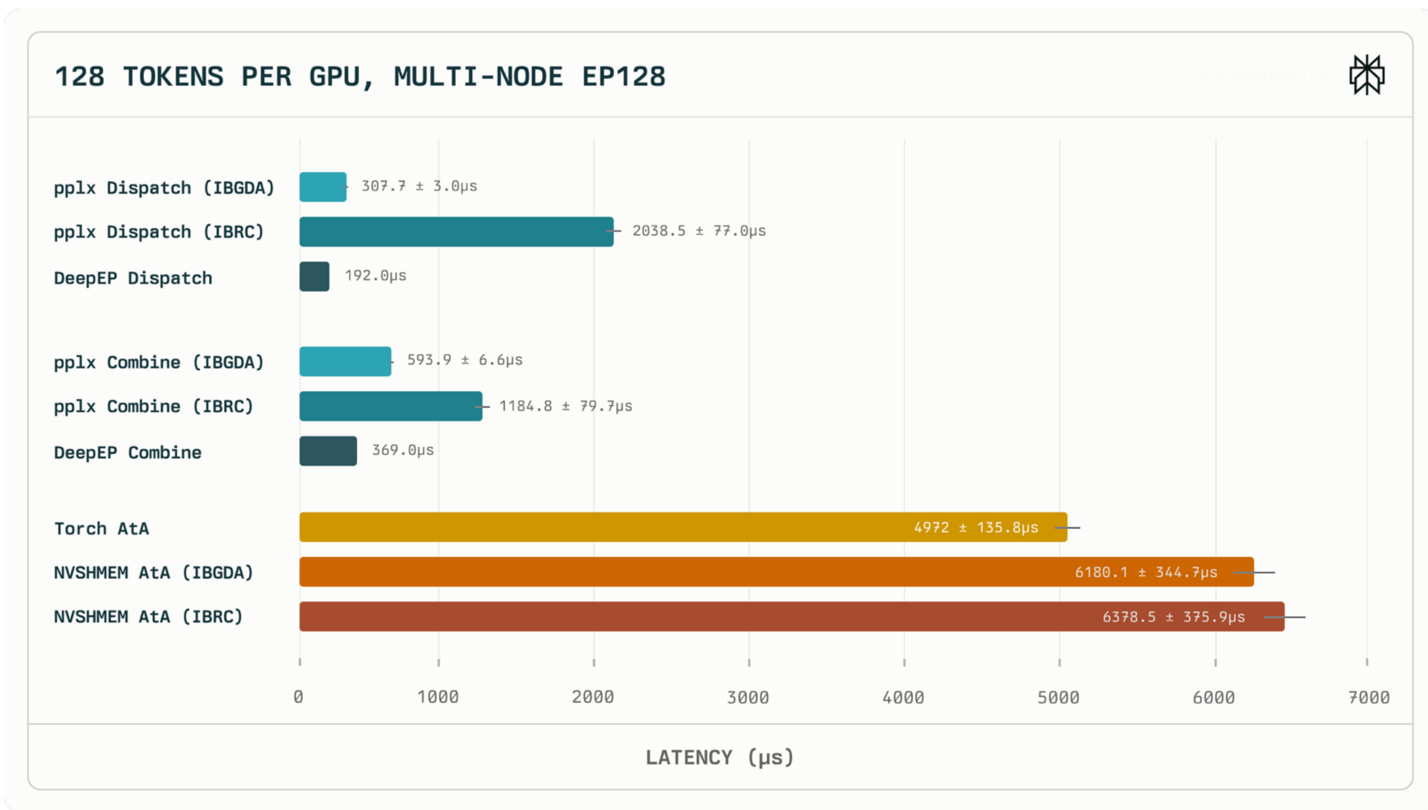
"Efficient and Portable Mixture-of-Experts Communication" by Perplexity.
https://www.perplexity.ai/hub/blog/efficient-and-portable-mixture-of-experts-communication

Mixture-of-Experts



"Efficient and Portable Mixture-of-Experts Communication" by Perplexity.
https://www.perplexity.ai/hub/blog/efficient-and-portable-mixture-of-experts-communication

Mixture-of-Experts



"Efficient and Portable Mixture-of-Experts Communication" by Perplexity.
https://www.perplexity.ai/hub/blog/efficient-and-portable-mixture-of-experts-communication

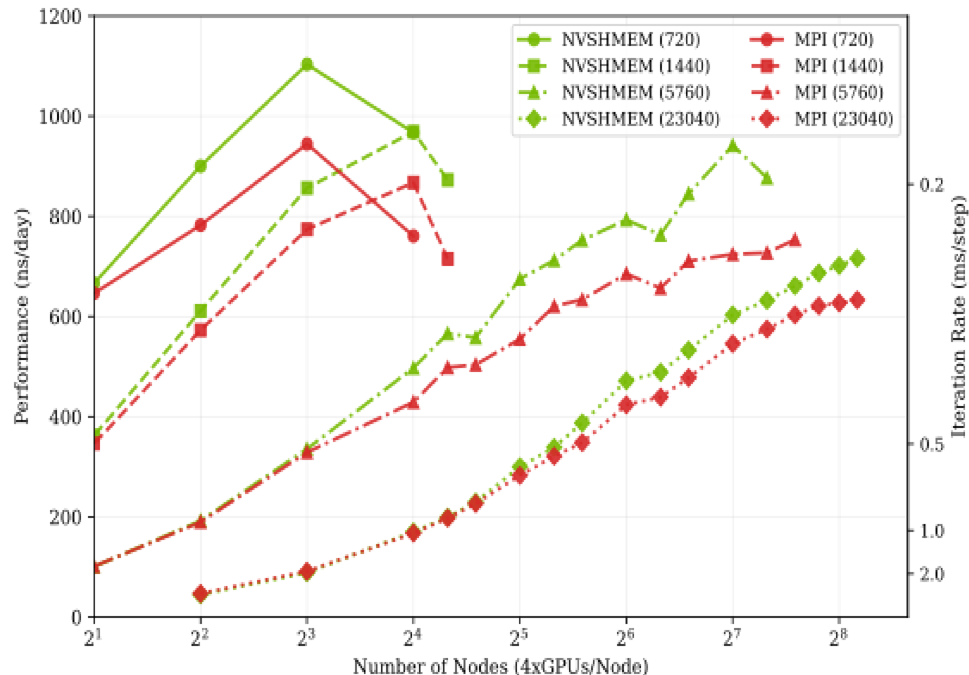
HPC Applications

GROMACS Strong Scaling NVSHMEM vs MPI

- MD strong scaling is latency-bound; CPU-centric MPI adds control-path syncs, limiting overlap on GPU-resident runs.
- GPU-initiated, NVSHMEM-based implementation executed entirely on GPU
- Eliminates CPU–GPU syncs, reduces latency, and improves compute–comm overlap; benefits grow with higher Domain Decomposition dimensionality (2D/3D) and scale.

- **5760k @ 128 nodes**: NVSHMEM ~**1.3×** faster

Credit: Mahesh Doijade and Alan Gray.



Eos - MPI vs NVSHMEM strong scaling simulation performance (ns/day) and iteration rate (ms/step) as a function of the number of nodes. The iteration rate is the average wall-time per time-step, computed across all time-steps. For grappa reaction field inputs 720k – 23040k

NVIDIA GB200 NVL72 Powers Quantum

State of the art accelerated computing provides speedups across quantum workloads

Speedup vs. CPU

	cuQuantum	cuQuantum	cuQuantum	CUDA-Q	CUDA-Q
Workload	Quantum Algorithm Development	Qubit EDA	Quantum Data Generation	Hybrid Applications	Quantum Error Correction
Speedup	800X	1,200X	4,000X	1,300X	500X
GPUs	64 GPUs	72 GPUs	72 GPUs	72 GPUs	64 GPUs

CPU: Qiskit, QuTiP, or LDPCv2 on 2-Socket AMD EPYC 9654, 96 Cores Per Socket, 2 Threads per Core, 3.7GHz Max Freq, 770GiB 4800MHz DDR5

Take-away messages

- HPC is changing from being CPU-centric to GPU-centric. Evolving MPI codes to MPI+X codes, where X=CUDA/OpenACC/etc. isn't enough anymore.
- NCCL is the natural evolution of MPI to GPUs and has MPI-3 features now. Python support is now available in nvshmem4py and nccl4py.
- GPU-native networking is exploding: 1.8 TB/s NVLINK bandwidth in GB300 with load-store connectivity to 72 GPUs (and more).
- GPUs do >40M FP64 operations (>2B FP16) in 1 microsecond (lower bound on network/synchronization costs). Asynchrony is critical to efficiency.

The end

Why MPI on GPUs is Hard

- Is this a CUDA GPU or an OpenCL GPU?
  - Forward-progress guarantees are important.
  - Unified memory is required for a complete MPI implementation.
- MPI doesn't provide a natural way to support feature subsetting.
- MPI support for NUMA doesn't easily extend to GPUs.
  - MPI endpoints were proposed for other reasons but are the answer here.

NCCL is MPI for GPUs

- Stream semantics in everything.
- Only implements patterns that make sense.
- Supports GPU endpoints…

Hopefully, MPI-6 will catch up to NCCL…

https://github.com/NVIDIA/mpi-acx