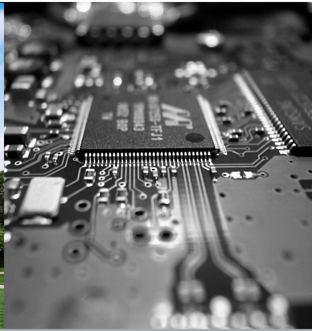
Optimizing Communication for CPU/GPU Nodes

Carl Pearson

March 11 2020



ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Carl Pearson

Ph.D. student, Electrical and Computer Engineering, University of Illinois Urbana-Champaign

- Advised by Professor Wen-Mei Hwu
- (Multi-)GPU communication
- Accelerating irregular applications

cwpearson
cwpearson
pearson at illinois.edu
https://cwpearson.github.io

Background

Application Acceleration

GPU Triangle counting
[HPEC ‘18]

Inverse Scattering
[IPDPS ‘18]

FGPA Triangle counting
[HPEC ‘18]

Stencil
[IPDPS Wksh. ‘20]

Triangle counting
[HPEC ‘19]

GPU Communication

Comm|Scope
[ICPE ‘19]
[best paper]

GPU Education

WebGPU
[IPDPS Wksh. ‘16]

RAI
[IPDPS Wksh. ‘17]

Neural Network Course
Material [‘18-’20]

Outline

- Research Background
- Benchmarking heterogeneous system communication
- Acceleration of a stencil code
- Future Directions

SCOPE Benchmarking Framework

GPU benchmarking framework

amd64 and ppc64le
CUDA

- Comm|Scope **(Pearson et al. ICPE '19 Best Paper)**
- TCU|Scope (Dakkak et al. ICS '19)
- NCCL|Scope
- CUDNN|Scope (Li et al. ICS' 19)

University of Illinois / IBM Center for Cognitive Computing Systems Research (C^3SR)

Prof. Wen-Mei Hwu (Illinois)
Jinjun Xiong (IBM T. J. Watson Research)

https://scope.c3sr.com

https://github.com/c3sr/scope

Comm|Scope

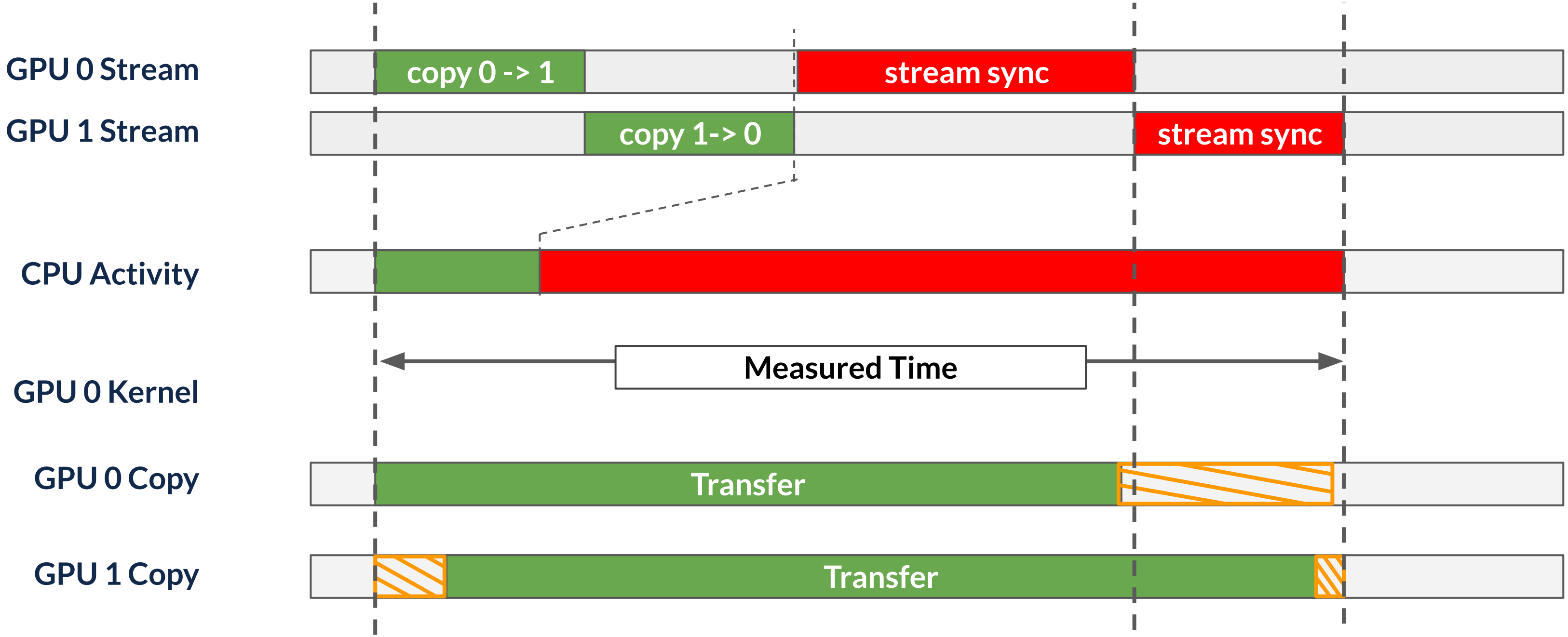
SCOPE plugin: multi-socket multi-GPU communication microbenchmarks

amd64 & ppc64le
CUDA
NUMA-aware allocation and pinning
cache control
asynchronous CUDA operations

"Final word" and examples for CUDA communication benchmarking

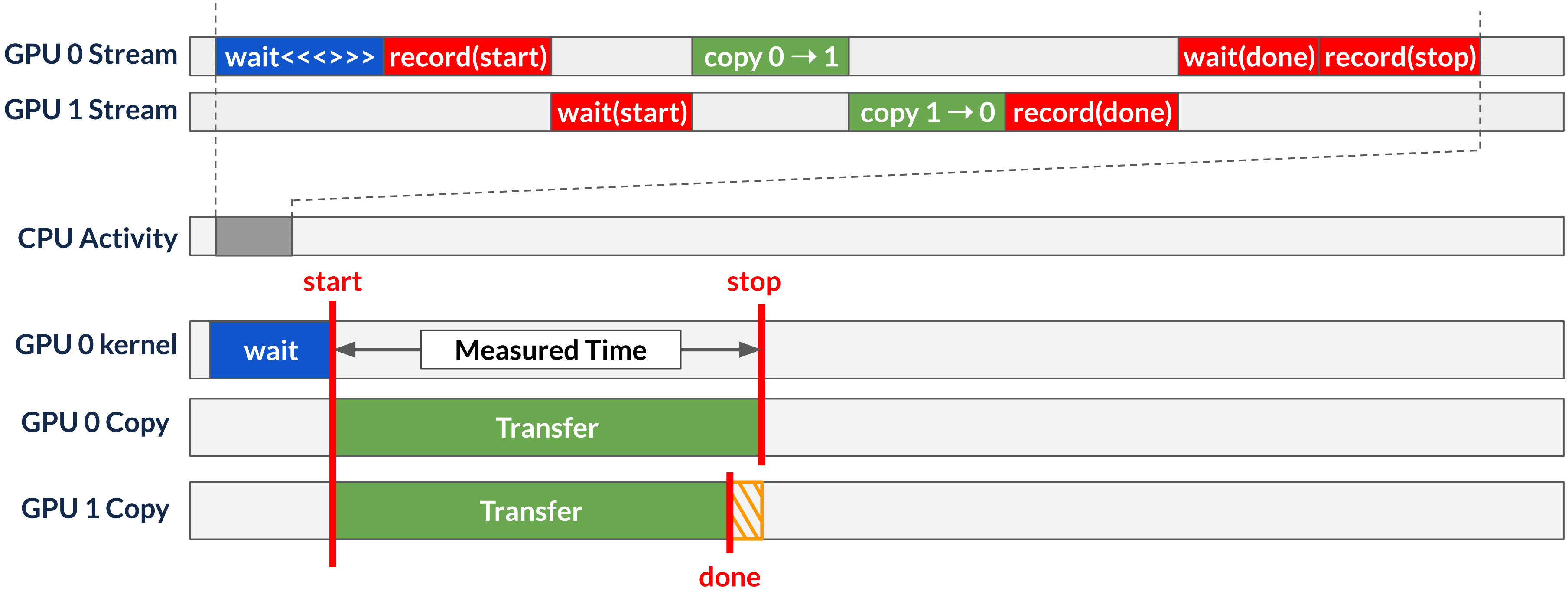
Transfer	Host Alloc.	Device Alloc.	Direction
`cudaMemcpy`	pageable (NUMA)	`cudaMalloc`	H2D / D2H / bi
`cudaMemcpy`	pinned (NUMA)	`cudaMalloc`	H2D / D2H / bi
zero-copy	mapped	-	H2D
zero-copy	-	`cudaMalloc`	D2D / bi
`cudaMemcpy`	-	`cudaMalloc`	D2D / bi
`cudaMemcpy` (peer)	-	`cudaMalloc`	D2D / bi
`cudaMemcpyPeer`	-	`cudaMalloc`	D2D / bi
`cudaMemcpyPeer` (peer)	-	`cudaMalloc`	D2D / bi
demand	`cudaMallocManaged`		H2D / D2H / D2D / bi
prefetch	`cudaMallocManaged`		H2D / D2H / D2D / bi

Measuring Bidirectional Transfers



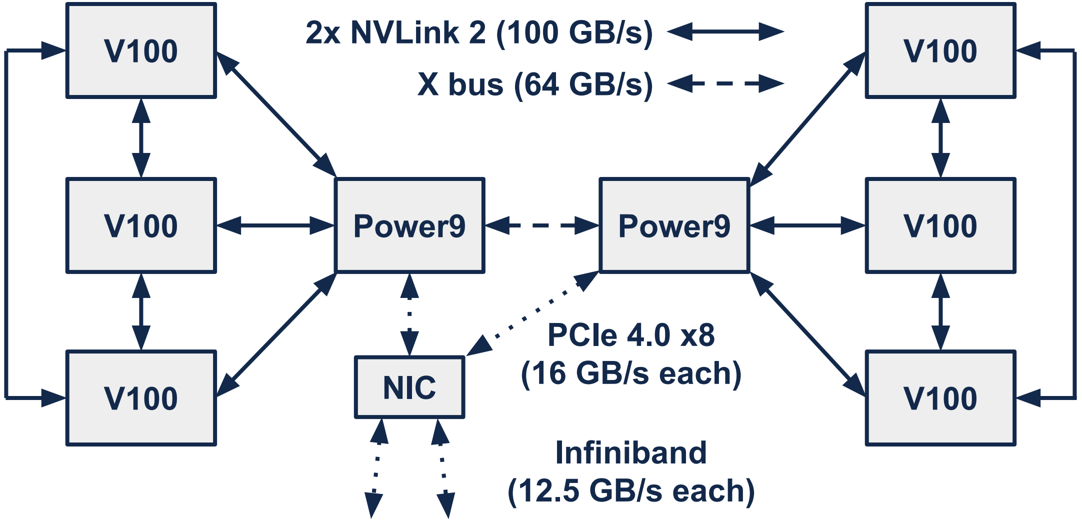
Measure runtime cost at start, and stream sync cost at end

Measuring Bidirectional Transfers



Kernel prevents copies from starting until both are issued.
Events minimize measured overhead.

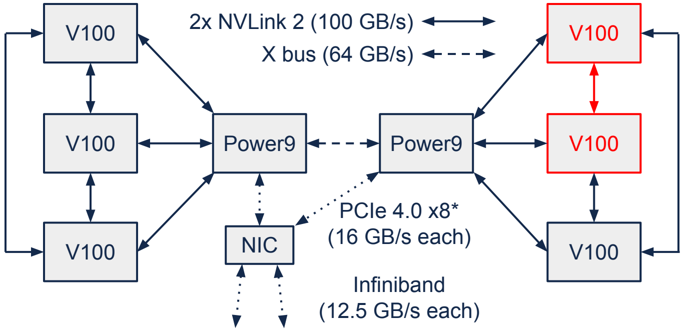
Why measure at all?



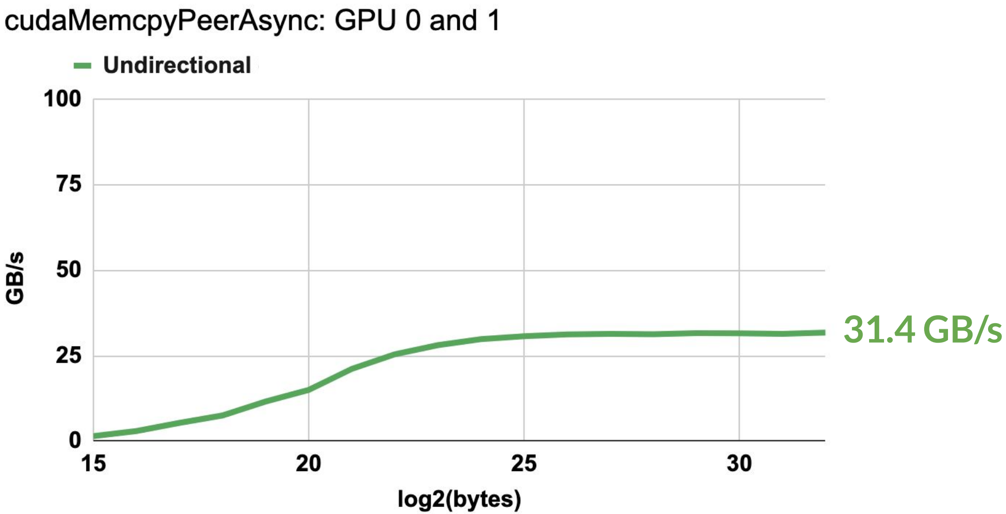
Summit Node
(bidirectional bandwidth)

* “shared” between CPUs.

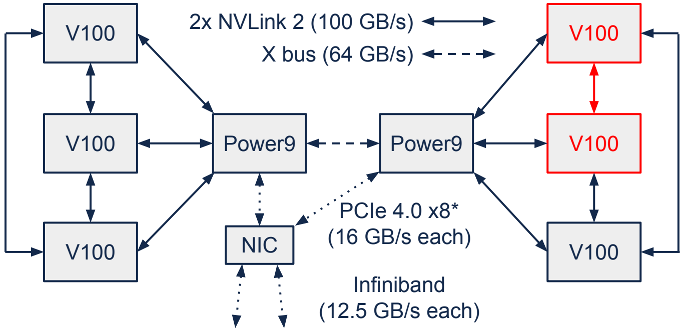
Why measure at all?



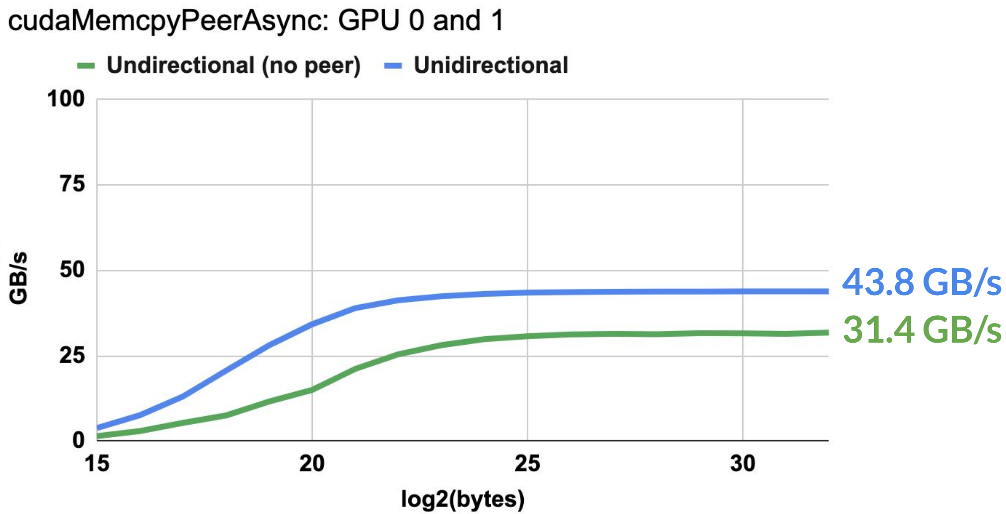
Summit Node
(bidirectional bandwidth)



Why measure at all?

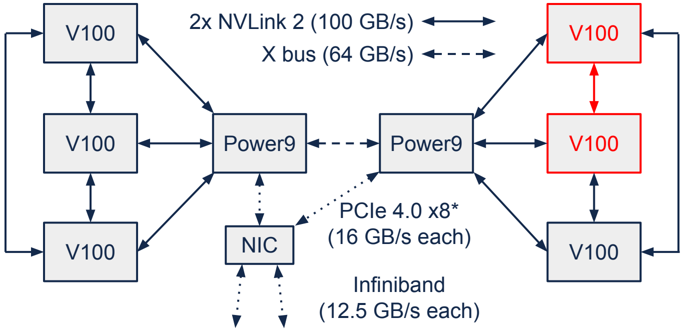


Summit Node
(bidirectional bandwidth)

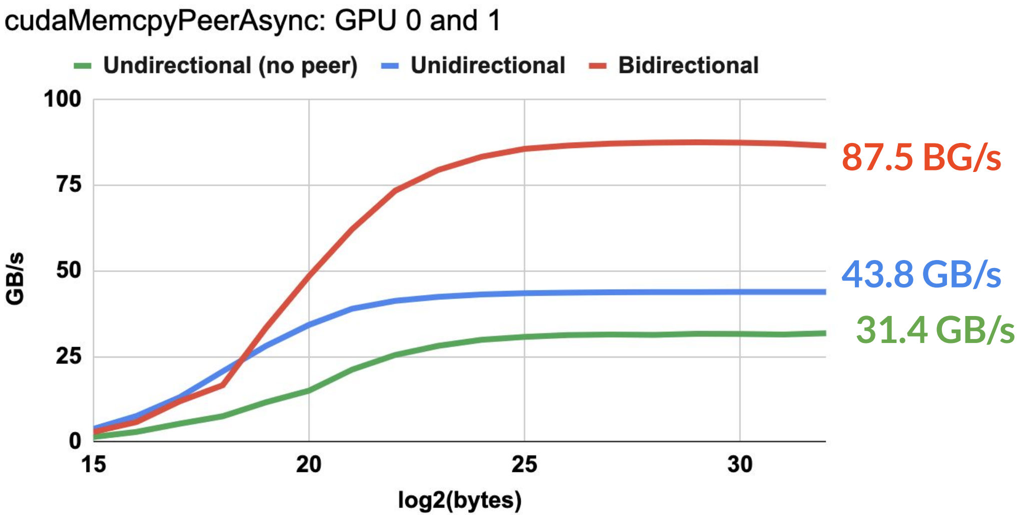


Enable peer access near beginning of program (`cudaDeviceEnablePeerAccess`)

Why measure at all?

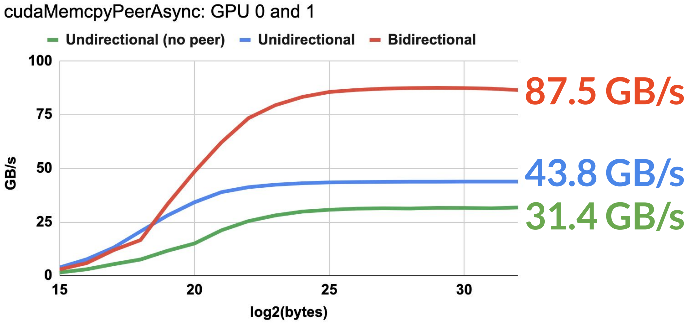


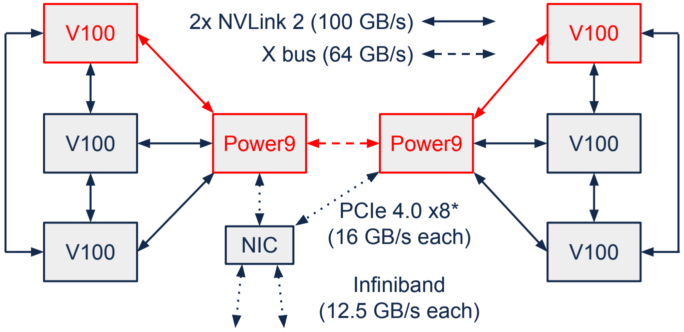
Summit Node
(bidirectional bandwidth)

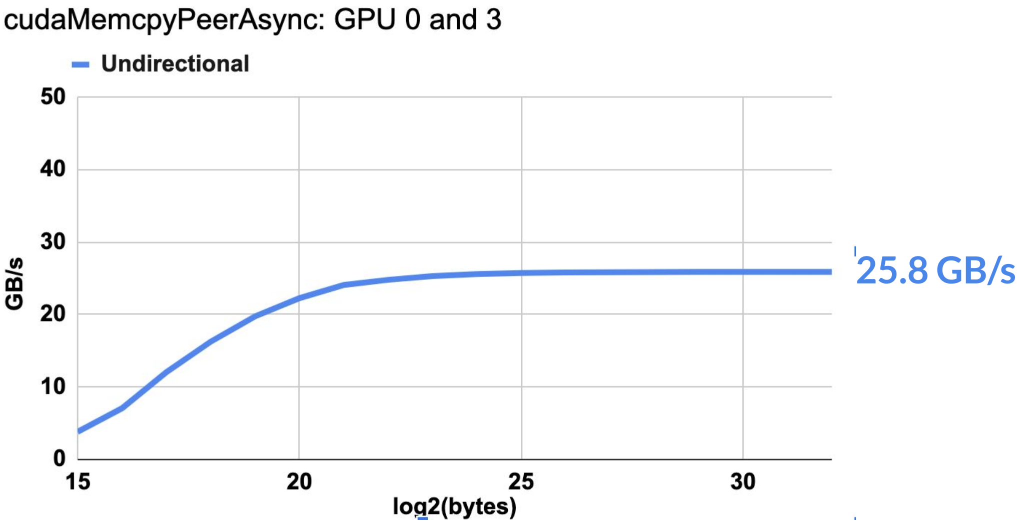


Bidirectional transfers double bandwidth

Why measure at all?



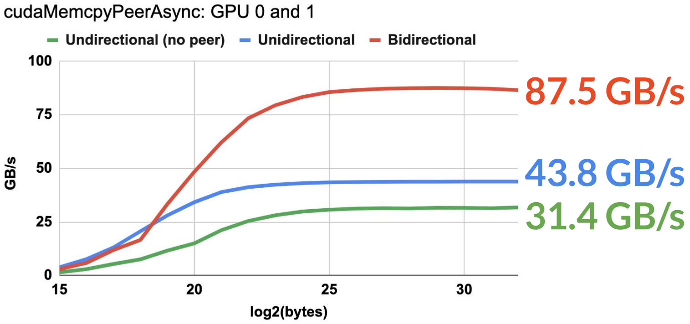


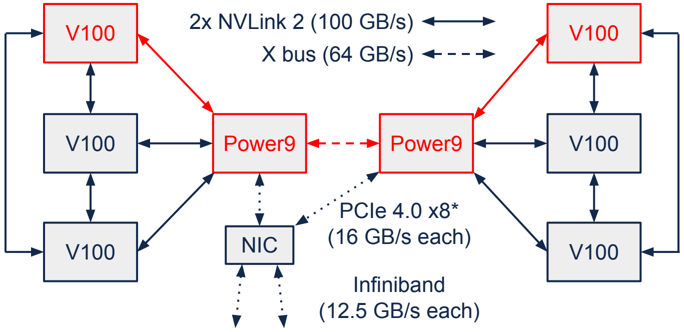


Summit Node
(bidirectional bandwidth)

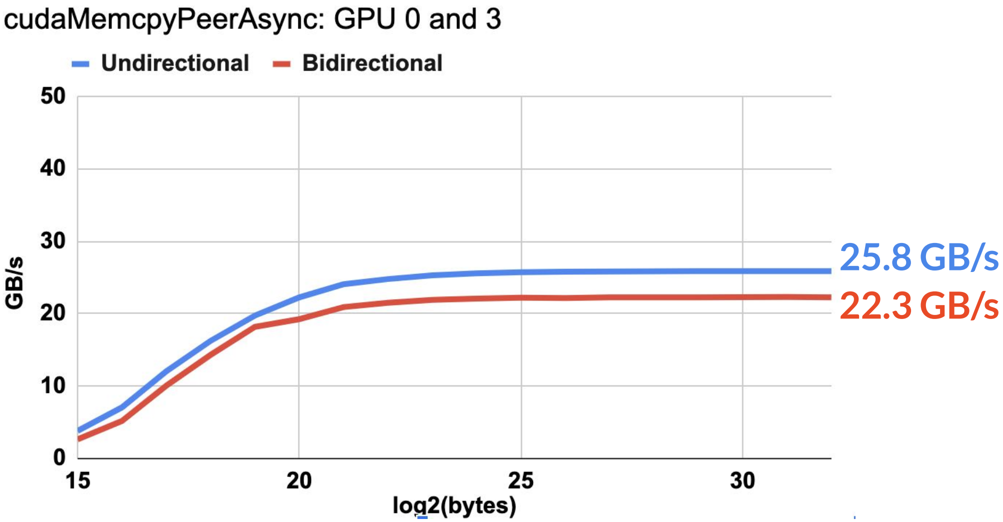
Transfers between sockets are slower

Why measure at all?



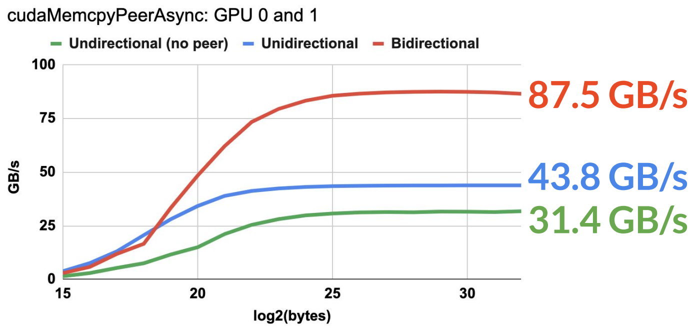


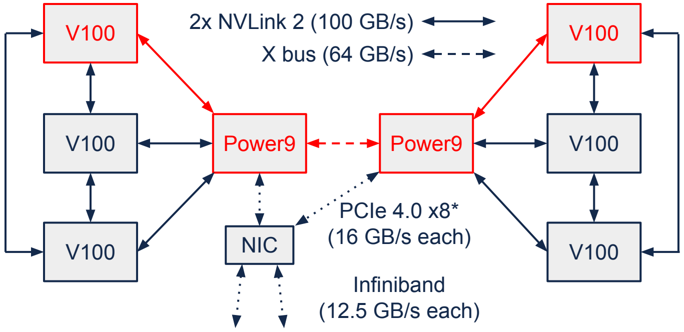
Summit Node
(bidirectional bandwidth)



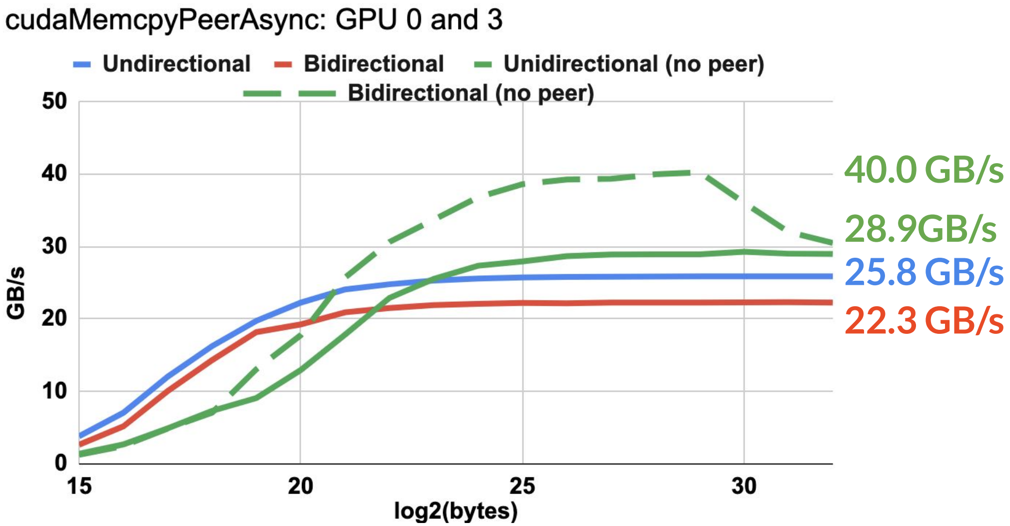
Bidirectional transfers are even slower

Why measure at all?





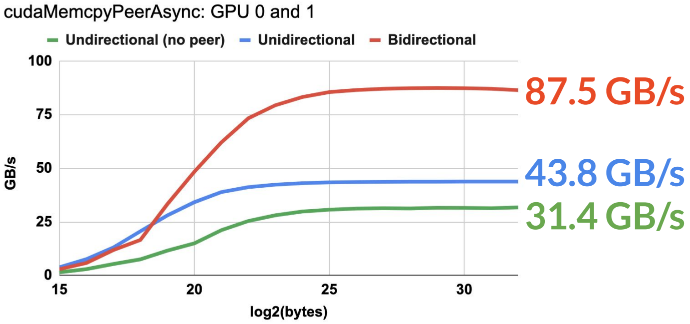
Summit Node
(bidirectional bandwidth)

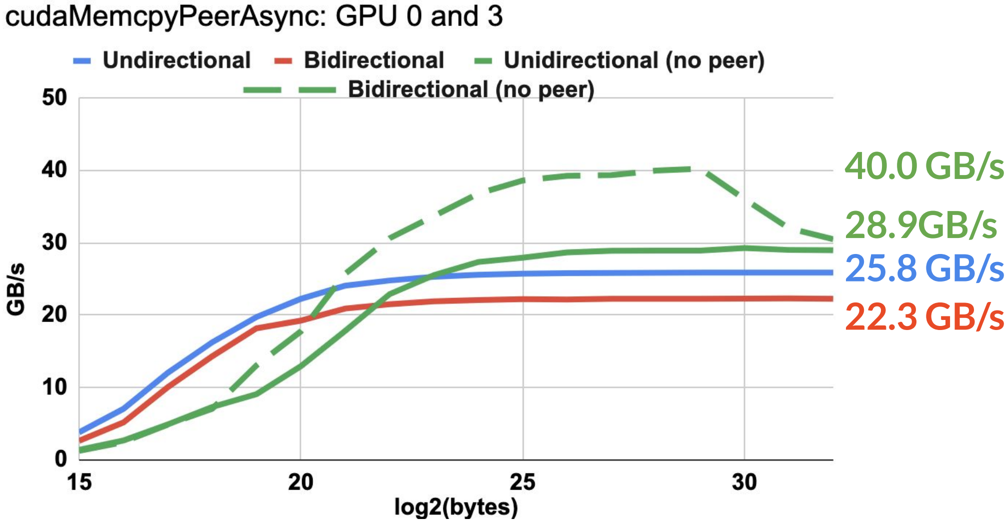


Disabling peer access is faster. Systems do not always behave according to expectations

Why measure at all?

- Peer access disabled ➡️ data staged through CPU
- X-bus for CPU-CPU works as promised, *not* for GPU-GPU
- Answering why as an outsider is difficult for closed drivers & firmware
- Some need for a high-level test to make sure system performs as advertised



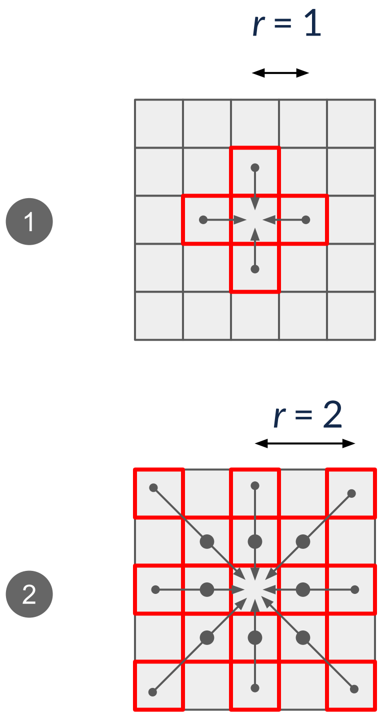


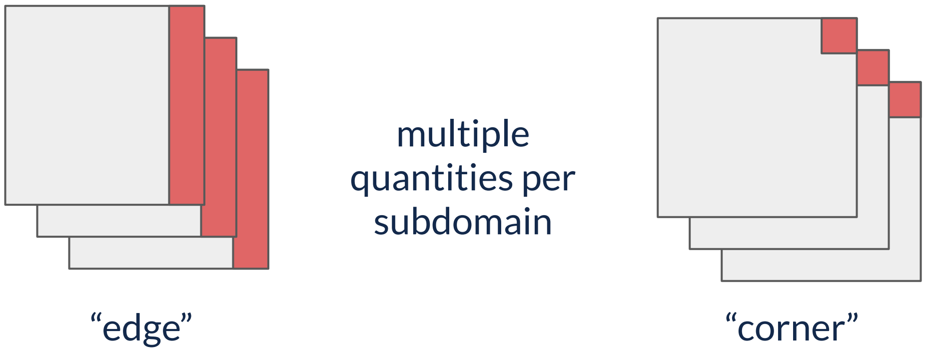
Distributed Stencils & Heterogeneous Nodes

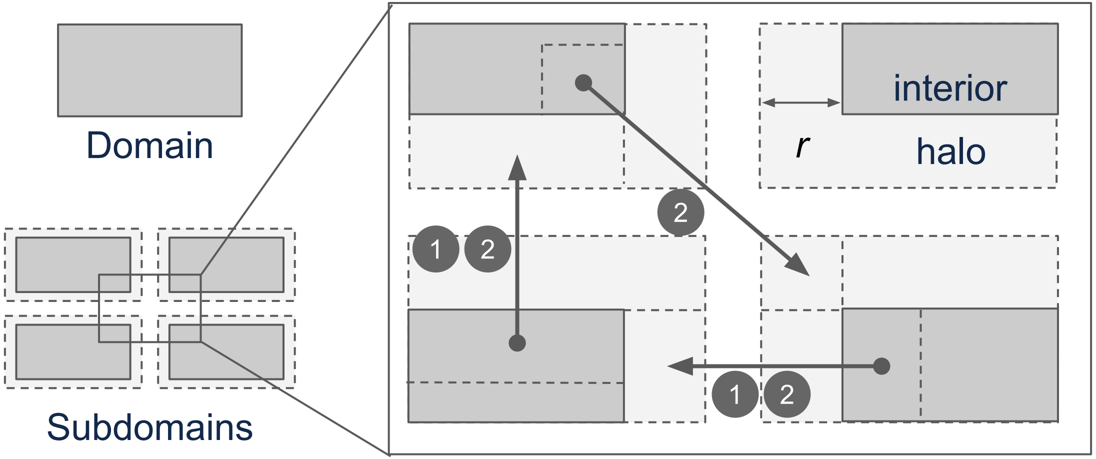
- Finite Difference Methods
- Regular computation, access, and structure reuse ➡️ stencil on GPU
- High-resolution modeling ➡️ Large stencils
- Limited GPU memory ➡️ distributed stencils with communication
- Fast stencil codes ➡️ larger impact of communication
- Heterogeneous nodes (“fat nodes”) ➡️ how to do communication

- Performance impact of the on-node optimizations
- Packaging this so science people don’t need to be GPU communications people too

Stencil Glossary



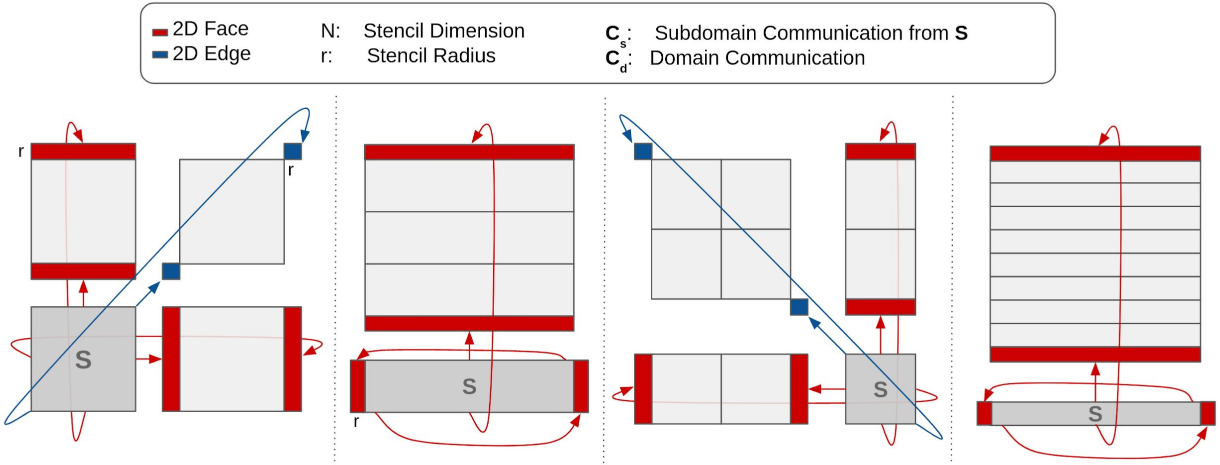




Approach

Parallelism	Scalable decomposition	Subdomain decomposition to minimize communication
Placement	Assign tasks according to theoretical performance	Node-aware placement to utilize interconnections
Primitives	Achieve theoretical performance	Asynchronous operations Communication specialization

Decomposition - Minimize Required Comm.



Partition: 2x2
S size = **N/2** x **N/2**
$\mathbf{C_s}$ = 4rN/4 + $2r^2$
$\mathbf{C_d}$ = 4$\mathbf{C_s}$ = 8Nr + $8r^2$

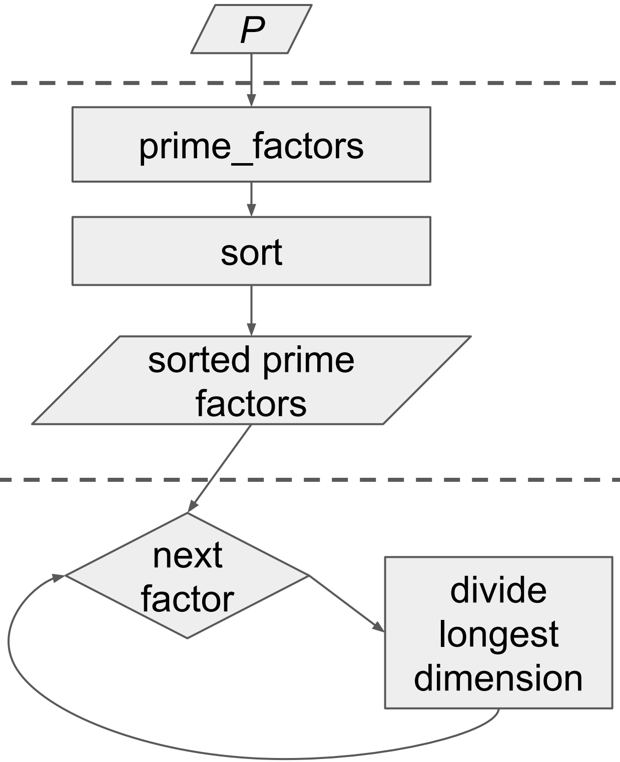
Partition: 4x1
S size = **N/4** x **N**
$\mathbf{C_s}$ = 2rN + 2rN/4
$\mathbf{C_d}$ = 4$\mathbf{C_s}$ = 10Nr

Partition: 3x3
S size = **N/3** x **N/3**
$\mathbf{C_s}$ = 4rN/3 + $2r^2$
$\mathbf{C_d}$ = 9$\mathbf{C_s}$ = 12Nr + $18r^2$

Partition: 9x1
S size = **N** x **N/9**
$\mathbf{C_s}$ = 2rN + 2rN/9
$\mathbf{C_d}$ = 9$\mathbf{C_s}$ = 20Nr

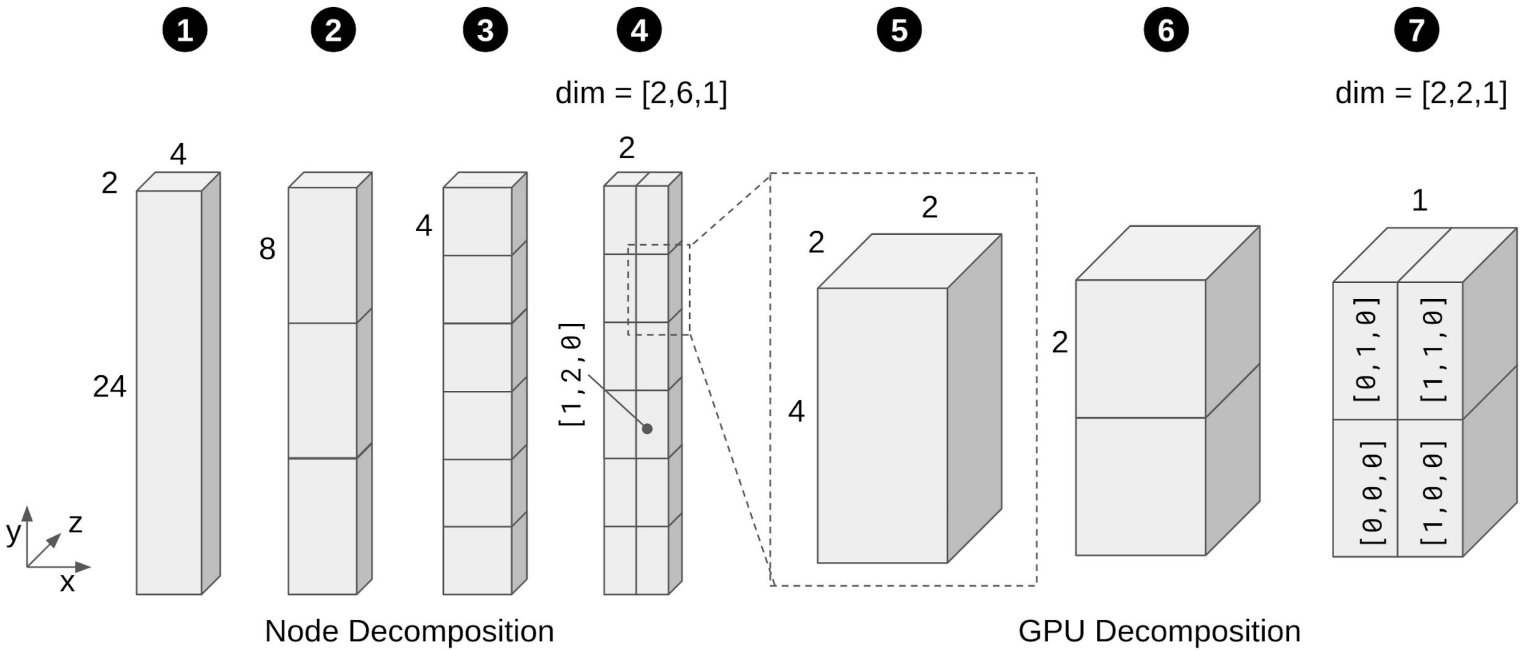
Intuition: less halo-to-interior ratio means less communication

Decomposition - Recursive Inertial Bisection



- Divide given domain into *P* subdomains
- Generate sorted prime factors, largest to smallest.
  - Evenly-sized subdomain require dividing by integers.
  - Prime factors is the largest number of integers that multiply to P
  - Most opportunity to divide into cubical subdomains
- Divide the longest dimension by prime factors
  - subdomains tend towards cubical
  - use smaller prime factors later to clean up

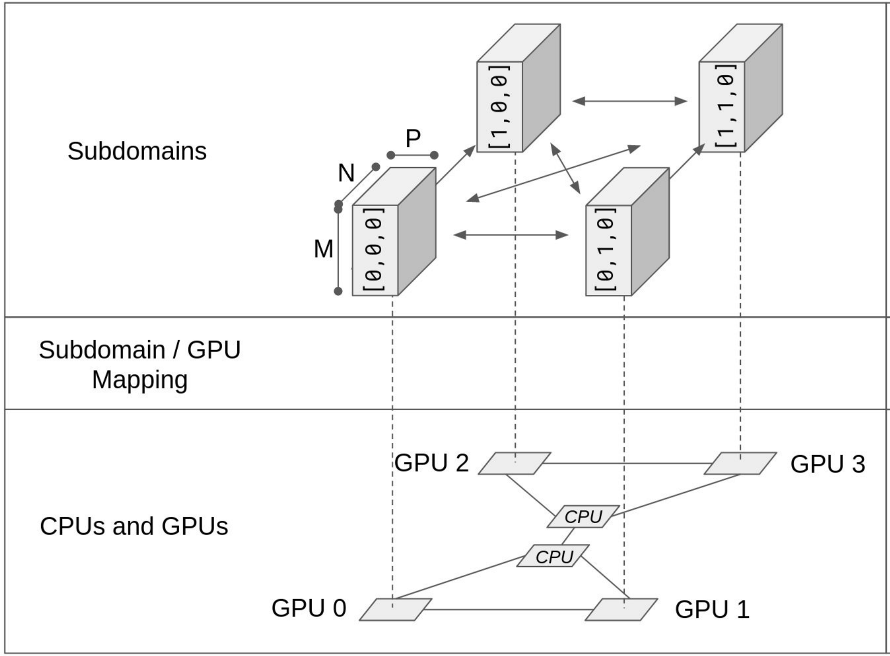
Hierarchical Decomposition



Minimize Communication Out of Node

Minimize communication between GPUs

Placement



$$\begin{array}{c|cccc} & [0,0,0] & [1,0,0] & [0,1,0] & [1,1,0] \\ \hline [0,0,0] & 0 & MP & MN & M \\ [1,0,0] & MP & 0 & M & MN \\ [0,1,0] & MN & M & 0 & MP \\ [1,1,0] & M & MN & MP & 0 \end{array}$$

$$\begin{array}{c|cccc} & \text{GPU 0} & \text{GPU 1} & \text{GPU 2} & \text{GPU 3} \\ \hline \text{GPU 0} & 0 & 75 & 64 & 64 \\ \text{GPU 1} & 75 & 0 & 64 & 64 \\ \text{GPU 2} & 64 & 64 & 0 & 75 \\ \text{GPU 3} & 64 & 64 & 75 & 0 \end{array}$$

How to place subdomains on GPUs to maximize bandwidth utilization?

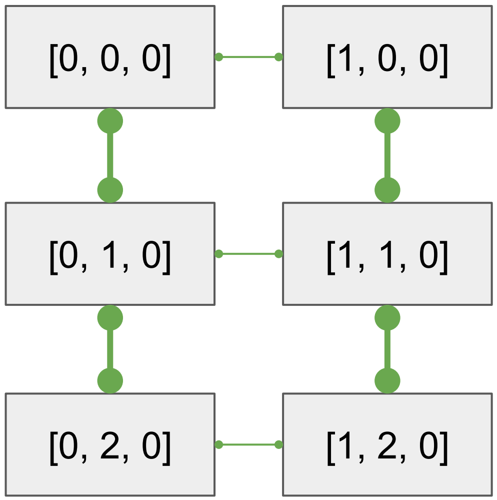
Quadratic Assignment Problem

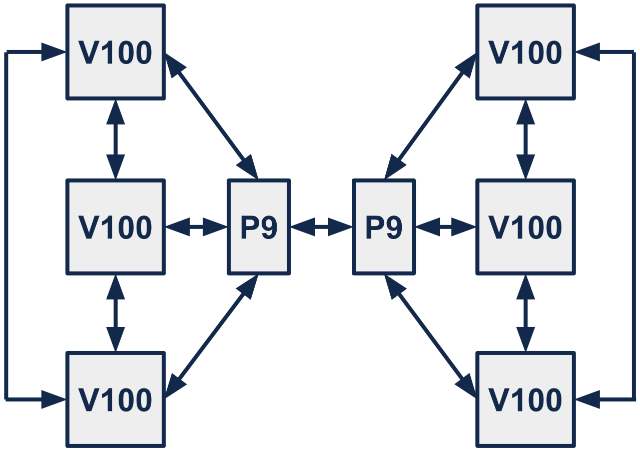
n facilities with "flow" between them.
n locations with "distance" between them.
Assign facilities to locations while minimizing total flow-distance product.
Facilities with a lot of flow should be close.

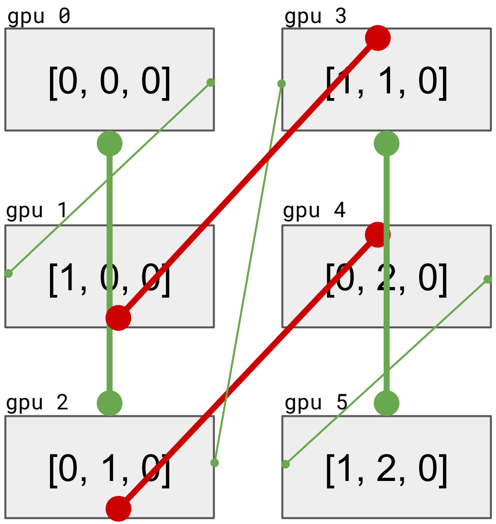
$$\sum_{i,j<n} w_{i,j} d_{f(i),f(j)}$$

	Abstract	Concrete
$w, w_{i,j}$	Matrix of "flow" between facilities *i* and *j*.	subdomain communication amount
$d, d_{i,j}$	Matrix of "distance" between locations *i* and *j*.	GPU distance matrix
f	$n \rightarrow n$ bijection assigning facilities to locations	*n* vector

Example Placement







Node-Aware Placement

20% reduced exchange time from placement alone

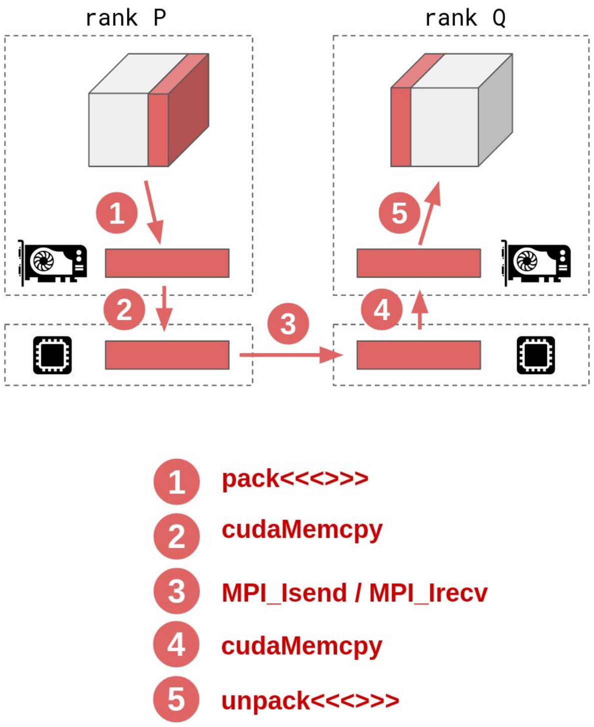
Another Placement

Capability Specialization

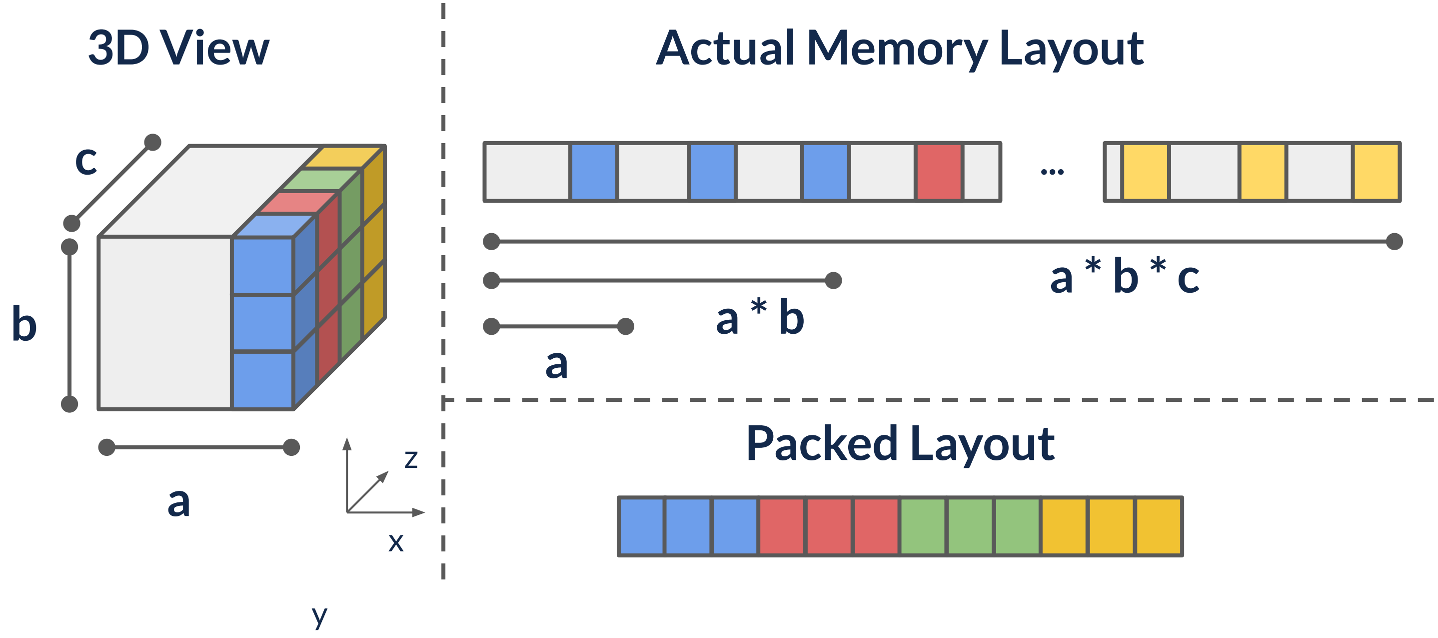
Achieve best use of bandwidth, regardless of ranks/node and GPUs/rank

- “Staged”: works for any 2 GPUs anywhere
    - pack from device 3D region into device 1D buffer
    - copy from device 1D buffer to host 1D buffer
    - MPI_Isend / MPI_Irecv to other host 1D buffer
    - copy from host 1D buffer to device 1D buffer
    - unpack from device 1D buffer to device 3D buffer

Optimizations are node-aware shortcuts on top of this

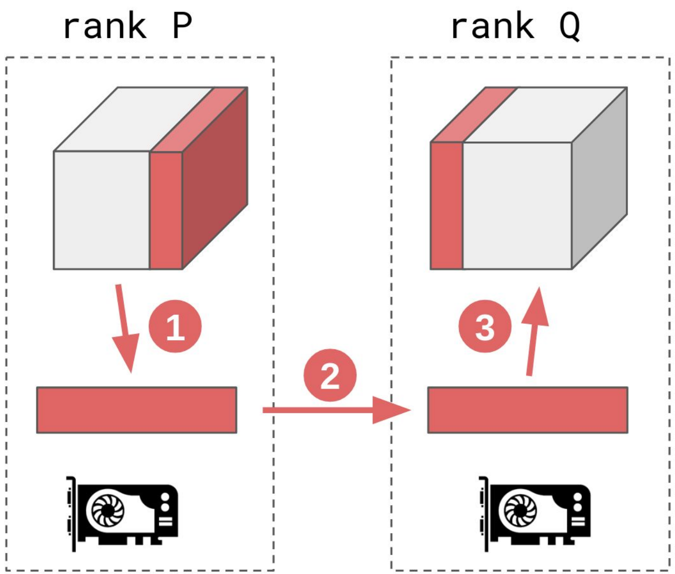


Pack and Unpack



CUDA-Aware MPI

1. pack<<<>>>
2. MPI_Isend / MPI_Irecv
3. unpack<<<>>>



Same as the staged, but MPI responsible for getting data between GPUs

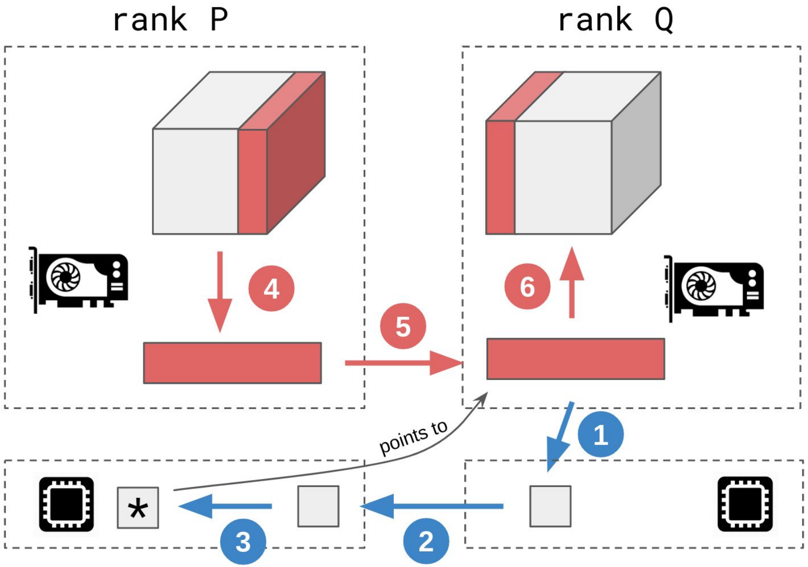
Colocated

setup

1. cudaIpcGetMemHandle
2. MPI_Isend / MPI_Irecv
3. cudaIpcOpenMemHandle

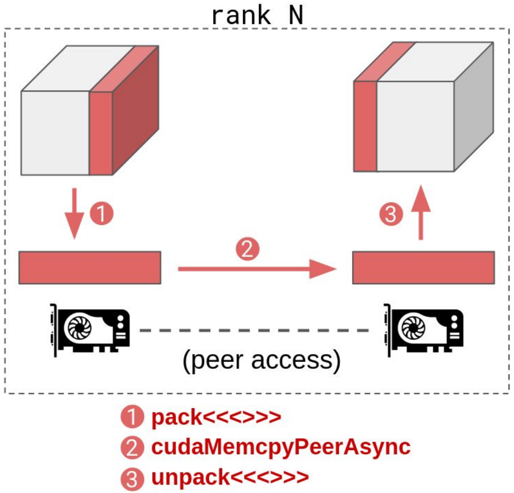
exchange

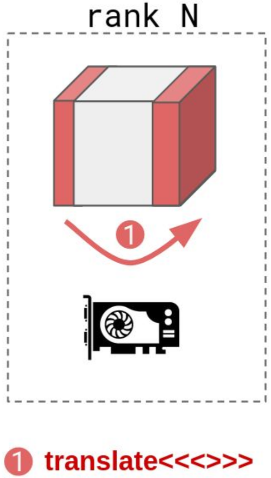
4. pack<<<>>>
5. cudaMemcpyPeerAsync
6. unpack<<<>>>



Exchange between different ranks on the same node
Different ranks are different processes with different address spaces
Use `cudaIpc*` to move a pointer between ranks, then cudaMemcpy*

Peer- and Self-exchange

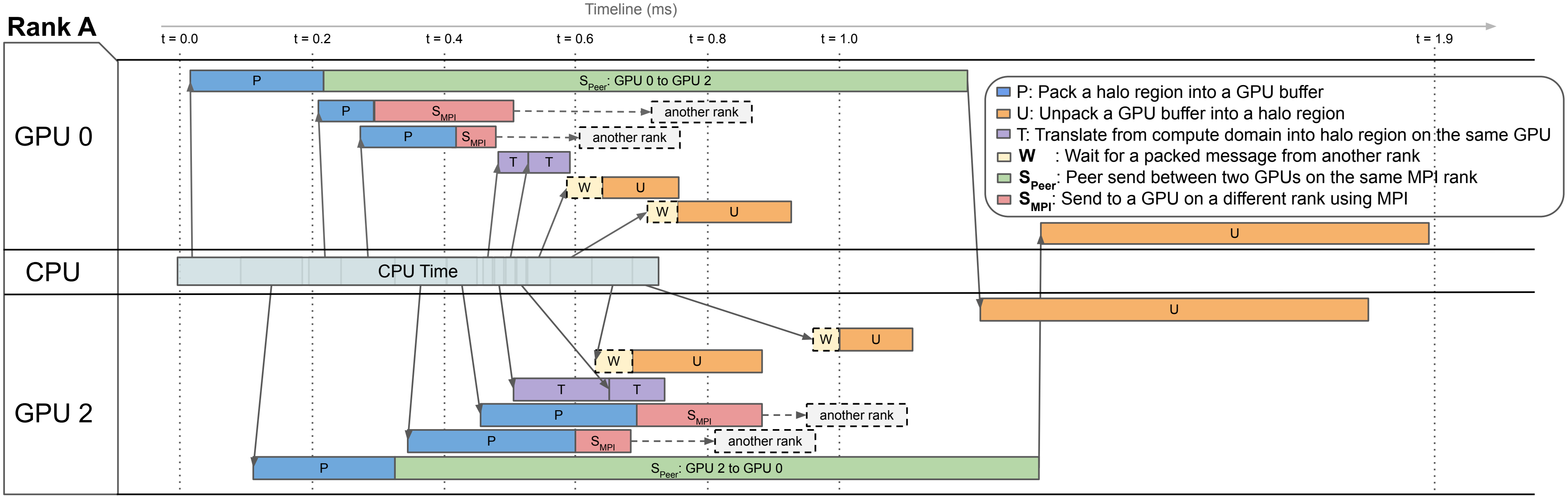




Peer: Two GPUs in the same rank

Self: Same GPU is on both sides of the domain
Only if decomposition has extent=1 in any direction

Overlap

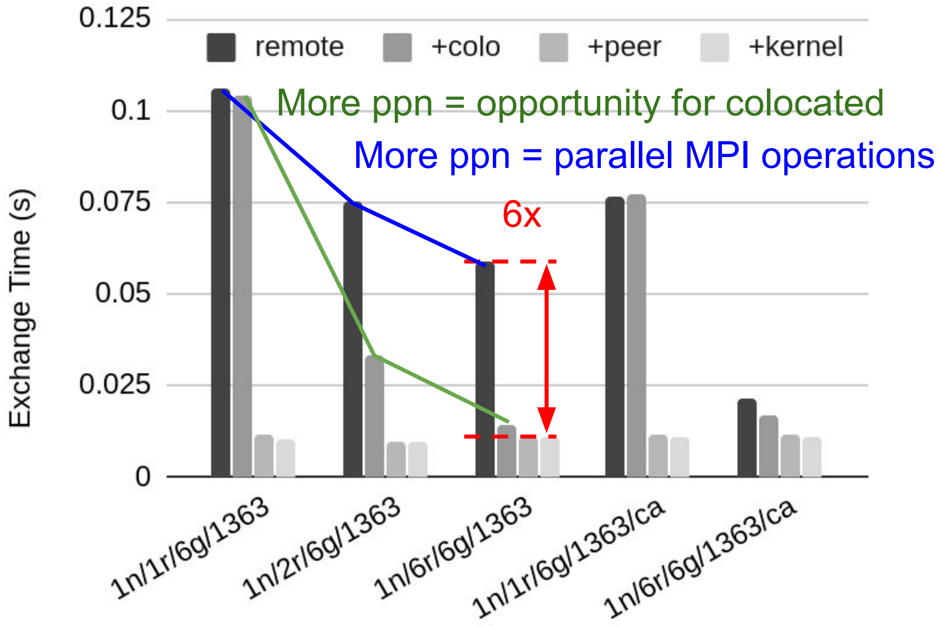


All operations are parallel and asynchronous

May be able to trade off kernel time with communication time by storing halos in a packed configuration

1 Node (Summit)

CPU	OS	Kernel	GPUs	CUDA Driver	MPI	nvcc	cc
22-core POWER9	RHEL 7.6	4.14.0-115.8.1.el7a.ppc64le	V100-SXM2-16GB	418.67	Spectrum 10.3.0.1	10.1.168	g++ 4.8.5



An/Br/Cg/N

A nodes

B ranks per node

C GPUs per node

N: total domain size is N^3

remote: staged or CUDA-Aware only

+colo: “remote” + colocated communicators

+peer: “+colo” + peer communicator

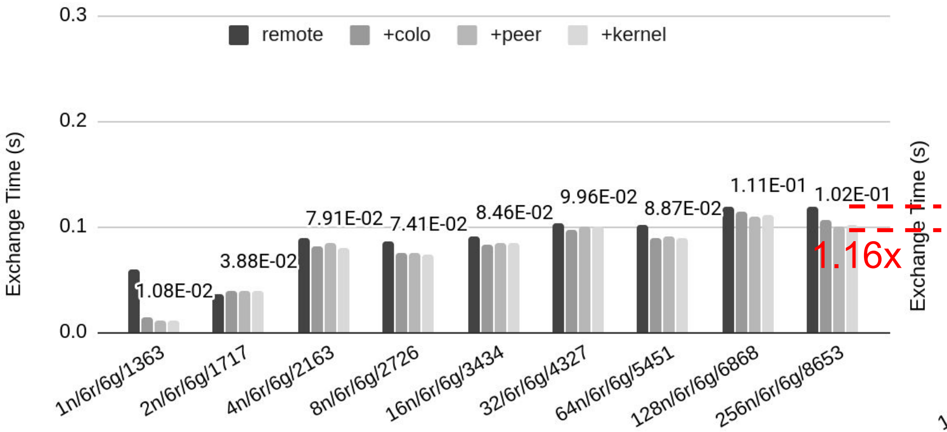
+kernel: “+peer” + self communicator

Specialization has a big impact in intra-node performance

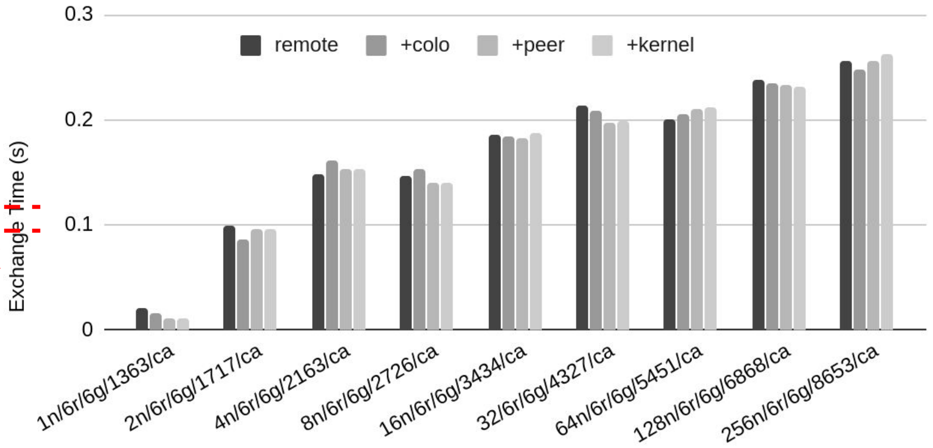
Weak Scaling (Summit)

CPU	OS	Kernel	GPUs	CUDA Driver	MPI	nvcc	cc
22-core POWER9	RHEL 7.6	4.14.0-115.8.1.el7a.ppc64le	V100-SXM2-16GB	418.67	Spectrum 10.3.0.1	10.1.168	g++ 4.8.5

Non-CUDA-aware MPI



CUDA-aware MPI



Exchange time stabilizes once most nodes have 26 neighbors

Specialization has a smaller impact on off-node performance (1.16x at 256 nodes)

CUDA-aware causes poor scaling

Implementation - CUDA/C++ Header-only Library

https://github.com/cwpearson/stencil

Fast stencil exchange for any configuration of CUDA + MPI

Support for any combination of quantity types (float, double)

“Patch-based” approach, for integrating existing GPU kernels

- Still has a few loose ends:
  - Multi-radius stencils (improve communication performance)
  - Export to standard visualization formats
  - Checkpointing
  - Convenience functions for overlapping communication and computation

Takeaways so Far

- Use (at least) one rank per GPU to maximize MPI injection bandwidth
- Data placement was good for 20% performance for one node
- Communication specialization was good for 6x on one node
  - still 1.16x at 256 nodes - allows MPI to just do off-node
- CUDA-Aware MPI seems like a proof-of-concept right now
- Some opportunities to improve partitioning and placement according to node topology
- May be able to trade off kernel time with communication time by storing halos in a packed configuration

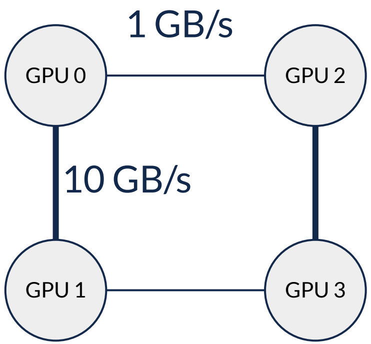
Future Directions

Assumed minimizing communication volume would maximize communication performance

- Do all transfer directions have equal bandwidth?
- Do all transfers have equal cost?

Example Node-Aware Partition

Minimal communication is not maximum performance



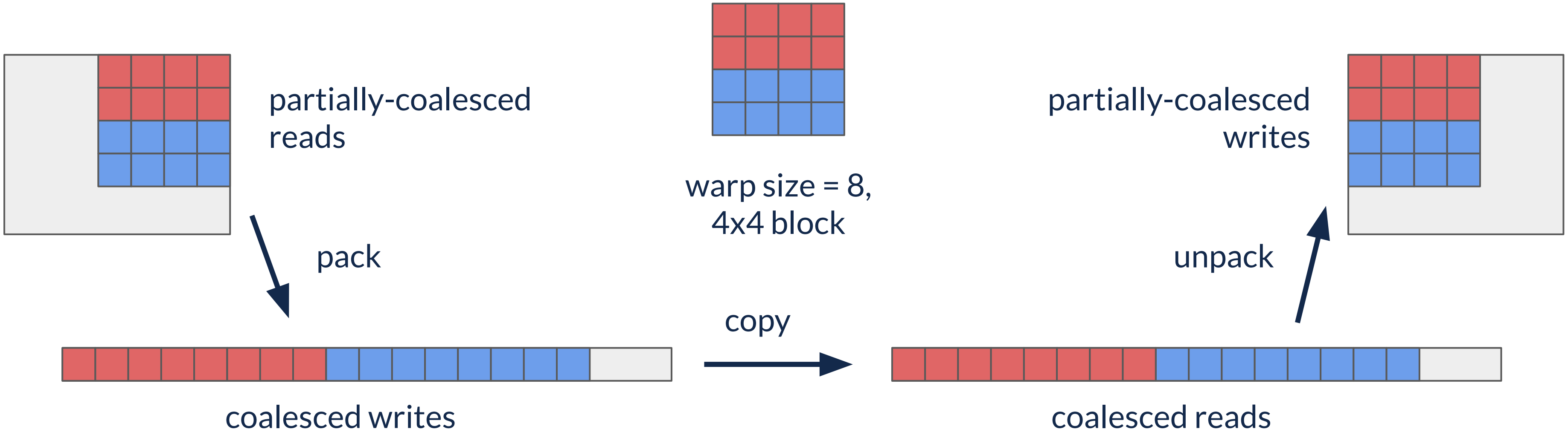
Hypothetical Node

	$4n$	$4n$
Partition	 n	 n
Time	n / 1GB/s	$2n$ / 10 GB/s
Comm.	$4n$	$5n$

All Pack Directions not Equal

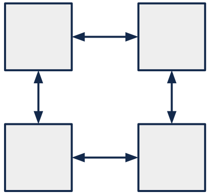
Not all communication directions have same performance on same link.

Pack / Unpack performance depends on strides

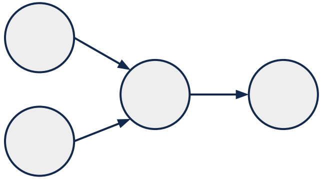


unpack is 2-3x slower than pack for non-contiguous regions

Future Directions



System Graph
vertices: PEs
edges: interconnects

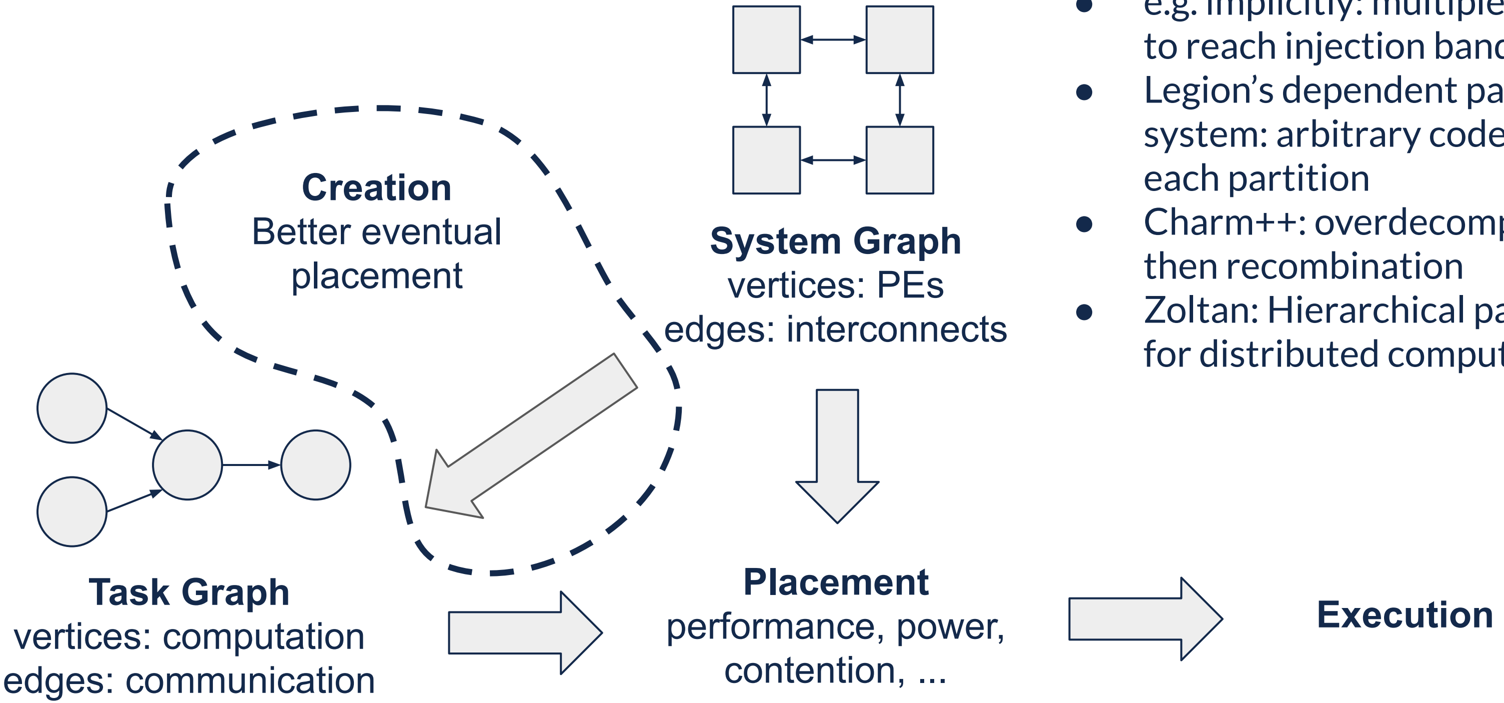


Task Graph
vertices: computation
edges: communication

Placement
performance, power,
contention, ...

Execution

Future Directions



- e.g. implicitly: multiple MPI ranks to reach injection bandwidth limit
- Legion's dependent partitioning system: arbitrary code to color each partition
- Charm++: overdecomposition and then recombination
- Zoltan: Hierarchical partitioning for distributed computing

Conclusion

- Careful measurement as a foundation for performance
- Examining the impact of heterogeneous communication performance
- Making successful approaches available through a library
- Algorithm-level communication performance is impacted by the system
    - Generalize to other applications?
    - Integrate with an existing task/placement/execution system

Thank you - Carl Pearson

Ph.D. student, Electrical and Computer Engineering, University of Illinois Urbana-Champaign

- (Multi-)GPU communication
- Accelerating irregular applications

cwpearson
cwpearson
pearson at illinois.edu
https://cwpearson.github.io

Extra Slides

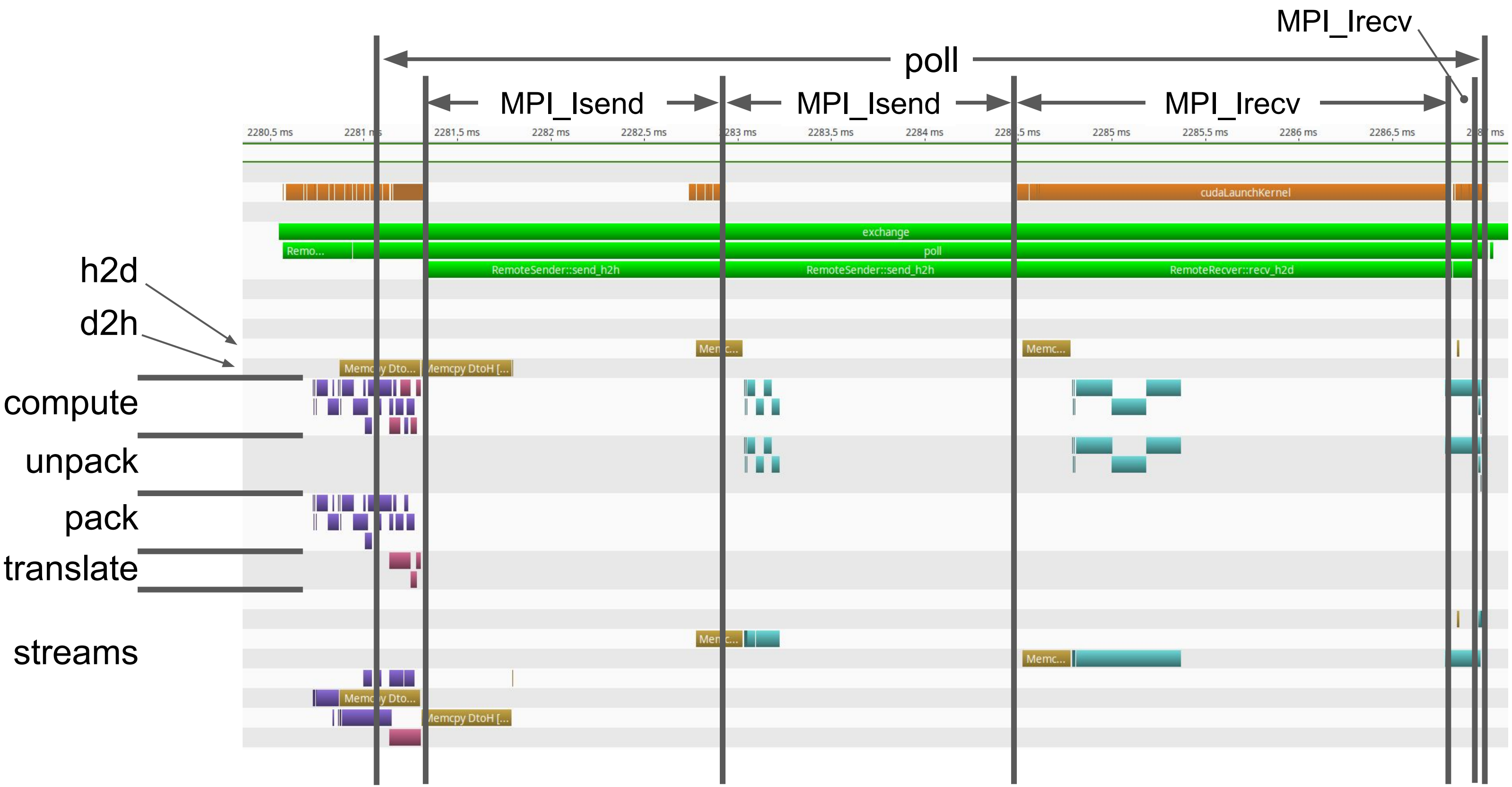
	pack	unpack
Issued Ld/St	393216	393216
L2 Transactions (Texture Reads)	327840	98464
L2 Transactions (Texture Writes)	98304	327680
Issue Stall (Mem Throttle)	0.3%	43.6%
Global Load Transactions	393216	163840
Global Store Transactions	98304	327680
L2 Read Transactions	327936	98560
L2 Write Transactions	98337	583340
Dev, Mem. Read Transactions	589836	415028
Dev. Mem. Write Transactions	171218	405348
Global Load Throughput (GB/s)	238.841	34.474
Global Store Throughput (GB/s)	59.71	68.949

Future Work: Store Halos Separately

Pros: no more packing and unpacking

Const: smart-pointer in cuda kernel to redirect accesses to the right buffer

Requires evaluation on real kernels

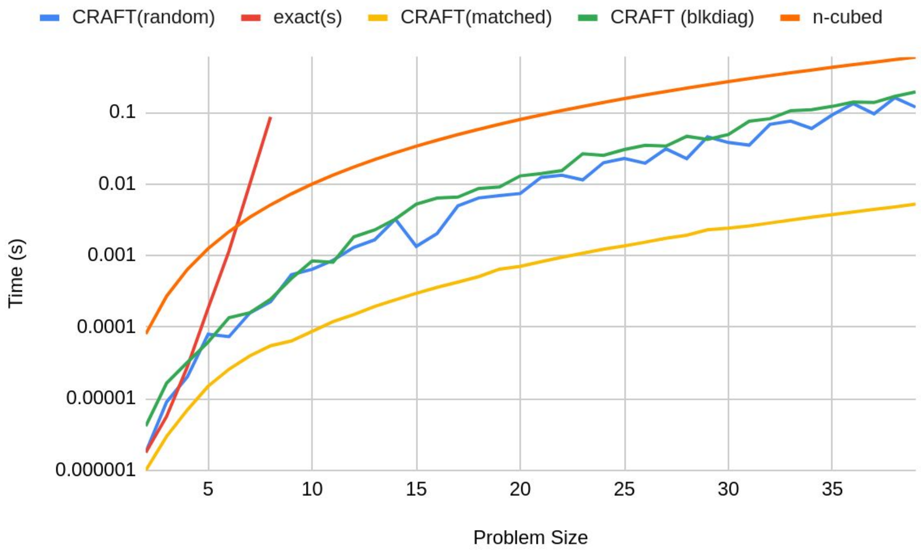


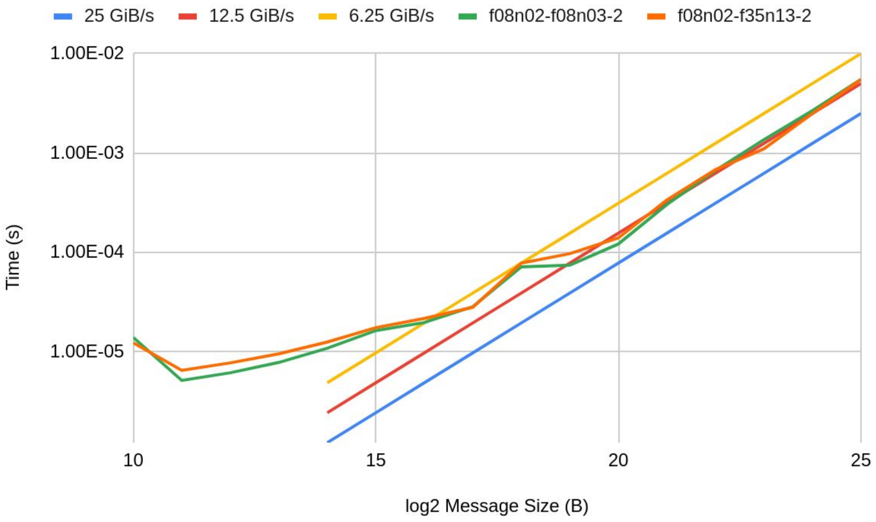
css-host-yz-20, 4 ranks, 1 GPU / rank, 71ff24, driver 440.33.01, CUDA 10.2, Ubuntu 18.04, kernel 4.14.0-74-generic, timeline_28038.nvvp

Future Work: Topology-Aware Placement

Extent QAP to n ~ 1k: need a better placement algorithm, SCOTCH or something?

No measurable locality on summit

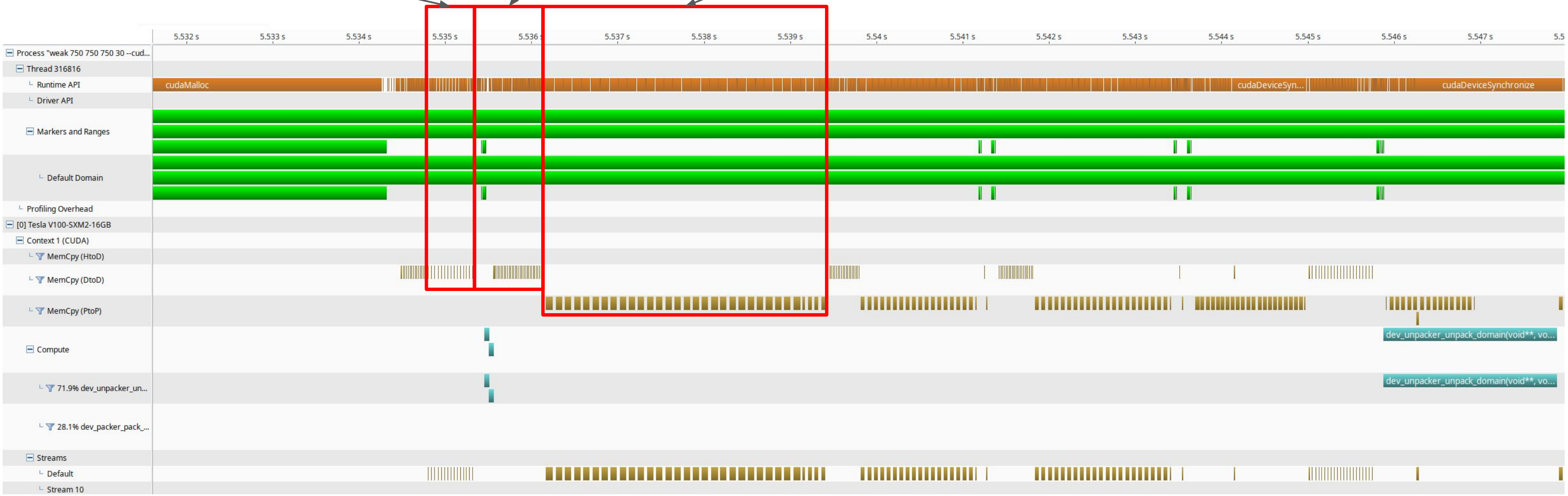




cudaMemcpyPeer same device

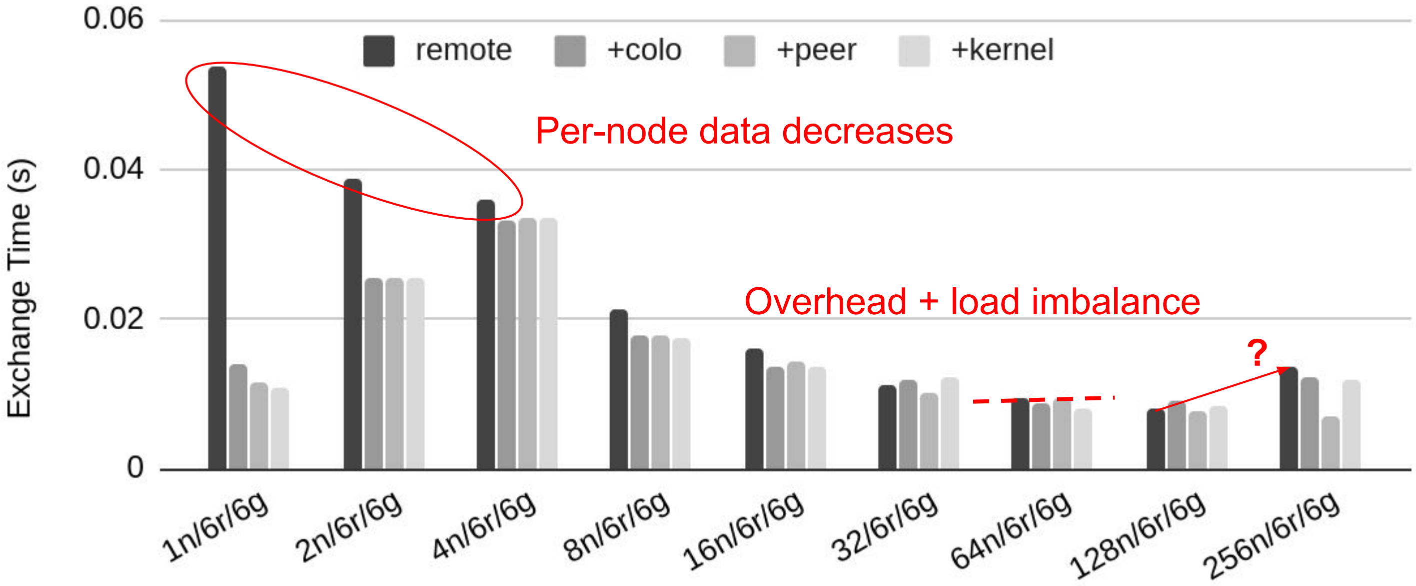
cudaMemcpyPeerAync same device

cudaMemcpyPeer between devices followed by cudaDeviceSynchronize

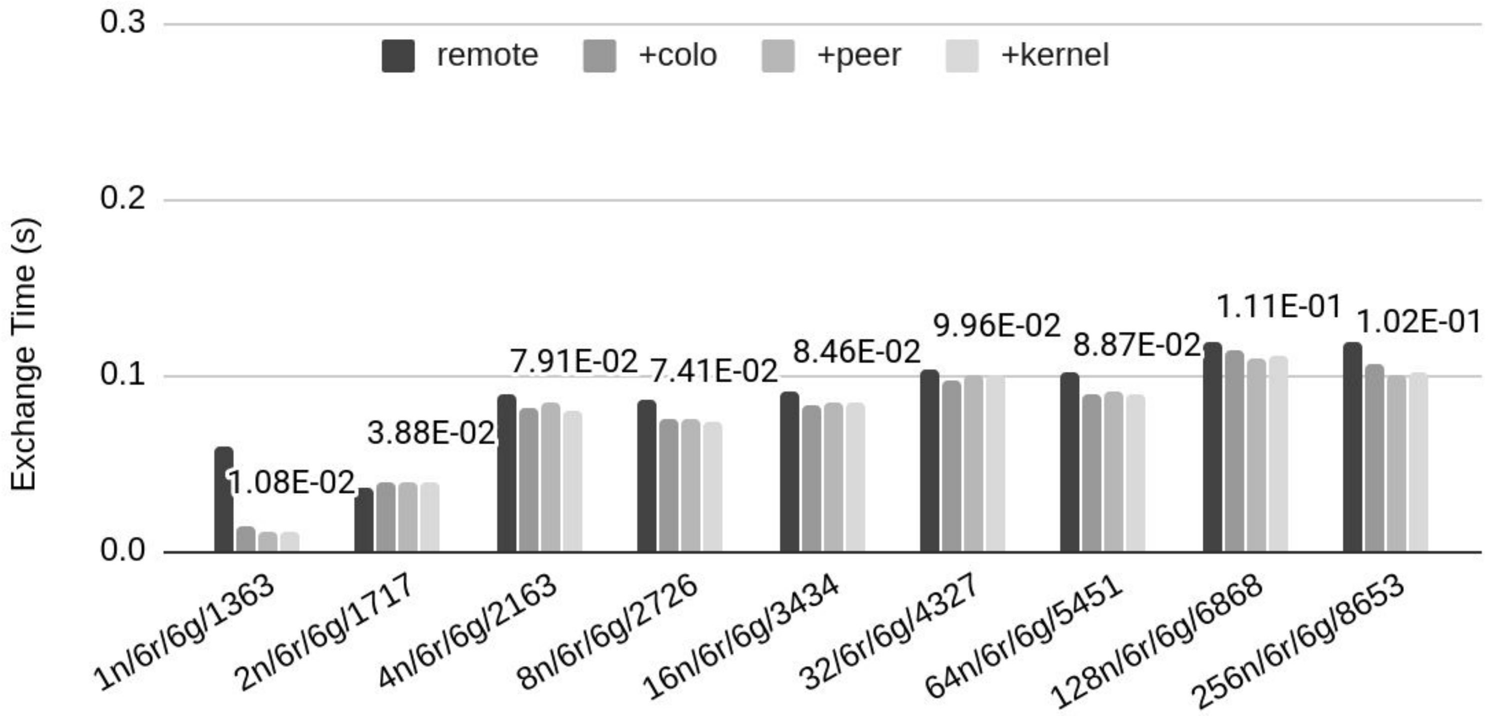


Spectrum MPI 10.3.0.1 puts many device-device copies in default stream, and also calls cudaDeviceSynchronize(), which synchronizes other asynchronous operations

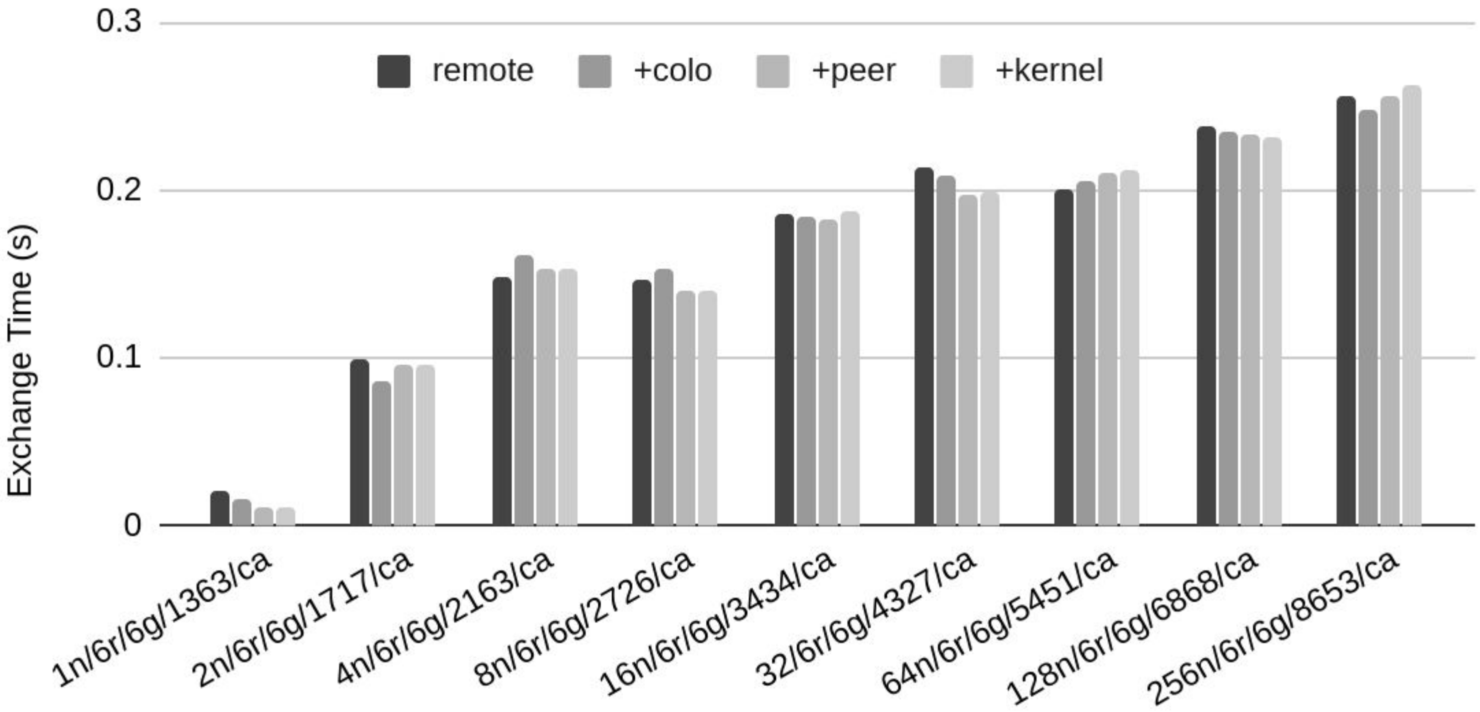
Strong Scaling: 1363^3



Weak Scaling (Summit) - Detail

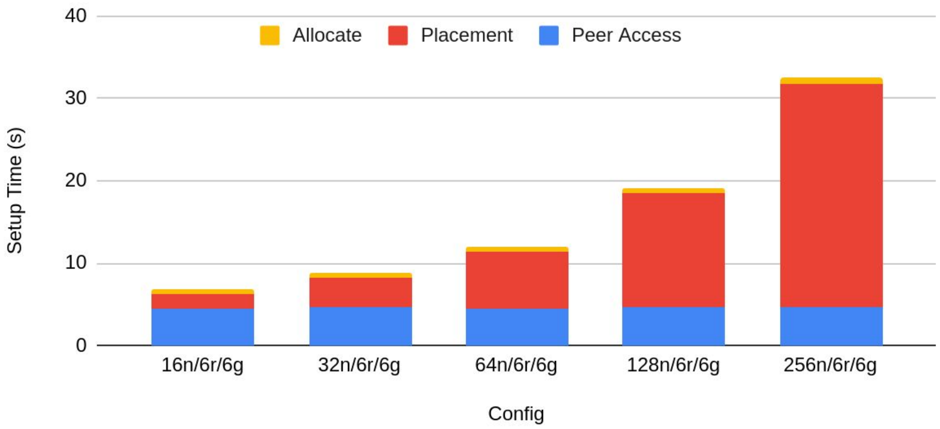


Weak Scaling (Summit) - CUDA-Aware Detail



Future Work: Placement Performance

- Naive implementation right now
- Same placement on all nodes -> only do it once, no need to broadcast full placement information



Future Work: Library Performance

Measure inter-node and intra-node tiny messages

Represents overhead

Future Work: Bandwidth Measurements

- CUDA-Aware MPI Performance
- MPI Performance
  - On-node vs off-node
- Can’t rely on specs to get actual bandwidth
- Use these instead distance for placement?

Future Work: Further Reduce MPI messages

Consolidate all messages to a remote node into a single buffer

Pros: fewer, larger MPI messages

Cons: Incurs intra-node messaging and synchronization overhead

Future Work: System-level heterogeneity

Whether in compute performance and communication contention

Could apply a similar placement scheme, but use ^ as inputs

Overlap with dynamic load balancing techniques?

Solving QAP

Allocating Facilities with CRAFT. Buffa, Armour, Vollman. 1962.

Start with some initial placement
while true:
  Check all possible location swaps
  Choose swap that lowers cost the most
  if no better swap:
    break
n^3 for n facilities (n swaps for n locations, roughly n iterations)
key to not recompute cost each time - each swap only changes a bit of the cost
matches exact solution for $n < 6$ in our case

Abstract

High-performance distributed computing systems increasingly feature nodes that have multiple CPU sockets and multiple GPUs. The communication bandwidth between those components depends on the underlying hardware and system software. Consequently, the bandwidth between these components is non-uniform, and these systems can expose different communication capabilities between these components. Optimally using these capabilities is challenging and essential consideration on emerging architectures. This talk starts by describing the performance of different CPU-GPU and GPU-GPU communication methods on nodes with high-bandwidth NVLink interconnects. This foundation is then used for domain partitioning, data placement, and communication planning in a CUDA+MPI 3D stencil halo exchange library.