CommBench: Micro-benchmarking hierarchical networks multi-GPU, multi-NIC nodes

Mert Hidayetoglu[1], Simon Garcia de Gonzalo[2], Elliott Slaughter[3], Yu Li[4], Christopher Zimmer[5], Tekin Bicer[6], Bin Ren[7], William Gropp[4], Wen-mei Hwu[8], Alex Aiken[1]

[1]Stanford University
[2]Sandia National Laboratories
[3]SLAC National Accelerator Laboratory
[4]University of Illinois at Urbana-Champaign
[5]Oak Ridge National Laboratory
[6]Argonne National Laboratory
[7]College of William & Mary
[8]Nvidia Research

Corresponding authors: merth@stanford.edu, simgarc@sandia.gov

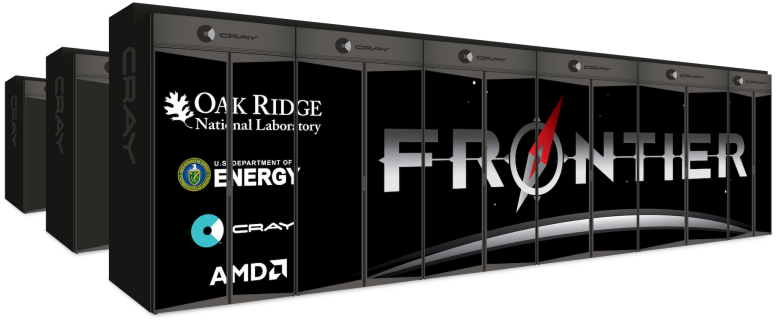
Summit (2018) 6 GPUs per Node

200 PFLOPS 4,608 Nodes

Aurora (2023) 12 GPUs per Node

2 EFLOPS 10,624 Nodes

Frontier (2022) 8 GPUs per Node

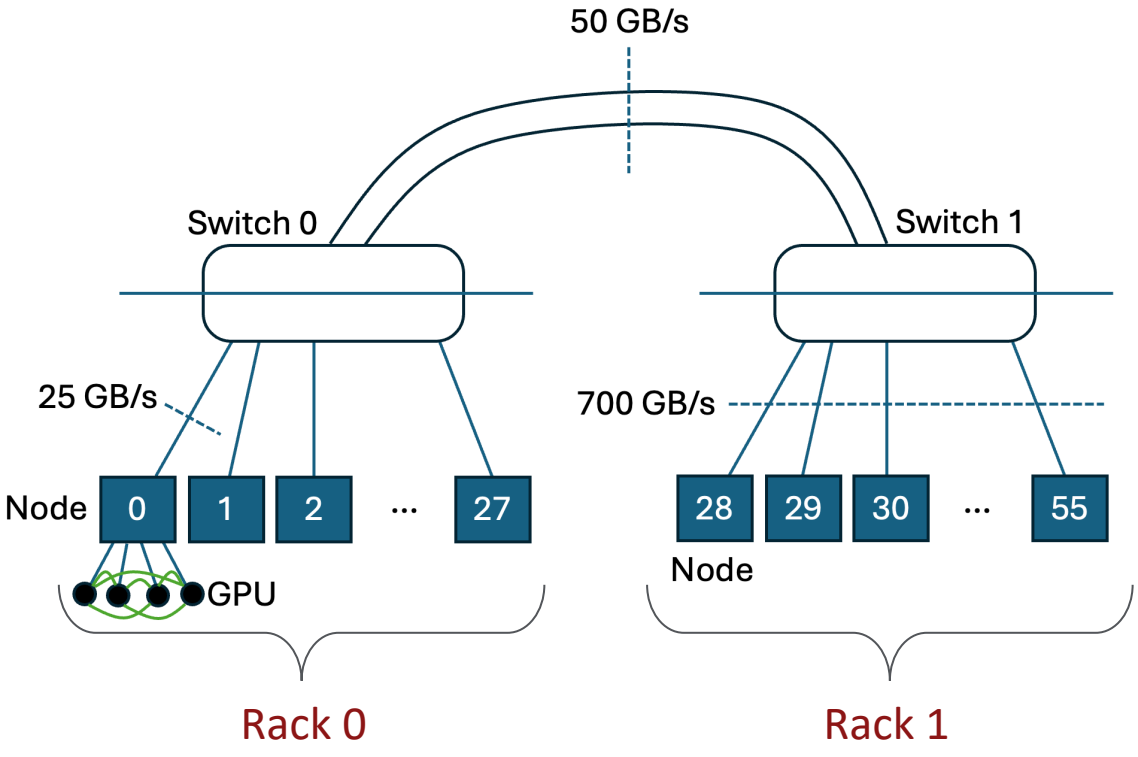


1.7 EFLOPS 9,472 Nodes

El Capitan (2024) 8 GPUs per Node

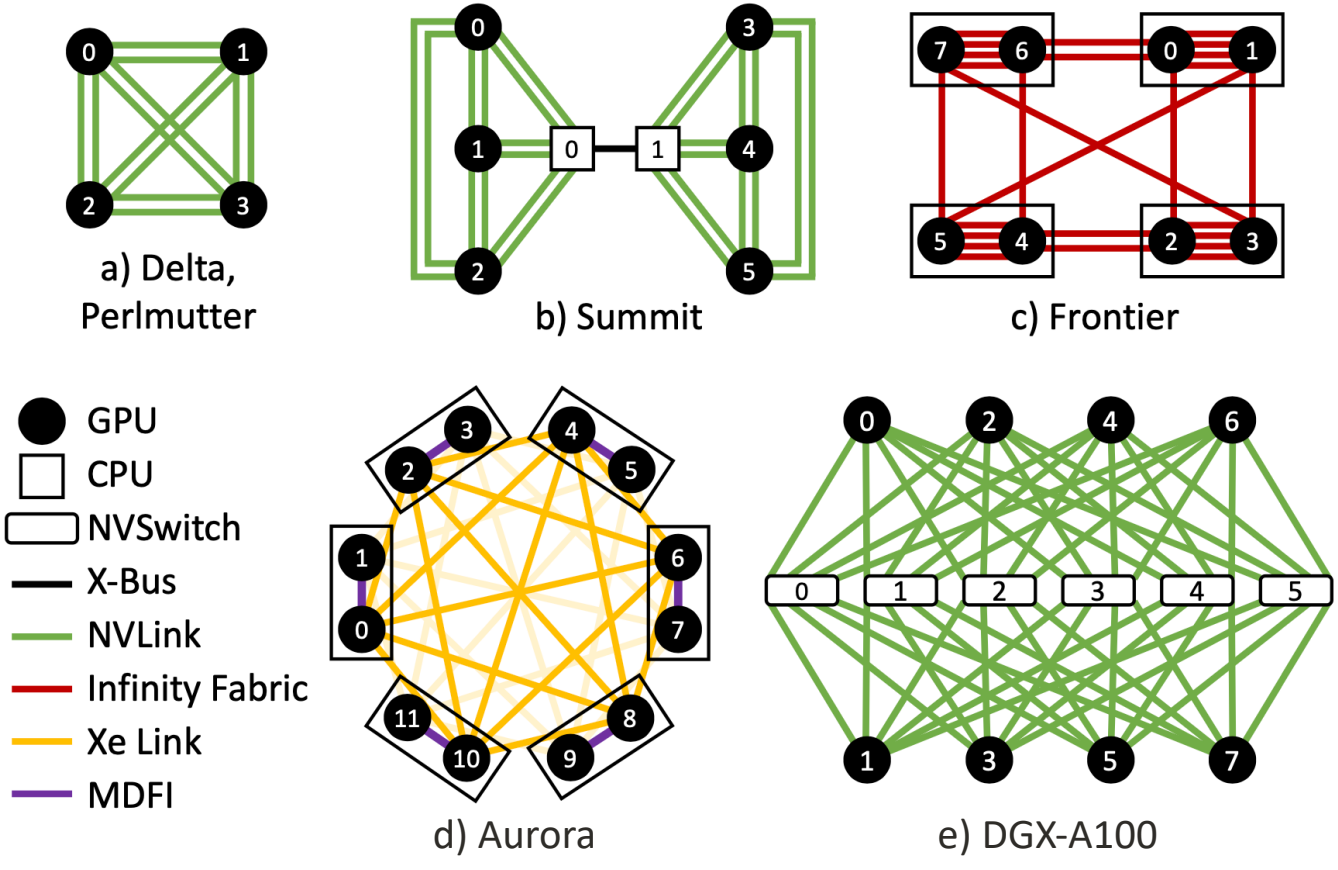
2 EFLOPS

GPU networks are hierarchical.

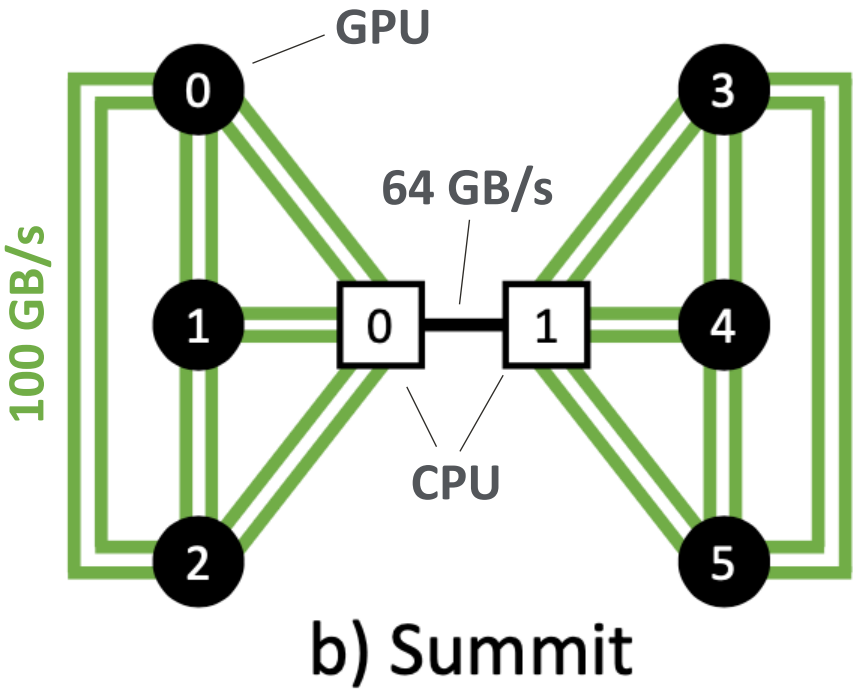


- Multiple levels across nodes, racks, and so on.
- Each level has fixed-size groups.
- Communication across groups are “costlier”.

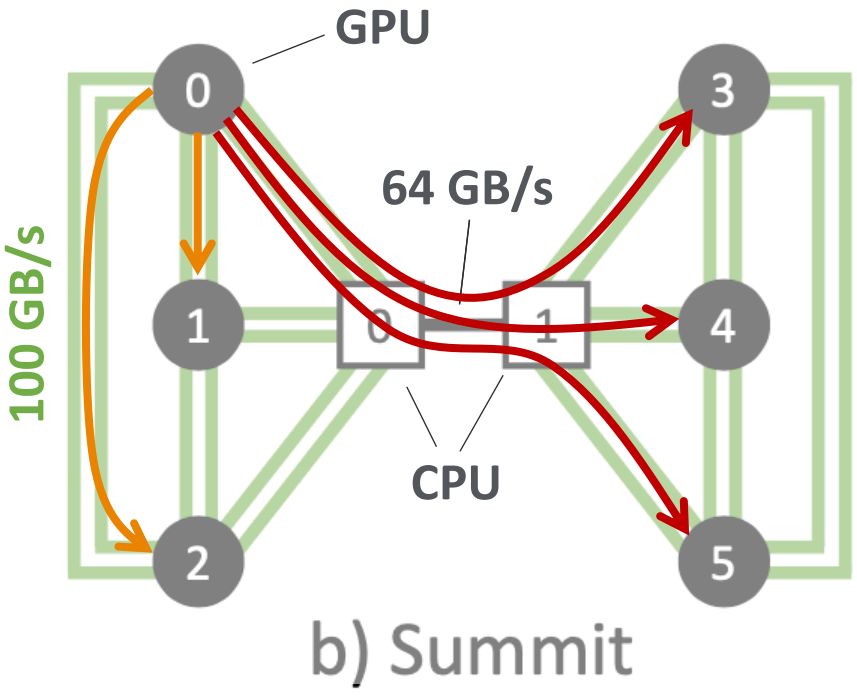
Intra-node hierarchies.



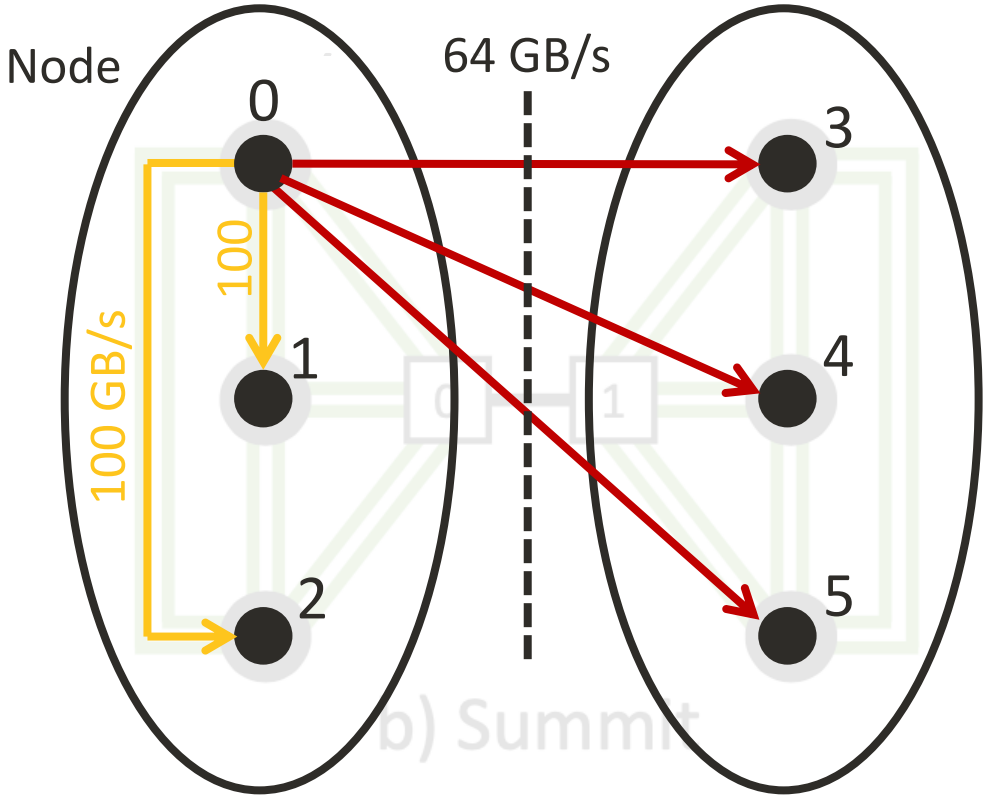
GPU networks are hierarchical.



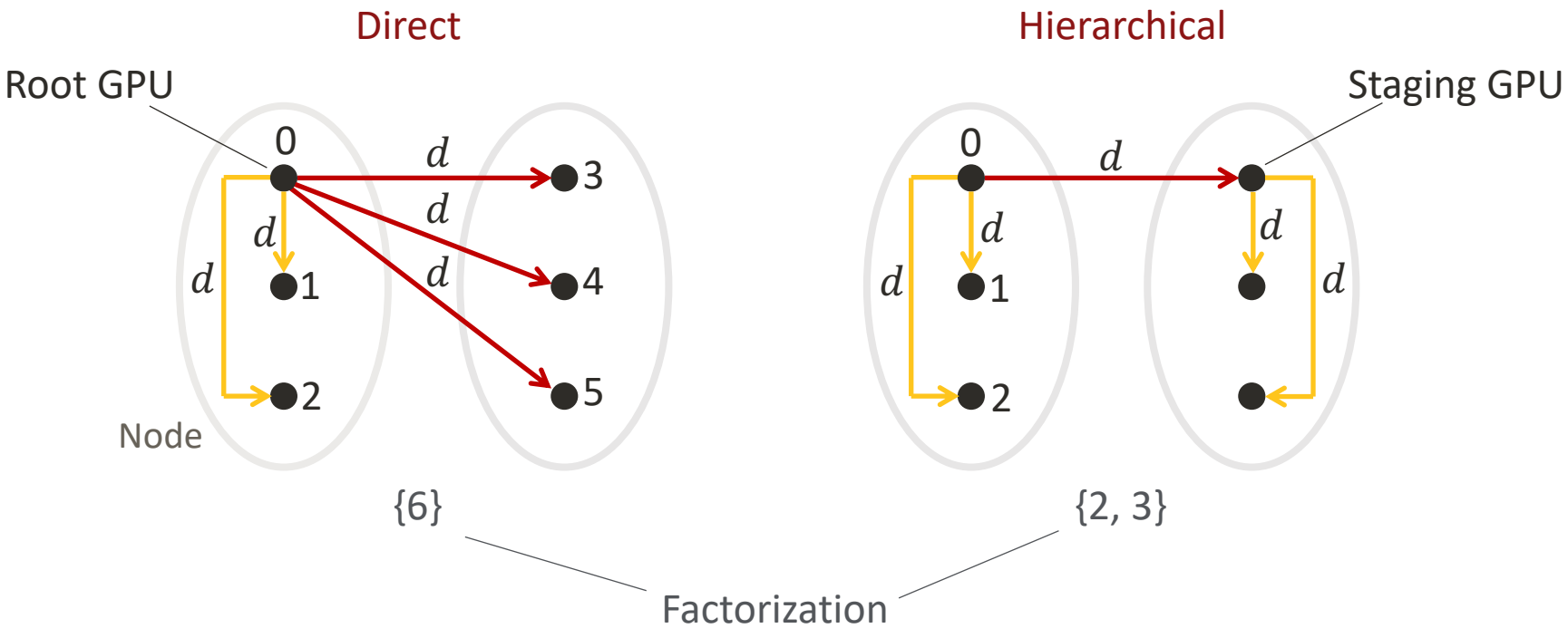
GPU networks are hierarchical.



Two-Level Hierarchy



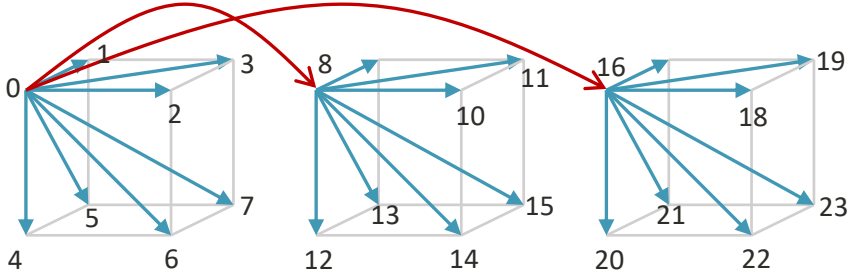
Hierarchical Broadcast



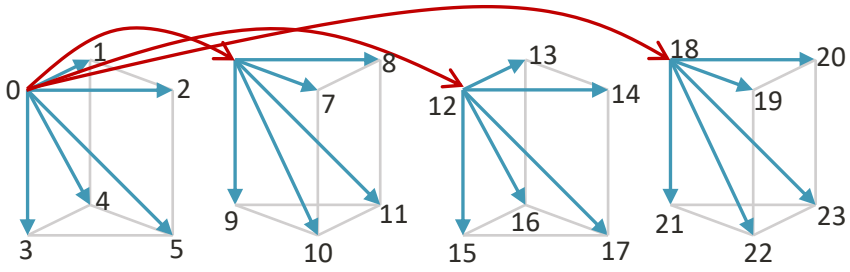
Tree Factorization

24 GPUs

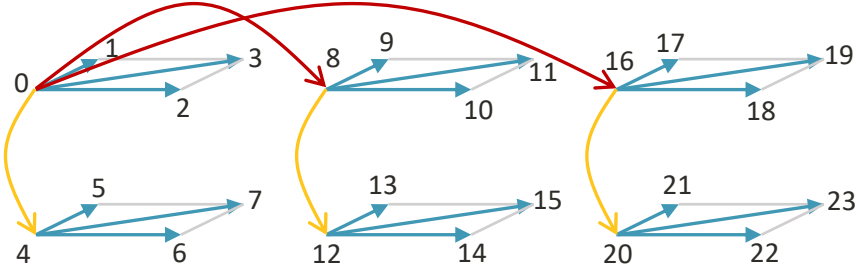
(a) {3, 8}



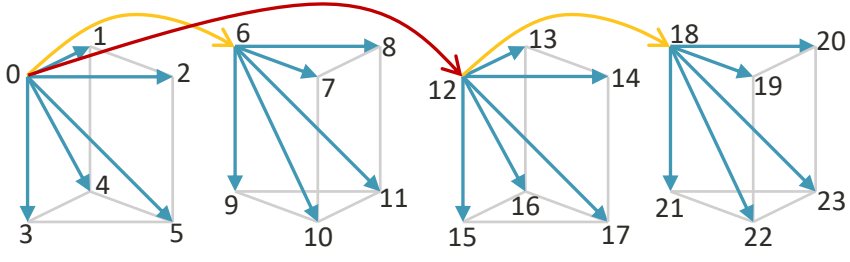
(b) {4, 6}



(c) {3, 2, 4}

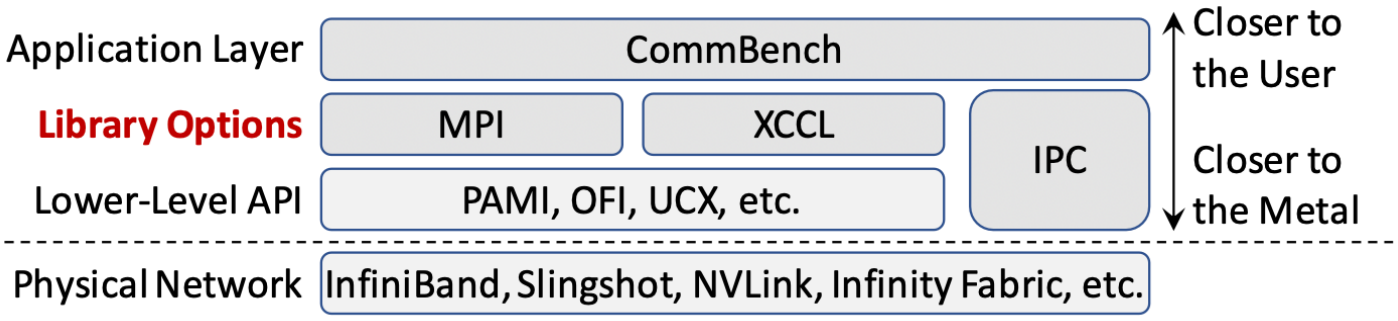


(d) {2, 2, 6}

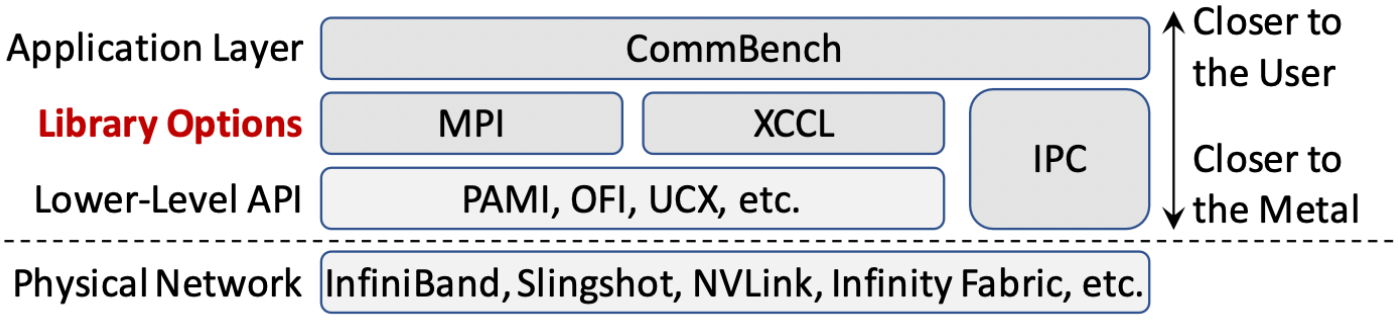


CommBench:
A micro-benchmarking tool

Software Stack

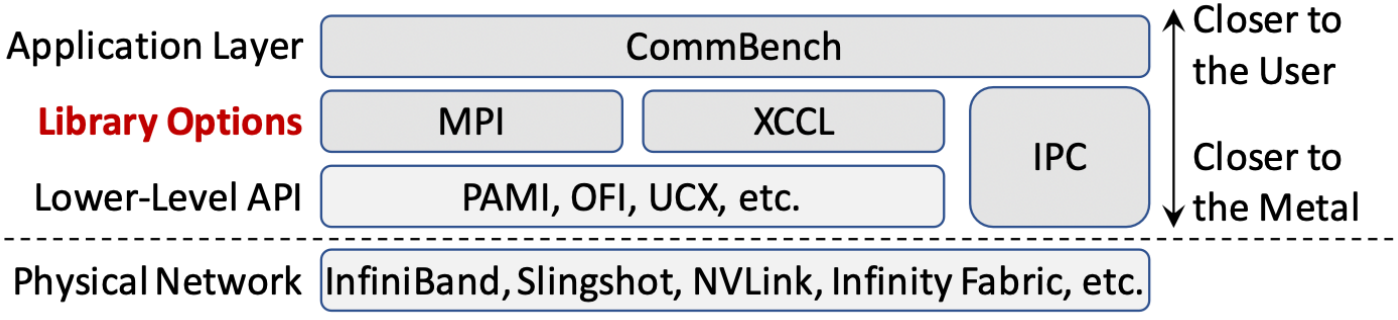


Software Stack




```
Comm<T>::Comm(Library); // choose implementation library

void Comm<T>::add(T *sendbuf, T *recvbuf, size_t count, int src, int dst); // register P2P

void Comm<T>::start(); // non-blocking
void Comm<T>::wait(); // blocking

void Comm<T>::measure(int warmup, int numiter);
```

Software Stack



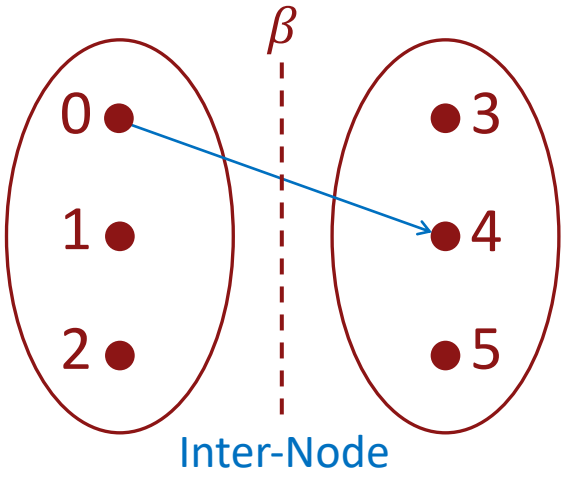
Ports & Libraries

	OneAPI	CUDA	HIP	(default)
MPI	GPU MPI	GPU MPI	GPU MPI	CPU MPI
XCCL	OneCCL[3]	NCCL	RCCL	n/a
IPC	ZE IPC	CUDA IPC	HIP IPC	n/a

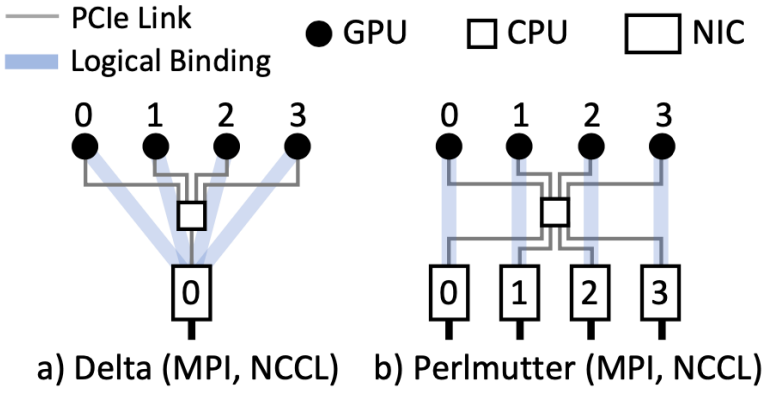
CommBench API

```
#define COMMBENCH_PORT_CUDA
#include “commbench.h”

#define Type int

using namespace CommBench;

int main() {

  Comm<Type> comm(NCCL); // initialize communicator

  Type* sendbuf, recvbuf;
  size_t count = 1e9 / sizeof(Type);
  allocate(sendbuf, count);
  allocate(recvbuf, count);

  comm.add(sendbuf, recvbuf, count, 0, 4); // register P2P

  comm.measure(5, 10);
}
```

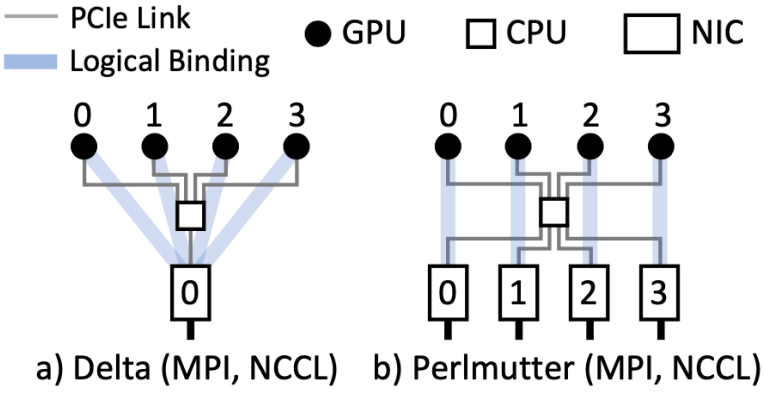
Point-to-Point Comm.



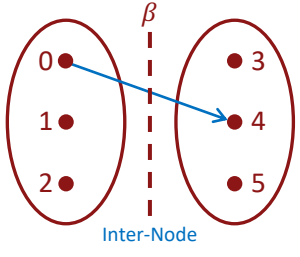
Optimization: Multi-Rail Striping



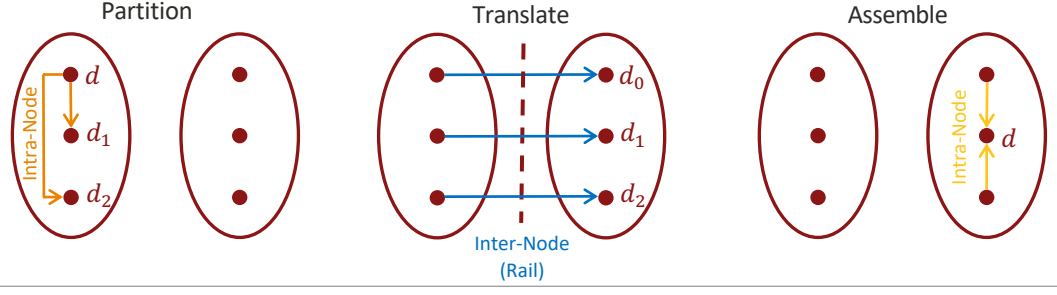
Optimization: Multi-Rail Striping



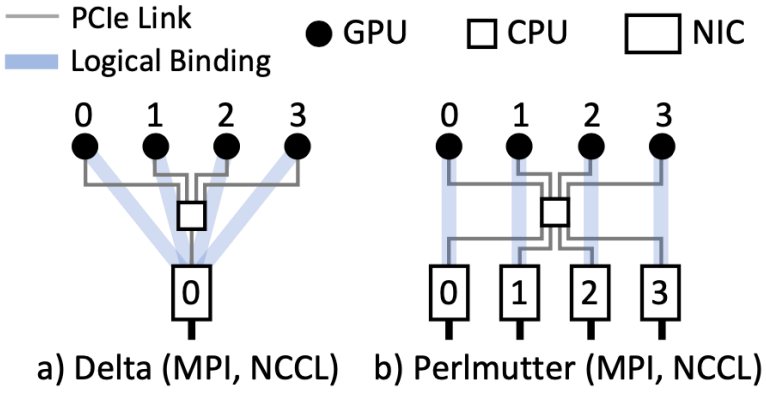
Point-to-Point Comm.



Striping Algorithm

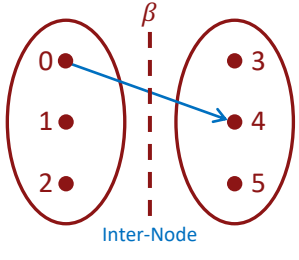


Optimization: Multi-Rail Striping

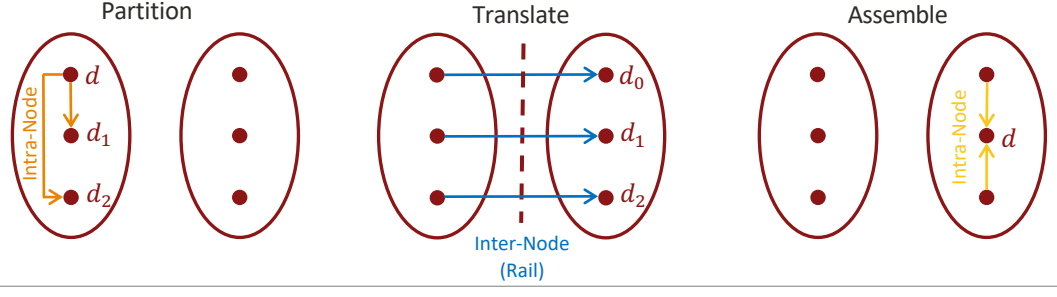



```
using namespace CommBench;

Comm<Type> partition(Library::IPC);
Comm<Type> translate(Library::NCCL);
Comm<Type> assemble(Library::IPC);

allocate(send_temp, count);
allocate(recv_temp, count);

partition.add(sendbuf, count, send_temp, 0, count, 0, 1);
partition.add(sendbuf, 2 * count, send_temp, 0, count, 0, 2);
translate.add(sendbuf, 0, recv_temp, 0, count, 0, 3);
translate.add(send_temp, 0, recvbuf, count, count, 1, 4);
translate.add(send_temp, 0, recv_temp, 0, count, 2, 5);
assemble.add(recv_temp, 0, recvbuf, 0, count, 3, 4);
assemble.add(recv_temp, 0, recvbuf, 2 * count, count, 5, 4);

partition.start();
partition.wait();
translate.start();
translate.wait();
assemble.start();
assemble.wait();

std::vector<Comm<Type>> stripe = {partition, translate, assemble};
measure_async(5, 10, count * 3);
```

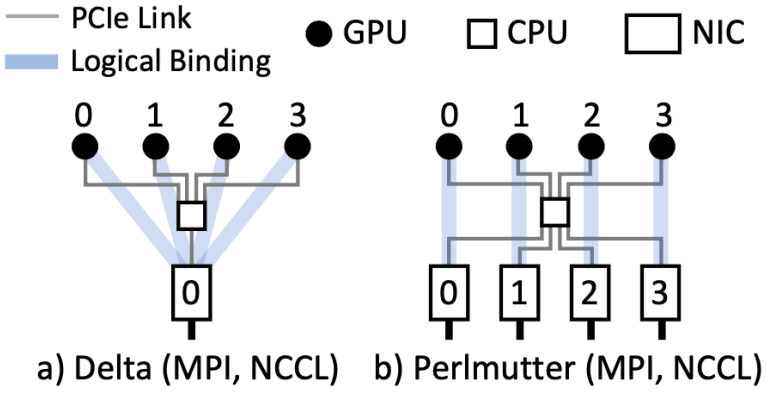
Point-to-Point Comm.



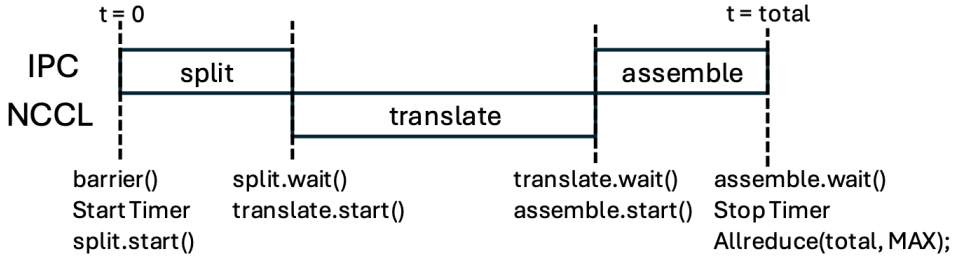
Striping Algorithm



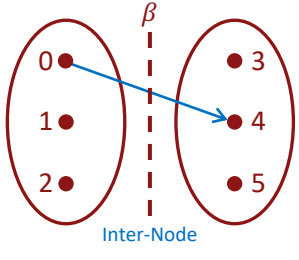
Optimization: Multi-Rail Striping



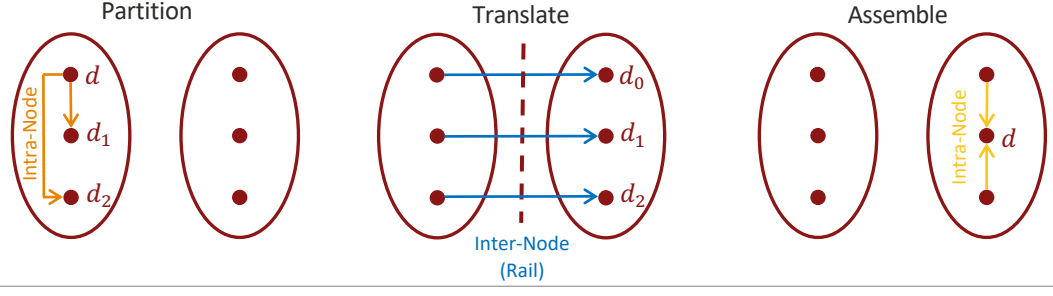
Striping μ-benchmark



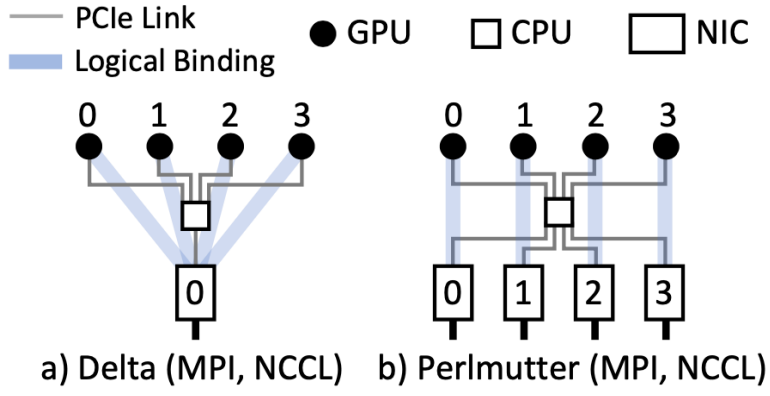
Point-to-Point Comm.

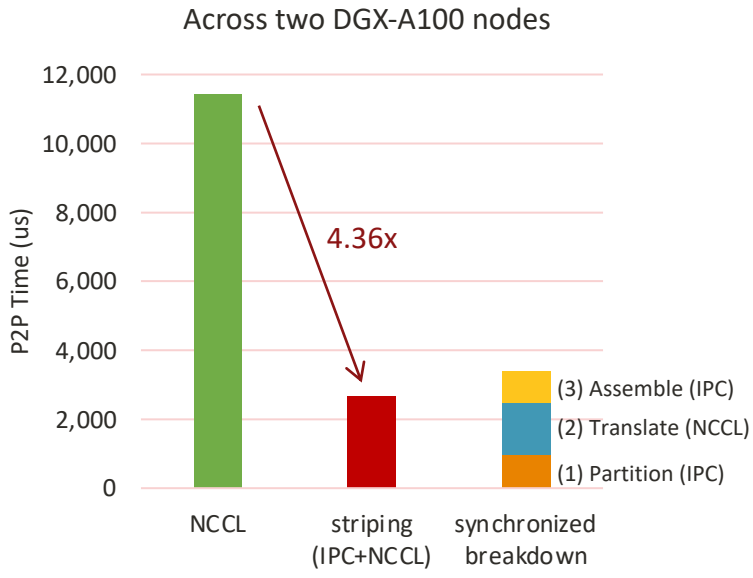


Striping Algorithm

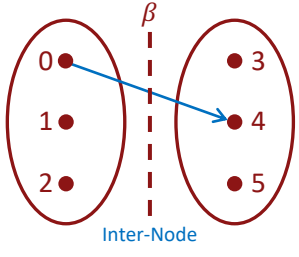


Optimization: Multi-Rail Striping

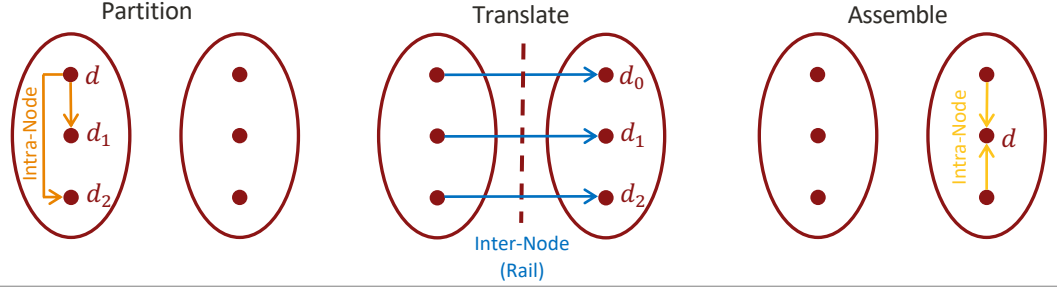




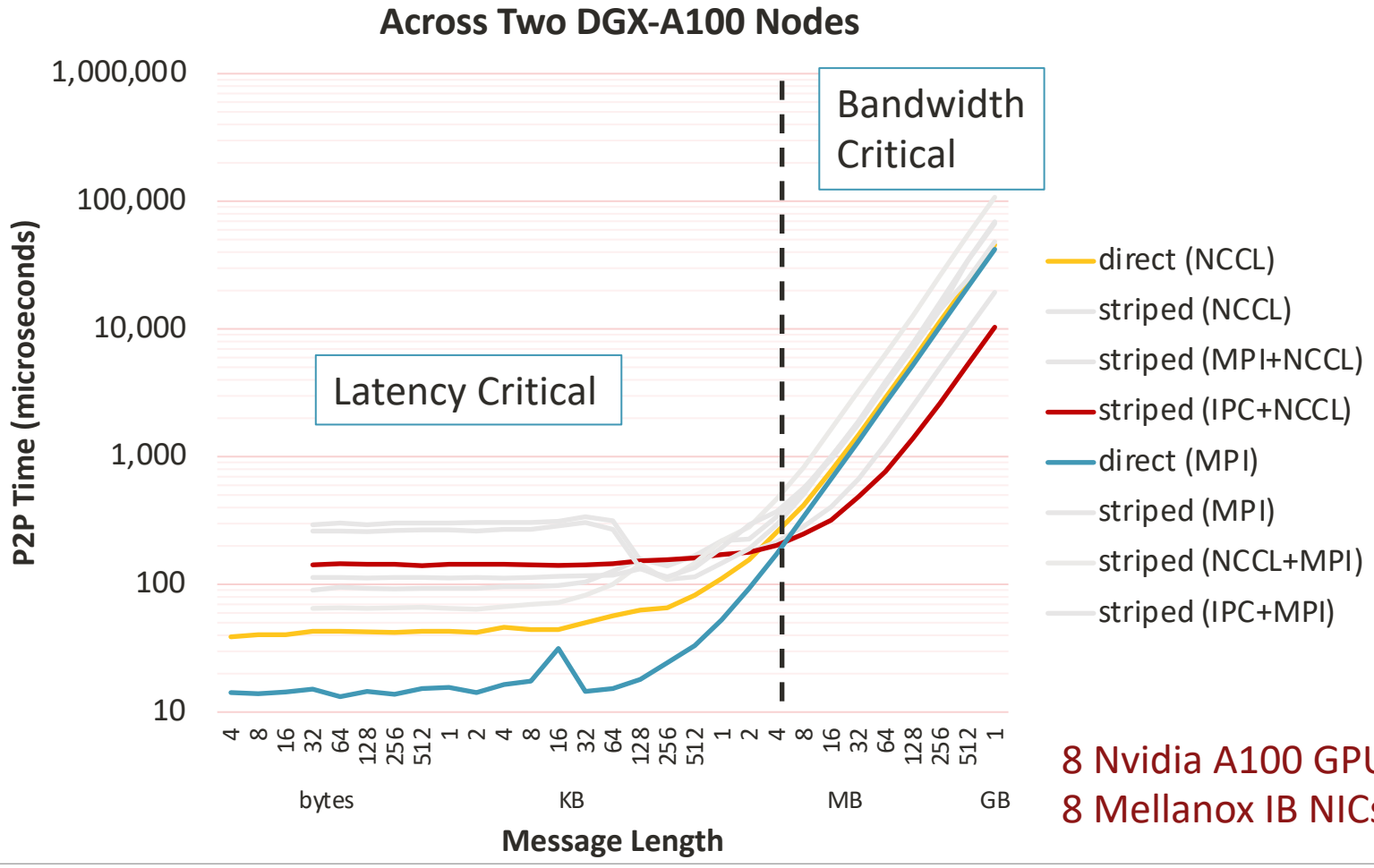
Point-to-Point Comm.



Striping Algorithm



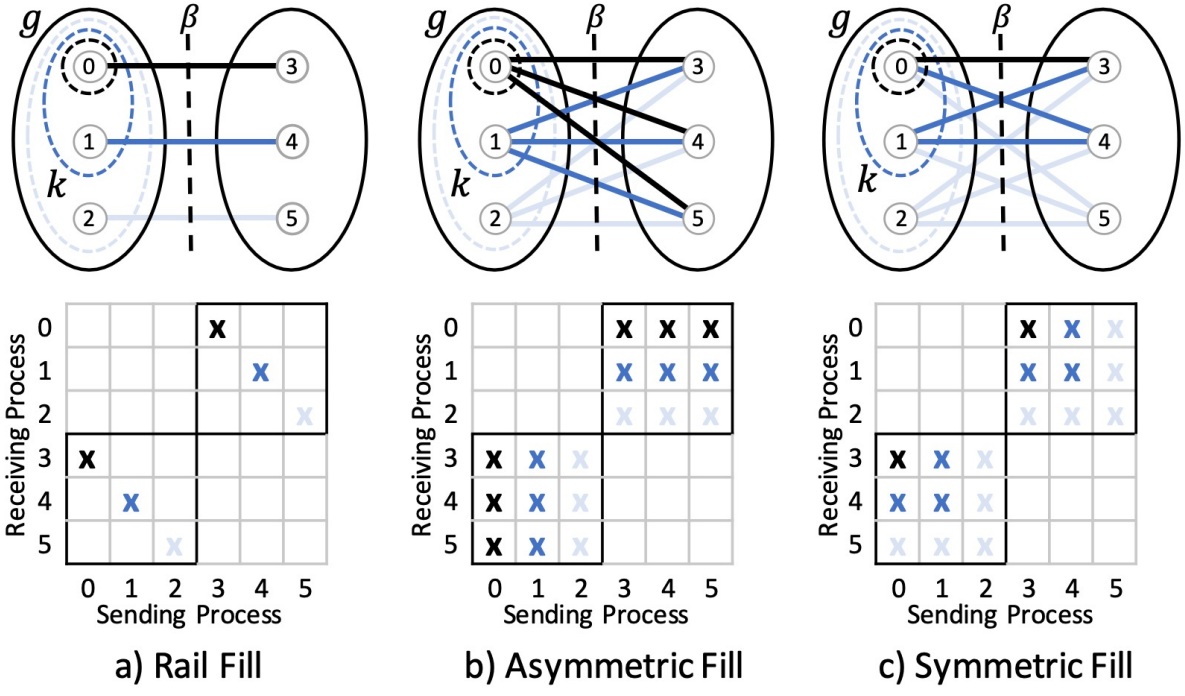
Bandwidth or Latency?



8 Nvidia A100 GPUs
8 Mellanox IB NICs

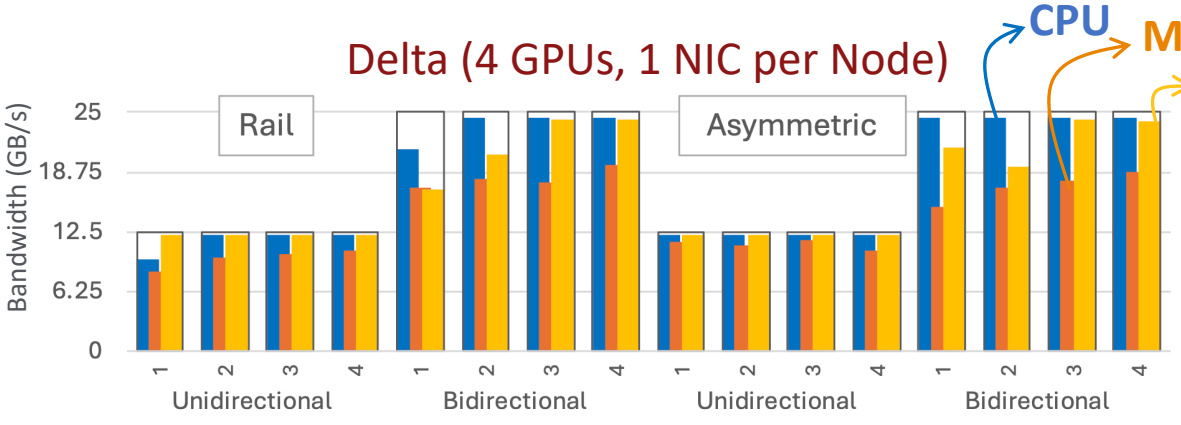
CommBench: Micro-benchmarks

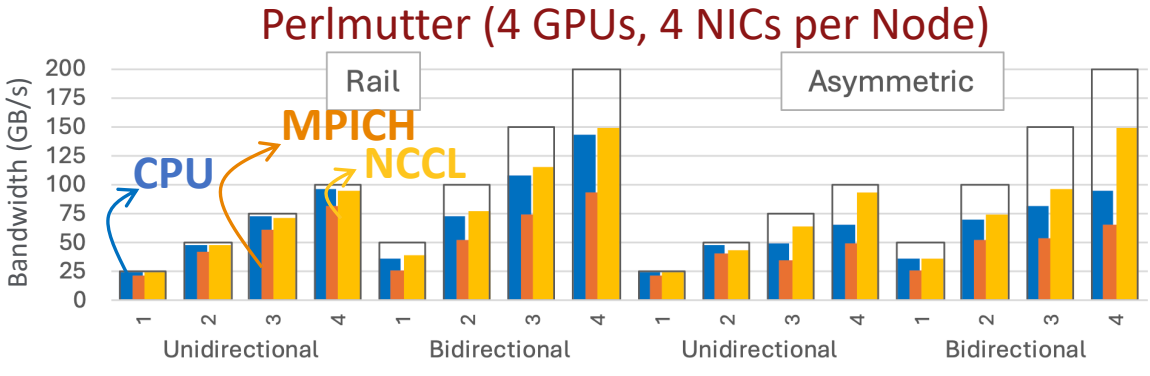
Group-to-group benchmarking

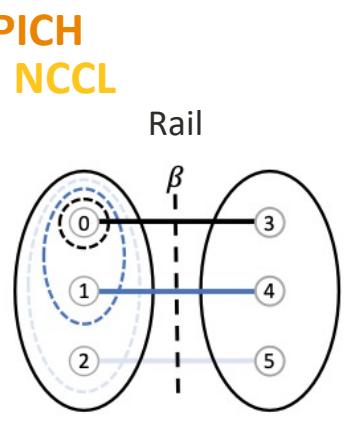


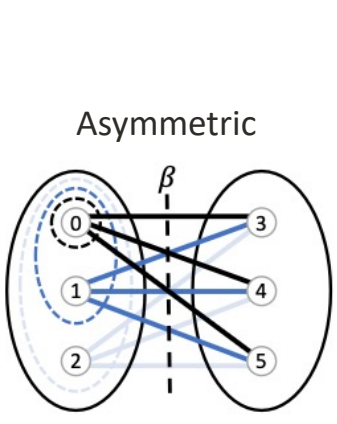
Hoefler, et al., Netgauge: A network performance measurement framework, HPCC'07
Hidayetoglu, et al., CommBench: Micro-benchmarking hierarchical networks with multi-GPU, multi-NIC nodes, ICS'24

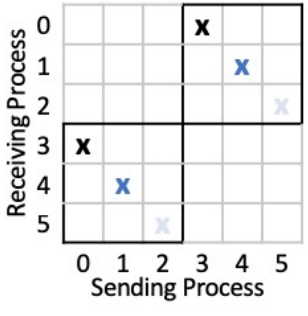
Group-to-group μ-benchmarking

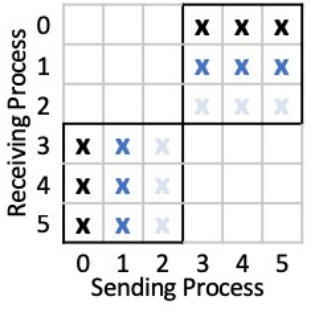




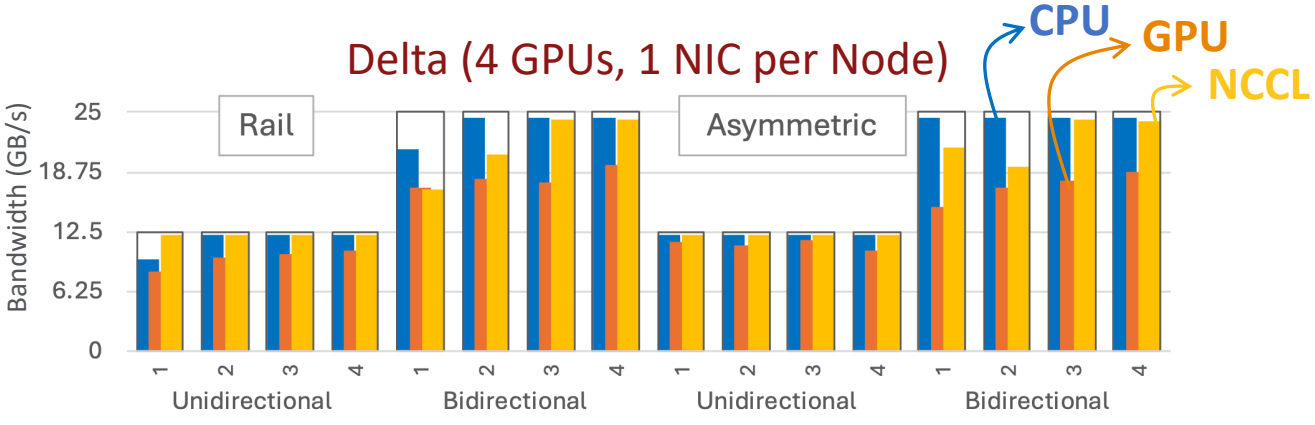


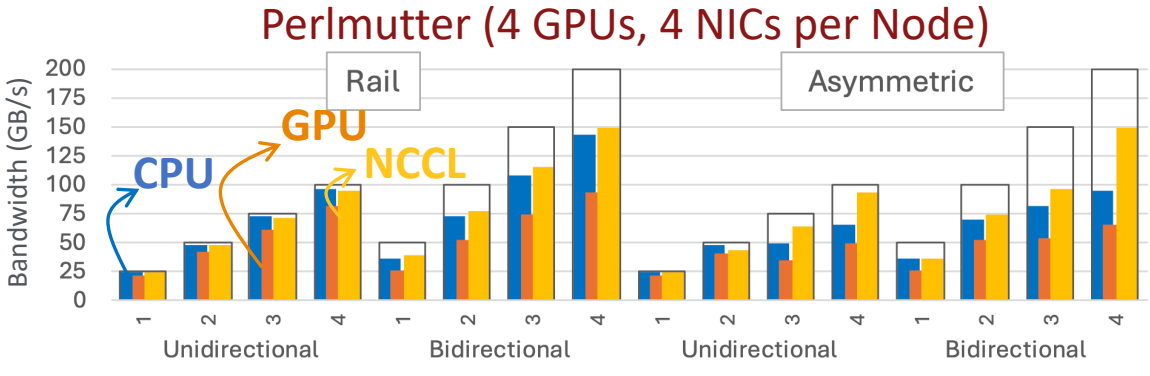




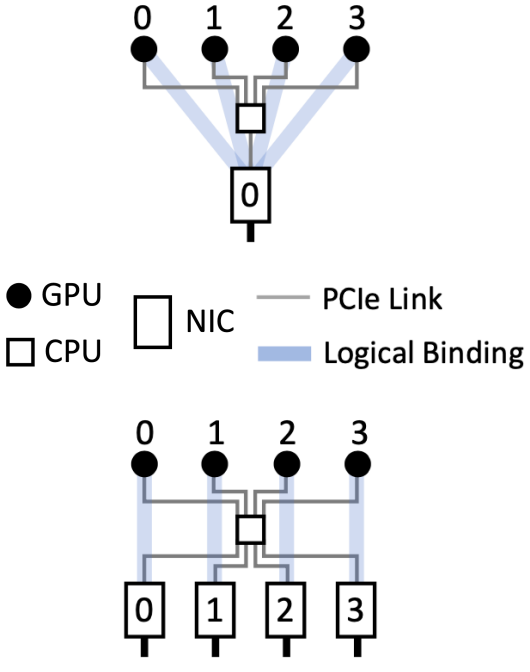


Group-to-group μ-benchmarking

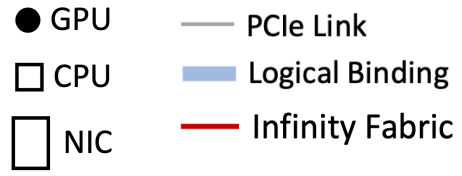




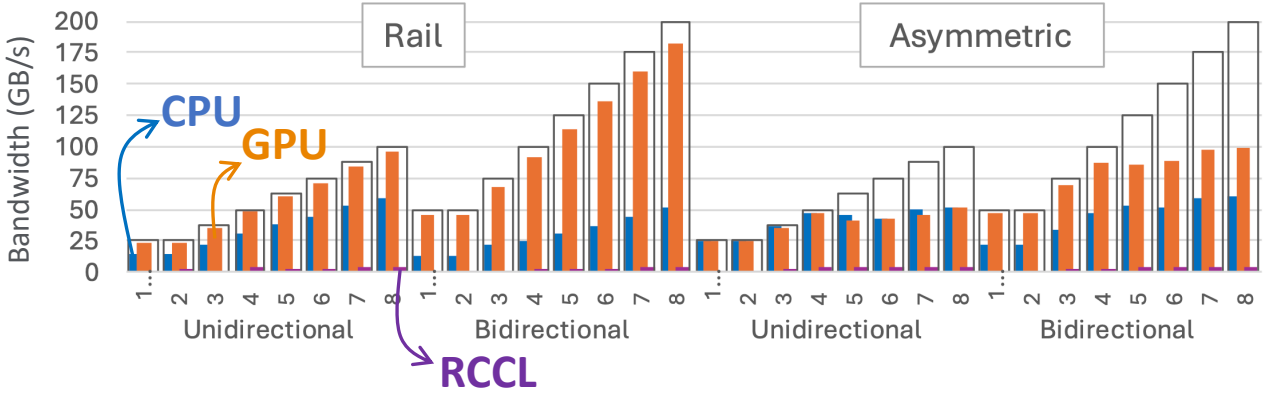
GPU-to-NIC Bindings



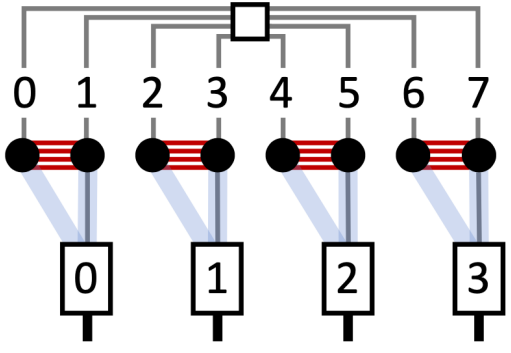
Group-to-group μ-benchmarking



Frontier (8 GPUs, 4NICs per node)



GPU-to-NIC assignments

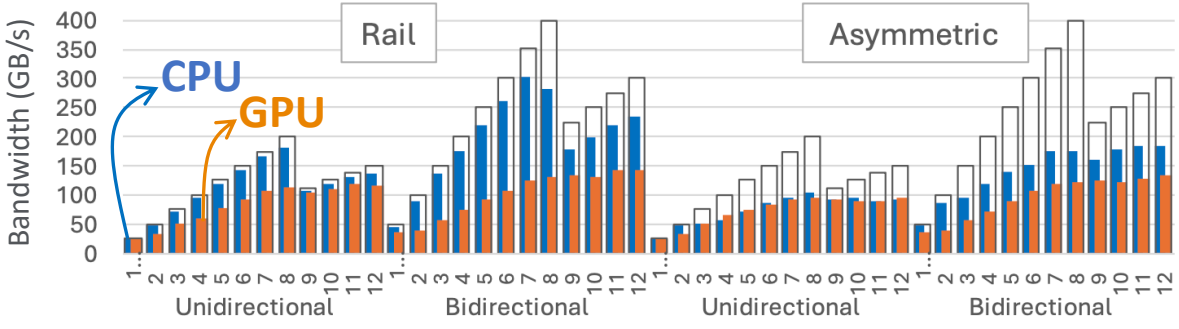


Packed assignment: $f_{\text{packed}} = f_{\text{NIC}}\left(1 + \frac{\text{ReLU}(k-p)}{p}\right)$

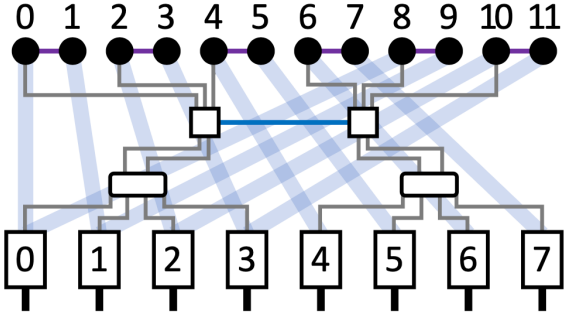
Number of GPUs per NIC

Group-to-group μ-benchmarking

Aurora (12 GPUs, 8 NICs per node)



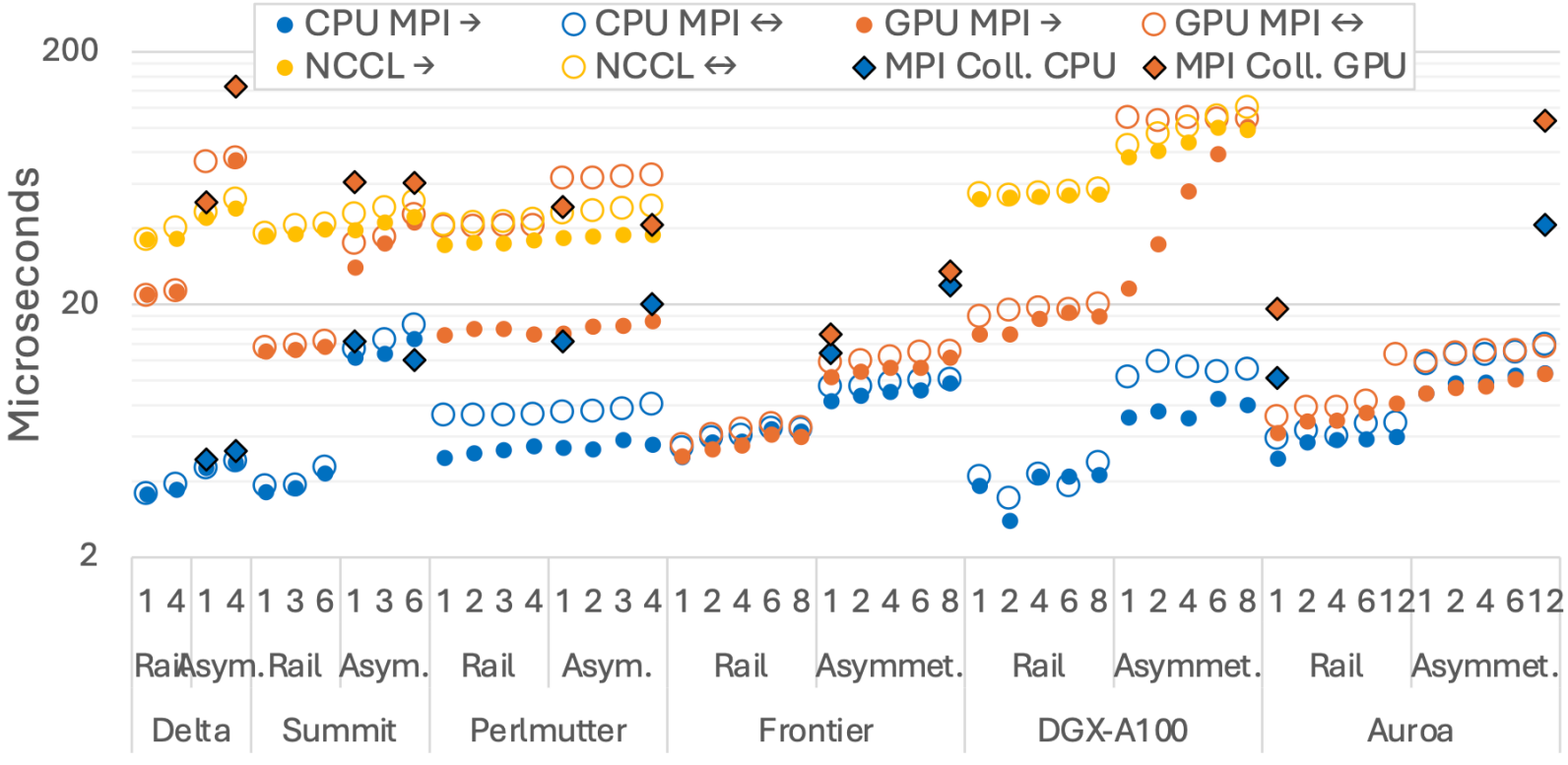
GPU-to-NIC assignments



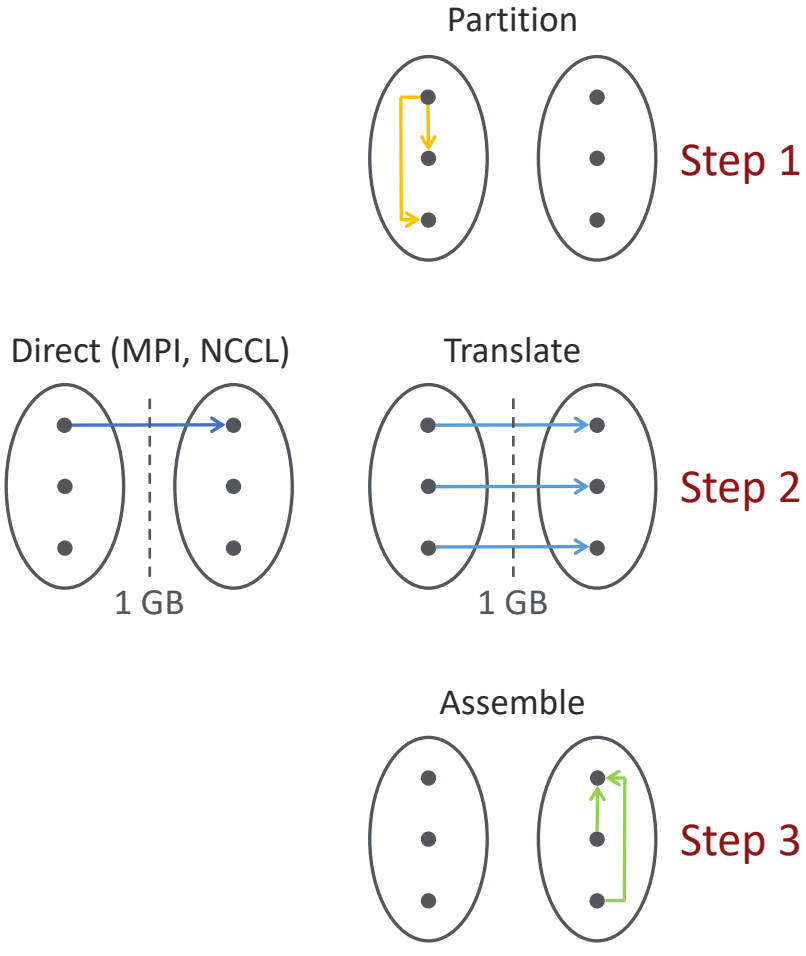
Round-robin assignment: $f_{\text{round-robin}} = f_{\text{NIC}} \dfrac{k}{\lceil k/r \rceil}$

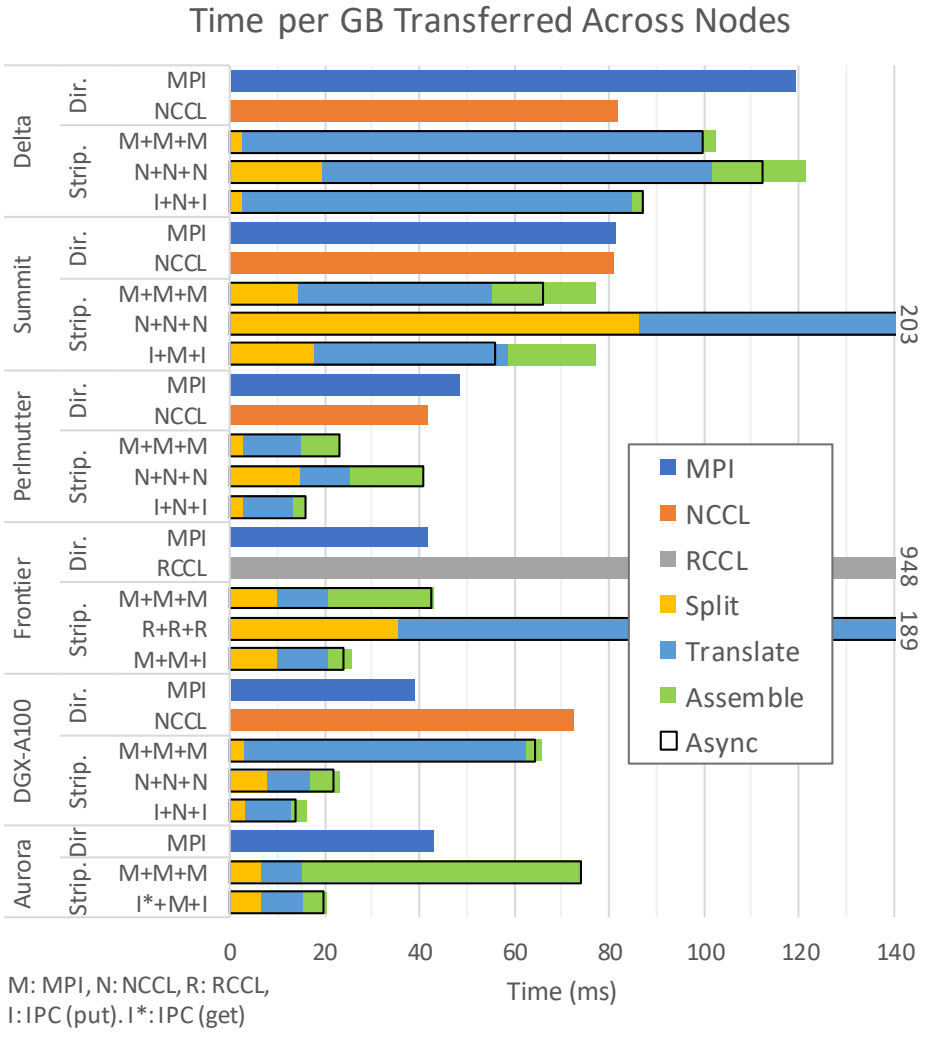
Number of NICs per node

Latency across two nodes.



Striping Micro-benchmark

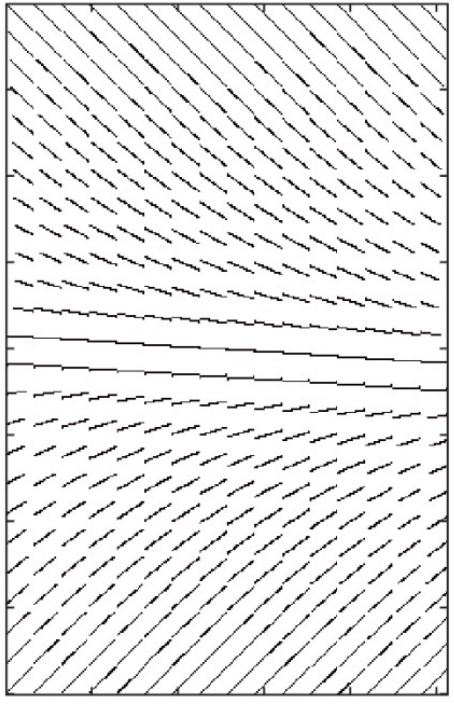




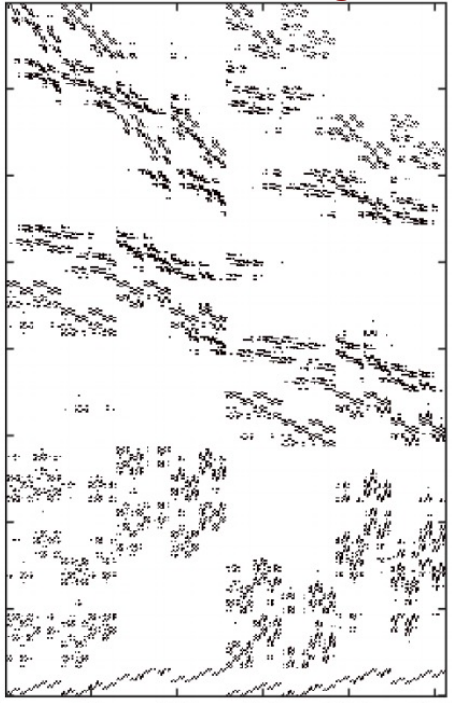
Irregular Micro-benchmark

Sparsity Patterns in X-ray imaging

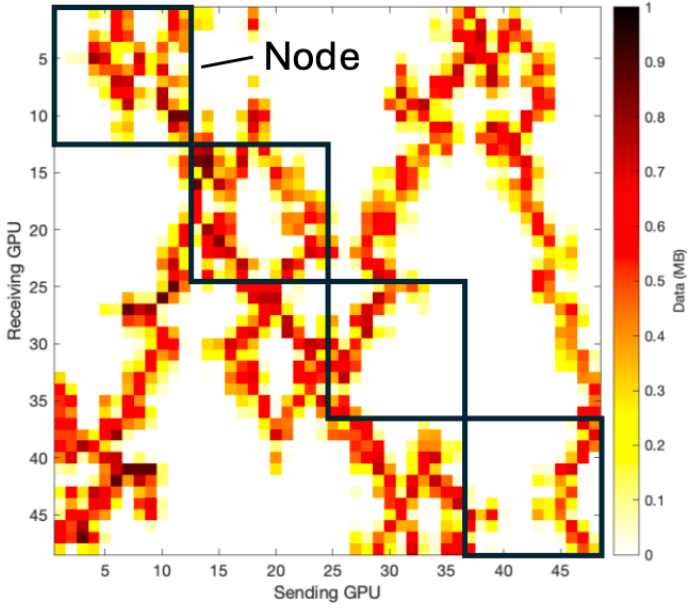
Cartesian Ordering



Hilbert Ordering

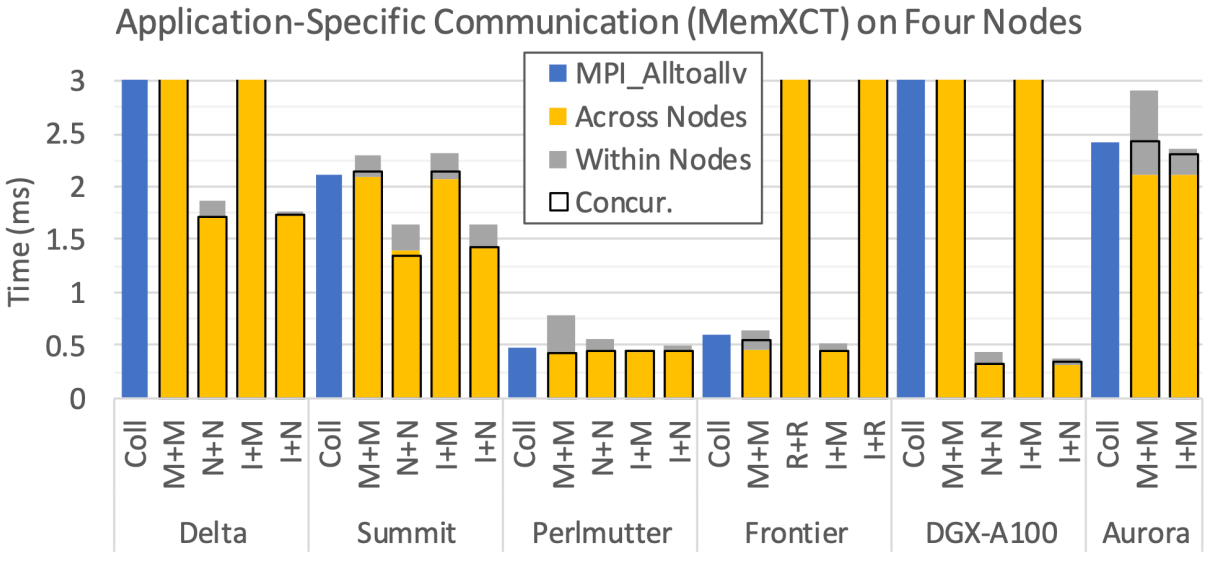


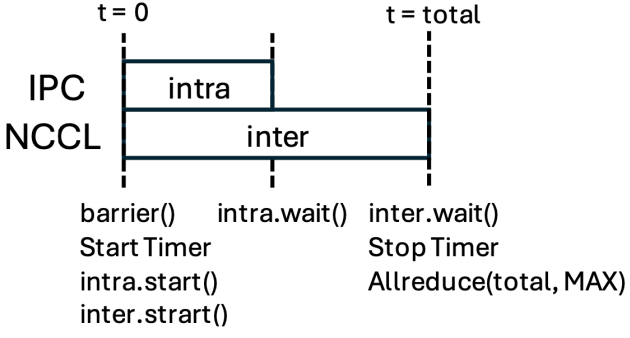
Resulting Sparse Communication Matrix



Hidayetoglu, et al., MemXCT: Memory-centric X-ray CT reconstruction with massive parallelization, SC19.

Irregular Micro-benchmark






```
std::vector<Comm<Type>> irregular = {across, within};
measure_concur(5, 10);
```

Conclusions

- GPU networks have become hierarchical
  - Group-to-group benchmarking is necessary
- Communication libraries have different behaviors across systems
  - MPI is better in terms of latency
  - NCCL is better in terms of throughput
  - GPU-to-NIC associations are crucial
- Compositional micro-benchmarking
  - Provides productivity in understanding performance implications
  - Helps developing performant applications & libraries

Thank You