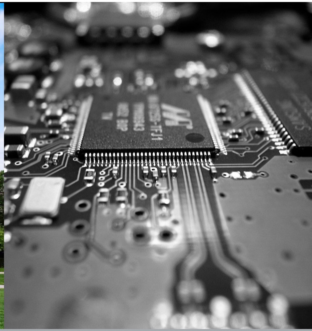
Node-Aware Stencil Communication on Heterogeneous Supercomputers

Carl Pearson[1], Mert Hidayetoglu[1], Mohammad Almasri[1], Omer Anjum[1], I-Hsin Chung[2], Jinjun Xiong[2], Wen-Mei Hwu[1]
[1]University of Illinois Electrical and Computer Engineering
[2]IBM T. J. Watson Research
May 22 2020



ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Carl Pearson

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

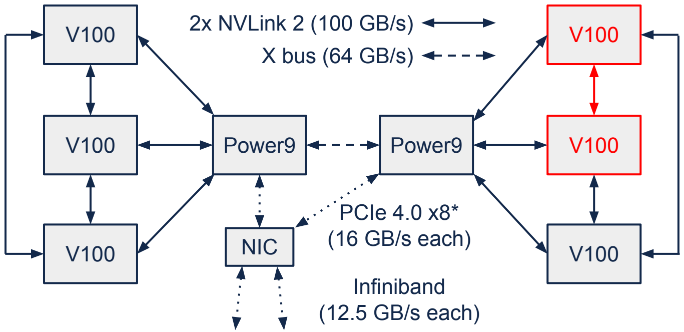
- Advised by Professor Wen-Mei Hwu
- (Multi-)GPU communication
- Accelerating irregular applications

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

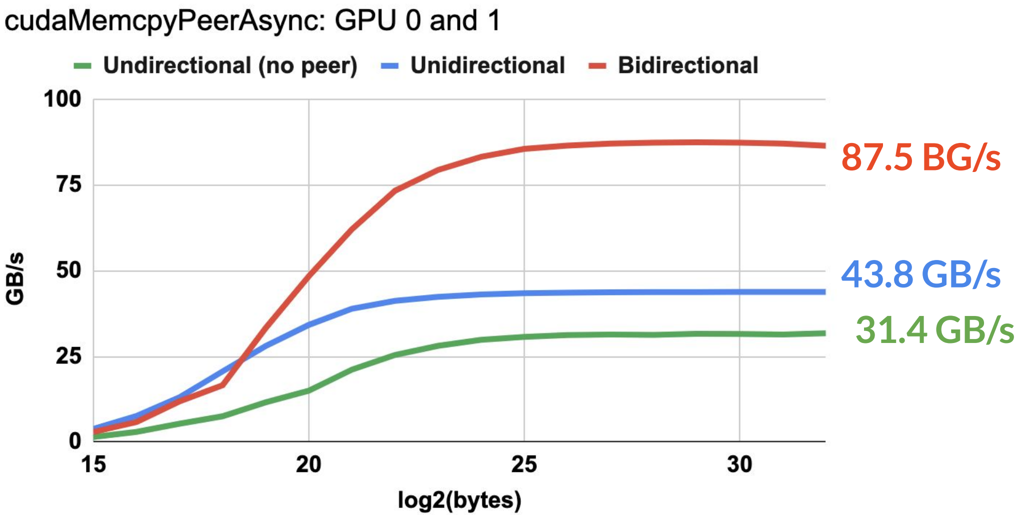
Outline

- Motivation
- Distributed Stencil & Glossary
- Parallelism
- Placement
- Primitives
- Future Work
- This talk: https://github.com/cwpearson/stencil

Single-Hop GPU Bandwidth

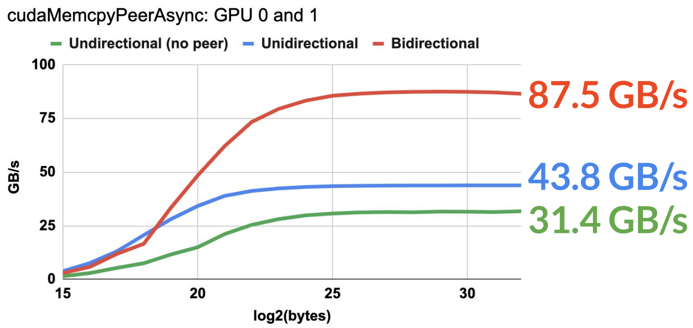


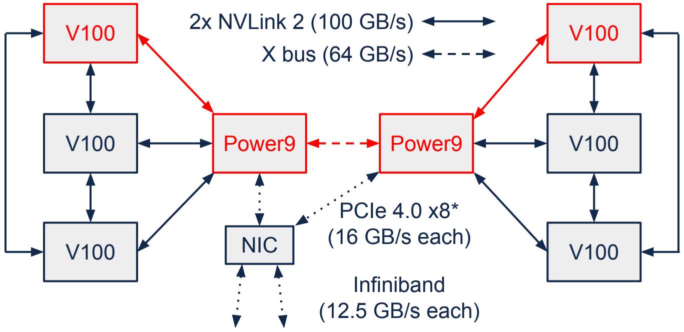
Summit Node
(bidirectional bandwidth)

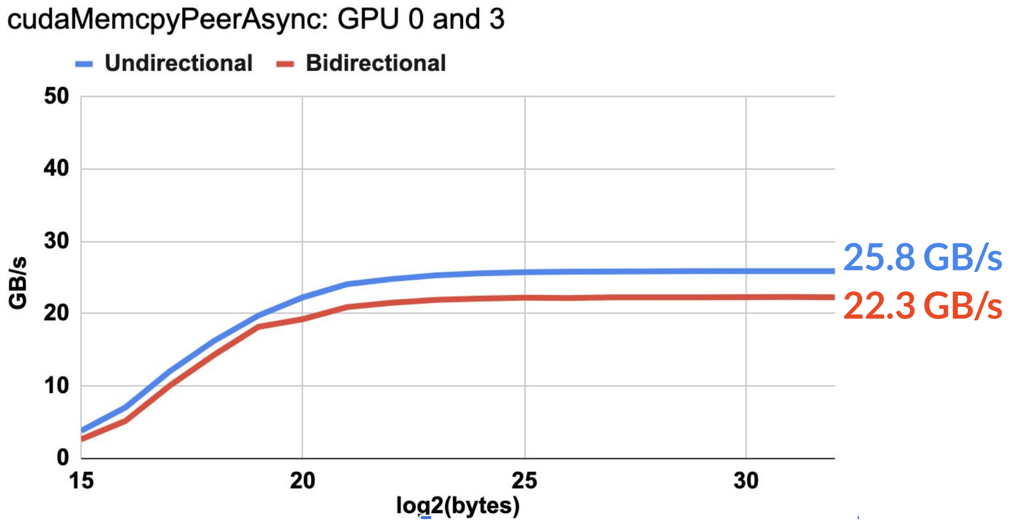


Bidirectional transfers double bandwidth

Multi-Hop Bandwidth







Summit Node
(bidirectional bandwidth)

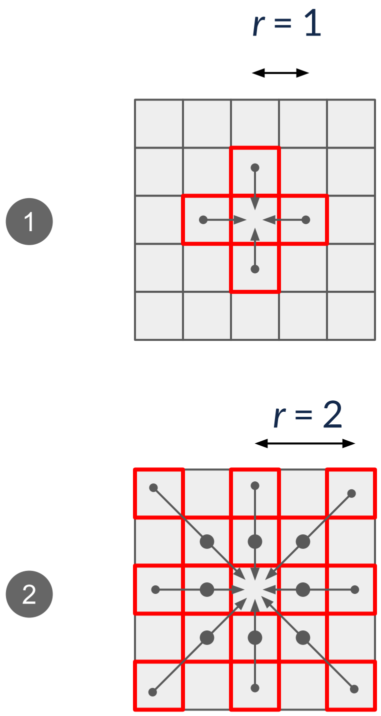
Bidirectional transfers are even slower

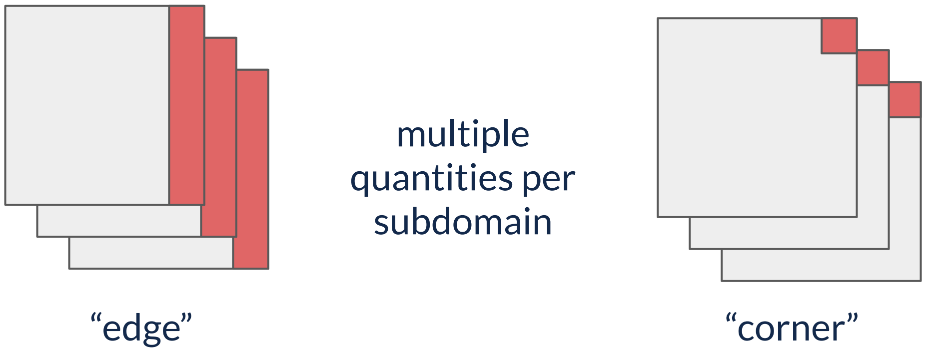
Distributed Stencils & Heterogeneous Nodes

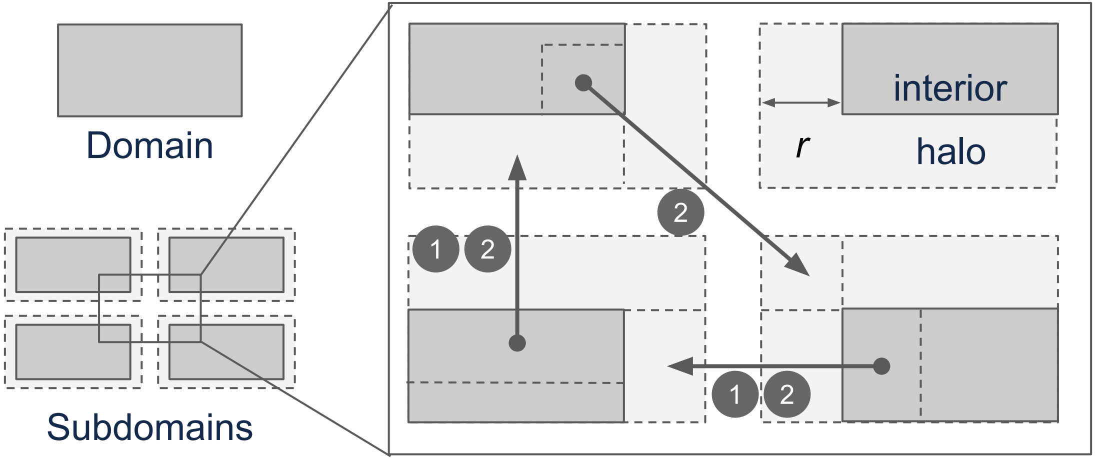
- Finite Difference Methods
- Regular computation, access, and structure reuse ➡️ stencil on GPU
- High-resolution modeling ➡️ Large stencils
- Limited GPU memory ➡️ distributed stencils with communication
- Fast stencil codes ➡️ larger impact of communication
- Heterogeneous nodes ("fat nodes") ➡️ how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



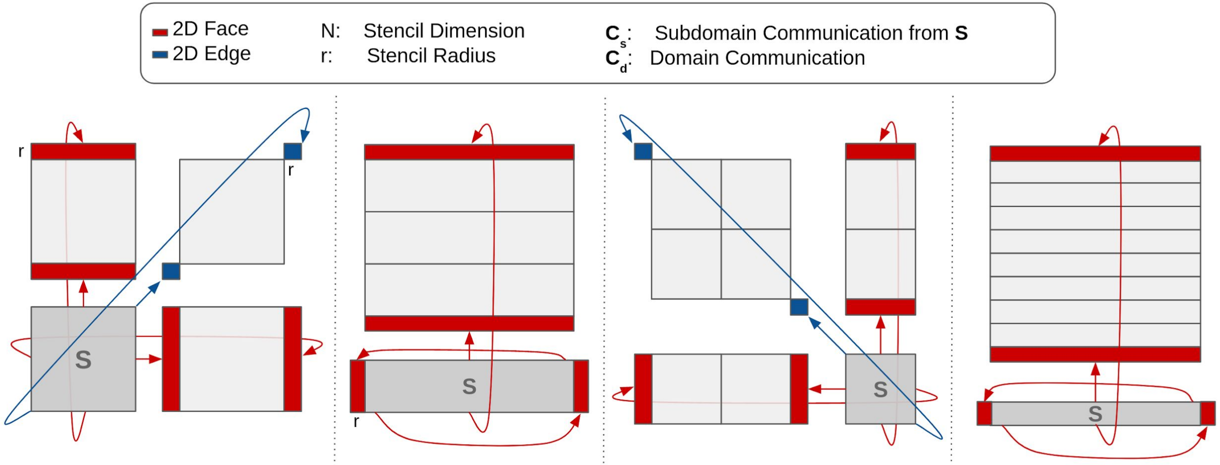




Approach

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

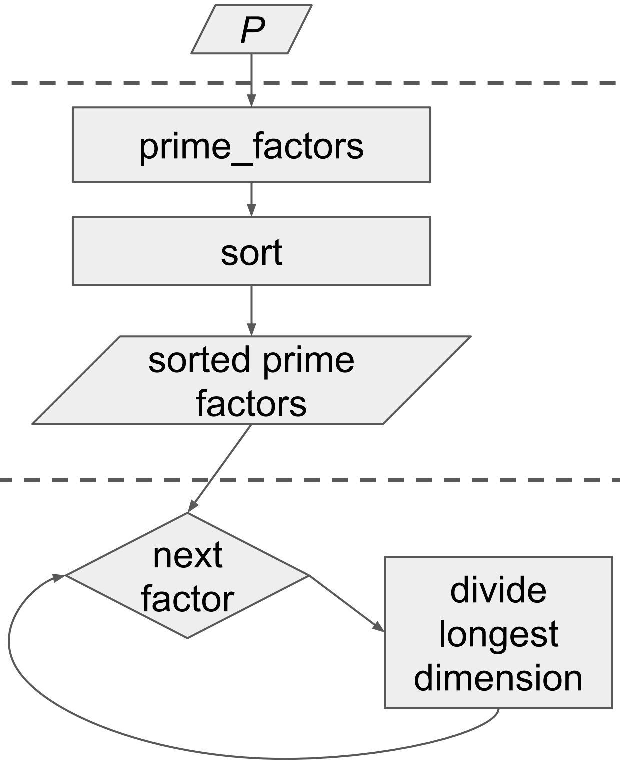
Partition: 4x1
S size = **N/4** x **N**
$\mathbf{C_s}$ = 2rN + 2rN/4
$\mathbf{C_d}$ = 4$\mathbf{C_s}$ = 10Nr

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

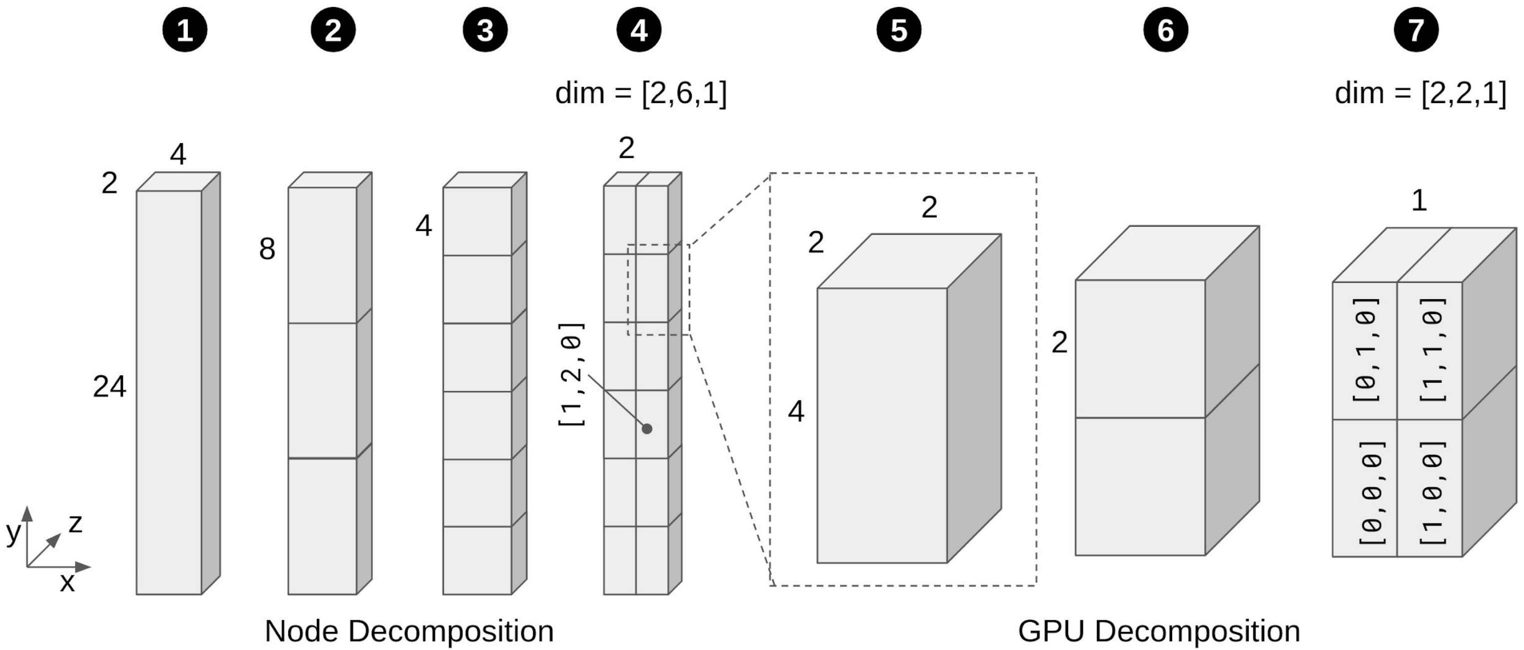


- Divide given domain into *P* subdomains

- Generate sorted prime factors, largest to smallest.
    - Evenly-sized subdomain require dividing by integers.
    - Fundamental Theorem of Arithmetic
    - 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

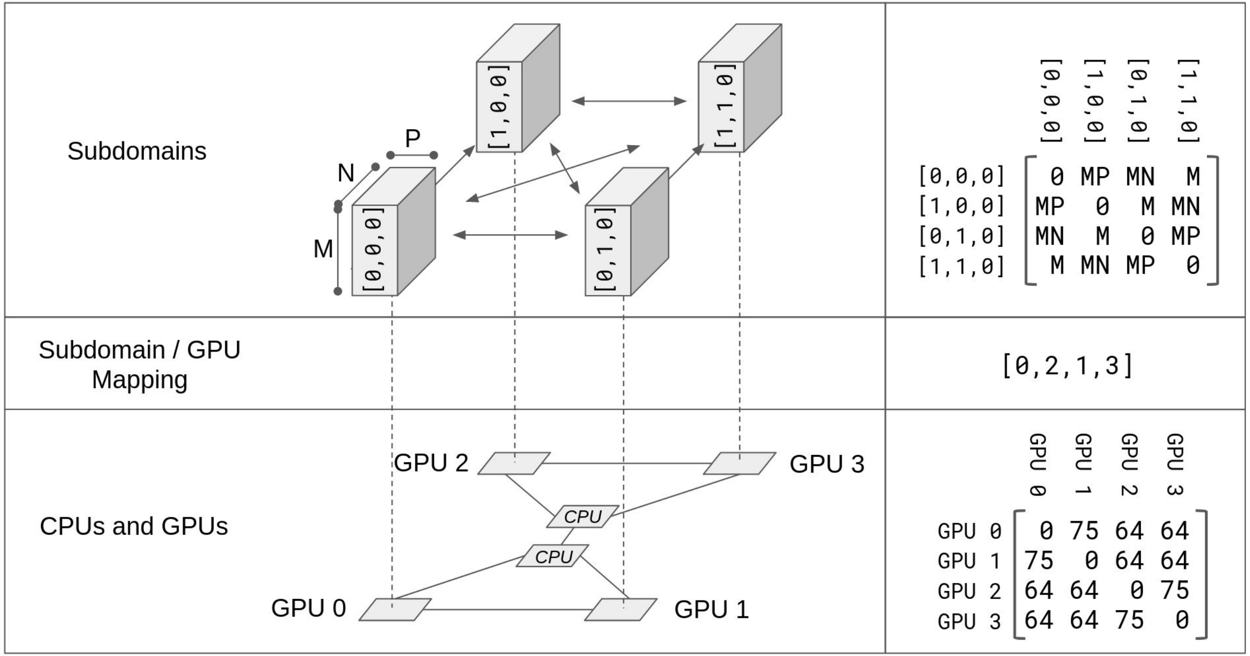
Hierarchical Decomposition



Minimize Communication Out of Node

Minimize communication between GPUs

Placement



How to place subdomains on GPUs to maximize bandwidth utilization?

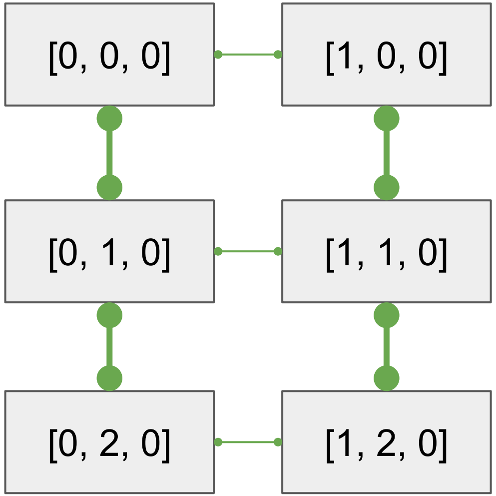
Quadratic Assignment Problem

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.

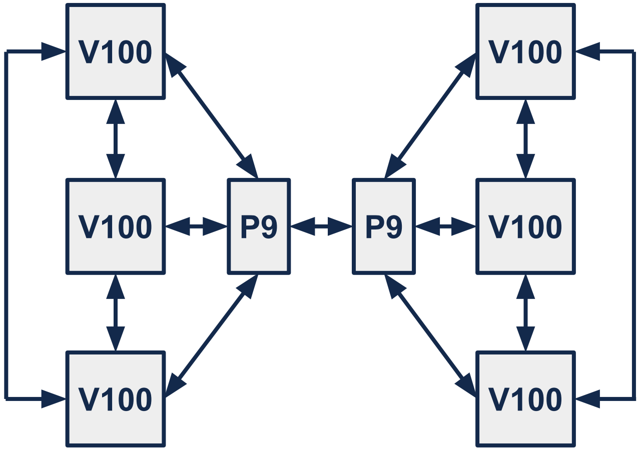
$$\sum_{i,j<n} w_{i,j} d_{f(i),f(j)}$$

	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

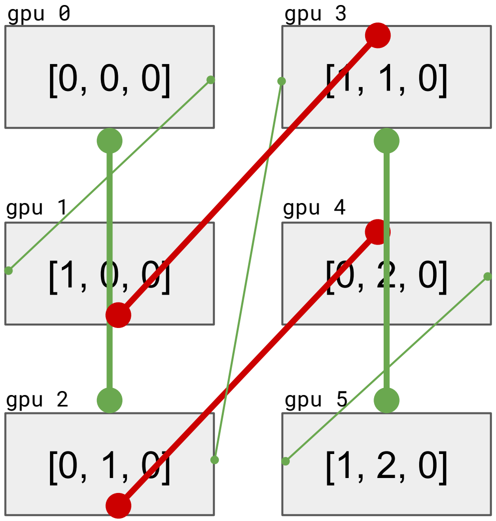
Example Placement



Node-Aware Placement



20% reduced exchange time
from placement alone



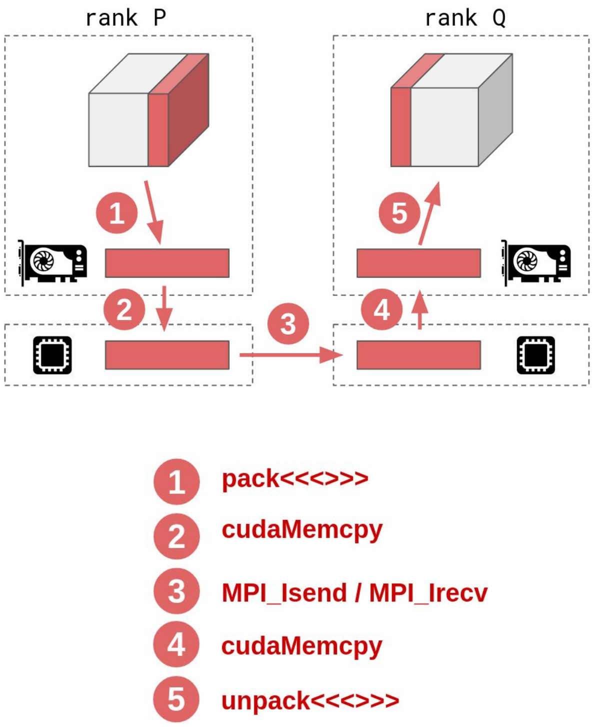
Another Placement

Capability Specialization Primitives

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

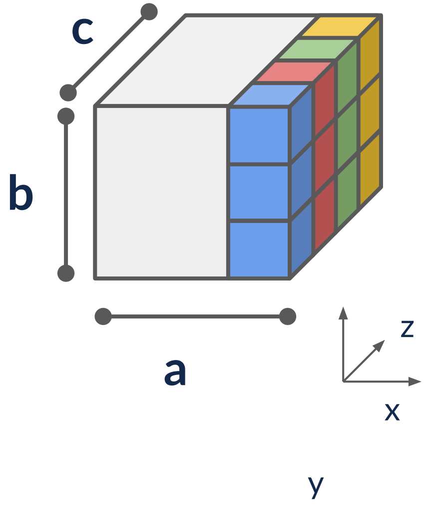
- "Staged": works for any 2 GPUs anywhere
  - pack from device 3D region into device 1D buffer
  - copy from device 1D buffer to host 1D buffer
  - MPI_Isend / MPI_Irecv to other host 1D buffer
  - copy from host 1D buffer to device 1D buffer
  - unpack from device 1D buffer to device 3D buffer

Optimizations are node-aware shortcuts on top of this

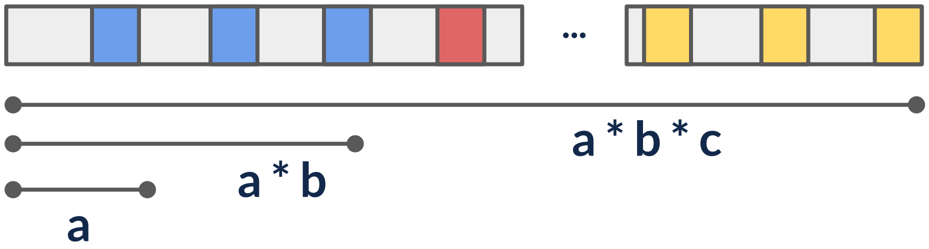


Pack and Unpack

3D View



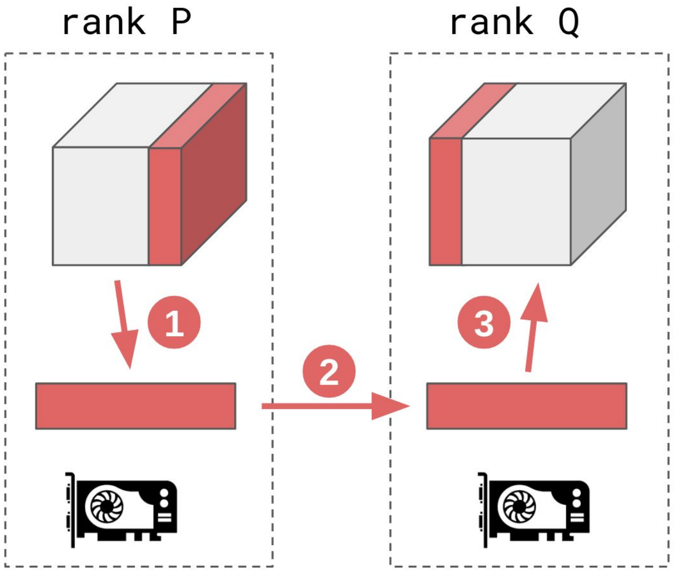
Actual Memory Layout



Packed Layout

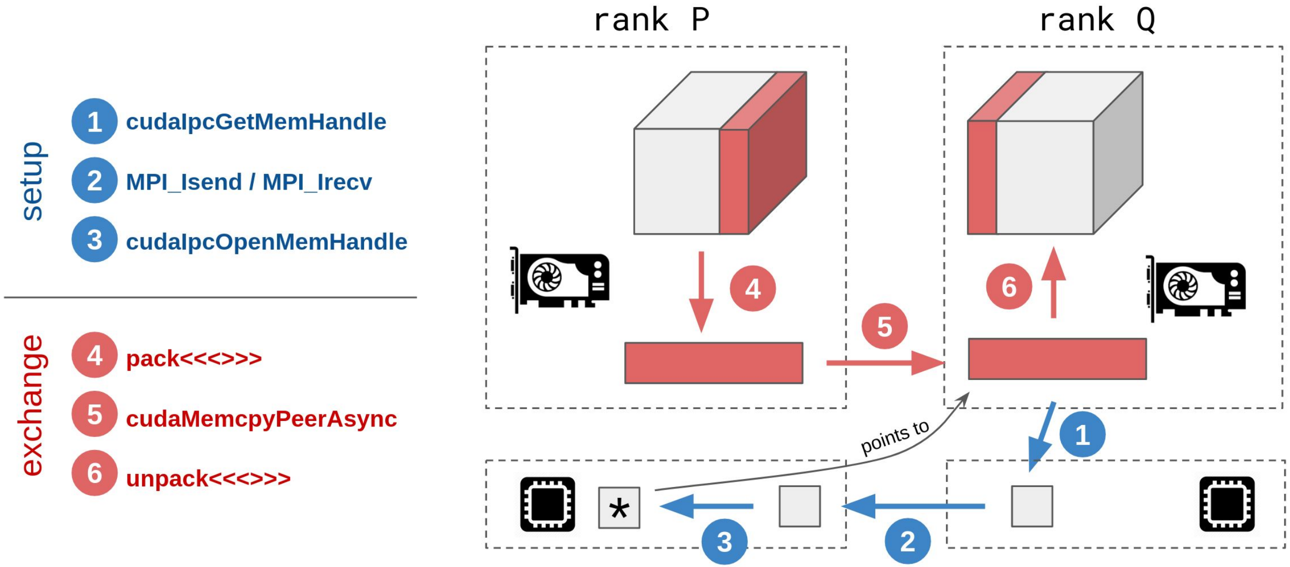
CUDA-Aware MPI

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



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

Colocated

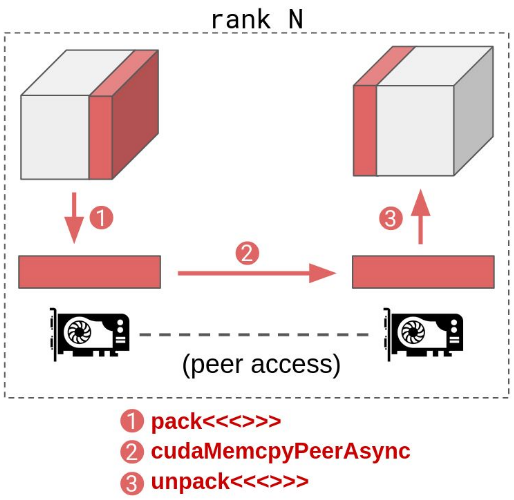


Exchange between different ranks on the same node

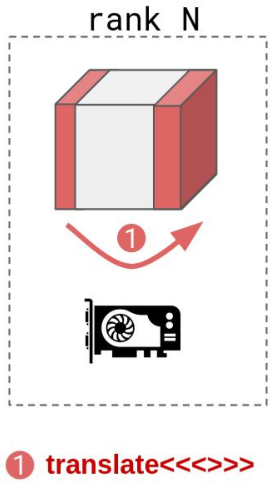
Different ranks are different processes with different address spaces

Use `cudaIpc*` to move a pointer between ranks, then cudaMemcpy*

Peer- and Self-exchange

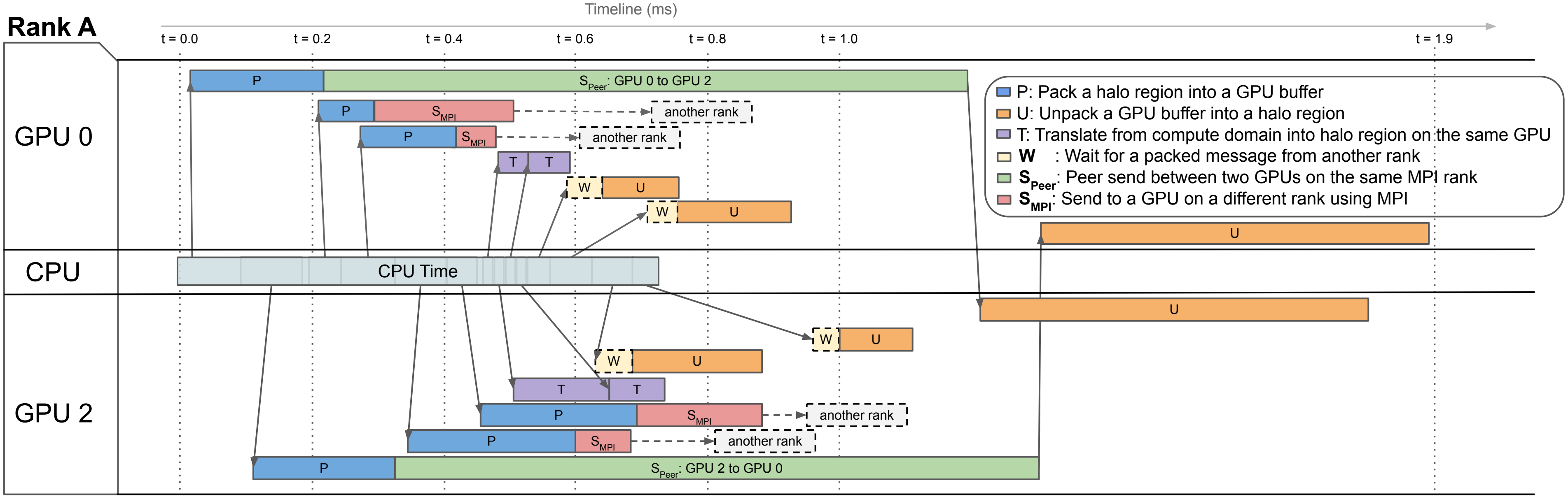


Peer: Two GPUs in the same rank



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

Overlap

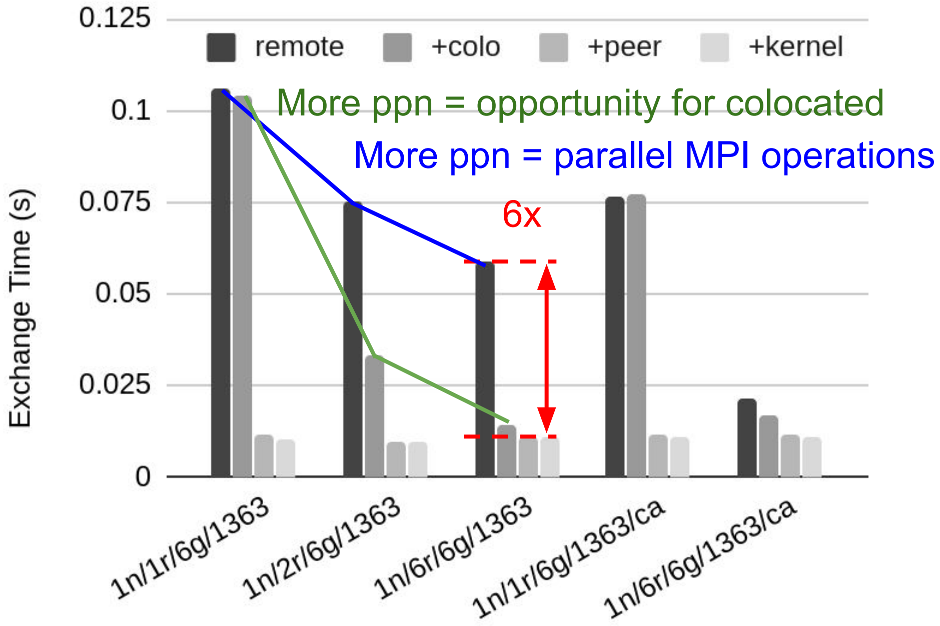


All operations are parallel and asynchronous

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

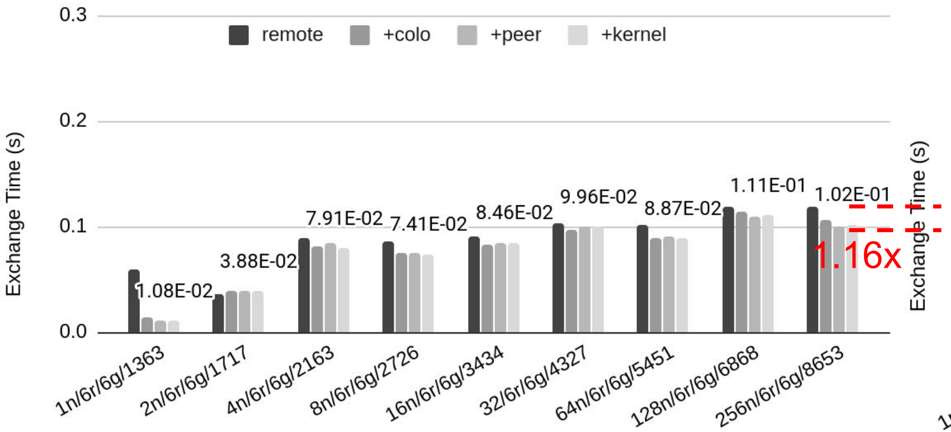
+kernel: “+peer” + self communicator

Specialization has a big impact in intra-node performance

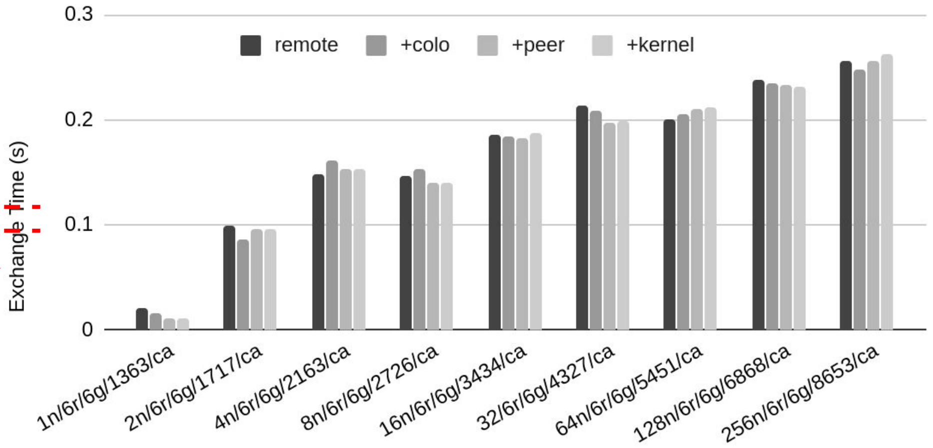
Weak Scaling (Summit)

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

Takeaways so Far

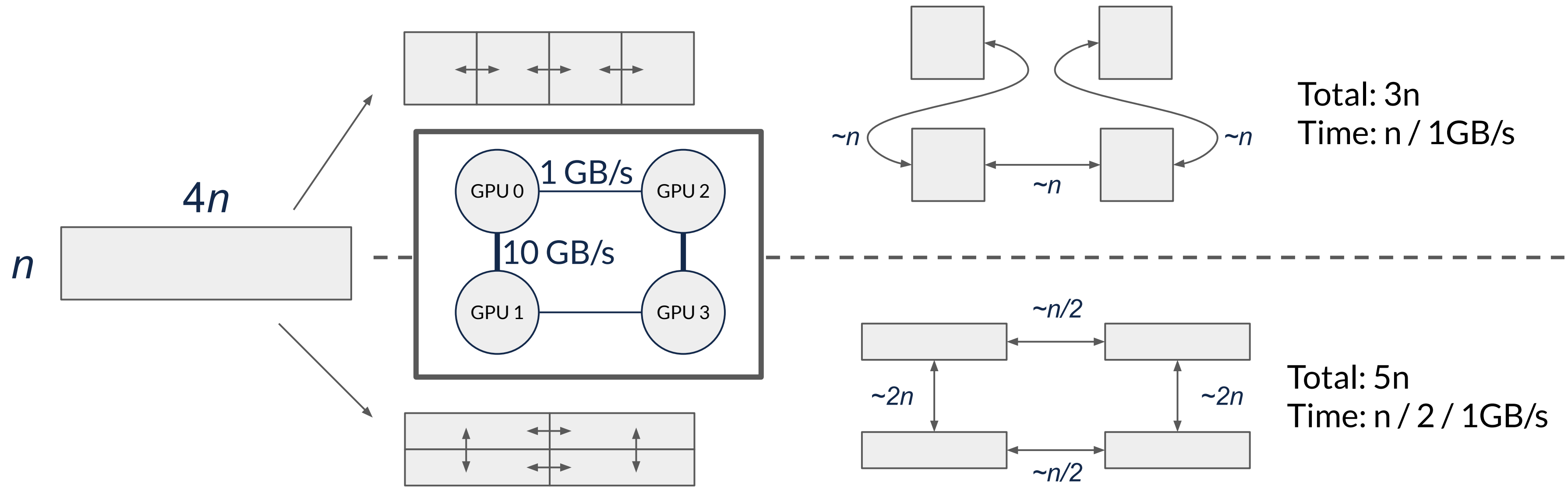
- Use (at least) one rank per GPU to maximize MPI injection bandwidth
- Data placement was good for 20% performance for one node
- Communication specialization was good for 6x on one node
  - still 1.16x at 256 nodes - allows MPI to just do off-node
- CUDA-Aware MPI seems like a proof-of-concept right now
- Some opportunities to improve partitioning and placement according to node topology
-

Future Directions (1/N)

Parallelism	Scalable decomposition	Subdomain decomposition to minimize communication
Placement	Assign tasks according to theo	Node-aware placement to ons
Primitives	Ach perf	ations cialization

Minimizing communication may not maximize performance. Both decomposition and placement informed by system

Example Node-Aware Partition



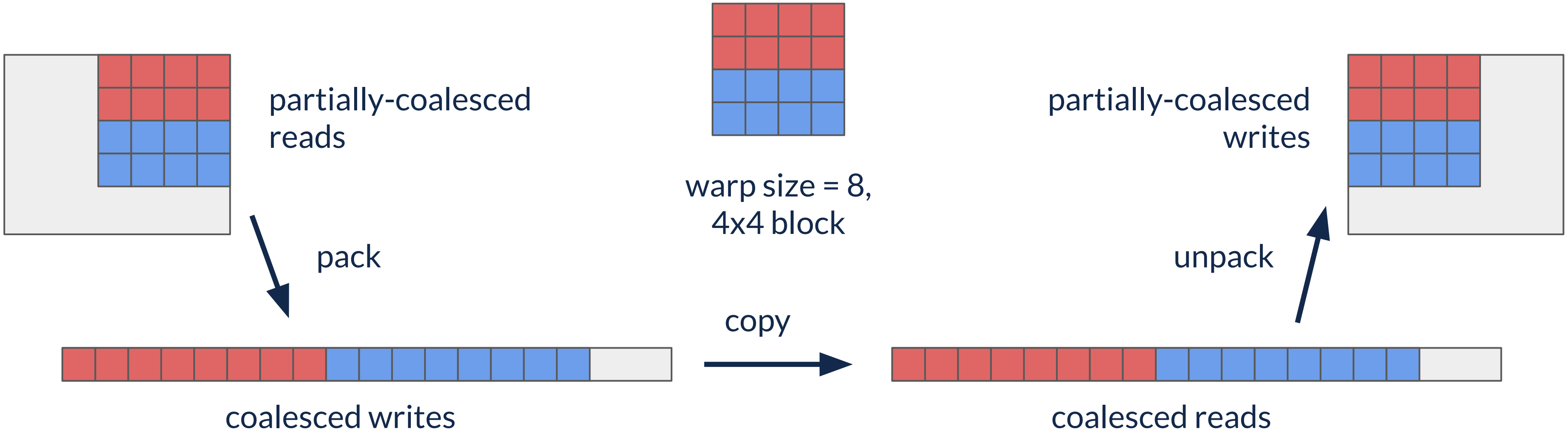
Consider 2 different partitions for target platform

Platform properties determine best partition, not just best placement

All Pack Directions not Equal

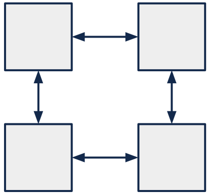
Not all communication directions have same performance on same link.

Pack / Unpack performance depends on strides

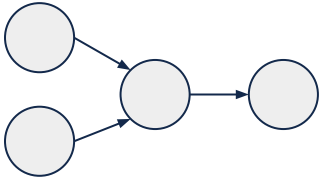


unpack is 2-3x slower than pack for non-contiguous regions

Future Directions



System Graph
vertices: PEs
edges: interconnects

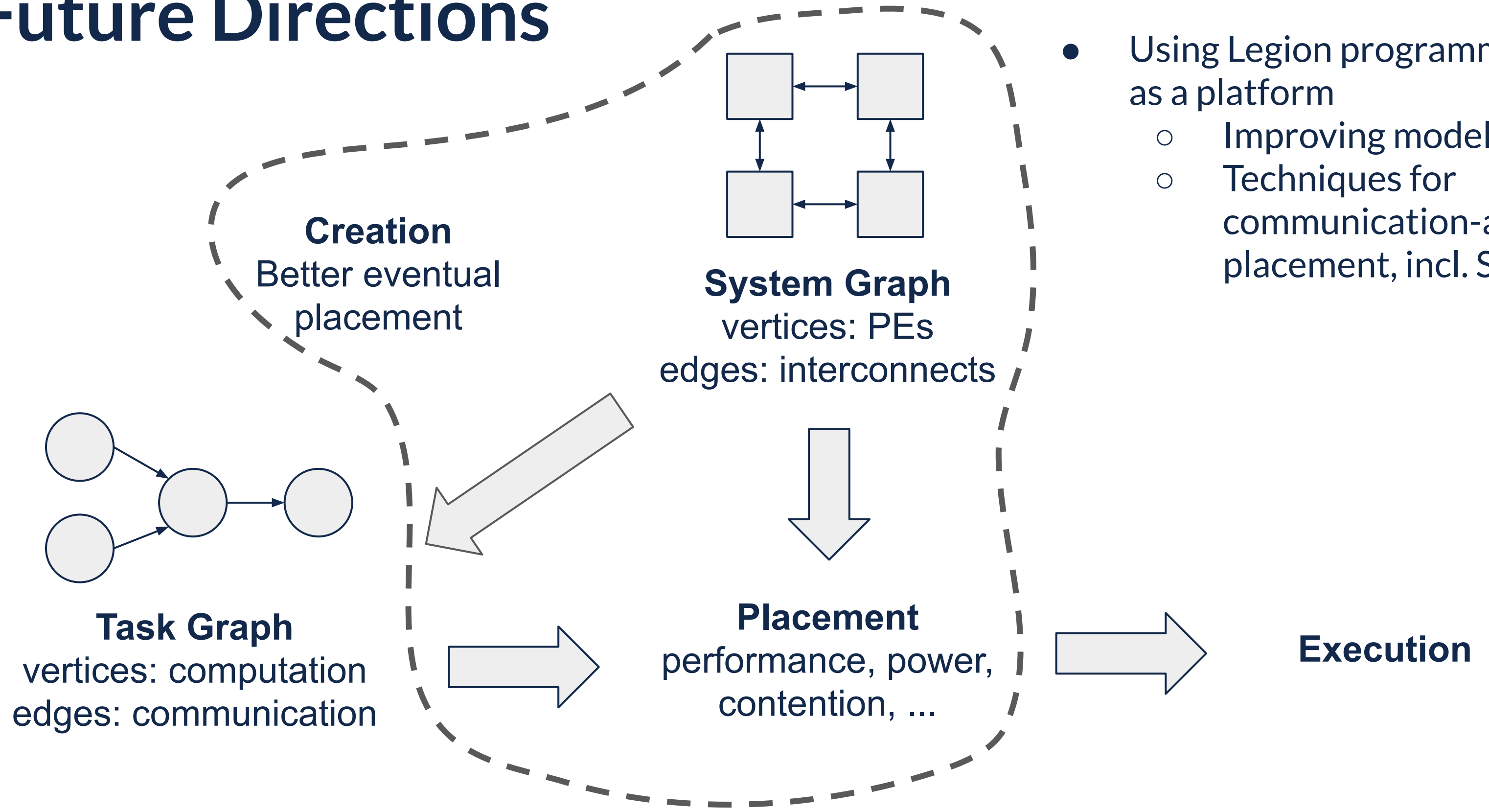


Task Graph
vertices: computation
edges: communication

Placement
performance, power,
contention, ...

Execution

Future Directions



- Using Legion programming system as a platform
  - Improving model of system
  - Techniques for communication-aware placement, incl. SMT solvers

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

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.