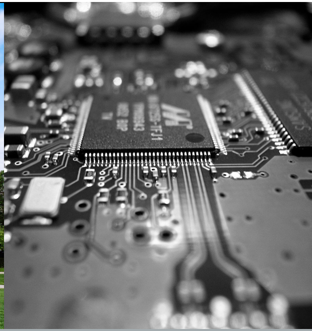
Node-Aware Stencil Communication for Heterogeneous Supercomputers

Feb 28 2020

Carl Pearson[1], Mert Hidayetoglu[1], Mohammad Almasri[1], Omer Anjum[1], I-Hsin Chung[2], Jinjun Xiong[2], Wen-Mei Hwu[1]

[1] Electrical and Computer Engineering, University of Illinois Urbana-Champaign
[2] IBM T.J. Watson Research, Yorktown Heights, NY



ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

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

Outline

- Motivation
- Stencils
- Decomposition
- Placement
- Specialization
- Results
- Future Directions

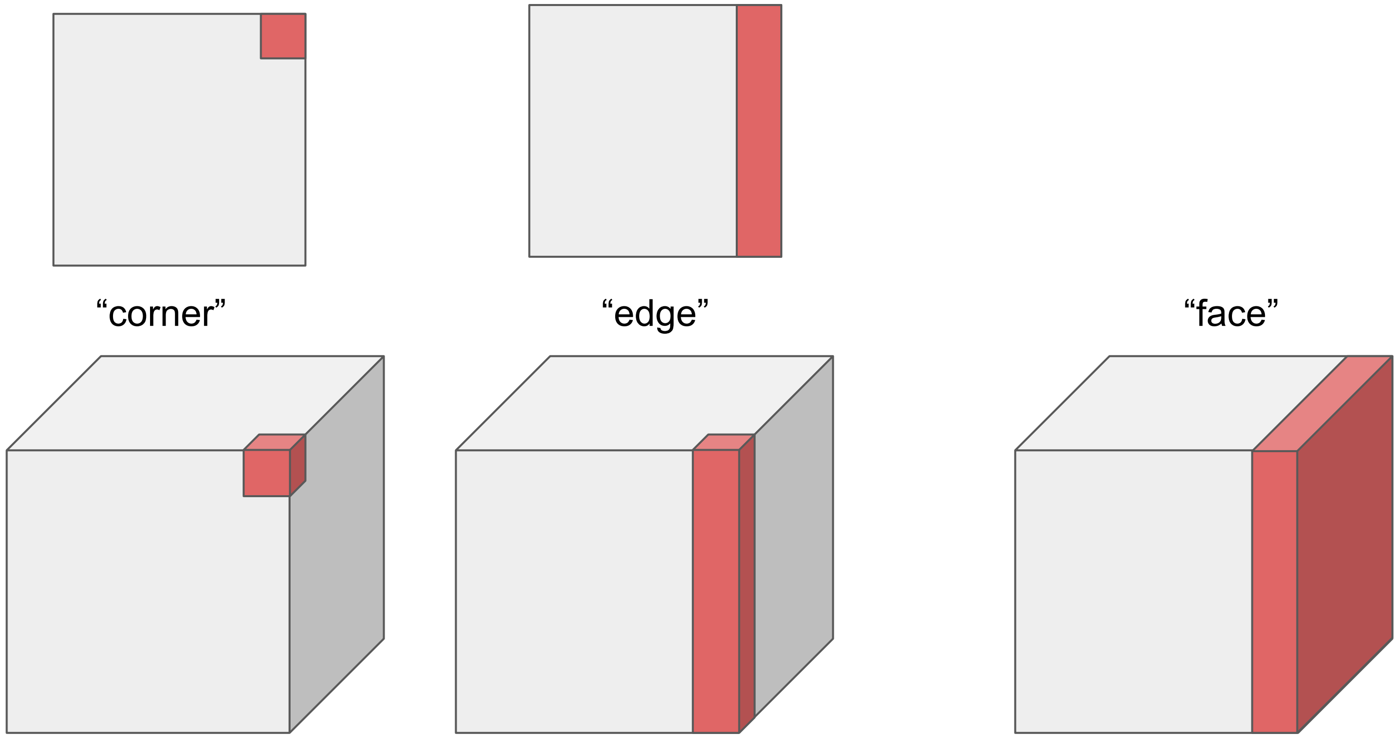
In submission: 2020 International Workshop on Automatic Performance Tuning (iWAPT)

Motivation

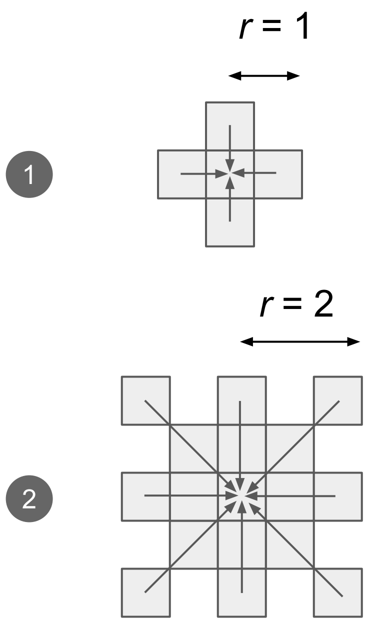
- Regular computation, access, and structure reuse ➡️ stencil on GPU
- High-resolution modeling ➡️ Large stencils
- Limited GPU memory ➡️ distributed stencils with communication
- Fast stencil codes ➡️ larger impact of communication
- Heterogeneous nodes (“fat nodes”) ➡️ how to do communication

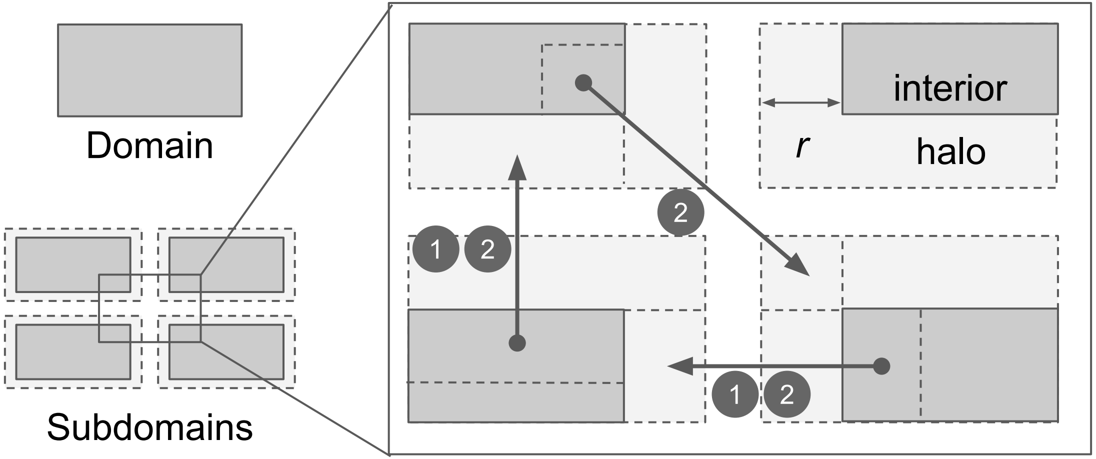
- Contributions:
  - A three-phase solution for optimized stencil communication on heterogeneous clusters
  - Capability-based communication specialization
  - Runtime node-aware data placement

Glossary



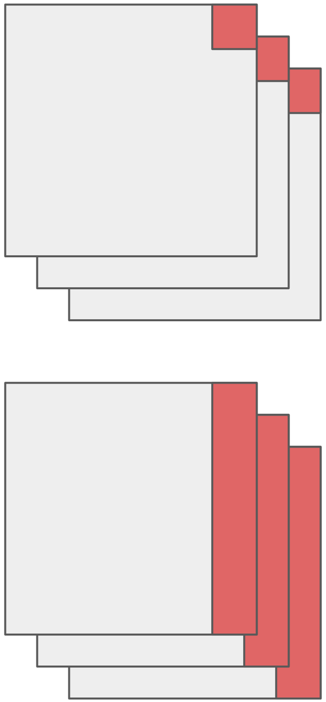
Stencil Overview





Required halo exchange depends on stencil complexity

Glossary

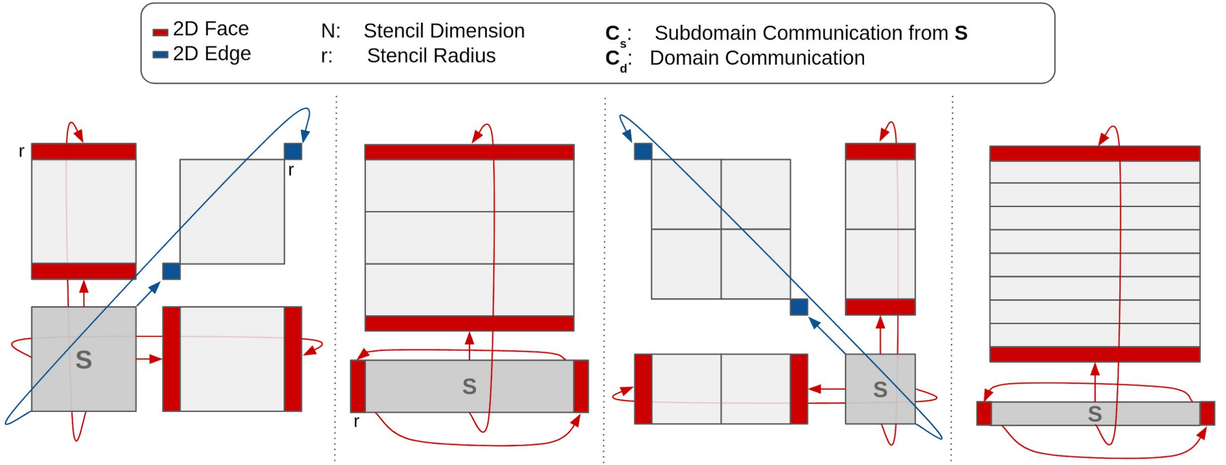


- Typically more than one quantity
  - problem-dependent
  - physical properties (pressure, temperature)
  - directional derivatives
- Each quantity's halo is exchanged with corresponding quantity in other subdomains

Intuition

- Off-node communication is expensive ➡️ minimize required, maximize injection bandwidth
  - “hierarchical decomposition”
  - multiple ranks per node
- On-node communication hardware ➡️ assign subdomains to GPUs to maximize use of bandwidth
  - “node-aware placement”
- On-node bandwidth depends on communication method ➡️ use best method to achieve hardware bandwidth
  - “capability-based specialization”
  - parallel, asynchronous exchanges

Decomposition - Minimize Required Comm.



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

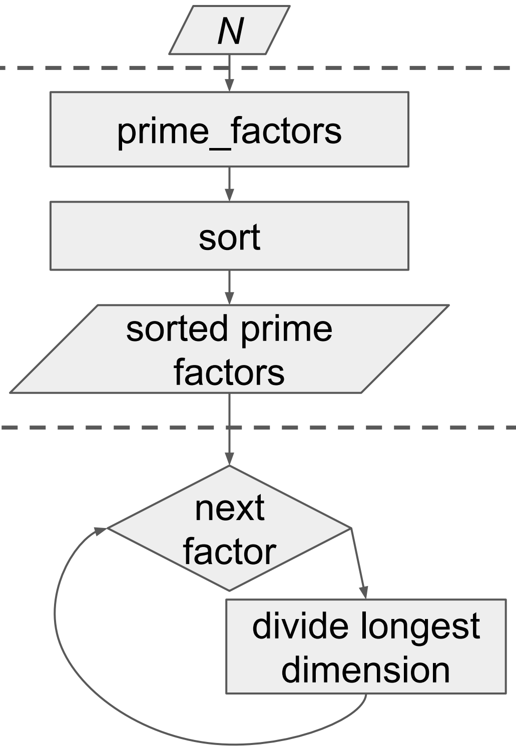
Partition: 4x1
S size = **N/4** x **N**
$\mathbf{C_s} = 2rN + 2rN/4$
$\mathbf{C_d} = 4\mathbf{C_s} = 10Nr$

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 - Approach

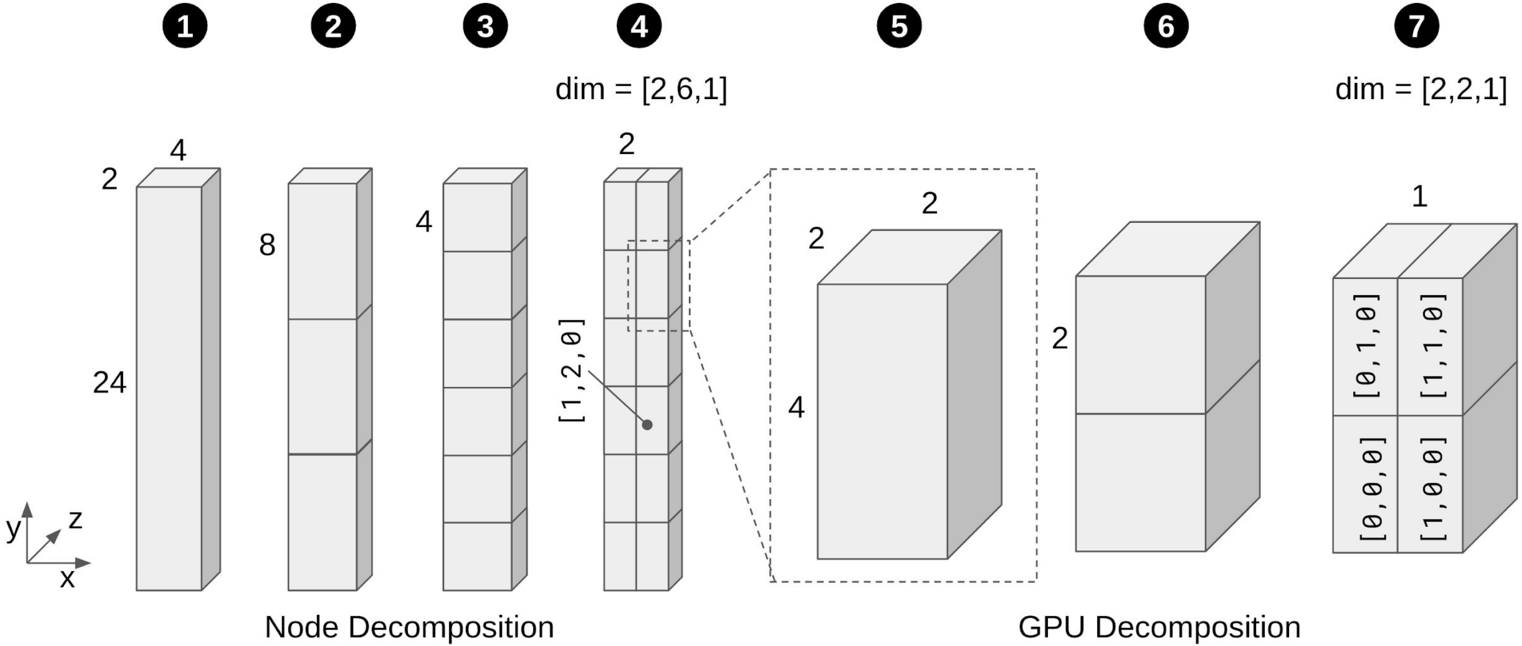


- Divide into *n* subdomains

- 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 N

- Divide the longest dimension by prime factors
  - use smaller prime factors later to clean up

Hierarchical Decomposition



Minimize Communication Out of Node

Minimize communication between GPUs

Communication In Fat Nodes

Different Bandwidths between GPUs

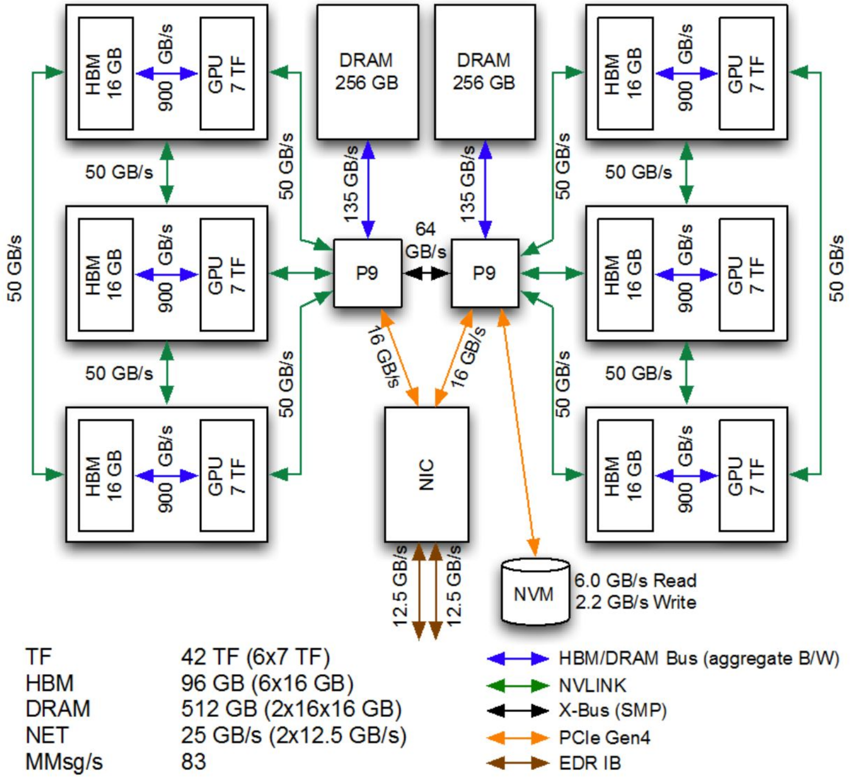
Not the same as theoretical [1]

X-bus: achieved **30 GB/s** unidirectional

NVLink: achieved **42 GB/s** unidirectional

NIC: achieved **12.5 GB/s** unidirectional

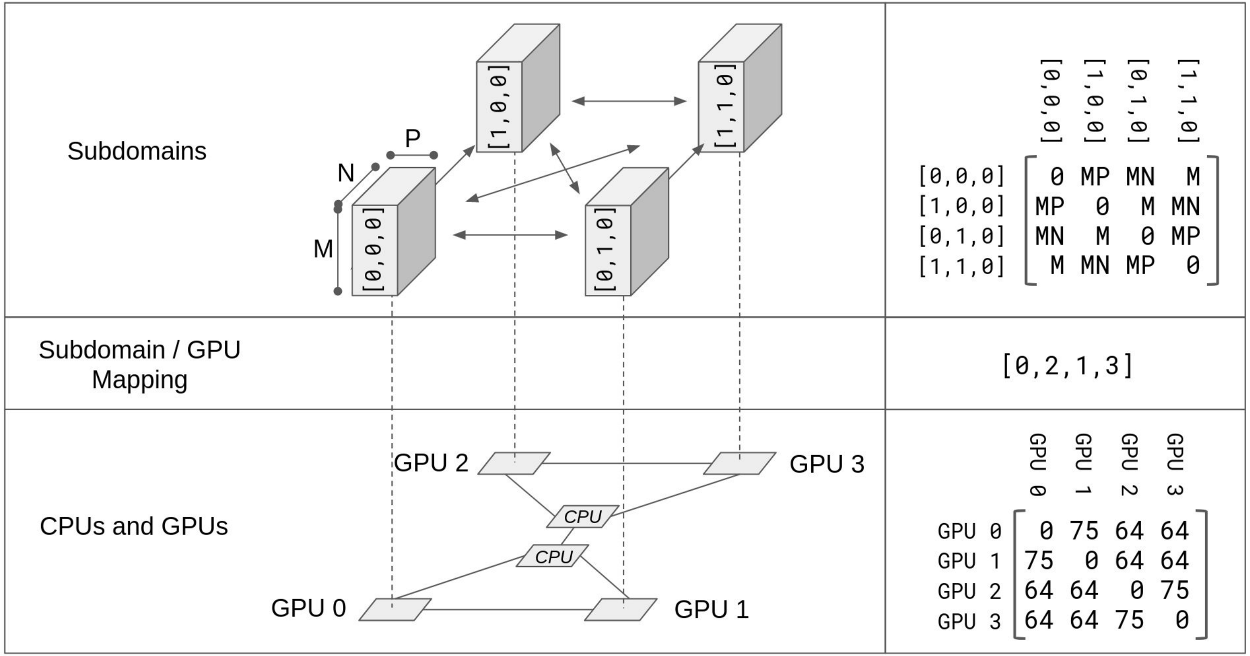
Neighbor GPUs have higher bandwidth



[1] Pearson et al. *Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects.* ACM/SPEC International Conference on Performance Engineering. 2019.

Placement

How to place subdomains on GPUs to maximize bandwidth utilization?



Quadratic Assignment Problem

n facilities and n locations

w: weight matrix: $w_{i,j}$ amount of "flow" between *i* and *j*.

d: distance matrix: distance between *i* and *j*

f: bijection n -> n "assignment" of facilities to location

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

Minimize cost function: sum of products of weights and distances under *f*.

GPUs: locations

subdomains: facilities

w: required communication

d: GPU bandwidth

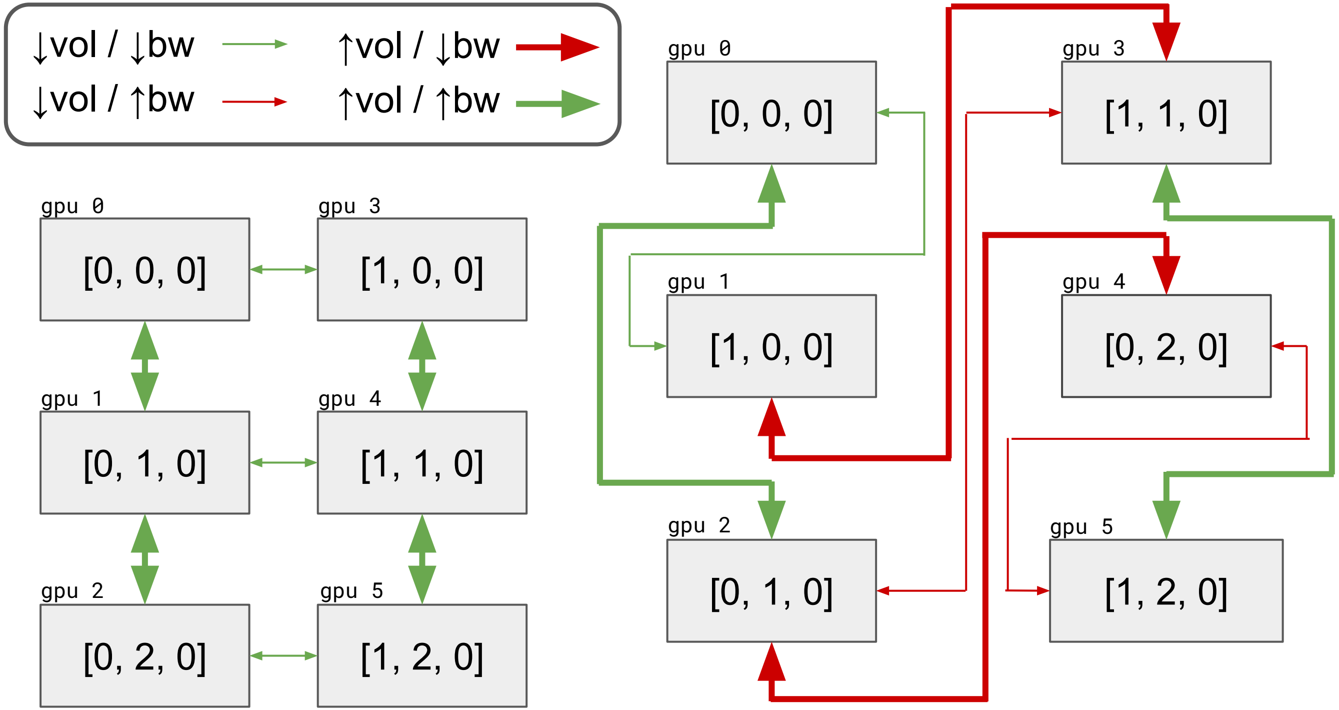
f: assignment of subdomains to GPUs

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

Example Placement



Node-Aware Placement

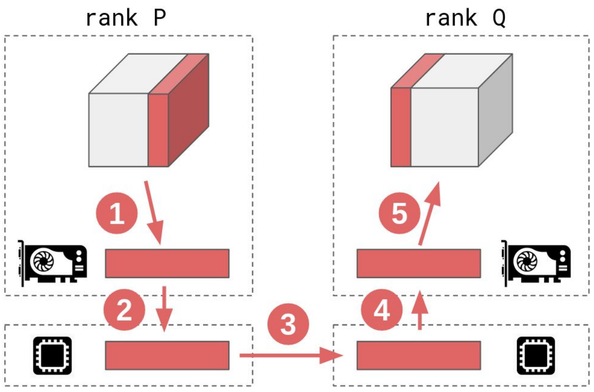
Trivial Placement

Capability Specialization

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

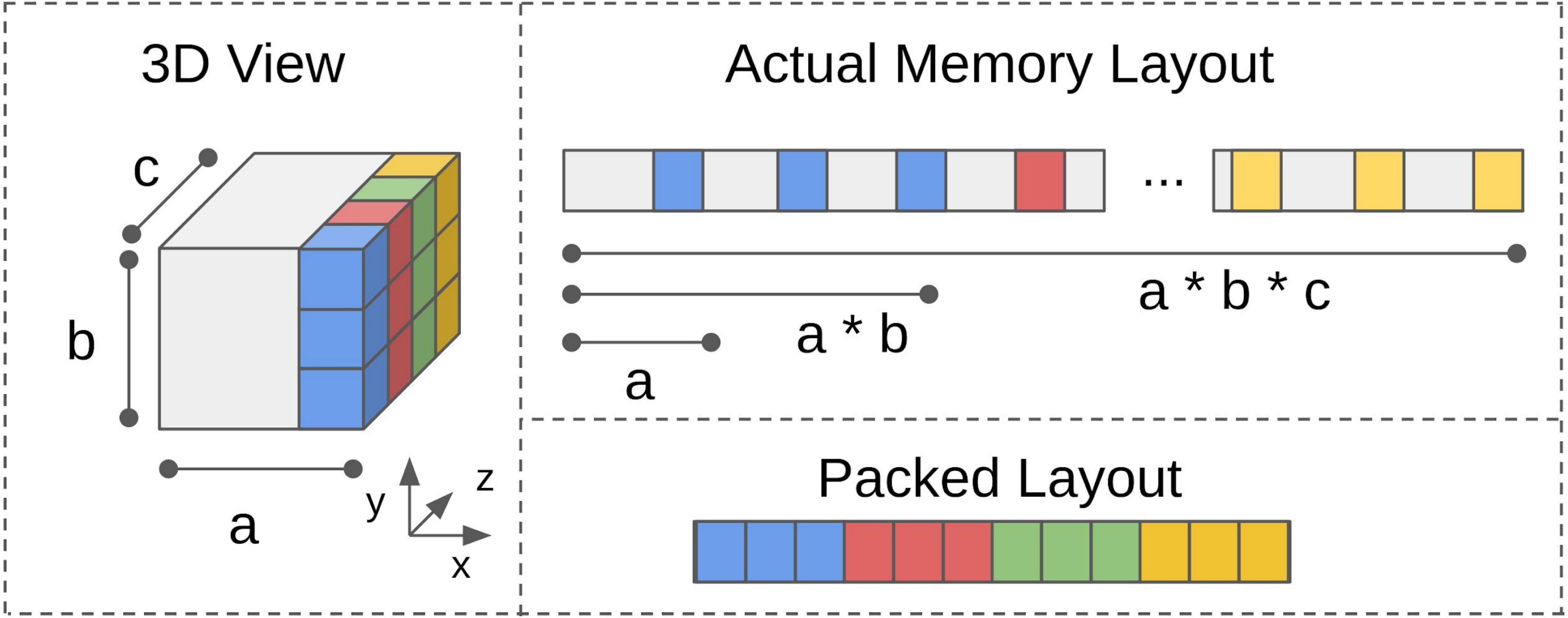
- “Staged”: works for any 2 GPUs anywhere
  - pack from device 3D region into device 1D buffer
  - copy from device 1D buffer to host 1D buffer
  - MPI_Send to other host 1D buffer
  - copy from host 1D buffer to device 1D buffer
  - unpack from device 1D buffer to device 3D buffer

Optimizations are node-aware shortcuts on top of this



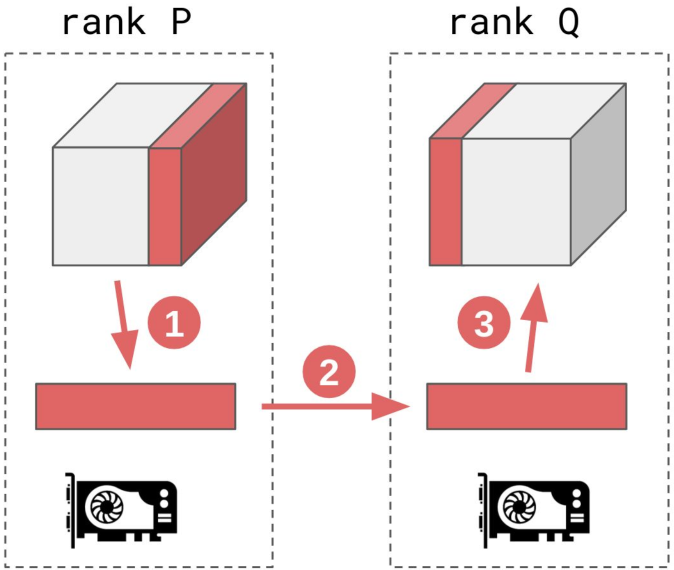
1 pack<<<>>>
2 cudaMemcpy
3 MPI_Isend / MPI_Irecv
4 cudaMemcpy
5 unpack<<<>>>

Pack and Unpack



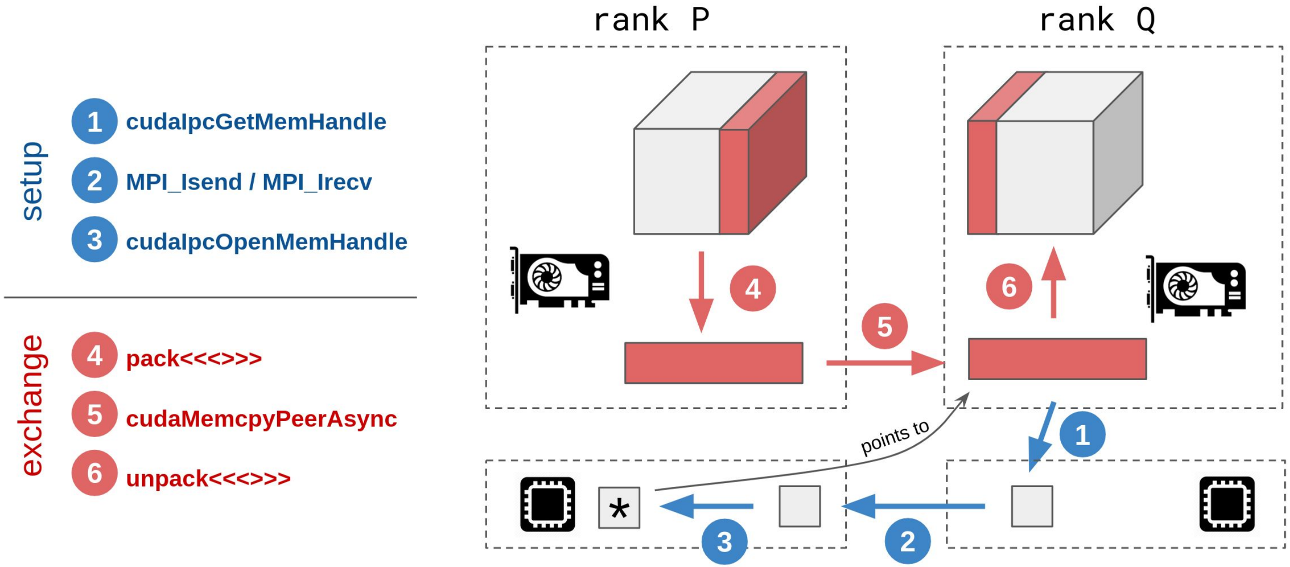
CUDA-Aware MPI

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



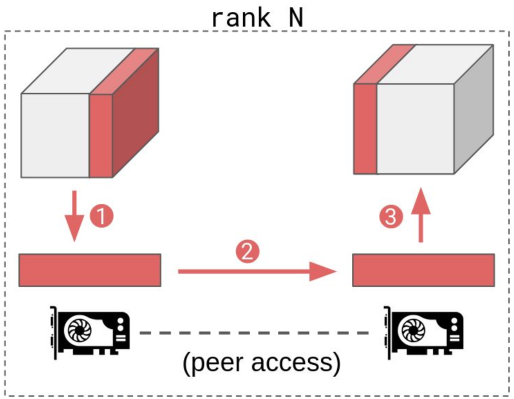
Same as the staged, but MPI responsible for getting data between GPUs

Colocated

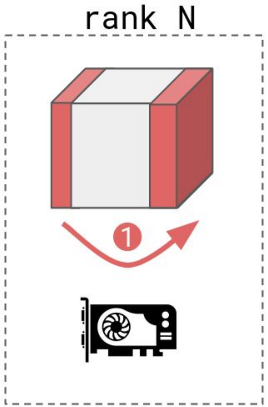


Exchange between different ranks on the same node
Different ranks are different processes with different address spaces
Use `cudaIpc*` to move a pointer between ranks, then `cudaMemcpy*`

Peer- and Self-exchange



1 pack<<<>>>
2 cudaMemcpyPeerAsync
3 unpack<<<>>>

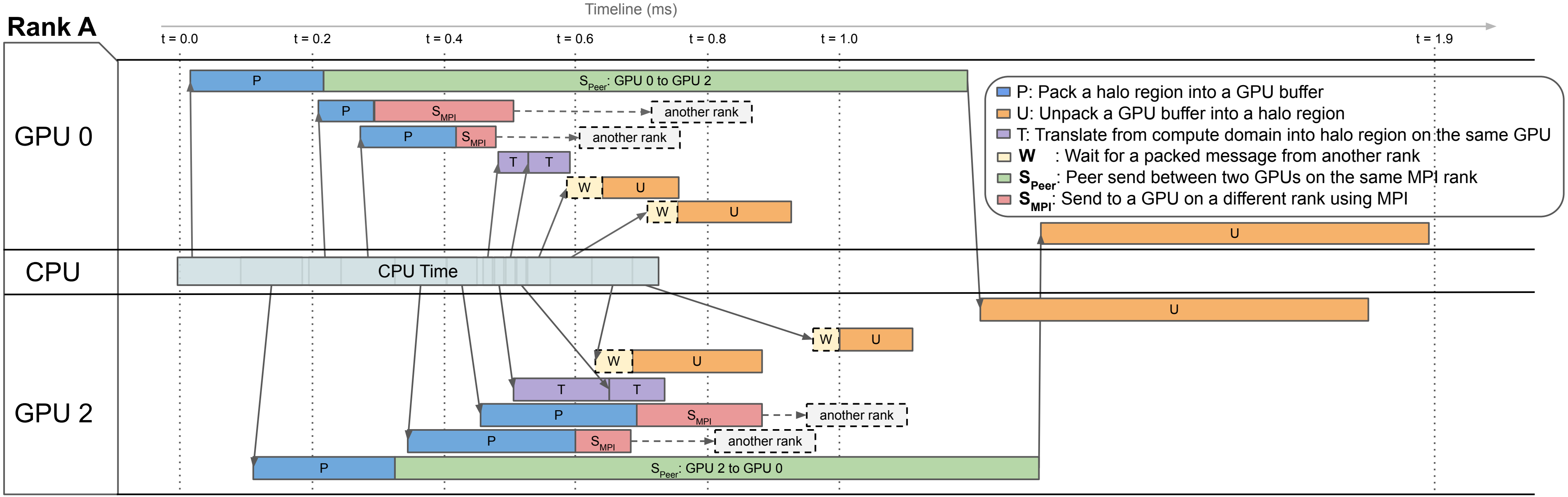


1 translate<<<>>>

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

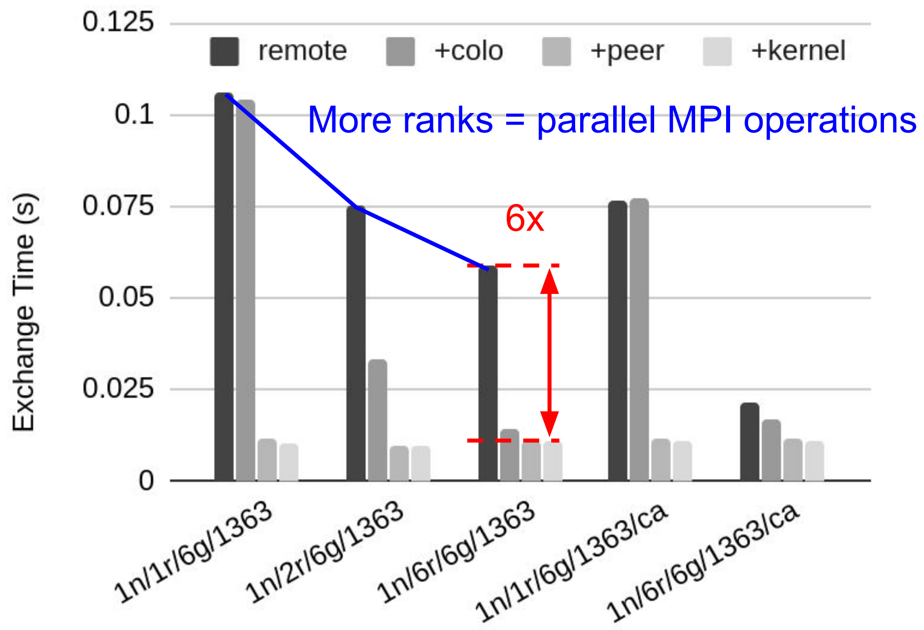
Overlap



All operations are parallel and asynchronous

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

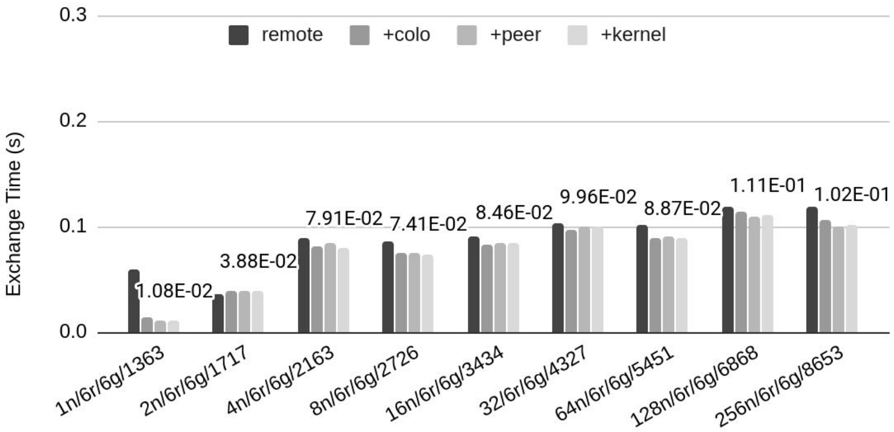
+kernel: “+peer” + self communicator

Specialization has a big impact in intra-node performance

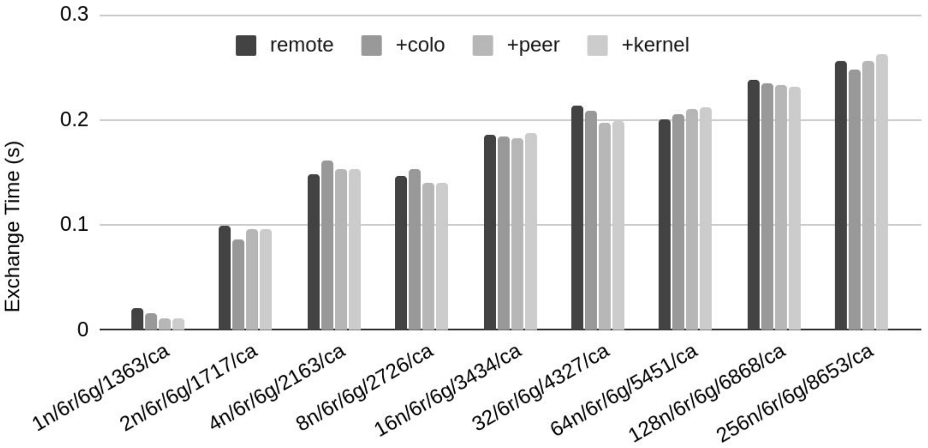
Weak Scaling (Summit)

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



CUDA-aware

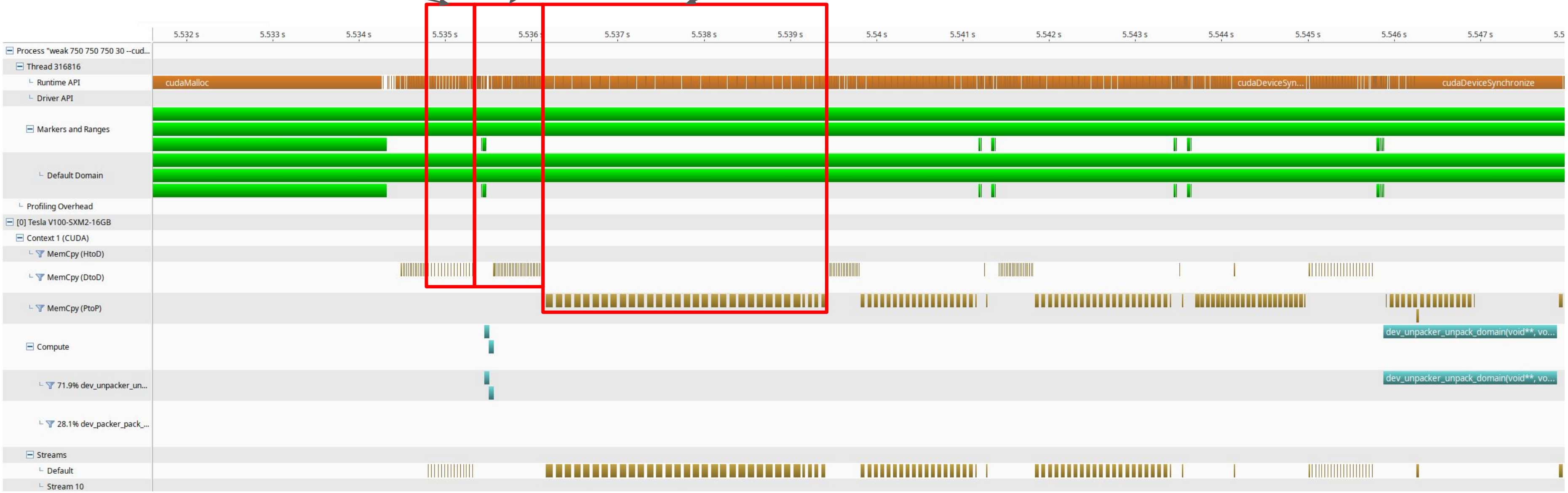


Exchange time stabilizes once most nodes have 26 neighbors
Specialization has a smaller impact on off-node performance (1.16x at 256 nodes)
CUDA-aware causes poor scaling

cudaMemcpyPeer
same device

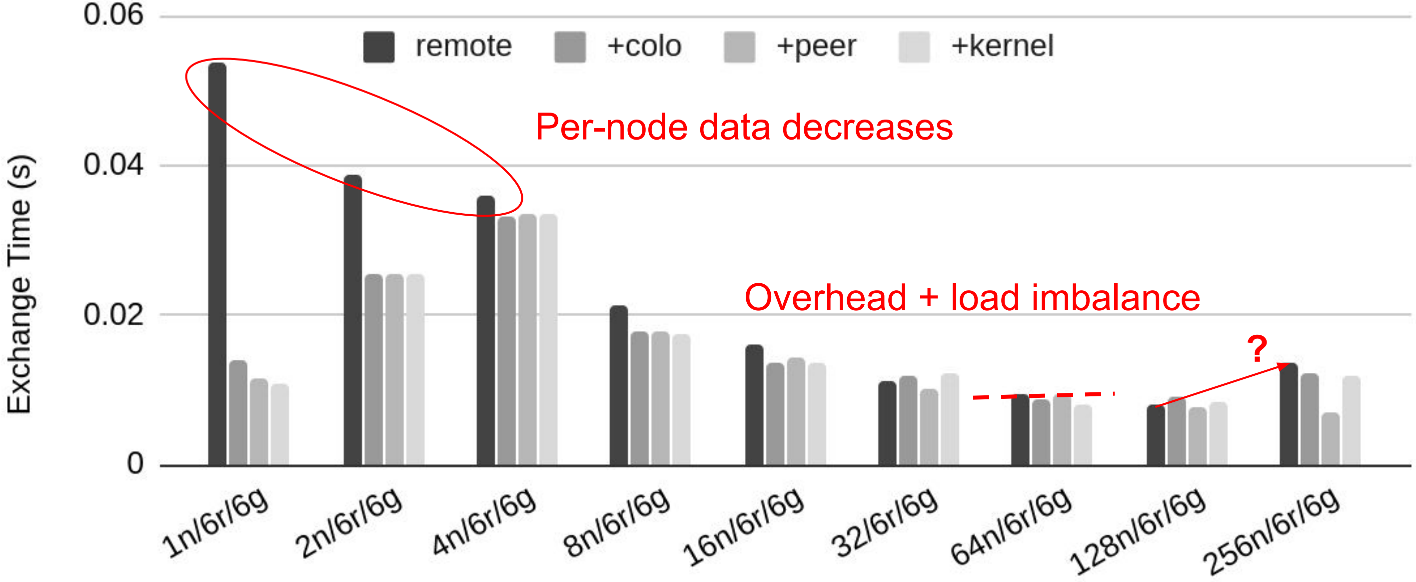
cudaMemcpyPeerAync
same device

cudaMemcpyPeer
between devices
followed by cudaDeviceSynchronize



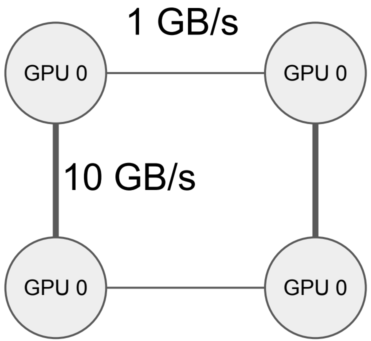
Spectrum MPI 10.3.0.1 puts many device-device copies in default stream, and also calls cudaDeviceSynchronize(), which synchronizes other asynchronous operations

Strong Scaling: 1363^3

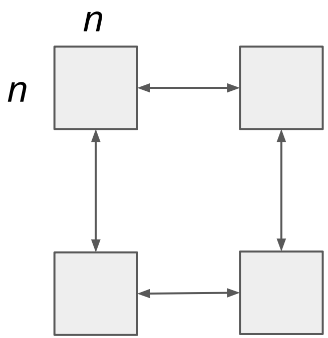


Future Work: Adjust Partition by Bandwidth

Minimal surface area for subdomain is not optimal

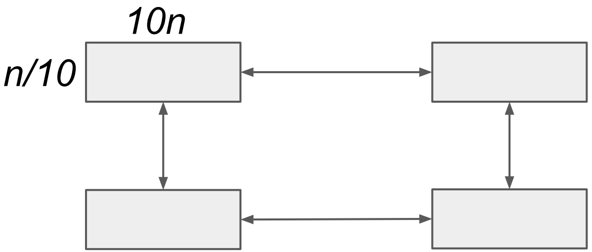


Hypothetical Node



Square Subdomains

max(n / 10GB/s, n / 1GB/s)
=
n / 1 GB/s

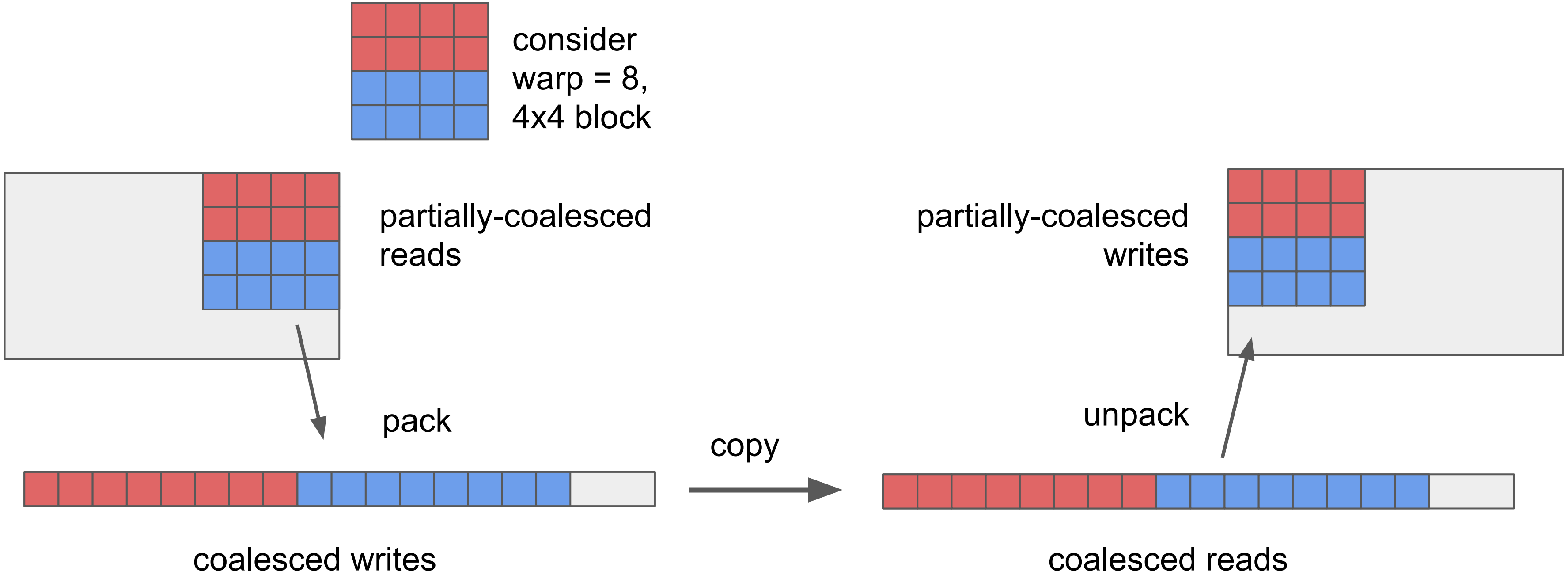


Stretched Subdomains

max(10n / 10GB/s, (n/10) / 1GB/s)
=
n / 10 GB/s

Future Work: All Pack Directions not Equal

Pack / Unpack performance depends on strides

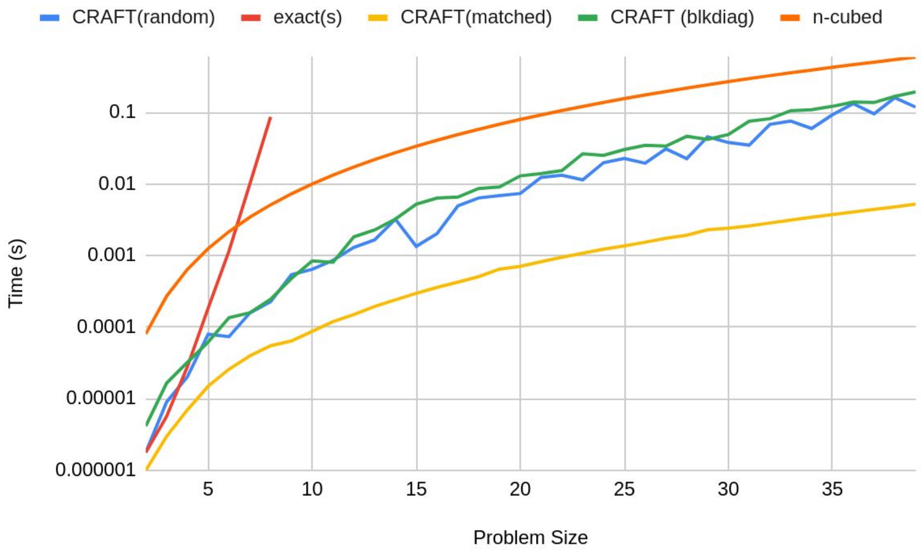


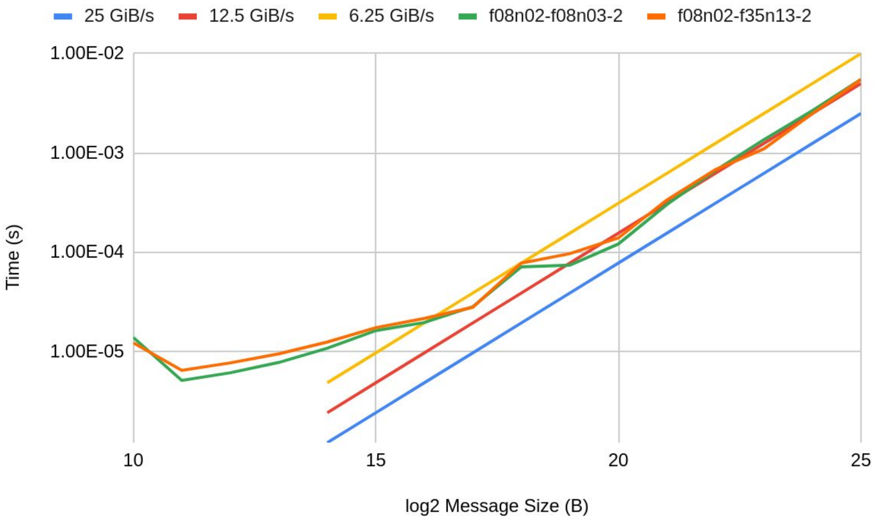
unpack is 2-3x slower than pack

Future Work: Topology-Aware Placement

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

No measurable locality on summit



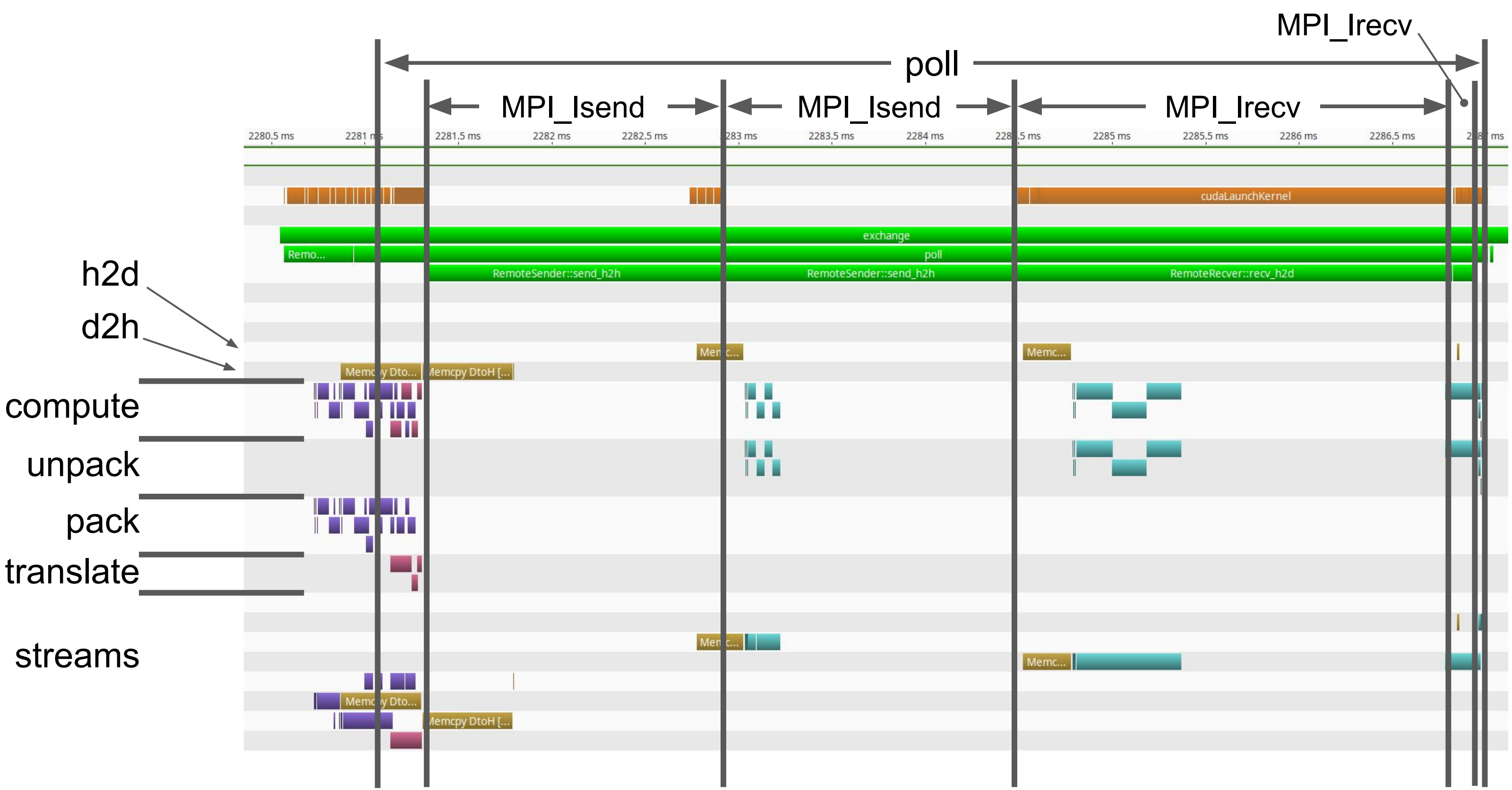


Future Work: Store Halos Separately

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

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

Implementation - CUDA/C++ Header-only Library

https://github.com/cwpearson/stencil - not quite public yet

Fast stencil exchange for any configuration of CUDA + MPI

Tested on Summit and Hal

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

- 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

Thank you - Carl Pearson

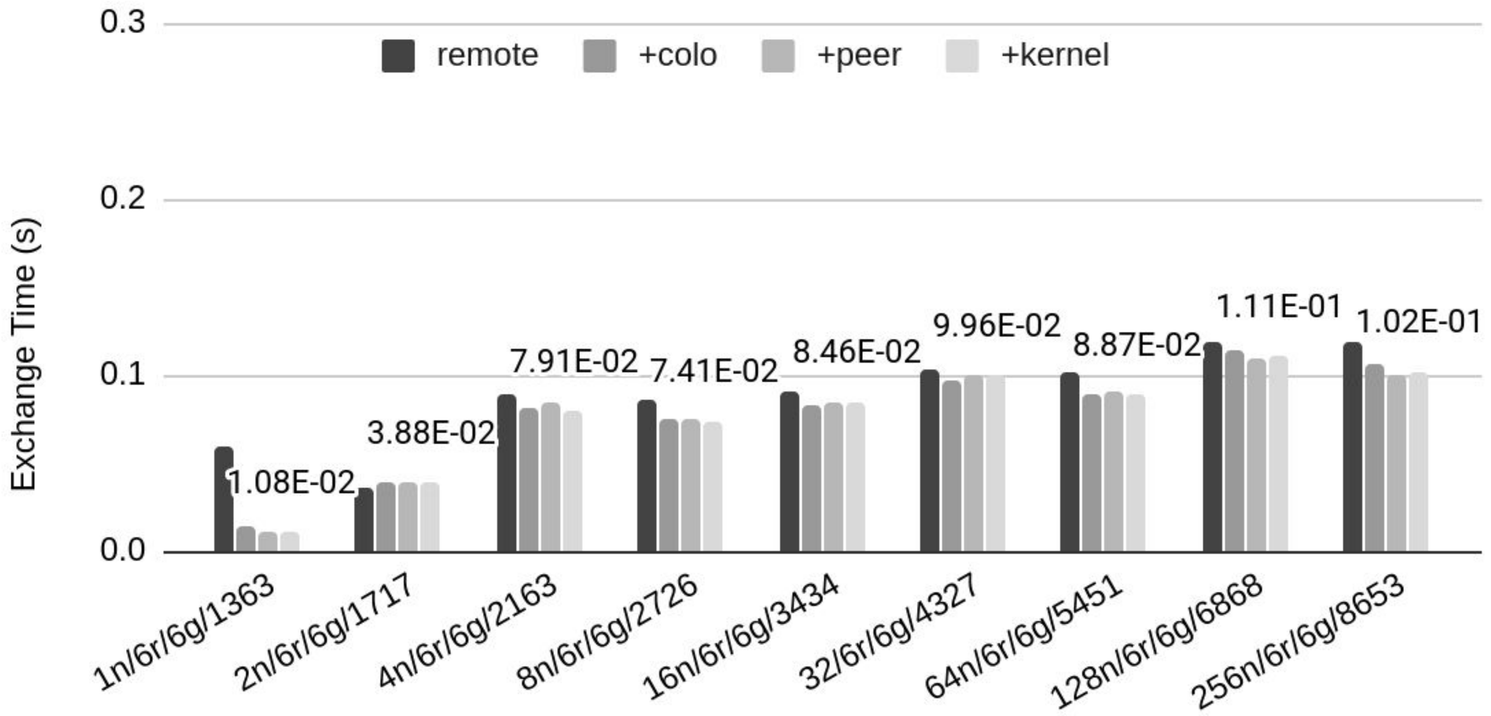
Ph.D. student, Electrical and Computer Engineering, University of Illinois Urbana-Champaign

- (Multi-)GPU communication
- Accelerating irregular applications

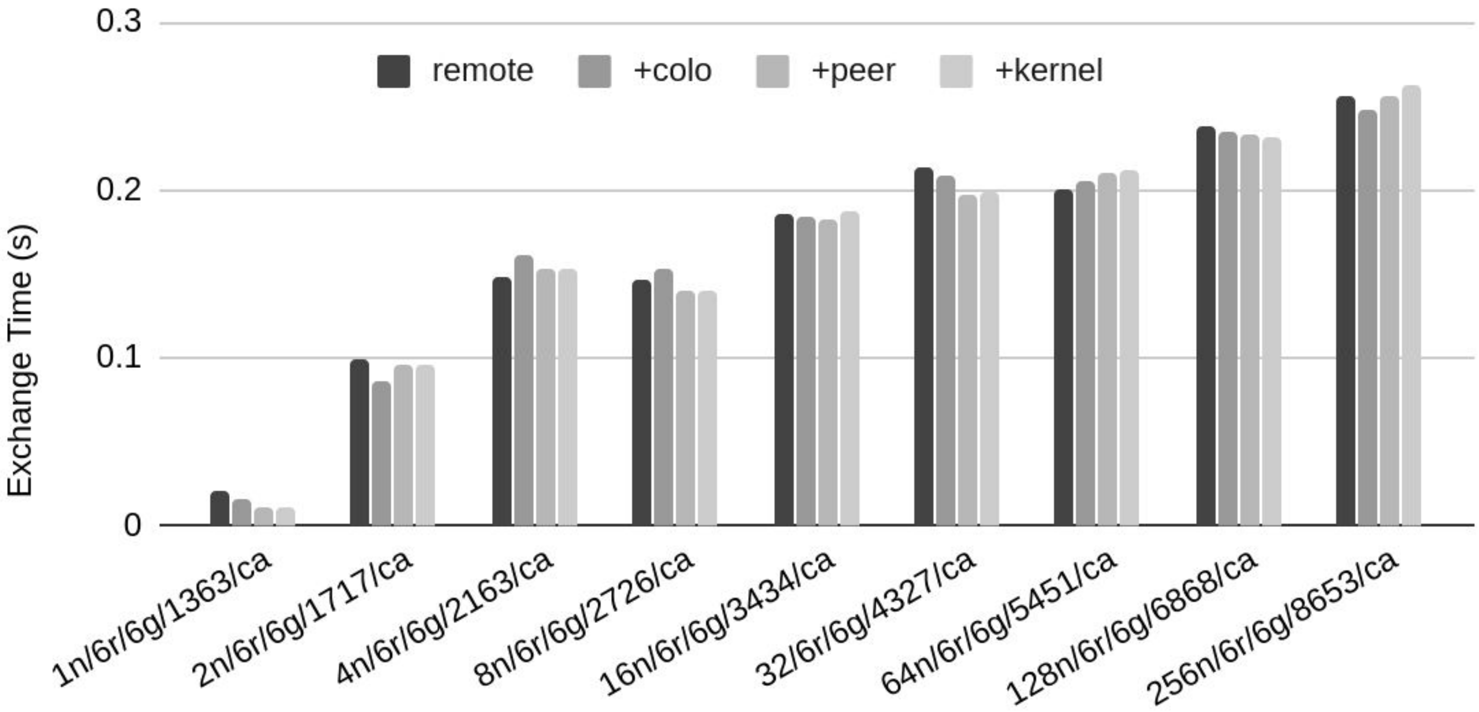
cwpearson
cwpearson
pearson at illinois.edu
https://cwpearson.github.io

Extra Slides

Weak Scaling (Summit) - Detail

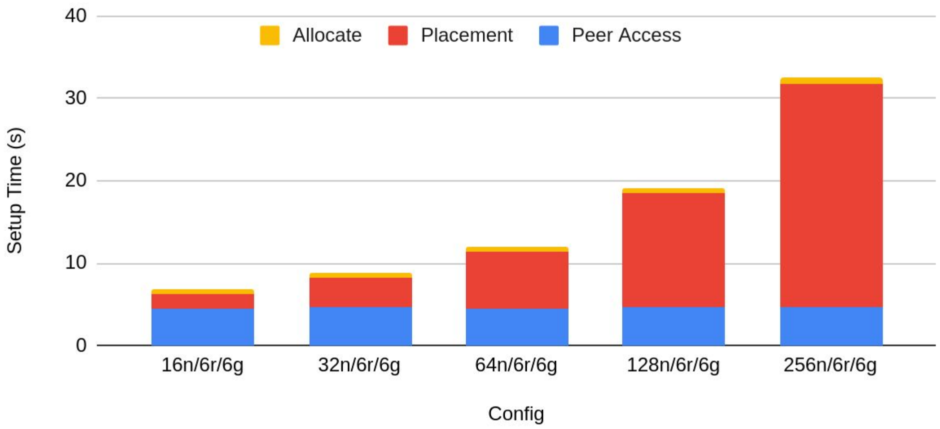


Weak Scaling (Summit) - CUDA-Aware Detail

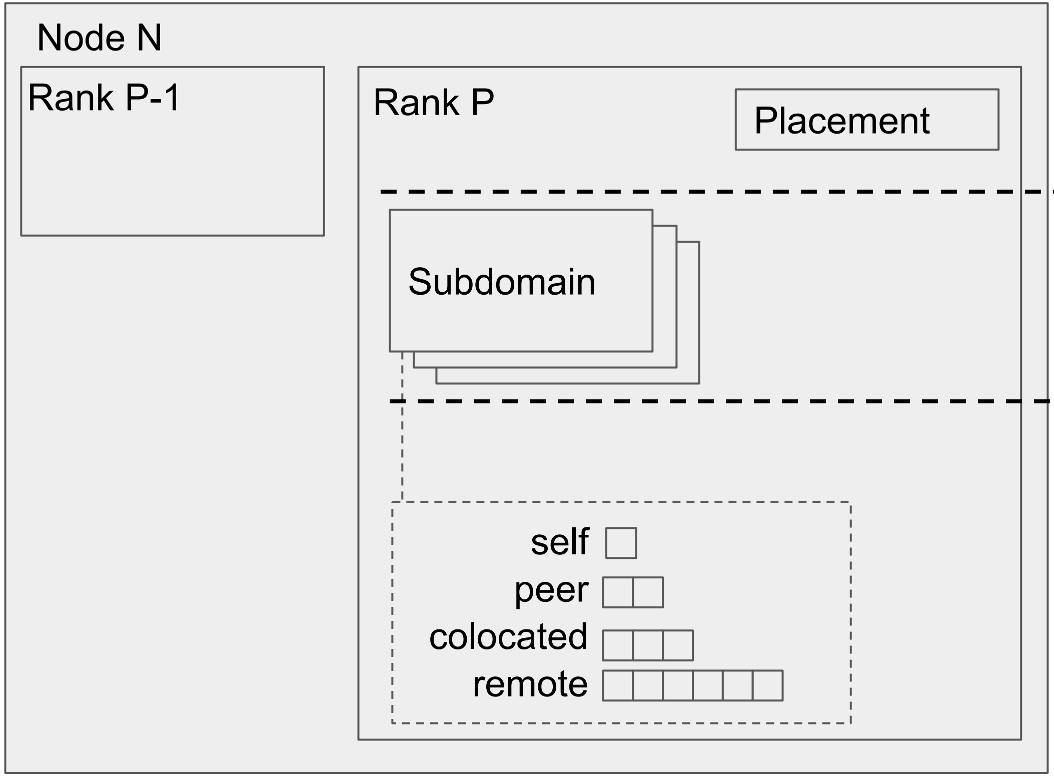


Future Work: Placement Performance

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



Communication Architecture



Holds placement information for all subdomains:
convert subdomain index to node, rank, GPU, and vis-versa

One subdomain per GPU

Communicators:
One group per subdomain
One self-communicator
Peer communicator per SD on same rank
Colocated communicator for each SD on same node
Remote communicator per SD on other node

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?