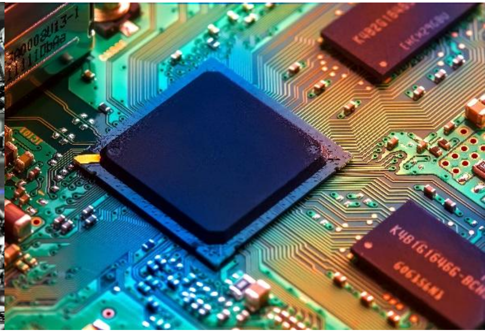
NUMA-Aware Data-Transfer Measurements for Power/NVLink Multi-GPU Systems

Carl Pearson[1], I-Hsin Chung[2], Zehra Sura[2], Wen-mei Hwu[1], Jinjun Xiong[2]
[1] Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign
[2] IBM T.J. Watson Research Center



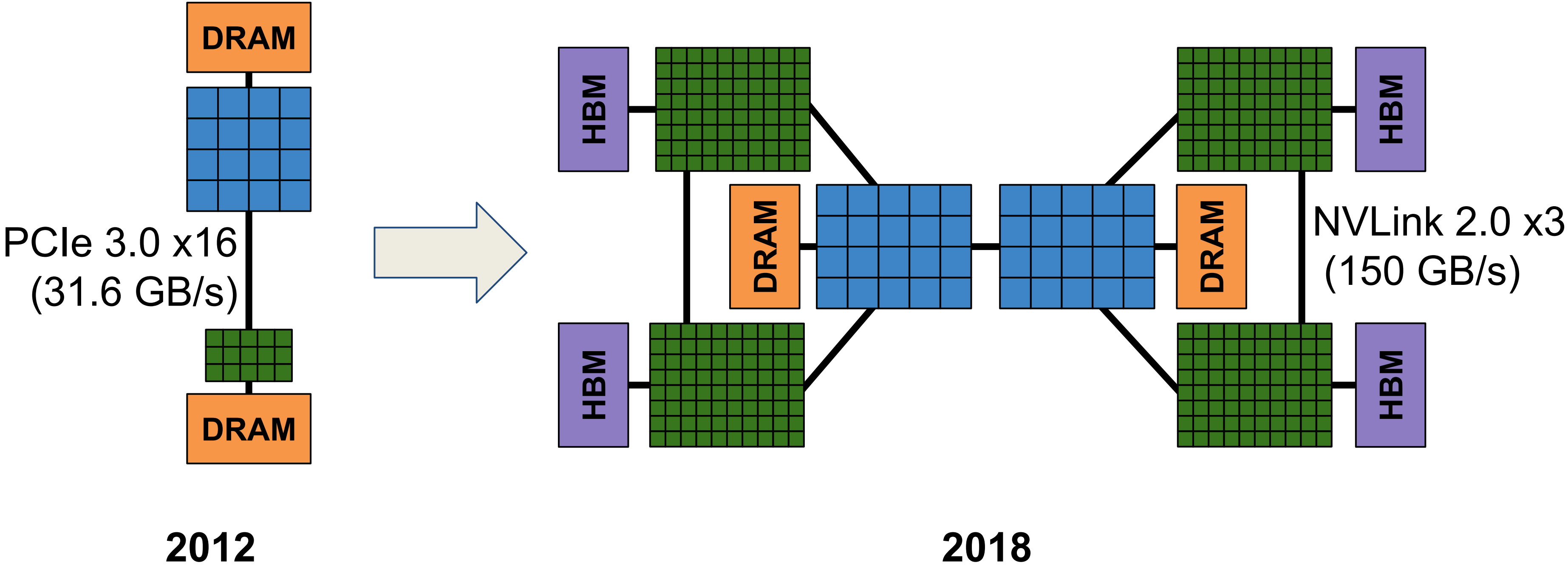
ILLINOIS
Electrical & Computer Engineering
COLLEGE OF ENGINEERING

Outline

- Motivation
  - Complex multi-cpu / multi-gpu nodes
- Measurement Approach
  - rai-project/microbench
  - Reference Systems
- Selected Results

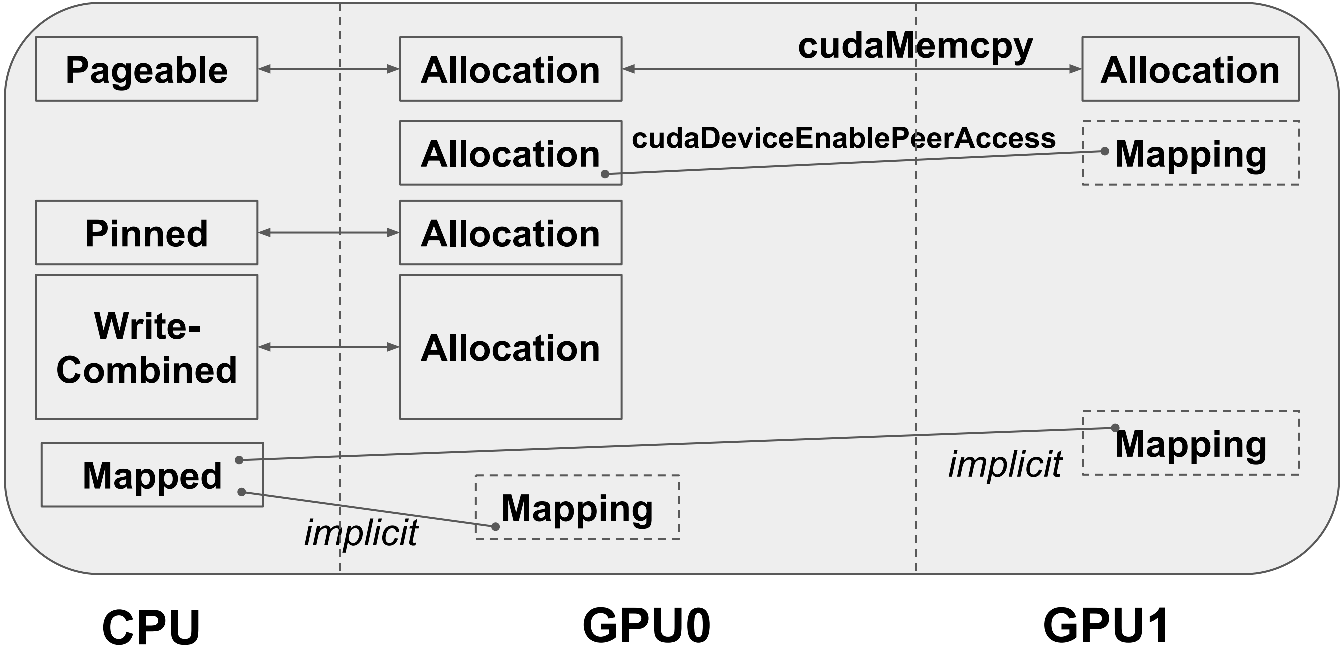
Motivation

Heterogeneous Hardware is Widely Available



System Software is Complicated

e.g. explicit CUDA memory management



System Software is Complicated

e.g. CUDA Unified Memory

	GPU 0	GPU 1	CPU	
cudaSetDevice(0); cudaMallocManaged(&a,...);				
a[**page0**] = 0; // **gpu0**				
a[**page1**] = 1; // **gpu1**				Page fault and migration
a[**page2**] = 2; // **cpu**				Page fault and migration
cudaMemAdvise(a, **gpu1**, cudaMemAdviseSetPreferredLocation); a[**page1**] = 1; **// cpu**				Write served over NVLink
cudaMemPrefetcAsync(a, **gpu1**);				Bulk page migration

Measurement Approach

“rai-project/microbench”

- NUMA / CPU / GPU Communication Microbenchmarks
  - libnuma
  - CUDA explicit memory management
  - CUDA unified memory coherence and prefetch
- Across all NUMA / GPU and GPU / GPU combinations

https://github.com/rai-project/microbench

High-Level Benchmark Approach

Repeat to find variability	`Loop repetitions`
Setup	`    Establish allocations`
Main loop	`    Loop iterations`
	`        Move data to src`
	`        Record time`
	`        Move data to dst`
	`        Record time`
Teardown	`    Free allocations`
	`    Metric = average`
	`Compute average, stddev of metric`

“rai-project/microbench” Other Microbenchmarks

- Present
  - CUDA primitive operations
    - Kernel launch, …
  - Neural Network primitives
    - CUDNN operations, parameters from published networks
- In Progress
  - Full-Duplex GPU-GPU communication
  - Multi-GPU collectives
  - Tensorcores
- Future
  - Disk / Network

https://github.com/rai-project/microbench

“rai-project/microbench” Infrastructure

- Google Microbenchmark Support Library for benchmarking functions
  - Benchmark filtering
  - Localized optimization controls
  - Manual or automatic timing
  - Automatic determination of number of runs
  - **Repeated runs and simple statistics**
  - JSON output files

https://github.com/google/benchmark

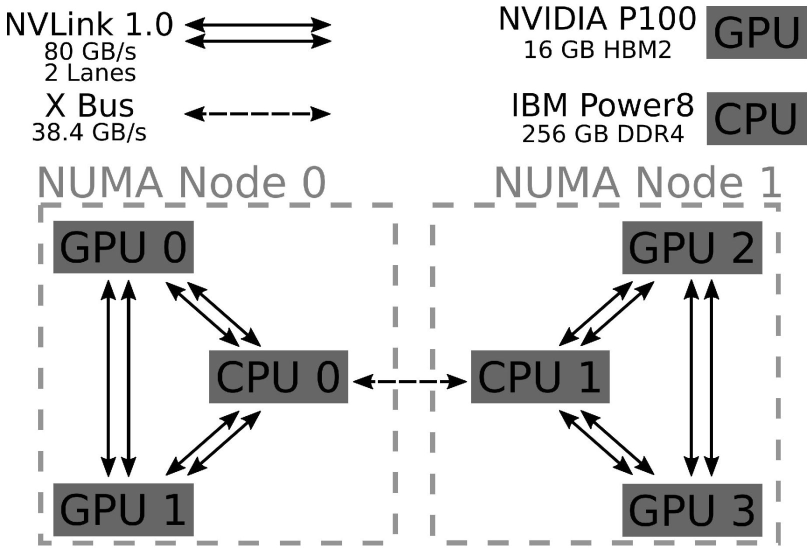
"rai-project/microbench" Infrastructure

- **CMake** - control build and installation process
  - **cotire**[1]: automate precompiled headers and single compilation unit builds
  - **hunter**[2]: cross-platform package manager for C++
- **Docker**
  - `raiproject/microbench:${arch}-${cuda}-${branch}`
  - **Have amd64 CUDA 7.5, 8.0, 9.2**
  - **Want ARM, POWER**
  - **Expect Docker has network performance hit**[3]

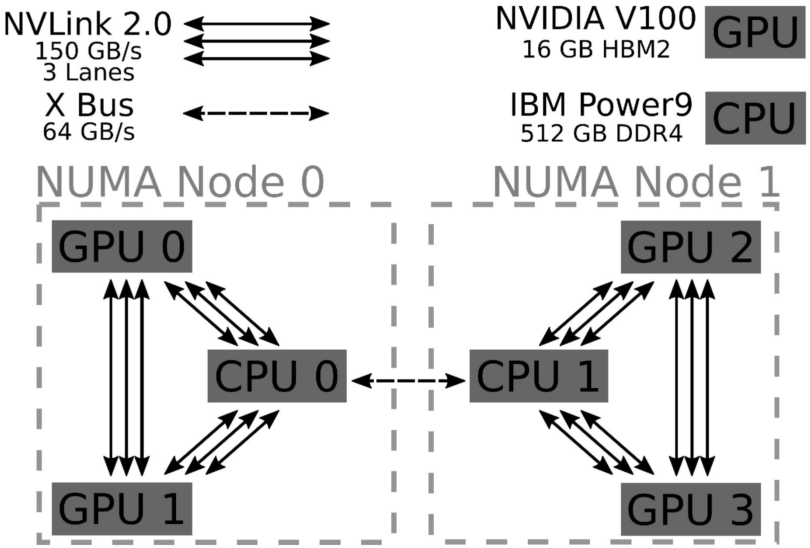
“rai-project/microbench_plot”

- Plotting google/benchmark results
- yaml plot specification format
- Parsing/filtering Benchmark data files
- Generate makefile dependencies
- Python 2 & 3

Reference Systems



Minsky
4.4.0-96-generic
CUDA 9.1.85
Driver 390.31

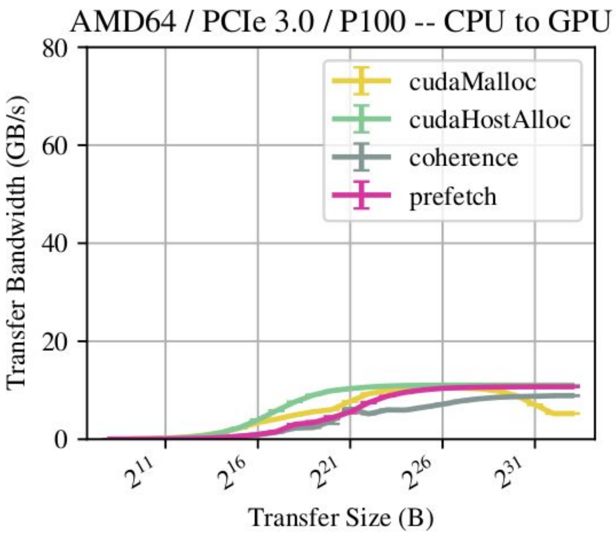


Newell
4.14.0-49.2.2.el7a.ppc64le
CUDA 9.2.88
Driver 396.26

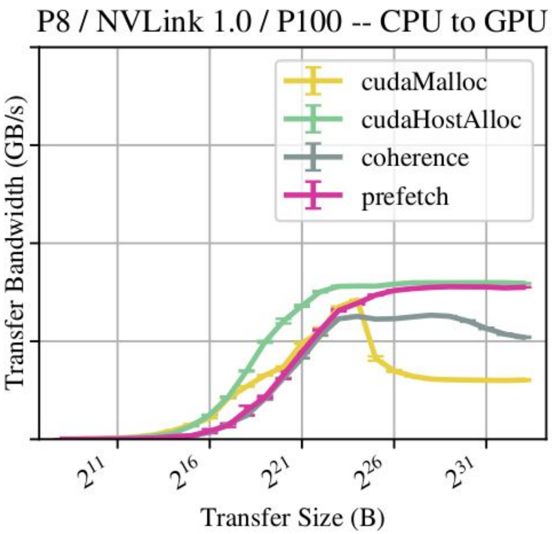
Selected Results

Faster Interconnects

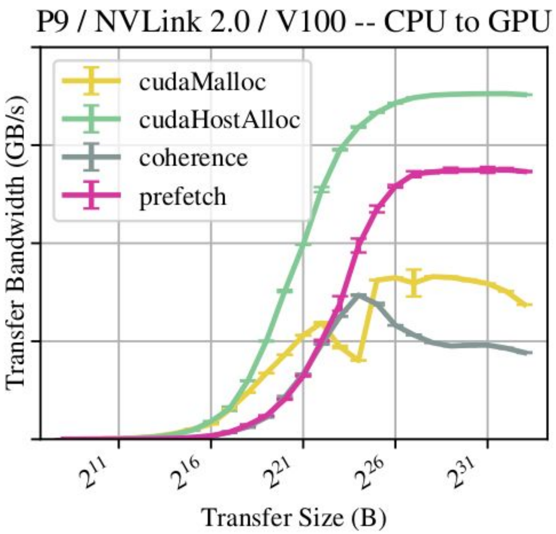
PCIe 3.0 x16 (15.8 GB/s)



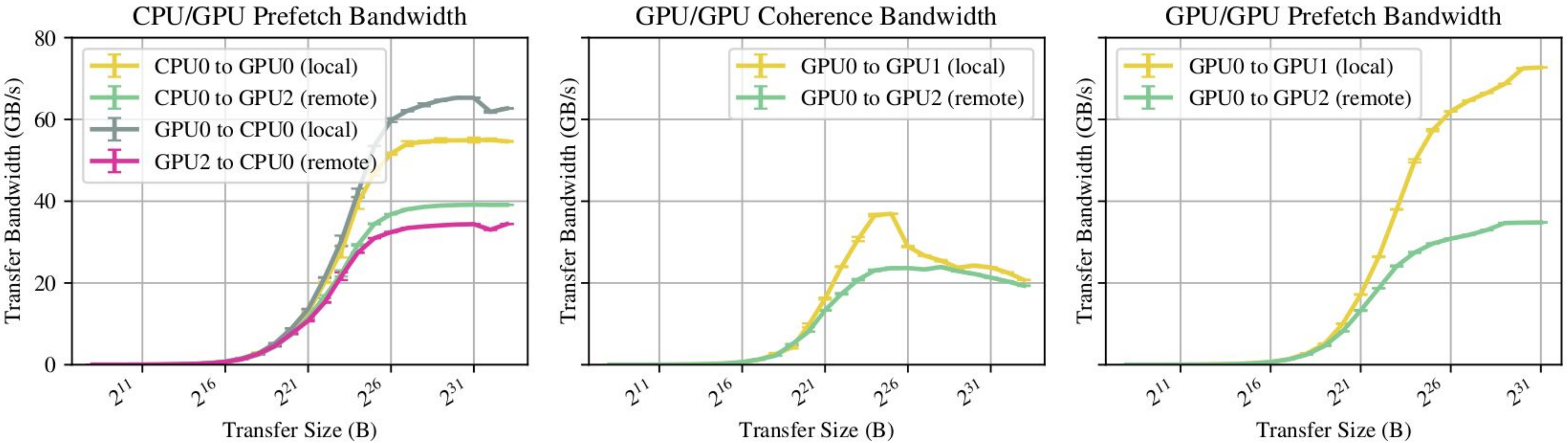
NVLink 1.0 x2 (40 GB/s)



NVLink 2.0 x3 (75 GB/s)



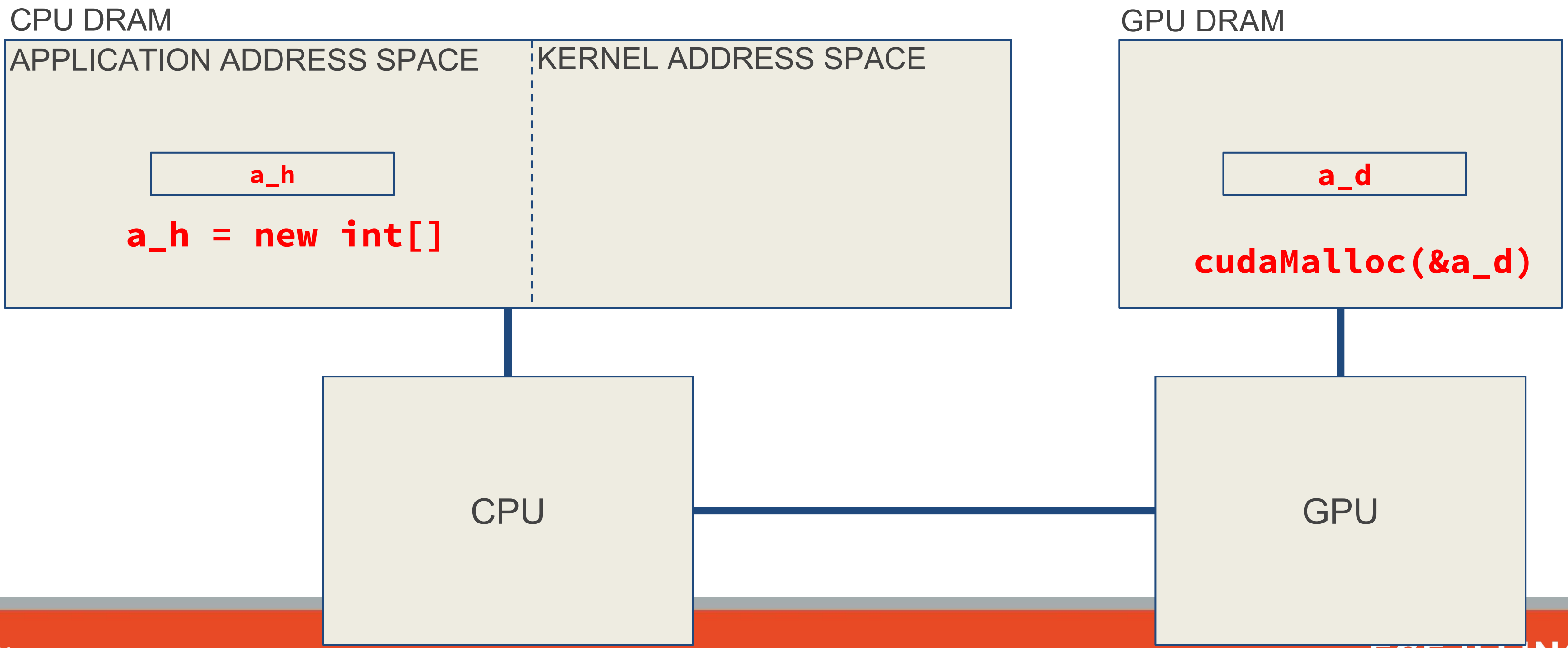
Device Affinity and Transfer Bandwidth (Newell)



Data placement has a big bandwidth impact

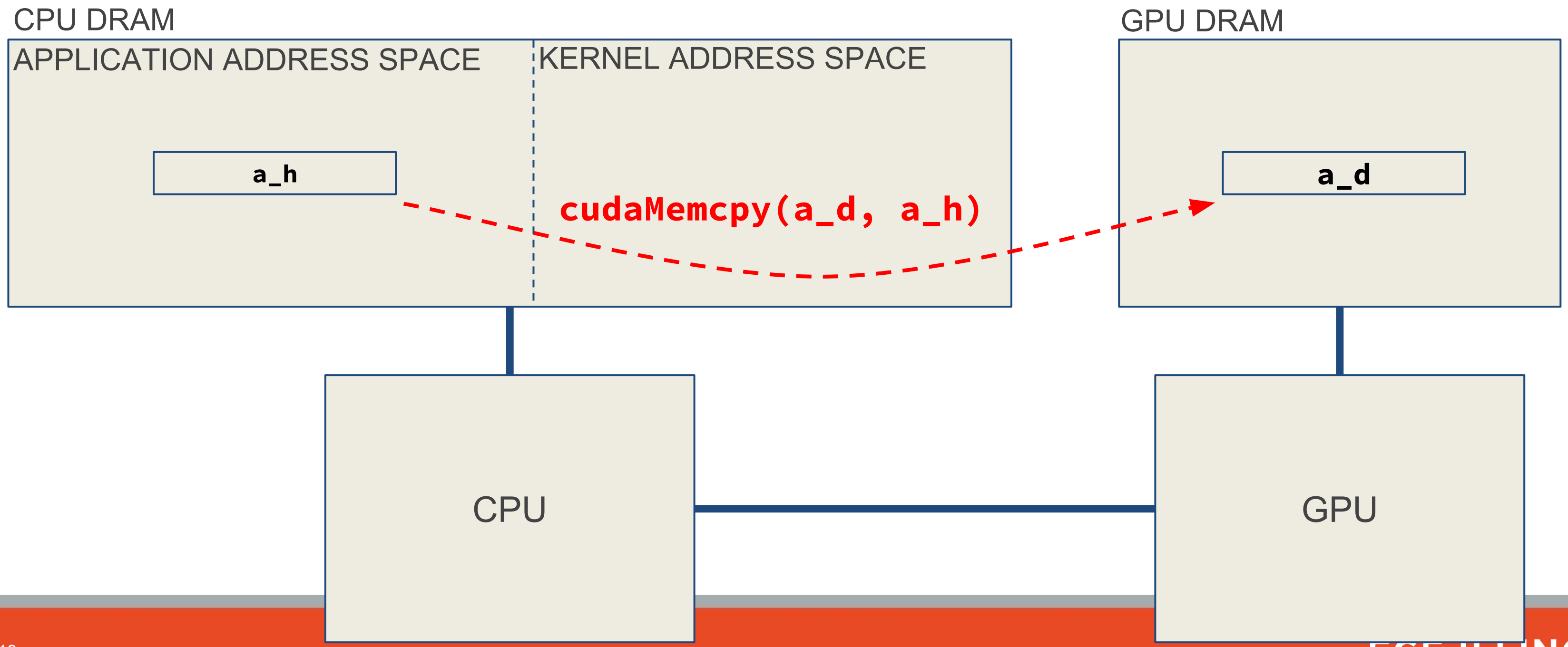
Pageable cudaMemcpy (1/4)

1) Allocate pageable memory



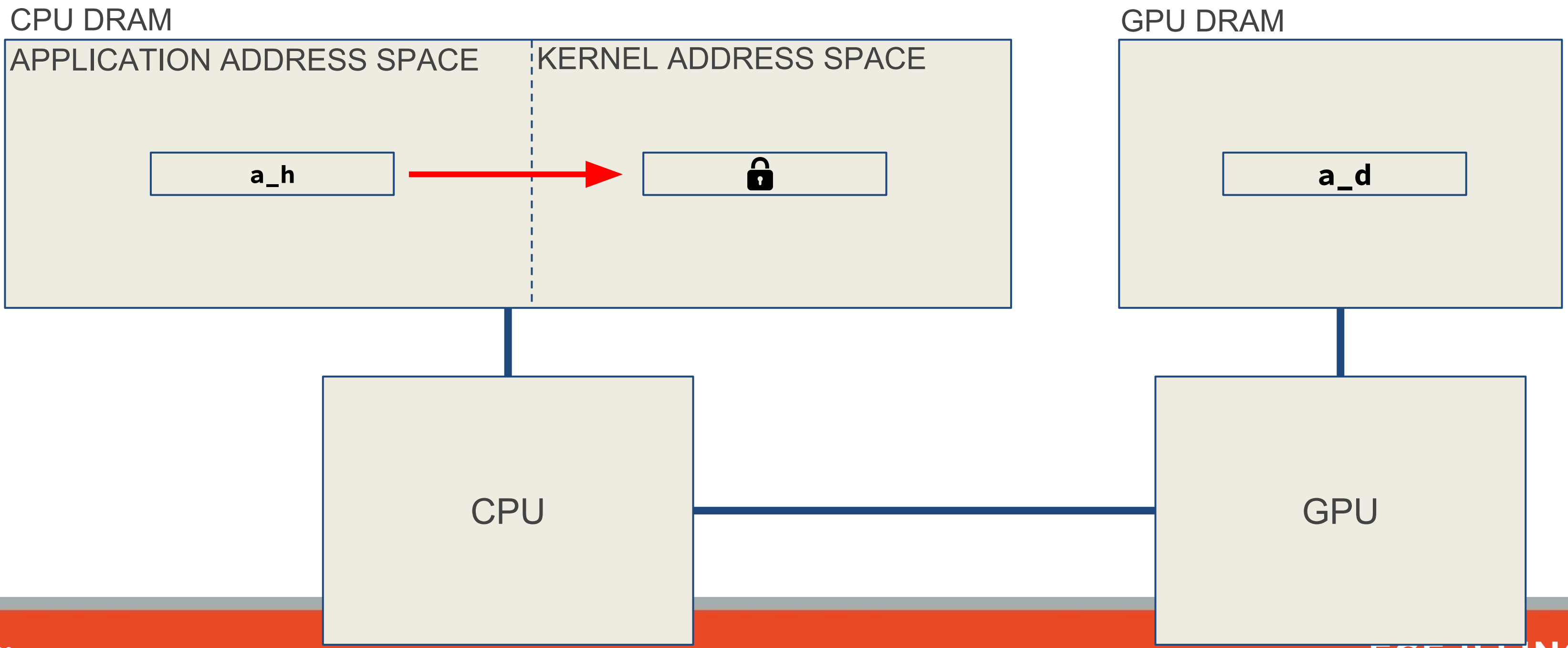
Pageable cudaMemcpy (2/4)

2) Initiate CUDA Memcpy



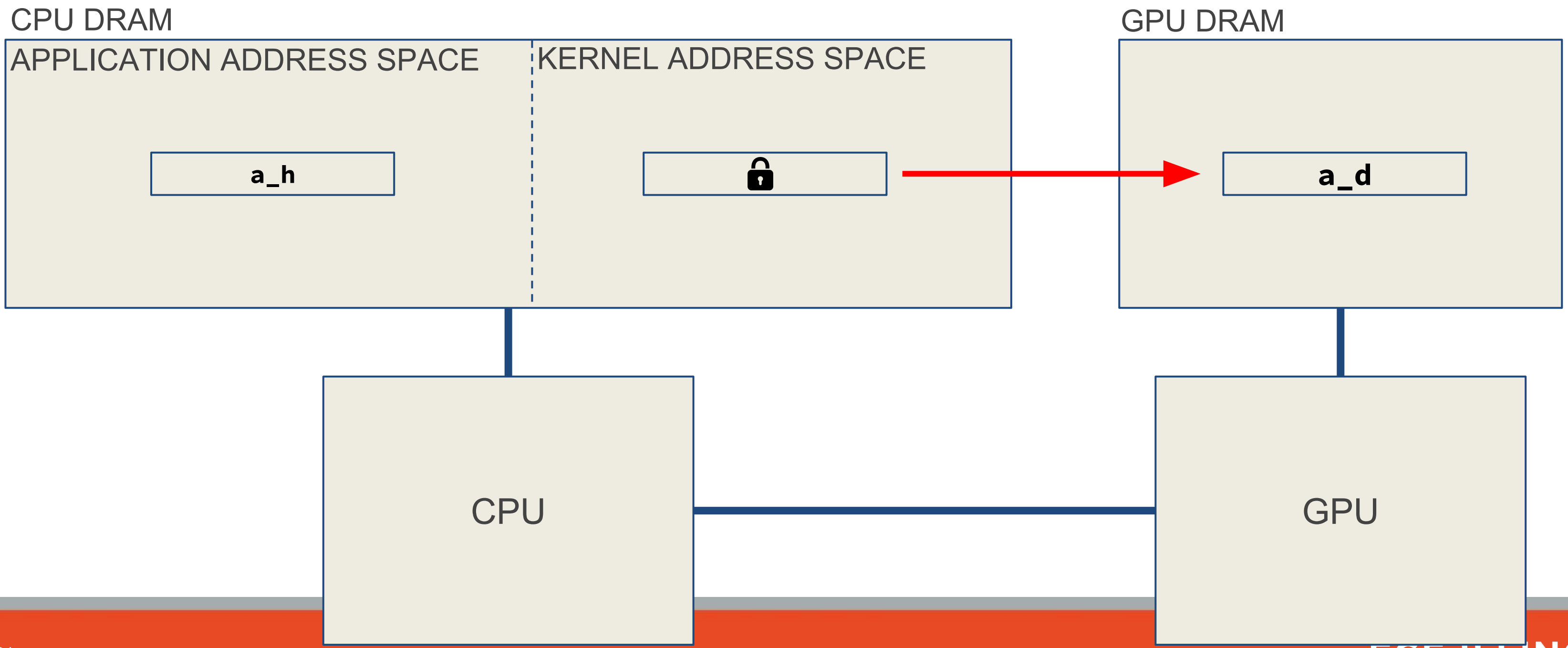
Pageable cudaMemcpy (3/4)

3) Driver copies to pinned internal buffer



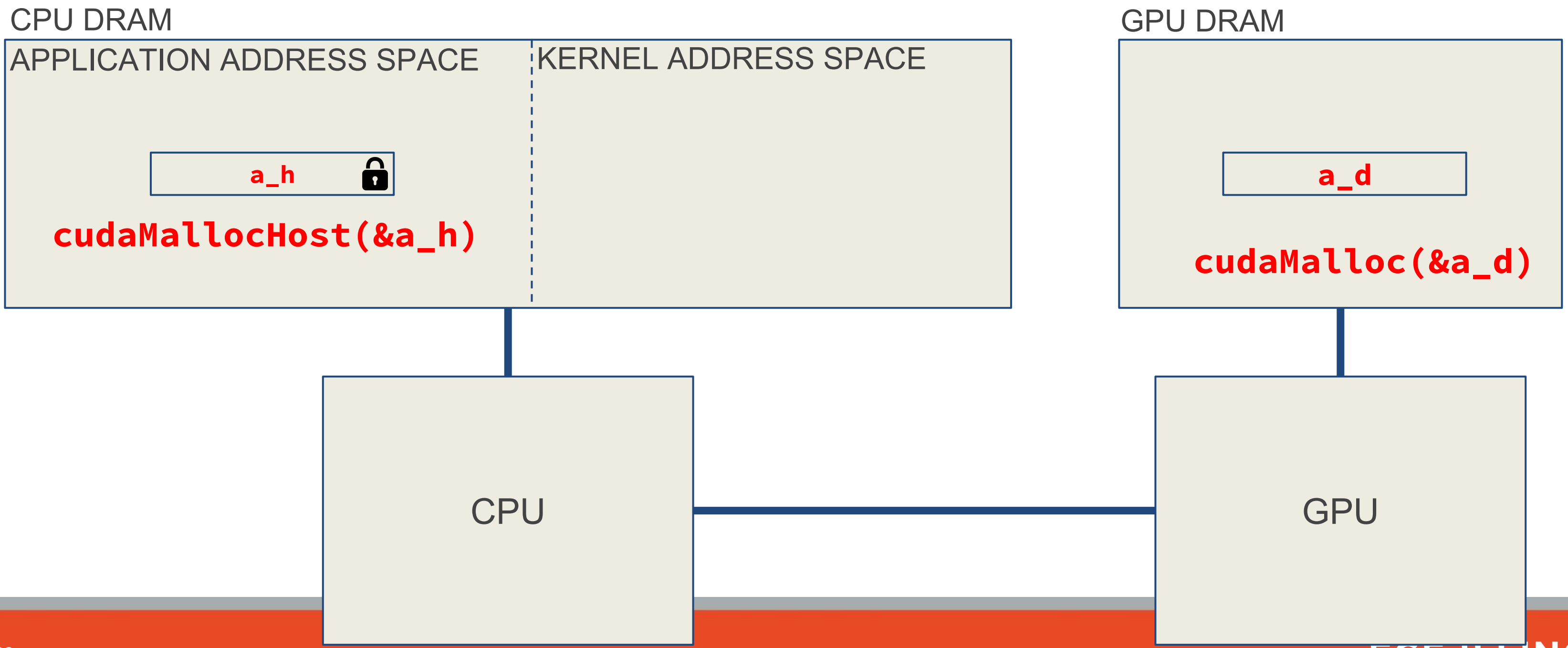
Pageable cudaMemcpy (4/4)

4) CPU instructs GPU to begin **D**irect **M**emory **A**ccess copy



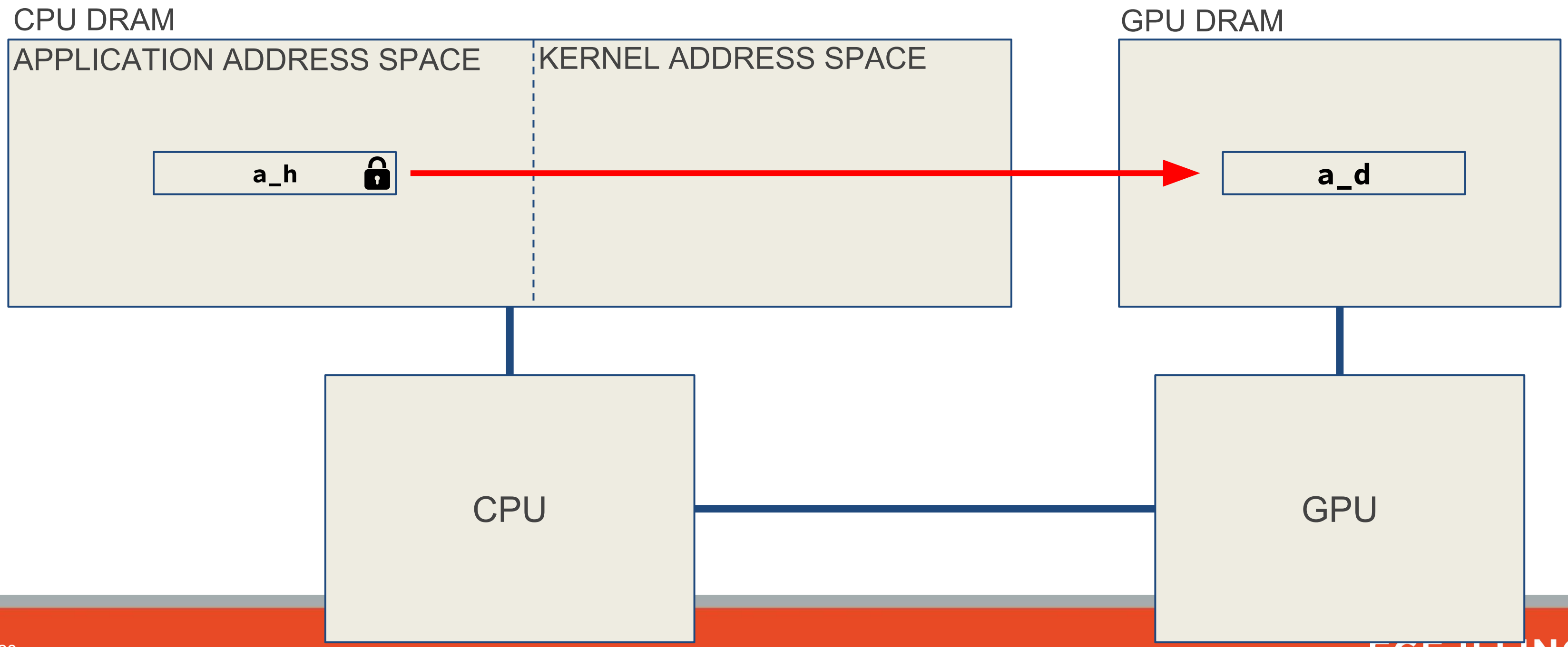
Pinned cudaMemcpy (1/2)

1) Allocate pinned memory

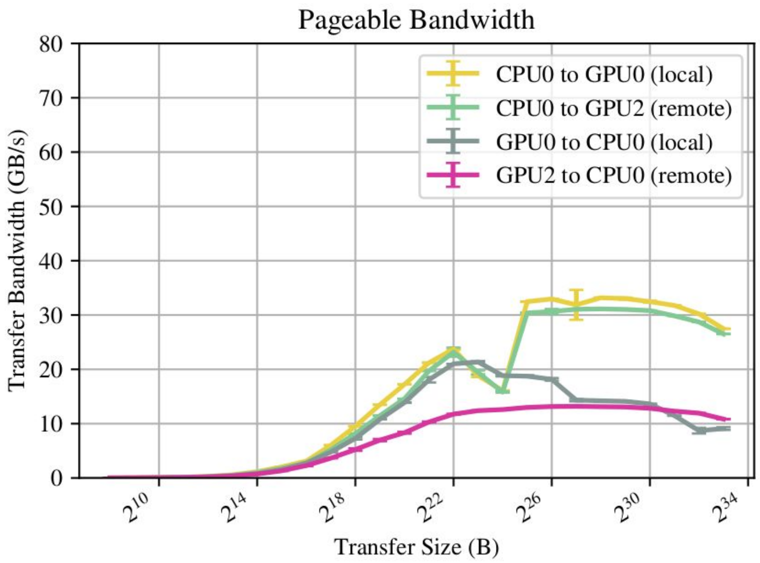


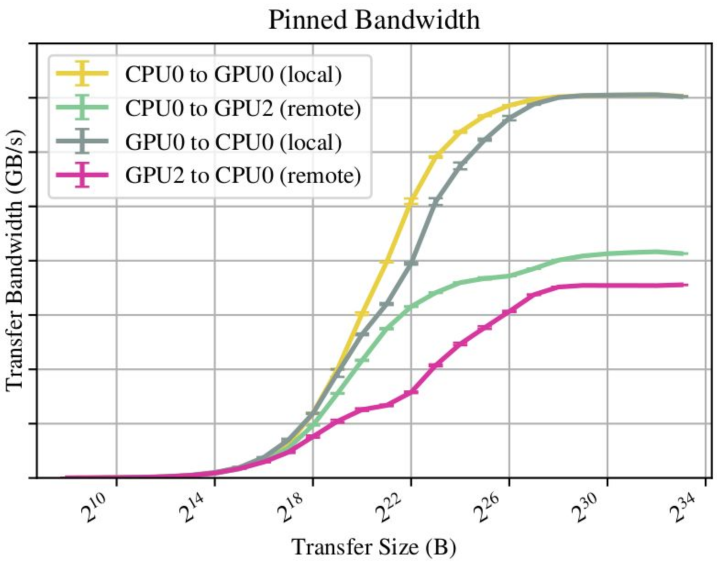
Pinned cudaMemcpy (2/2)

2) CPU instructs GPU to begin **D**irect **M**emory **A**ccess copy



CPU-to-GPU Transfers from Pageable Allocations



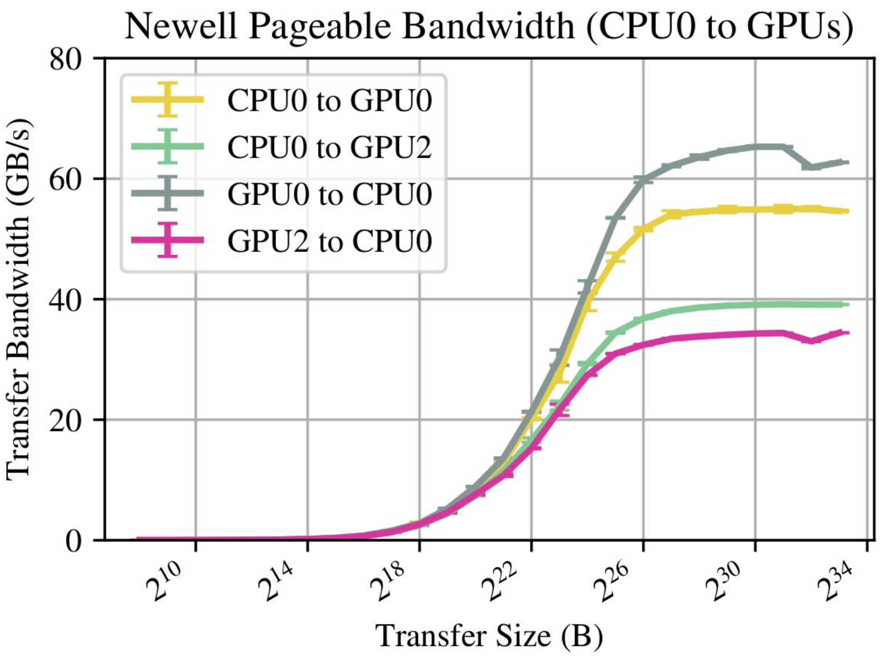


Pageable copies introduce strange performance

Transfer Anisotropy

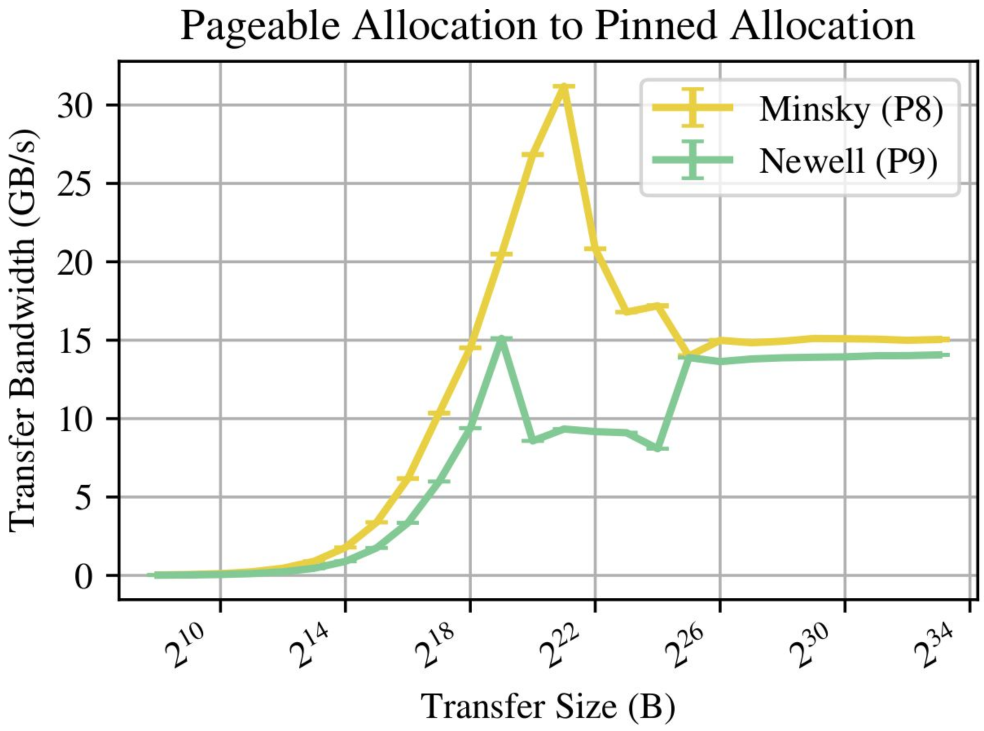
Local: GPU-to-CPU is faster

Remote: CPU-to-GPU is faster

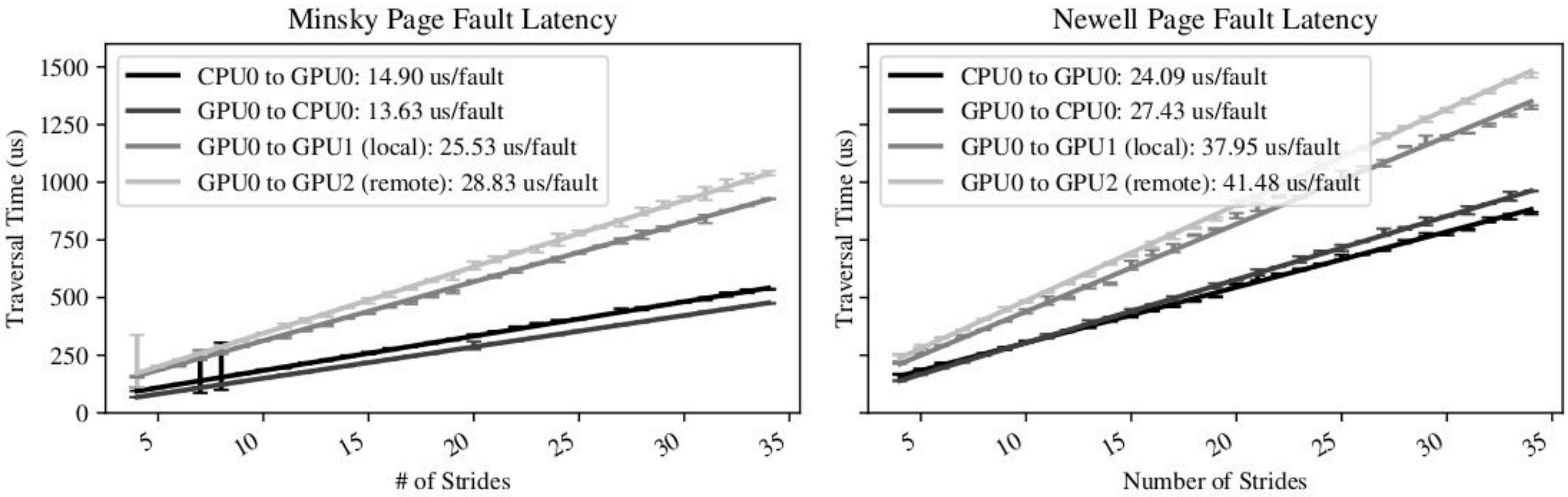


Intra-CPU CudaMemcpy()

P9 single-thread memory copy bandwidth lower than P8



Page Fault Latency



P9 higher page fault latency than P8 (no ATS)

Google Benchmark Lessons Learned

- Multithreaded Benchmarks
  - No built-in sync, ended up using OpenMP
- Needs some hints about computing reasonable runtime when CPU time >> wall time
- Benchmark function can only take integer arguments
  - Can't pass in a set of GPU ids, for example

Release Plan

- Pre-release version available now
  - `github.com/rai-project/microbench`
- 1.0 (this summer)
  - Unified and explicit memory
  - Plotting
  - PCIe / NVLink, POWER / x86, Pascal / Volta
- 1.x
  - Collective communication and contention
- 2.0
  - Neural network & Tensorcore primitives
  - Website with hosted results
- 2.x
  - Disk / network / multi-node

Future Directions

- Sanity check for system developers
- Empirical data for machine performance models

Summary

- CUDA / NUMA communication microbenchmarks
  - github.com/rai-project/microbench
  - github.com/rai-project/microbench_plot
- Some unexpected results
  - Need for open, comprehensive measurement techniques
- Underlying communication primitives
  - Sanity checks
  - Performance models

Thank You

pearson@illinois.edu

References

[1] https://github.com/sakra/cotire

[2] https://github.com/ruslo/hunter

[3] Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015, March). *An updated performance comparison of virtual machines and linux containers*. In Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On (pp. 171-172). IEEE.