



GPU Native Communications and Agentic Development in HPC Software

Jeff Hammond
Distinguished Engineer
GPU Communication Software

40 million

2 billion

40 million FP64 ops...

2 billion FP16 ops...

40 million FP64 ops...

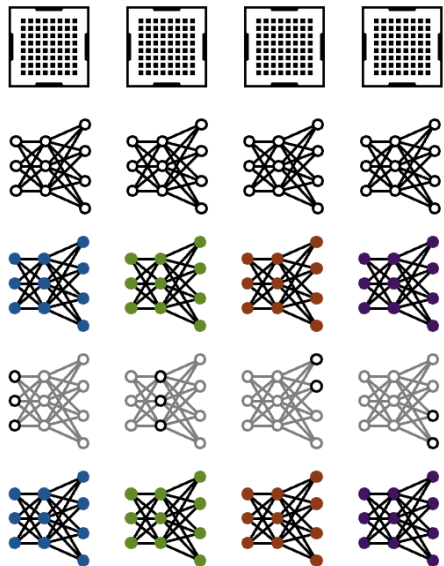
2 billion FP16 ops...

...in 1 microsecond



Overview of NCCL and NVSHMEM

Overview of NCCL



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



PCI Server



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



ANTHROPIC



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);
```

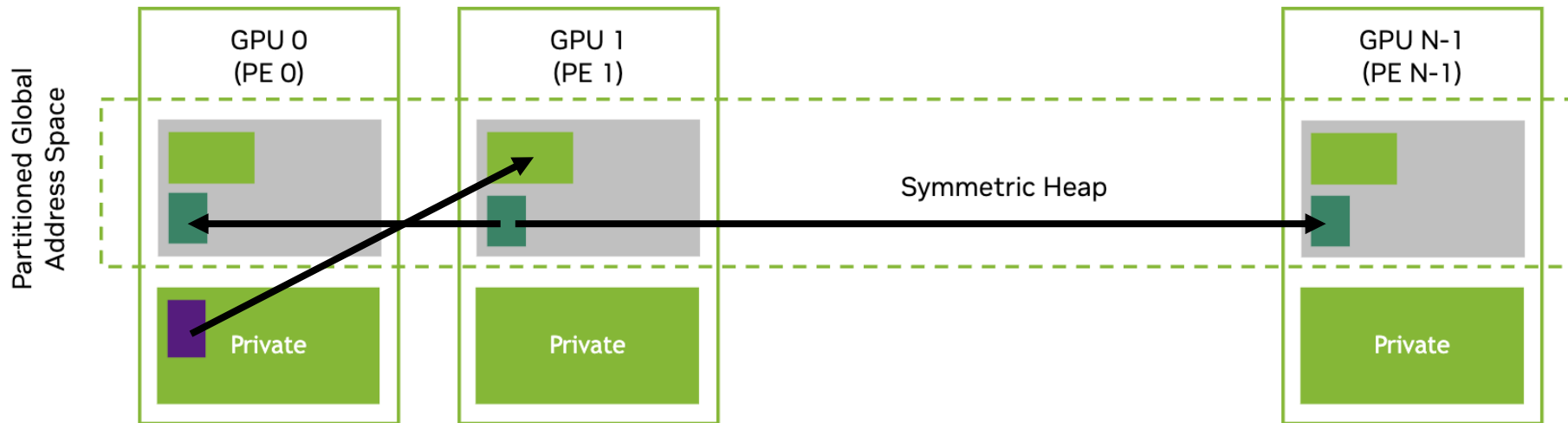
```
ncclResult_t ncclGroupStart();
```

```
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



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 nvshmemx_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 nvshmemx_putmem_block(void *dest, const void * src, size_t nb, int pe)
void nvshmemx_putmem_warp(void *dest, const void * src, size_t nb, int pe)
```

// host- and device-initiated collectives...

// (cooperative group launch required for synchronizing operations)

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 for NVLINK**

```
void * ncclGetLsaPointer(win, offset, peer);  
void * ncclGetLsaMultimemPointer(win, offset, devComm);
```

// **device APIs for the network**

```
ncclGin::put(..)  
ncclGin::flush(..)
```



A Brief History of Communication Libraries

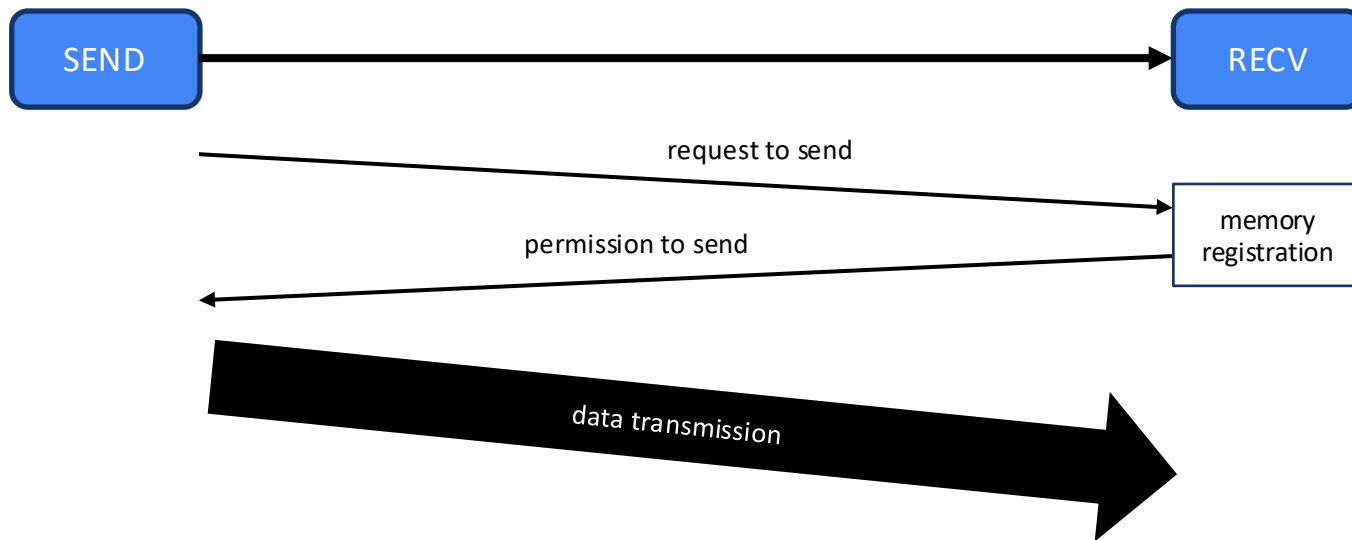
The Two Paradigms of HPC Communication

- 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

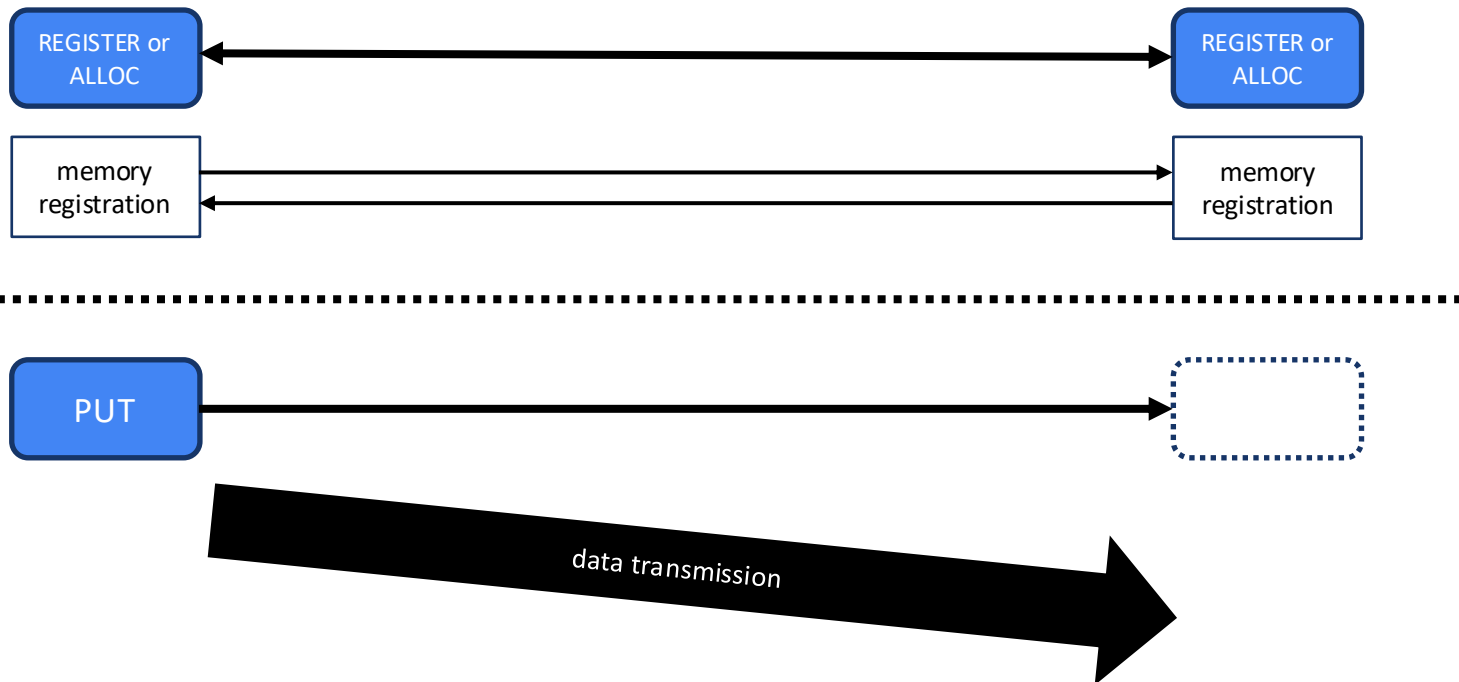
Two-sided communication



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



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.

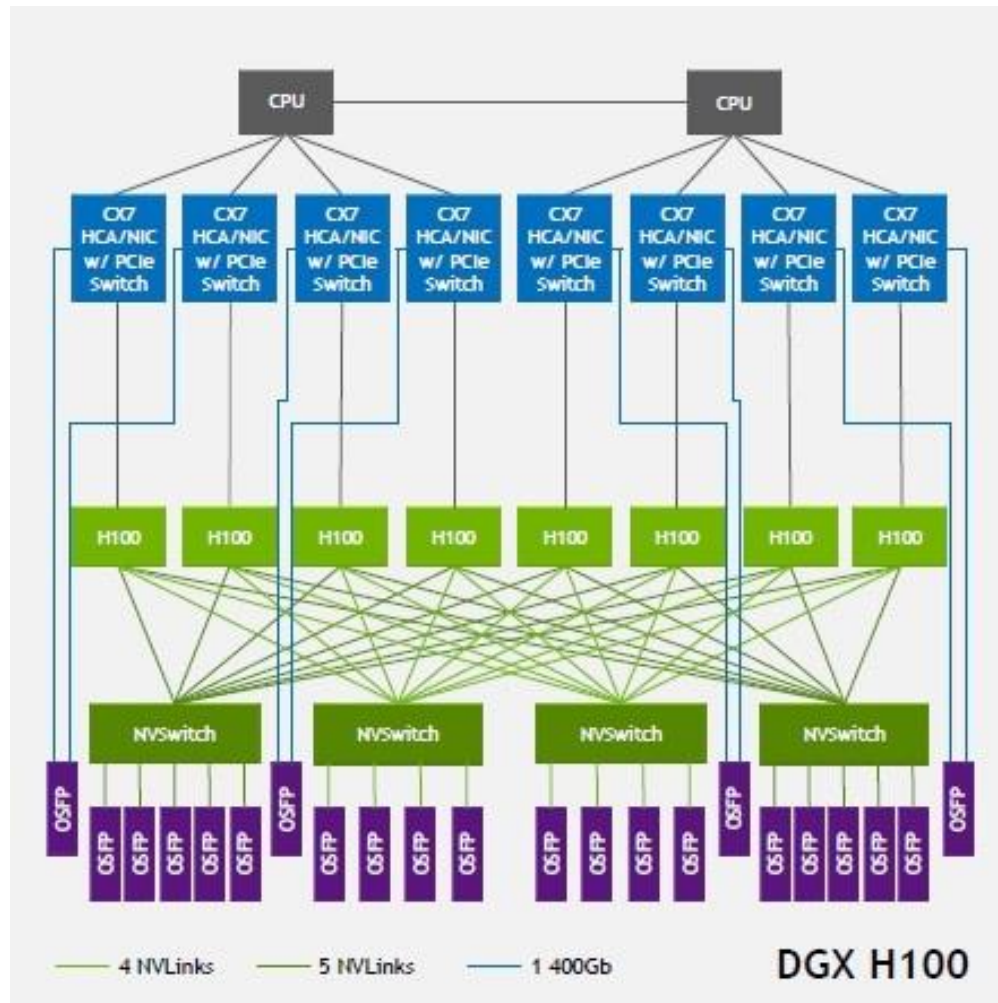


GPU Communication

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



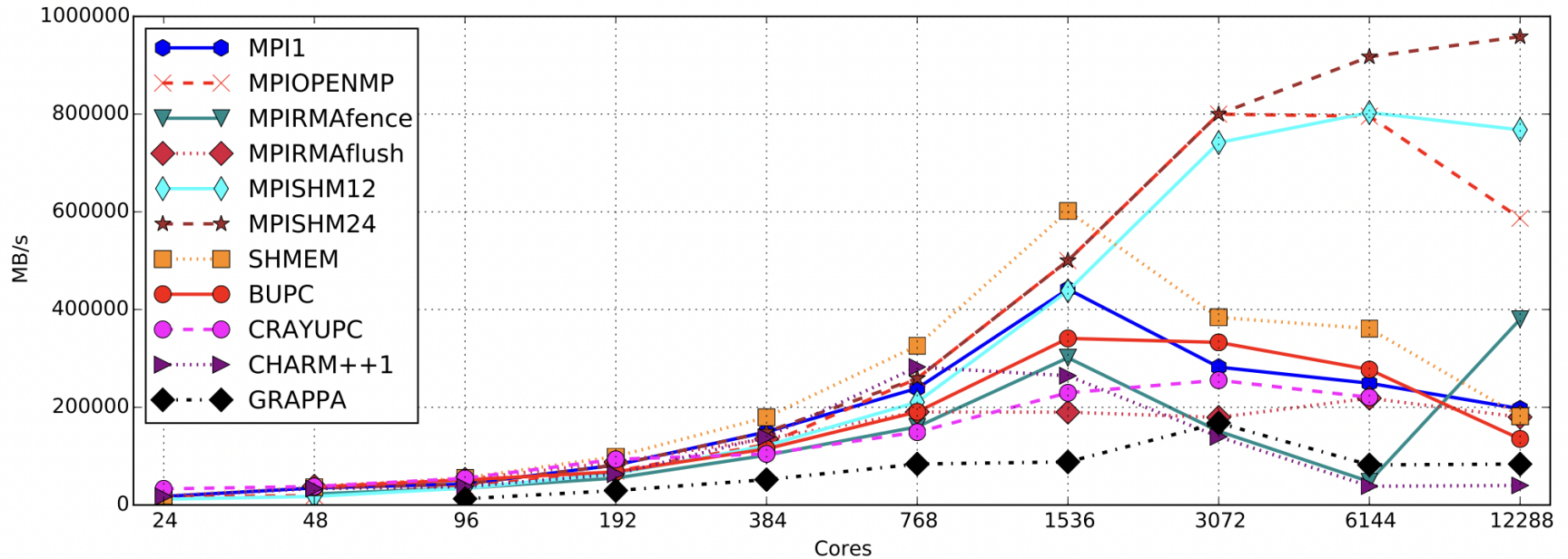
See NVLink @ <https://hc34.hotchips.org/>

Comparing runtime systems with exascale ambitions using the Parallel Research Kernels

R. F. Van der Wijngaart¹, A. Kayi¹, J. R. Hammond¹, G. Jost¹, T. St. John¹,
S. Sridharan¹, T. G. Mattson¹, J. Abercrombie², and J. Nelson²

¹ Intel Corporation, Hillsboro, Oregon, USA.

² University of Washington, Seattle, WA, USA.



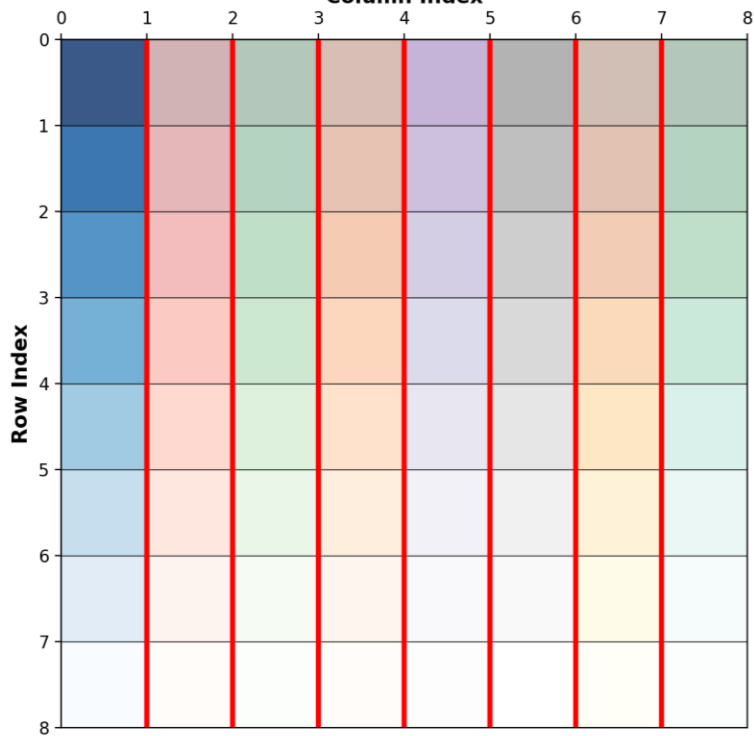
<https://link.springer.com/book/10.1007/978-3-319-41321-1>

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

<https://youtu.be/HTbjM5GDIRM>

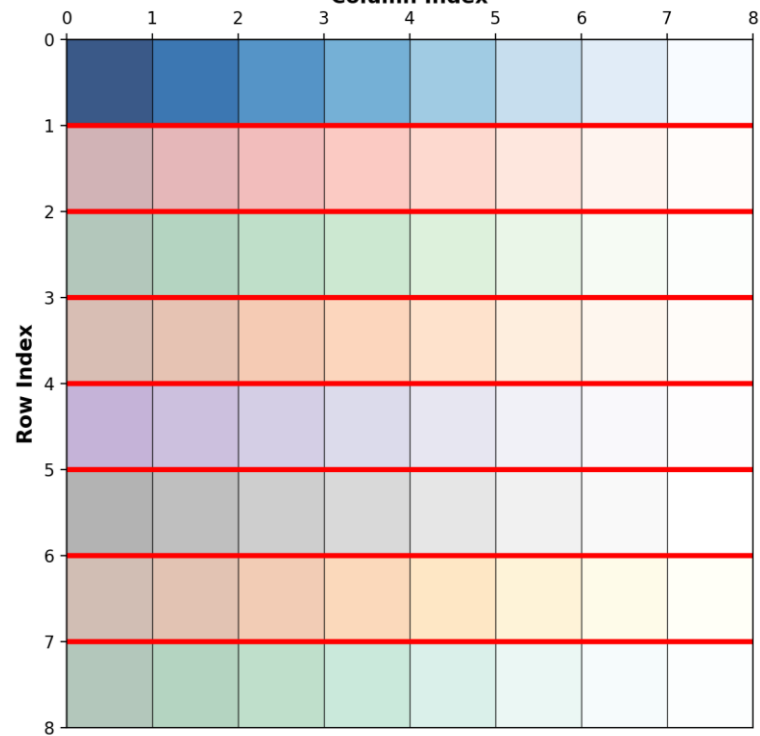
Matrix Transpose Communication - Processing Rank 0

Original Matrix A
Column Index



Column blocks are mapped onto GPUs 0..7

Transposed Matrix B
Column Index



NVSHMEM implementations of transpose

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

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

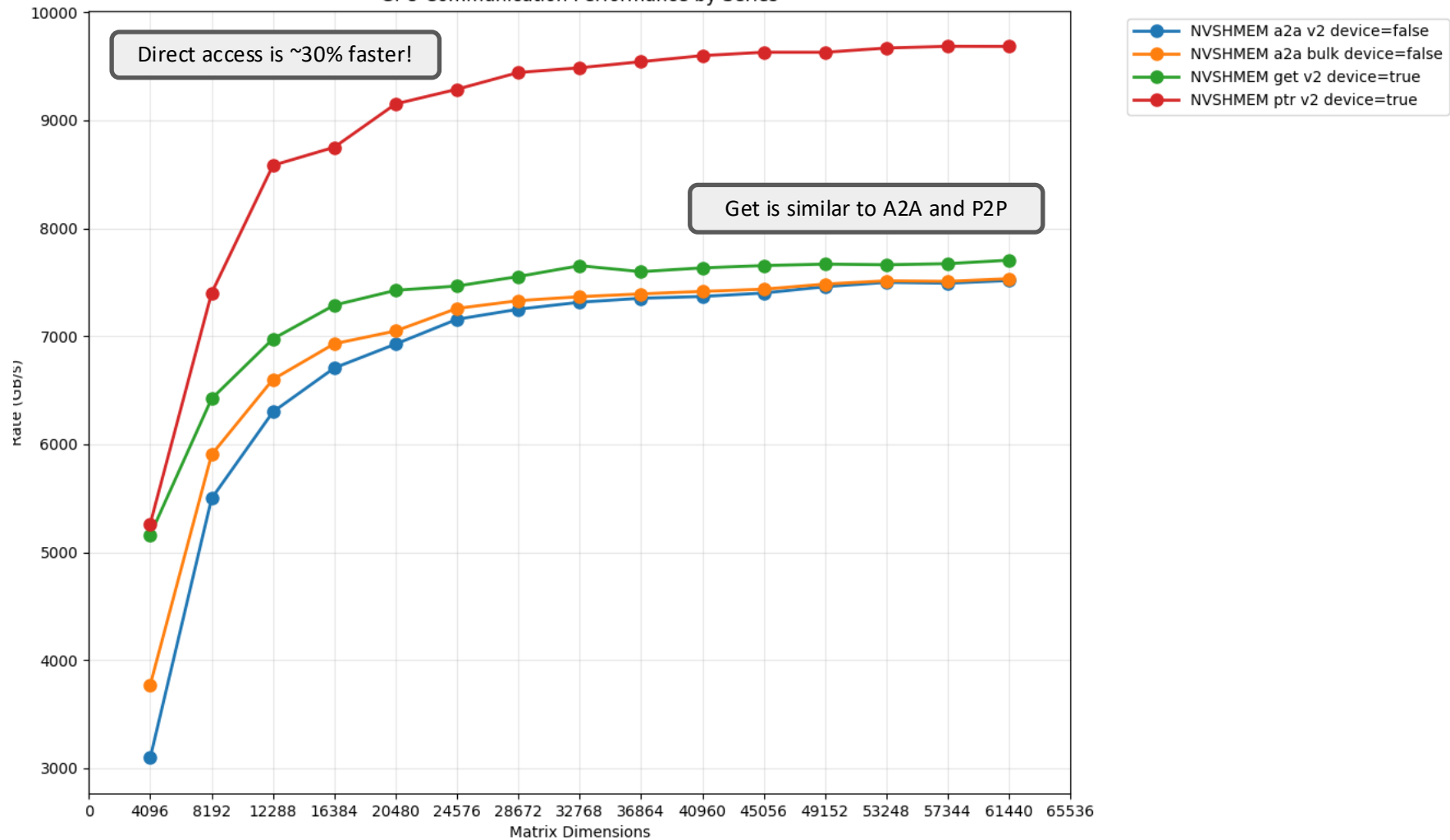
```
Barrier
Increment input matrix
```

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

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

```
Barrier
Increment input matrix
```

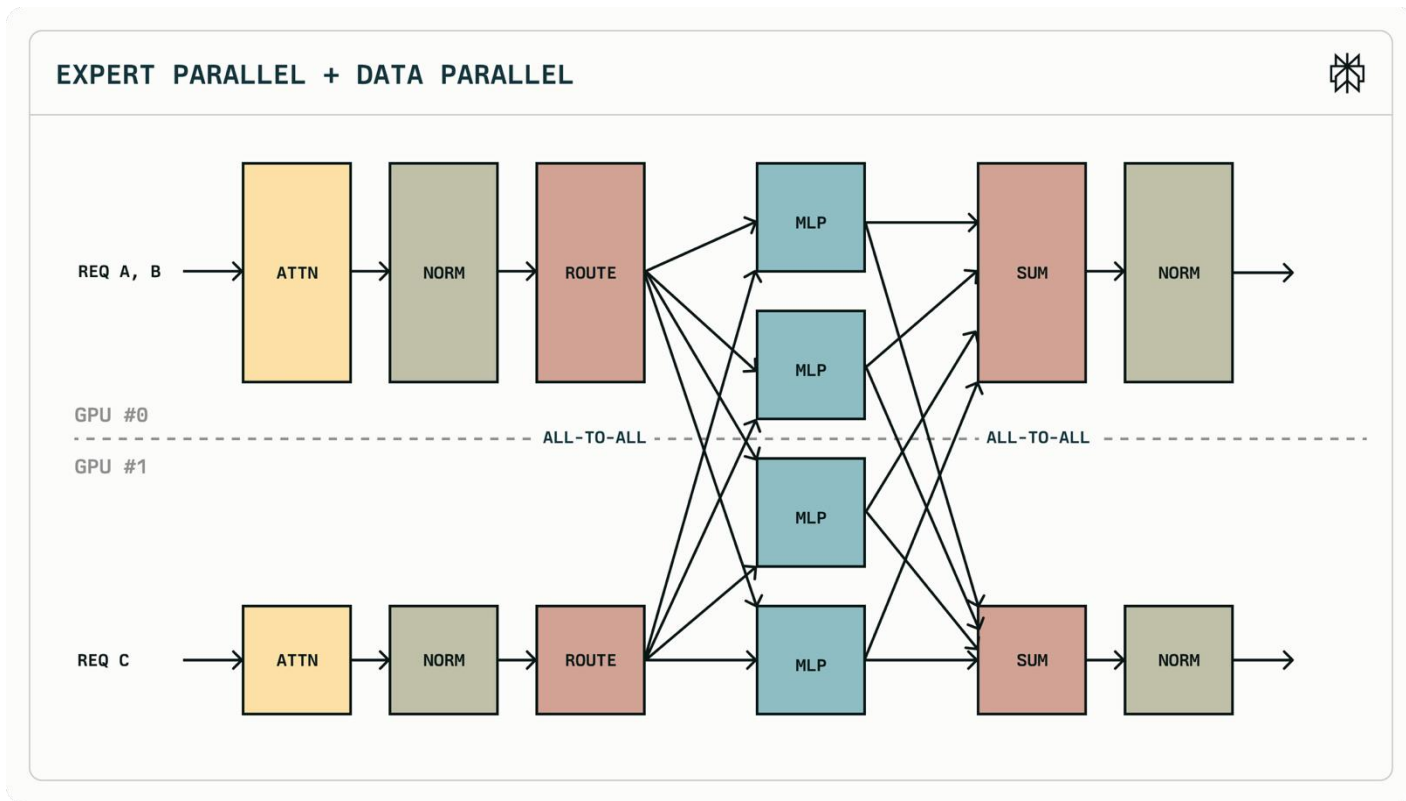
GPU Communication Performance by Series





Real-world example: AI Inference

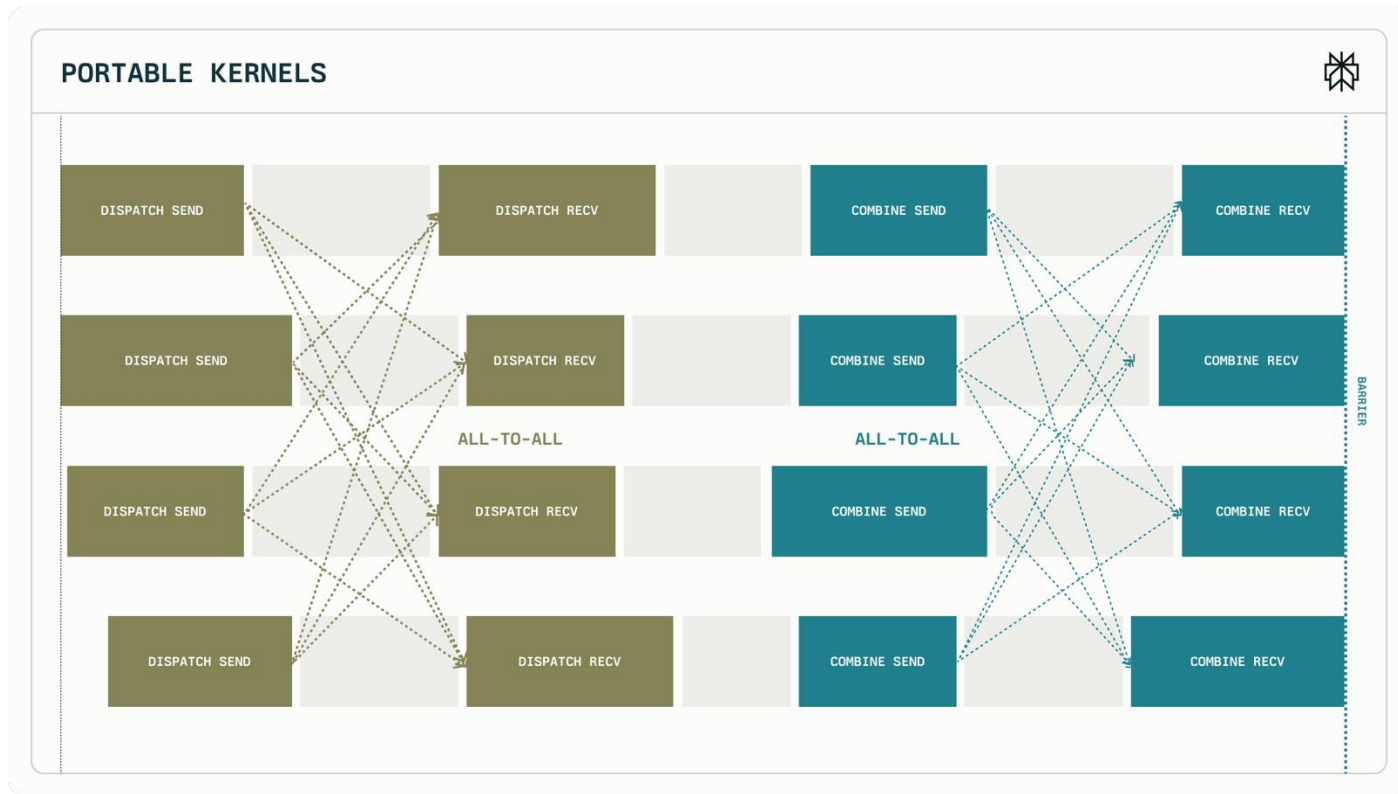
Mixture-of-Experts



“Efficient and Portable Mixture-of-Experts Communication” by Perplexity.

<https://www.perplexity.ai/hub/blog/efficient-and-portable-mixture-of-experts-communication>

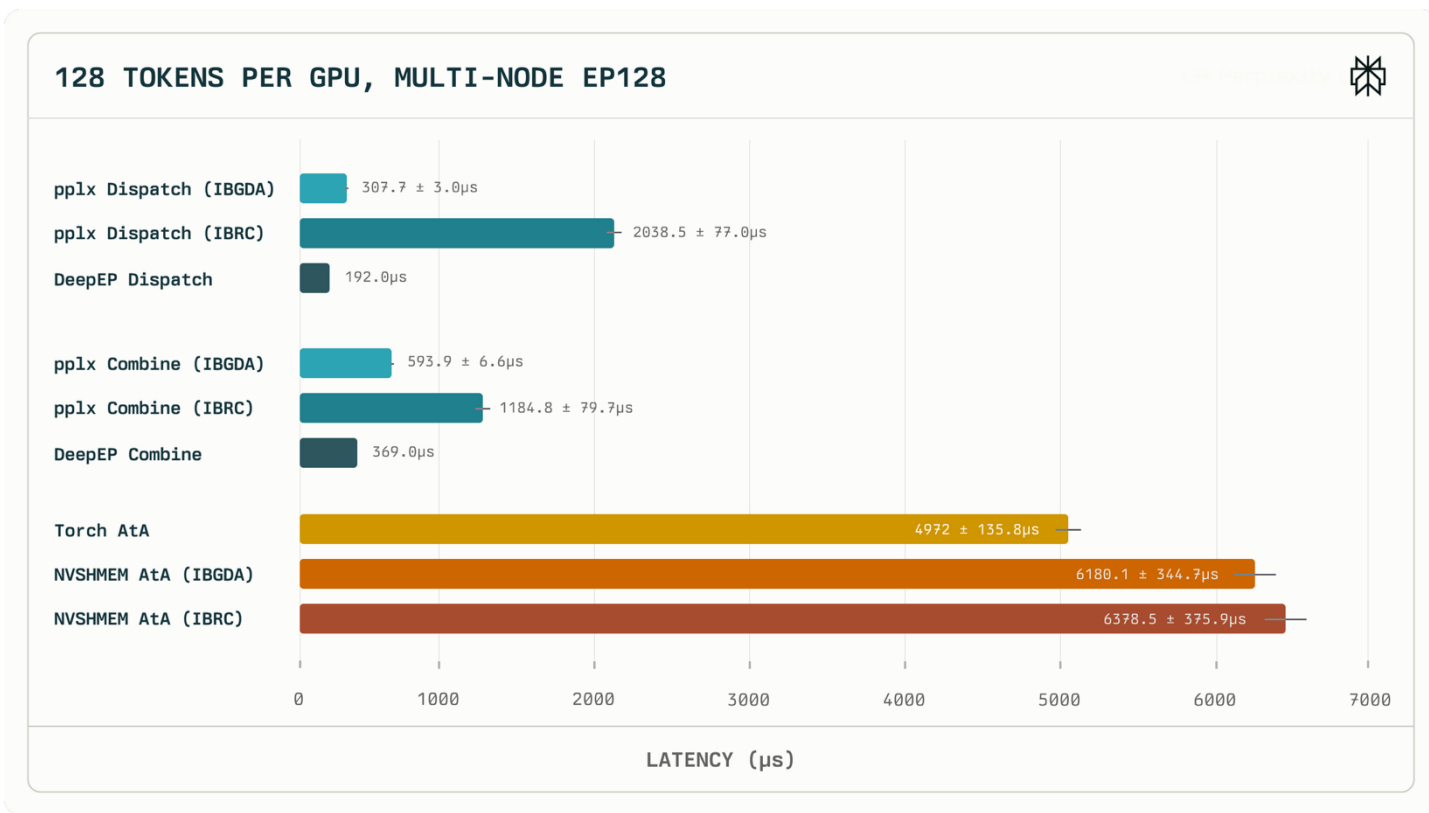
Mixture-of-Experts



“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>

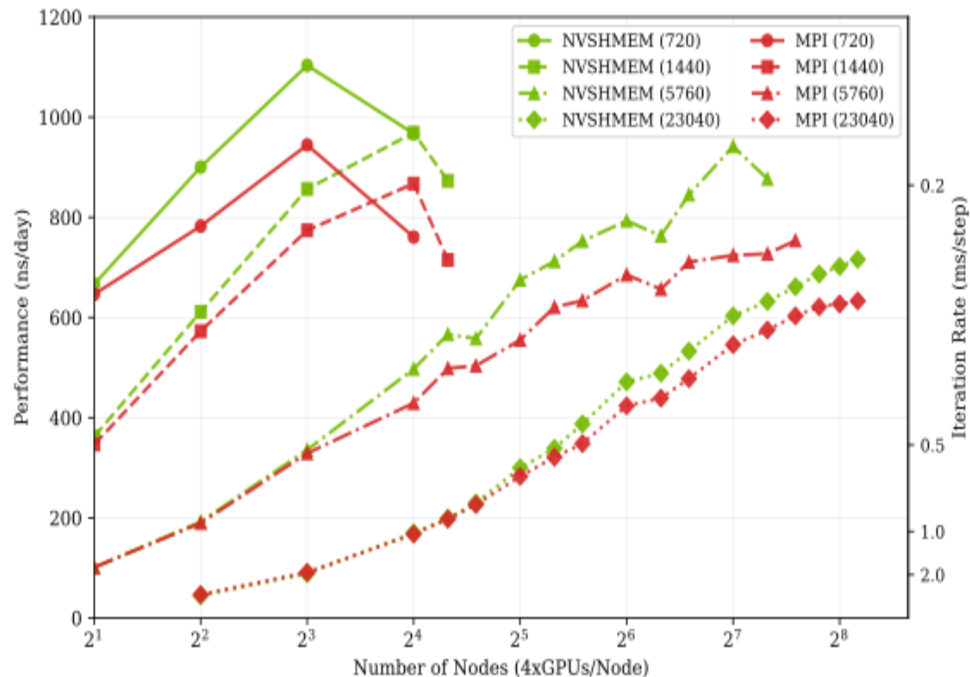


Real-world example: Molecular dynamics

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

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 with load-store connectivity to 72 GPUs (and growing).
- GPUs do >40M FP64 operations (>2B FP16) in 1 microsecond (lower bound on network/synchronization costs). Asynchrony is critical to efficiency.



AI-Driven Development in HPC Software

Tool Usage History

I am a programming Luddite who types fast...

- Until 2023: exclusively Vim, usually with syntax highlighting plugins but nothing else.
- 2024: Cursor IDE for autocomplete. Poor results with agents.
- 2025: Mixed results with agents, no major results beyond tutorial content.
- 2026: Went all-in with agents in May (good timing)...

Tools I Use

- **Claude Code (CLI)**
- **OpenAI Codex (CLI)**
- Cursor IDE and Agent (used for a quantum chemistry project)
- Perplexity (replacement for Google search)
- Claude Co-work and OpenAI ChatGPT for local projects



VibeMPI

It's an MPI implementation that sounds like it's vibe-coded, but isn't...

VibeMPI Motivation

- Can agentic development be an effective tool for communication software **in general**?
- MPI has a good specification, although it was not intended to be machine-readable (WIP) or machine-comprehensible.
- MPI is widely used, has multiple implementations already, and hundreds of tests.

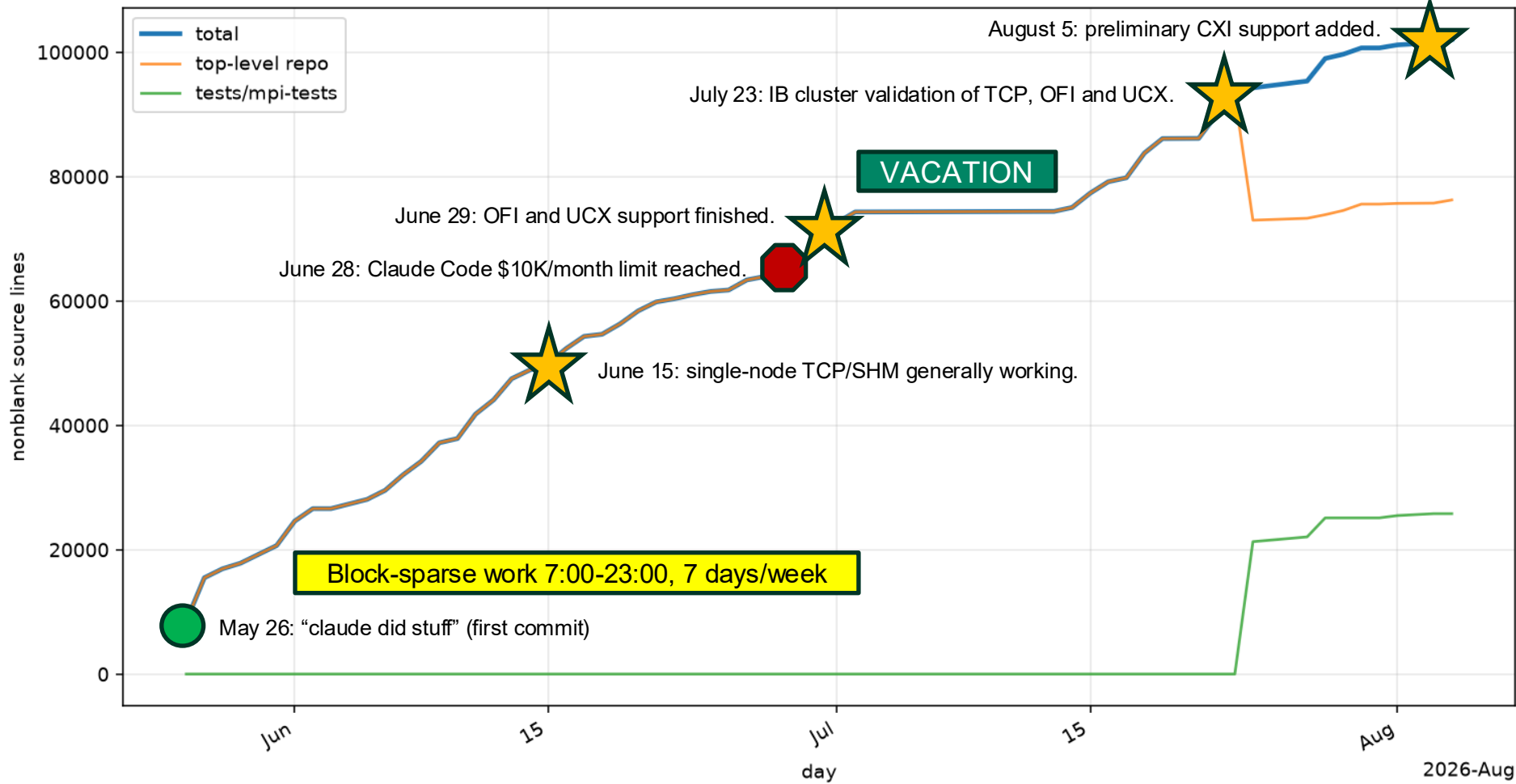
Testing, not the standard, is where the magic happens.

“What I cannot create, I do not understand.”
- Richard Feynman

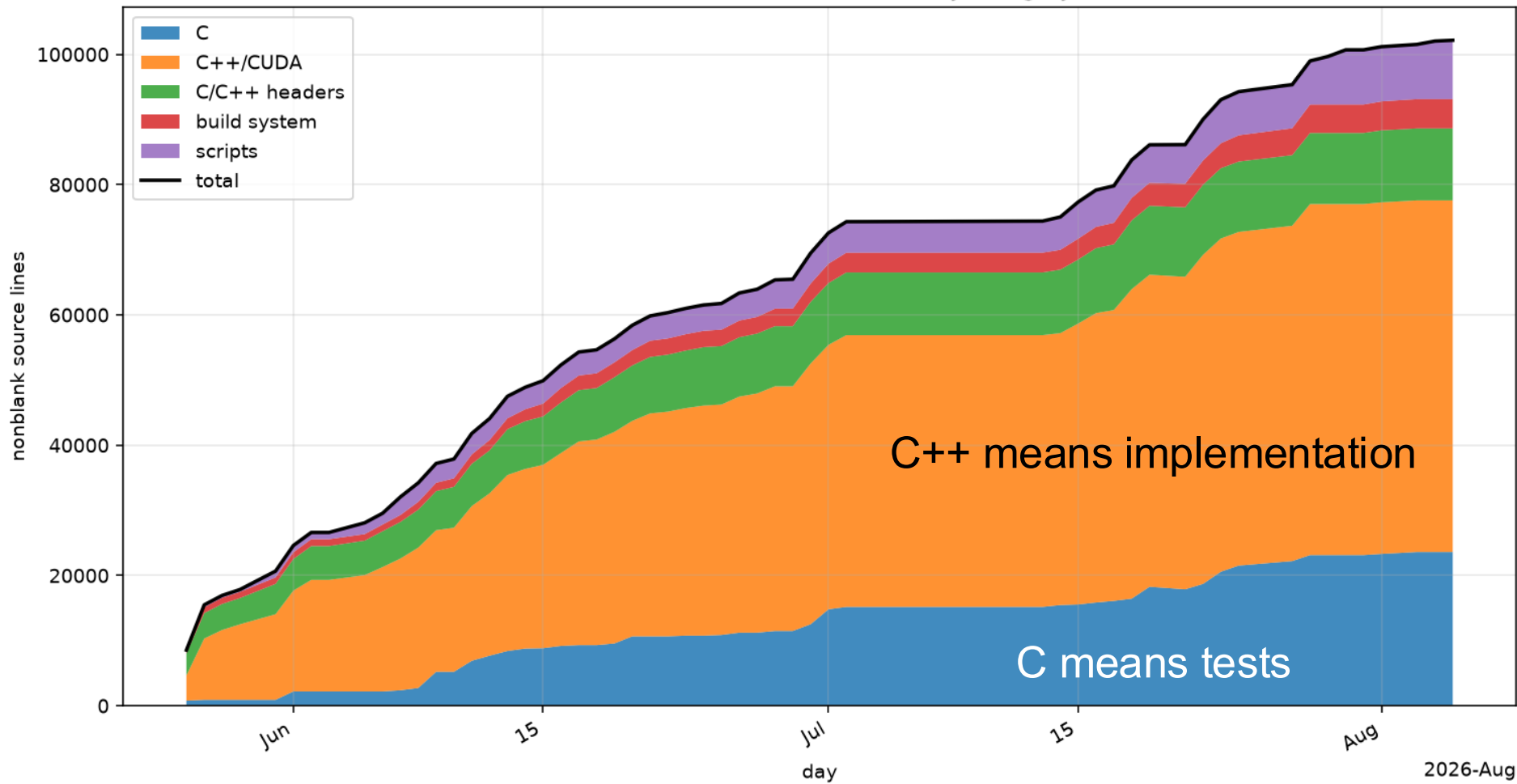
VibeMPI Design

- **Little to no actual vibe-coding.**
Apply everything the MPI community has taught me over the years, etc.
- No legacy debt. Use modern C++ for the implementation.
- Design for MPI-5 ABI, MPI_THREAD_MULTIPLE, large-count, fast+correct RMA, etc.
- *Use CMake, even though I hate it, because configure+build+test is on the critical path of agentic development and CMake+Ninja moves much faster than Autotools+GNU Make.*
- Not focused on HPC networks initially. Added later after an extensive redesign.

VibeMPI lines of code over time



VibeMPI lines of code over time by category



VibeMPI Testing

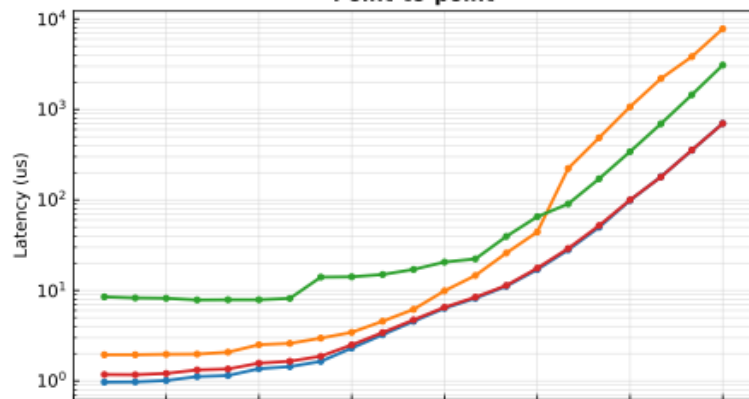
- Agents drive all the testing: Slurm submission, parsing, result aggregation, report writing.
- The standard sweep is ~2000 tests, with a handful of XFAIL/WONTFIX that look for implementation-specific behavior.
- AI ran GCOV analysis and test coverage is ~99% in most places. This required dozens of new tests. In total, I added hundreds of new tests.
- The VibeMPI test suite will be released as a separate project for others to use.

“What is not tested is wrong.”

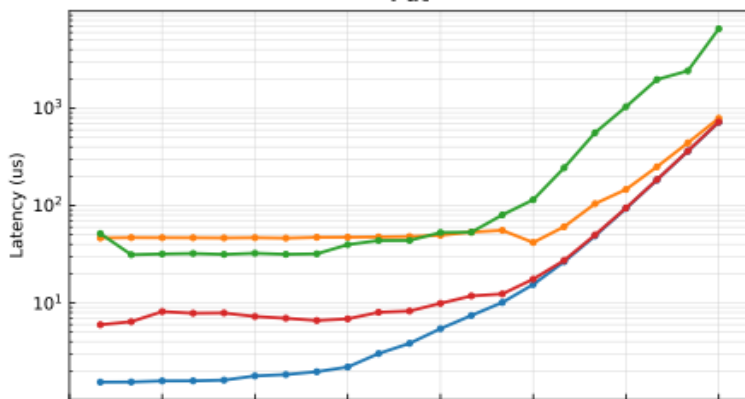
MPI inter-node latency on Iris

— HPC-X 4.1.9 / UCX-IB — VibeMPI / TCP-IPoIB
— VibeMPI / OFI verbs-RxM — VibeMPI / UCX-IB

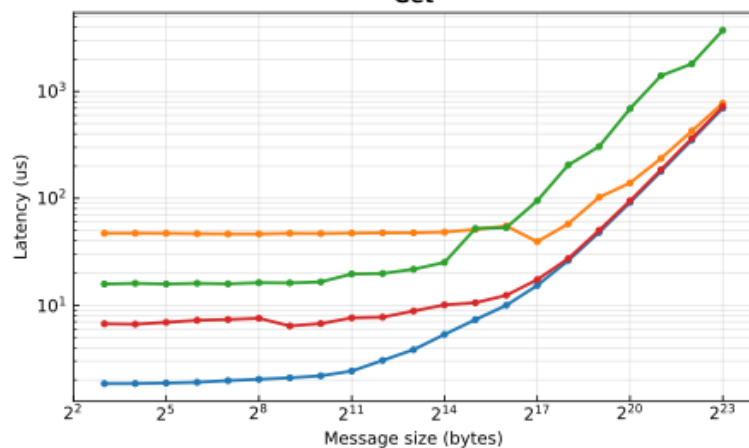
Point-to-point



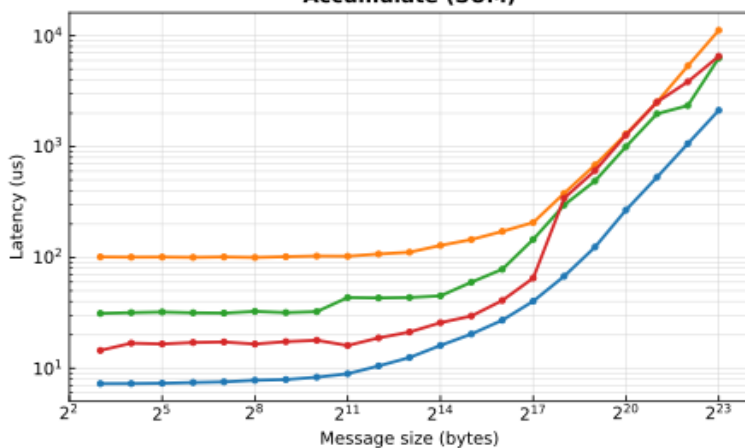
Put



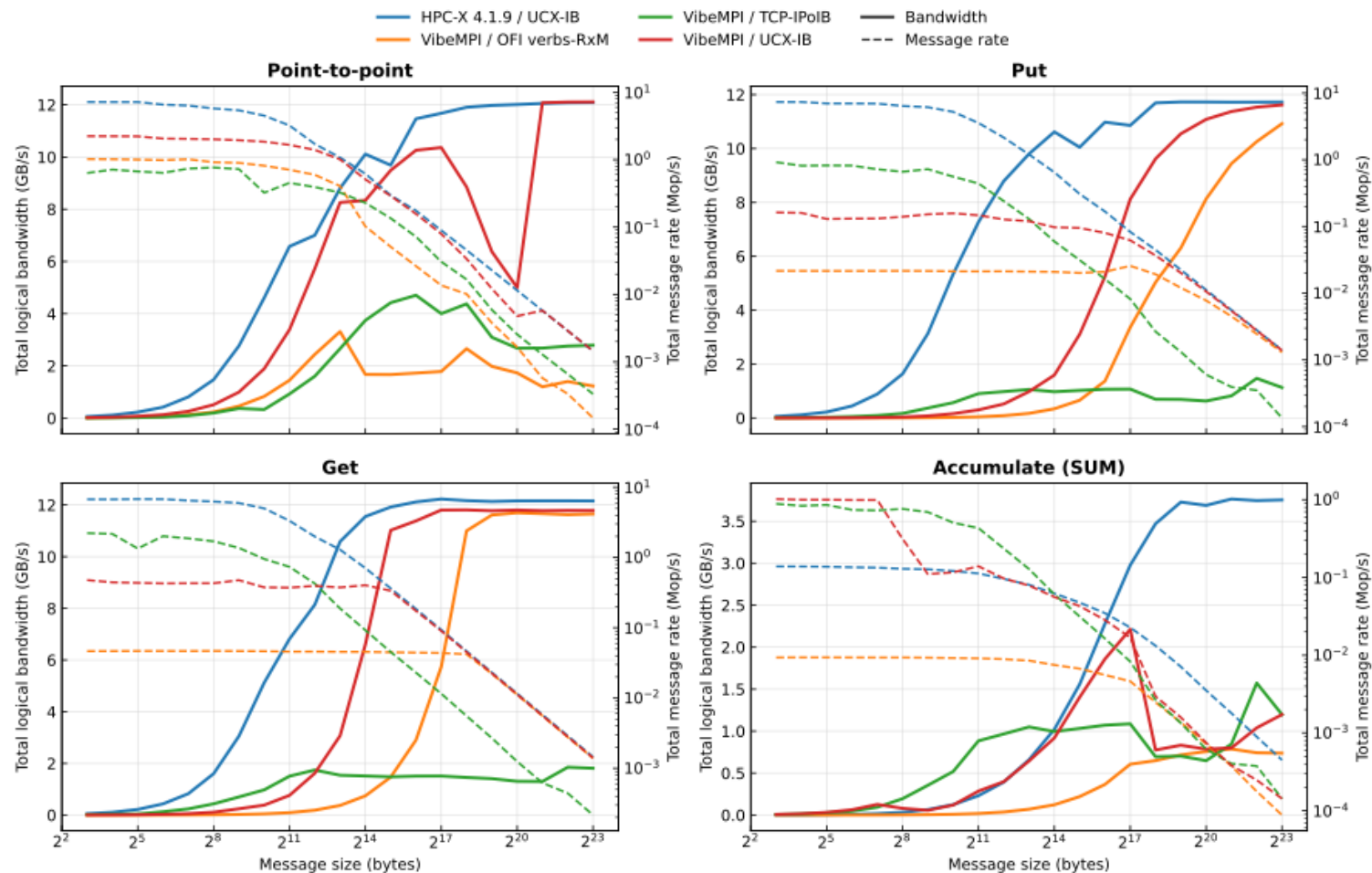
Get



Accumulate (SUM)

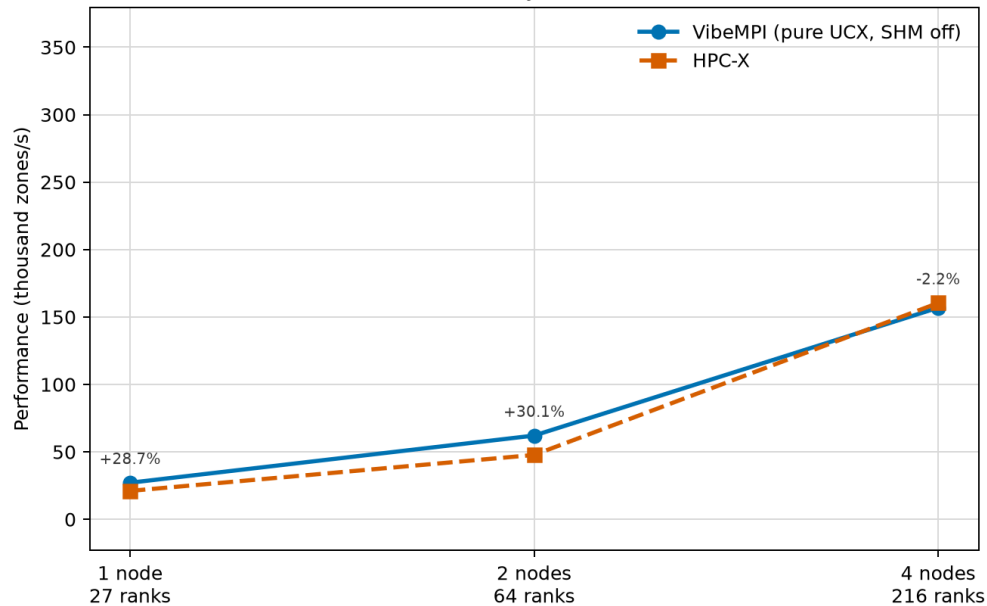


MPI inter-node throughput on Iris



LULESH node/process scaling

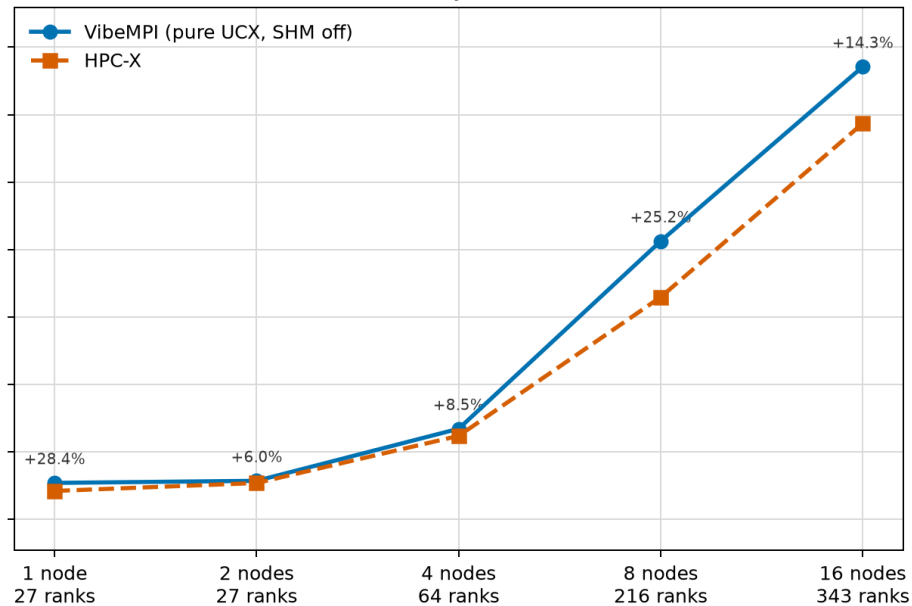
Iris — Full-density allocation, s=40



Allocated nodes and total MPI processes

Problem size is fixed per rank, so total work grows with the annotated cubic process count. Labels show VibeMPI relative to HPC-X.

Iris — Half-density allocation, s=40



Allocated nodes and total MPI processes



NCCL Libraries

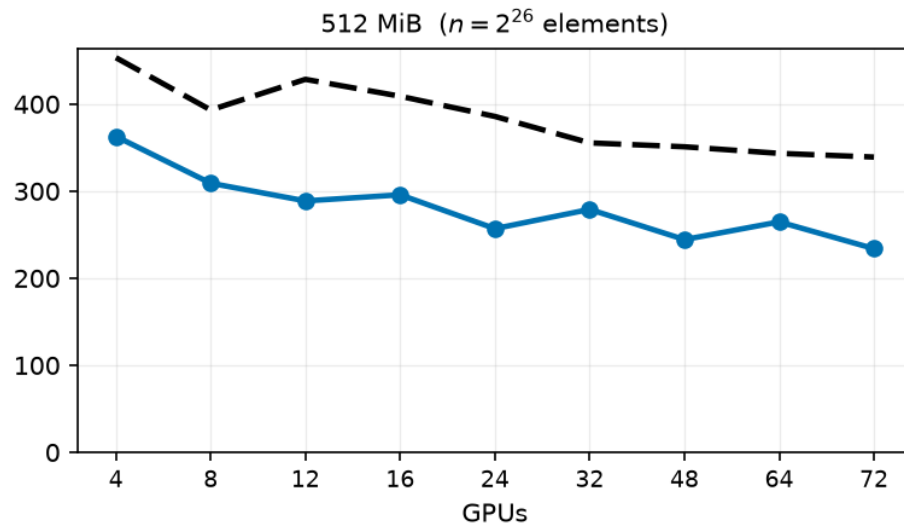
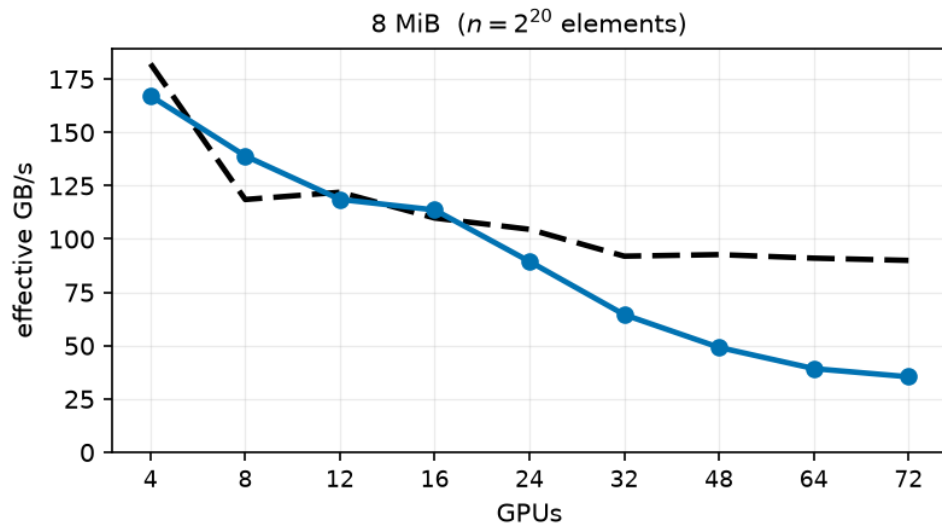
NCCL Development

- Tell the agent to read all the NCCL docs and source code at the beginning of the project; once you have one good project, tell AI to learn from it in the next one.
- Wait for the agent to produce correct code that performs terribly, then tell it to use more than one SM.
- Agents can attach NCU and do substantial kernel optimization, including register/occupancy optimizations.
- AIDD works for NCCL device-initiated communication for both NVLINK and networking (GIN), despite very few published examples. It took 1-2 days to figure it out.

NCCL NVL Reductions

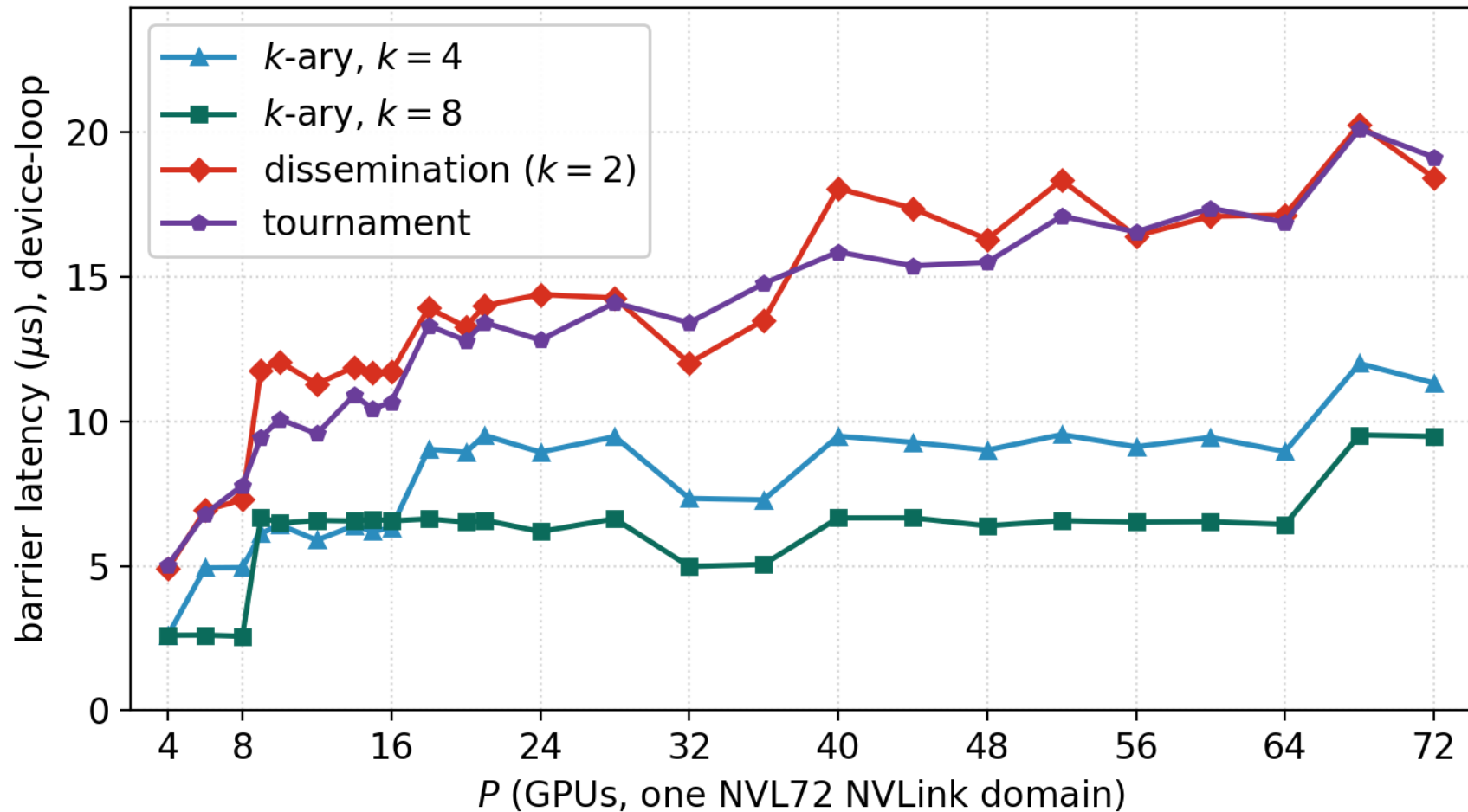
FP64 allreduce on GB200 NVL72 — effective GB/s, higher is better

— NCCL built-in — this library



Disclaimer: This is not a marketing number.

NVL72 barrier: k -ary, dissemination, tournament





Summary

Outlook for Agentic Development

- AI-driven development has progressed rapidly, with massive increase in utility around Q2 2026.
- AI is fantastic at menial tasks like Git (“rebase and fix conflicts”) and CMake. It can drive GDB, Valgrind and GCOV workflows completely autonomously. It understands Slurm.
- AI can script almost anything and generate diagrams and plots with matplotlib.
- AI will follow your instructions, even when they are bad.
- AI will be pathologically stupid mistakes, such as allocating unlimited buffering in MPI and generating obvious deadlocks.
- I rate AI at 100-1000x productivity for some tasks, and never less than 10x.

To build a building, you need an architect, an engineer, a construction manager, and a lot of carpenters.

An architect can't build a building without carpenters, but if you have an thousand carpenters and no architect, you won't get a building either.

AI is the thousand carpenters. You are the architect the engineer and the construction manager.



The end

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.

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 subsets.
- 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>