ComP-Net

Command Processor Networking for Efficient Intra-kernel Communications on GPUs

Michael LeBeane[1,2], Khaled Hamidouche[1], Brad Benton[1], Mauricio Breternitz[3], Steven K. Reinhardt[4], Lizy K. John[2]

[1] Advanced Micro Devices Inc., [2] The University of Texas at Austin, [3] INSEC-ID Lisboa, [4] Microsoft Corporation,

Presented by Sooraj Puthoor[1]

GPUS AND NETWORKS IN THE WILD

Introduction

- GPUs are everywhere in HPC, machine learning, and beyond
  - Excellent performance/watt for many classes of data-parallel computation

- Many GPUs are required to solve the biggest computational problems
  - Can only fit so many GPUs in a single node!
  - GPUs talk to each other through Network Interface Controllers (NICs)
  - Path between GPU and NIC must be efficient

- Vendors are selling machines filled with many GPUs and NICs
  - Inventec Project 47 Node
    - **4** Radeon Instinct GPUs
    - **2** Mellanox 100G NICs
    - **1** EPYC 7601 32-Core CPU
    - **2:1** GPU/NIC Ratio

OVERVIEW OF GPU NETWORKING

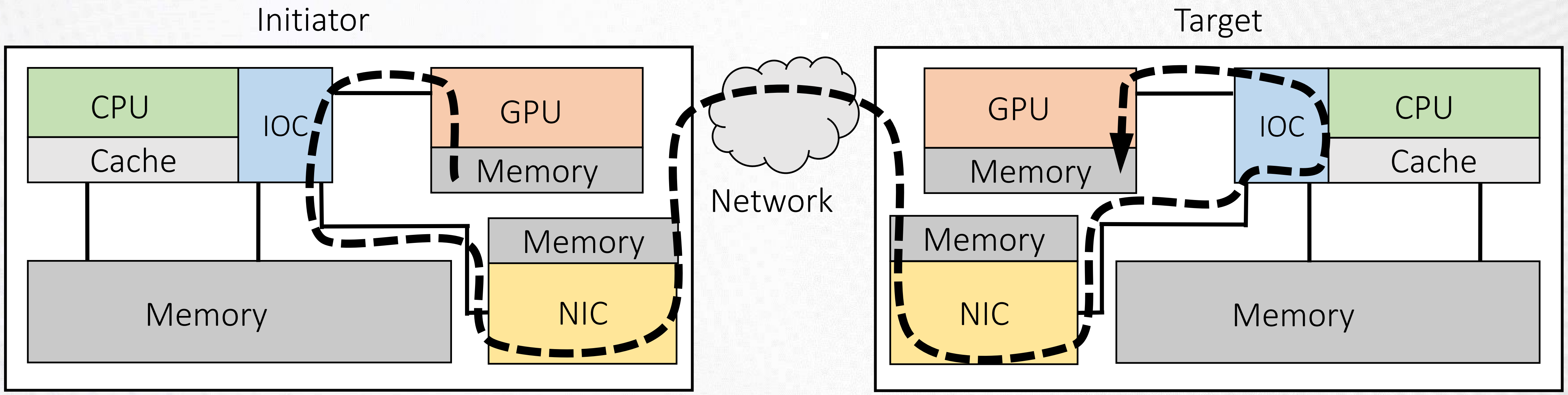
Introduction

- Much industry and academic work in the area
- Can largely be broken down into two domains:
  - ***Data Path***
    - i.e., where the data that goes across the network flows
  - ***Control Path***
    - i.e., who tells the NIC to move the data across the network

DATA PATH OPTIMIZATIONS

Introduction

- Direct path from discrete GPU memory to NIC
  - No bounce buffers or host memory copies
  - Implemented in Mellanox's PeerDirect interface for their NICs
  - Used by AMD's ROCn RDMA[1] and Nvidia's GPUDirect RDMA[2]



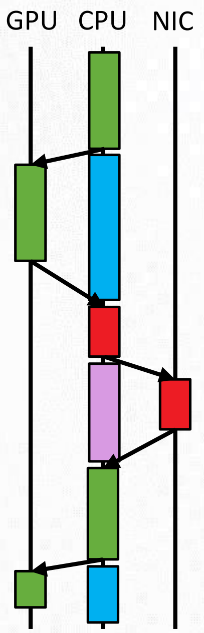
[1] AMD. 2017. ROCn RDMA. https://github.com/RadeonOpenCompute/ROCnRDMA
[2] Mellanox. 2017. Mellanox OFED GPUDirect RDMA. http://www.mellanox.com/page/products_dyn?product_family=116

CONTROL PATH OPTIMIZATIONS

Introduction

Host Driven Networking

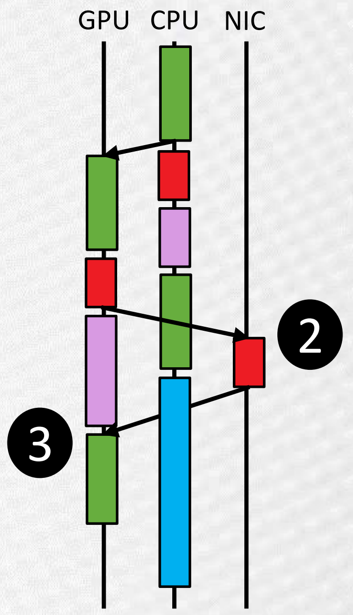
1
```
a_kernel<<<…, stream>>>(buf);
cudaStreamSynchronize(stream);
```
2
```
netSend(buf);
netWait();
```
3
```
b_kernel<<<…, stream>>>(buf);
cudaStreamSynchronize(stream);
```



1. CPU schedules kernel and waits for completion
2. CPU posts network operation and waits for completion
3. CPU schedules and waits on final kernel

GPU Direct Async (GDS)[3]

1
```
a_kernel<<<…, stream>>>(buf);
netPost_async(stream, qp, buf);
netWait_async(stream, txcq);
b_kernel<<…, stream>>(buf);
cudaStreamSynchronize(stream);
```



1. CPU schedules kernel, network operation, and, final kernel
2. GPU triggers initiation of a network operation after kernel
3. GPU launches final kernel

- GDS removes the CPU from the critical path and avoids control flow switches
- Communication events triggered at kernel boundaries

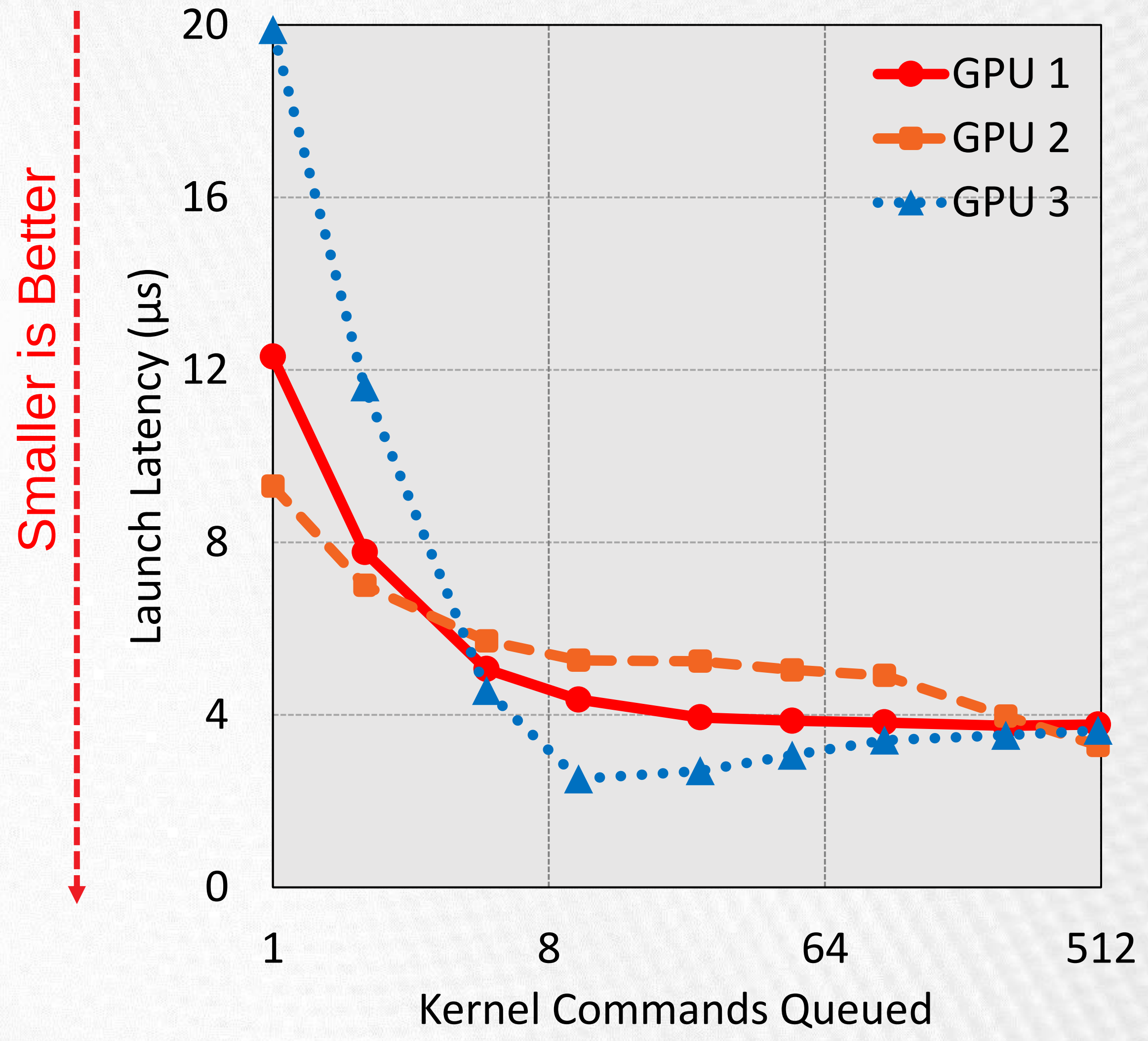
[3] Elena Agostini, Davide Rossetti, and Sreeram Potluri. 2017. Offloading communication control logic in GPU accelerated applications. In Intl. Symp. on Cluster, Cloud and Grid Computing (CCGrid). DOI:https://doi.org/10.1109/CCGRID.2017.

OVERHEAD OF KERNEL BOUNDARY COMMUNICATION

Introduction

- **Kernel launch latencies much higher than HPC network overheads!**
  - Can be up to 20µs for a kernel launch!
  - Compare that to < 1µs it takes to get to another node over the network

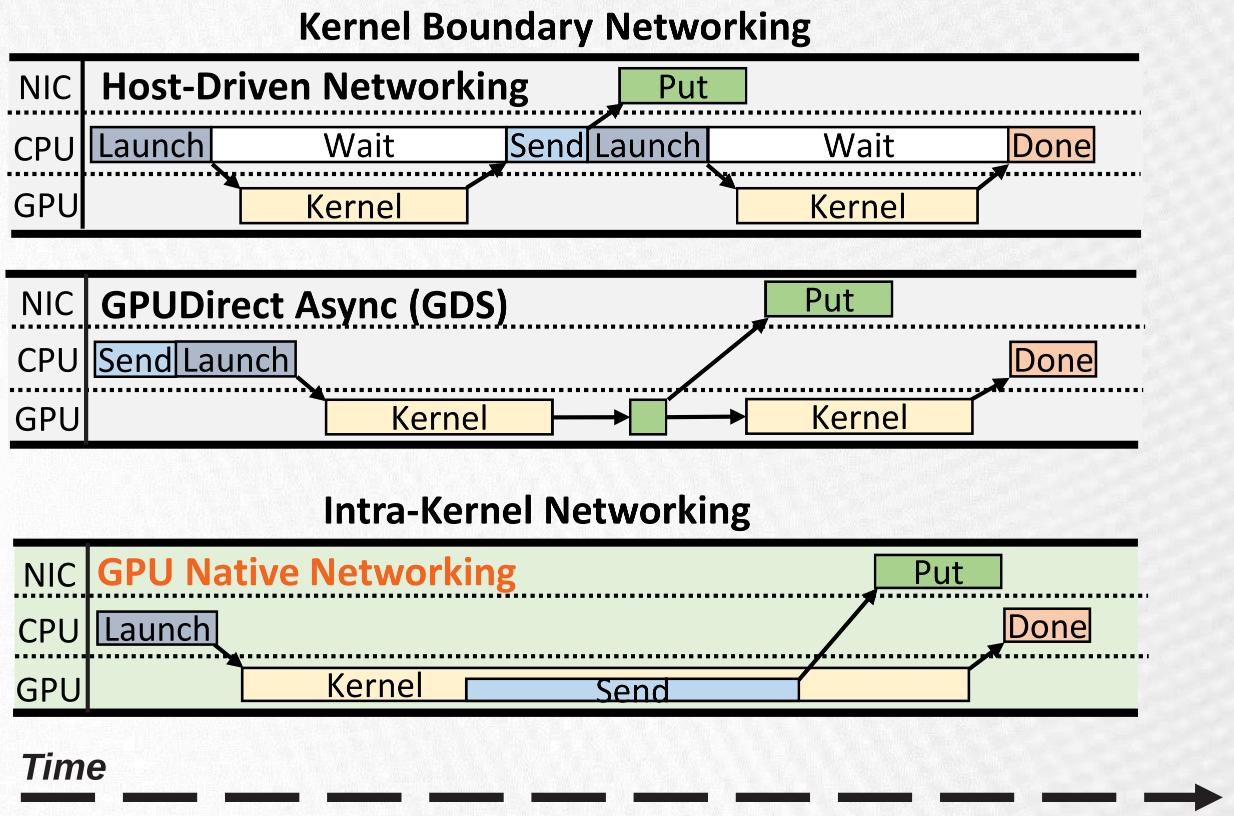
- **Obvious Solution**: Can you do networking from within a kernel?
  - Absolutely!
  - Two main schools of thought here…



GPU NATIVE NETWORKING[4, 5, 6, 7]

Introduction

- Run a networking stack on the GPU
- Allow the GPU work-items to directly interact with the network adaptor
- Pros
  - Completely decoupled from the CPU
  - Can be performant/low-latency
- Cons
  - Hard to talk to network interface designed for CPUs
  - Can suffer from significant control flow divergence and register pressure

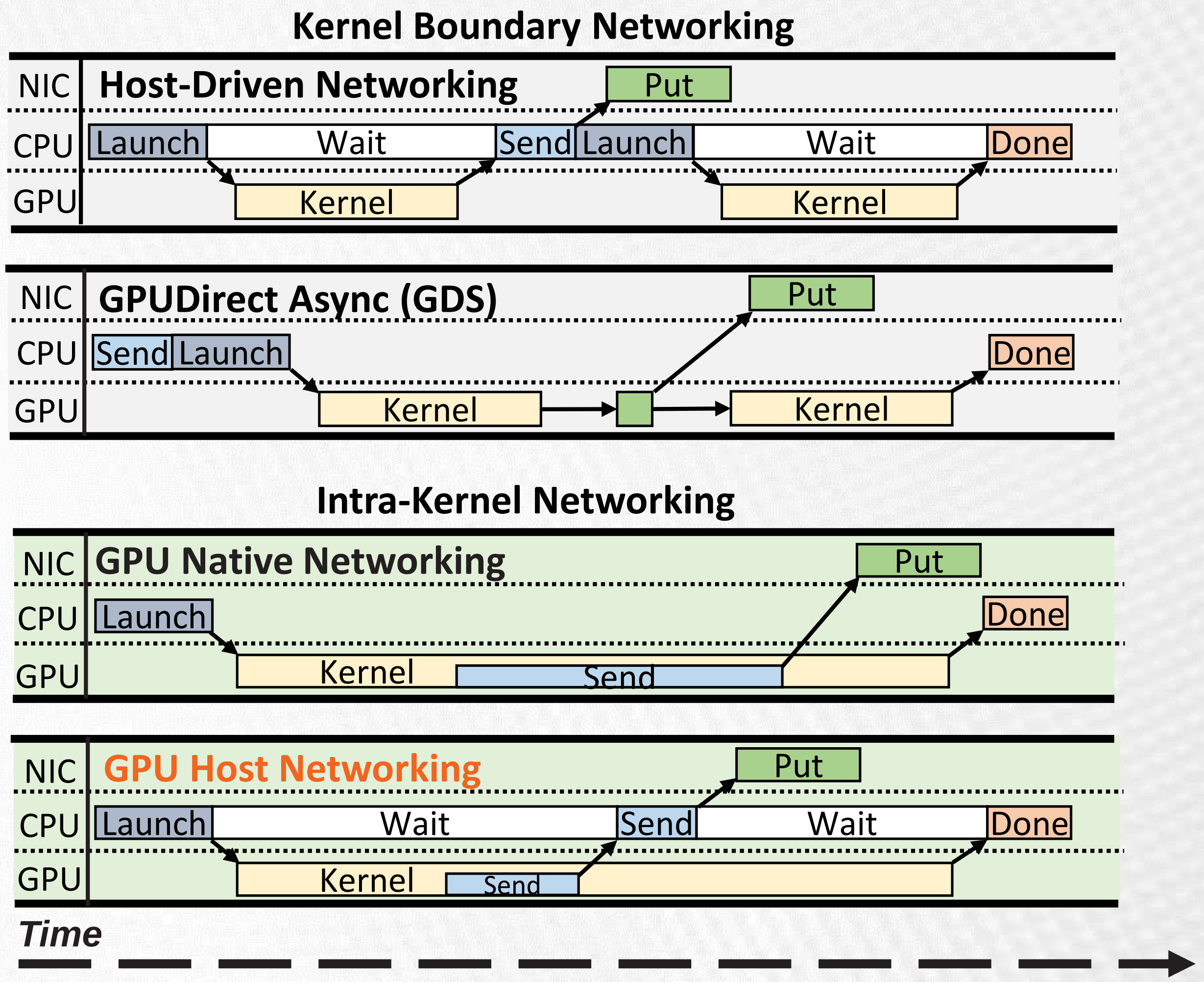


[4] Lena Oden, Holger Froning, and Franz-Joseph Pfreundt. 2014. Infiniband-Verbs on GPU: A Case Study of Controlling an Infiniband Network Device from the GPU. In Intl. Conf. on Parallel Distributed Processing Symposium Workshops (IPDPSW). 976–983.
[5] Benjamin Klenk, Lena Oden, and Holger Froning. 2014. Analyzing Put/Get APIs for Thread-Collaborative Processors. In Intl. Conf. on Parallel Processing (ICPP) Workshops.
[6] Benjamin Klenk, Lena Oden, and Holger Froning. 2015. Analyzing communication models for distributed thread-collaborative processors in terms of energy and time. In Intl. Symp. on Performance Analysis of Systems and Software (ISPASS).
[7] Feras Daoud, Amir Watad, and Mark Silberstein. 2016. GPUrdma: GPU-side Library for High Performance Networking from GPU Kernels. In Intl. Workshop on Runtime and Operating Systems for Supercomputers (ROSS). 6:1–6:8.

GPU HOST NETWORKING[8, 9, 10]

Introduction

- Run your networking stack on the CPU
- Have the GPU place network requests in a producer/consumer queue for the CPU
- Use threads on the CPU to process messages and synchronize with system atomics
- Pros
  - Lots of flexibility on the CPU to improve performance through coalescing, etc.
- Cons
  - Additional latency imposed by the indirection
  - Scales poorly with more or bigger GPUs

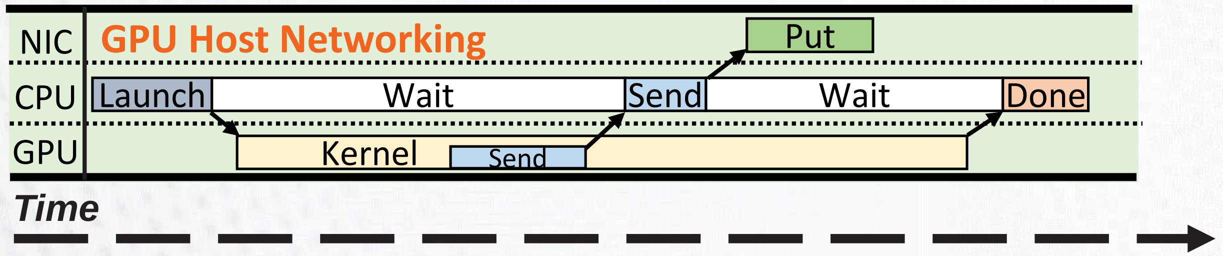


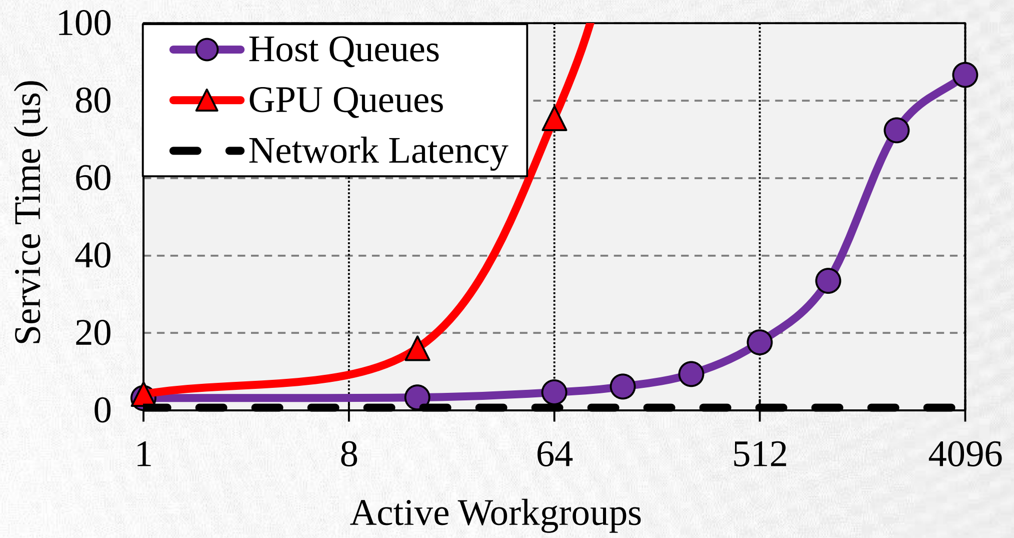
[8] Jeff A. Stuart and John D. Owens. 2009. Message passing on data-parallel architectures. In Intl. Symp. on Parallel Distributed Processing (IPDPS). 1–12.
[9] Sangman Kim, Seonggu Huh, Yige Hu, Xinya Zhang, Emmett Witchel, Amir Wated, and Mark Silberstein. 2014. GPUnet: Networking Abstractions for GPU Programs. In USENIX Conf. on Operating Systems Design and Implementation (OSDI). 201–216.
[10] Tobias Gysi, Jeremia Bär, and Torsten Hoefler. 2016. dCUDA: Hardware Supported Overlap of Computation and Communication. In Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC) (SC '16). Article 52, 12 pages.

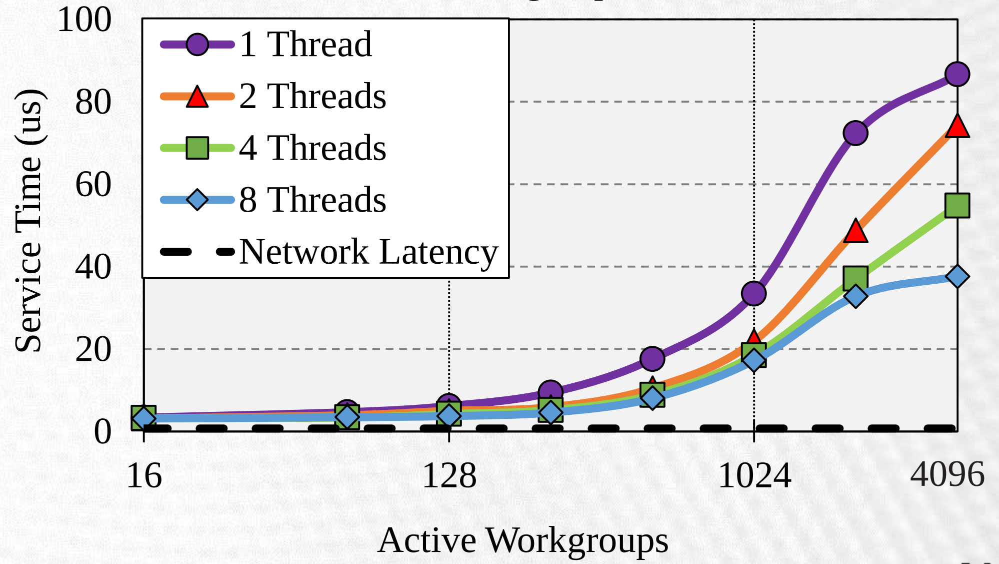
PERFORMANCE PROBLEMS WITH GPU HOST NETWORKING

Introduction

- Need multiple trips over IO bus
- Where to place queues?
  - GPU memory vs. host memory
  - High latency in both cases
- Not scalable
  - 40µs latency with 8 threads
  - Bigger/more GPUs reduce scalability



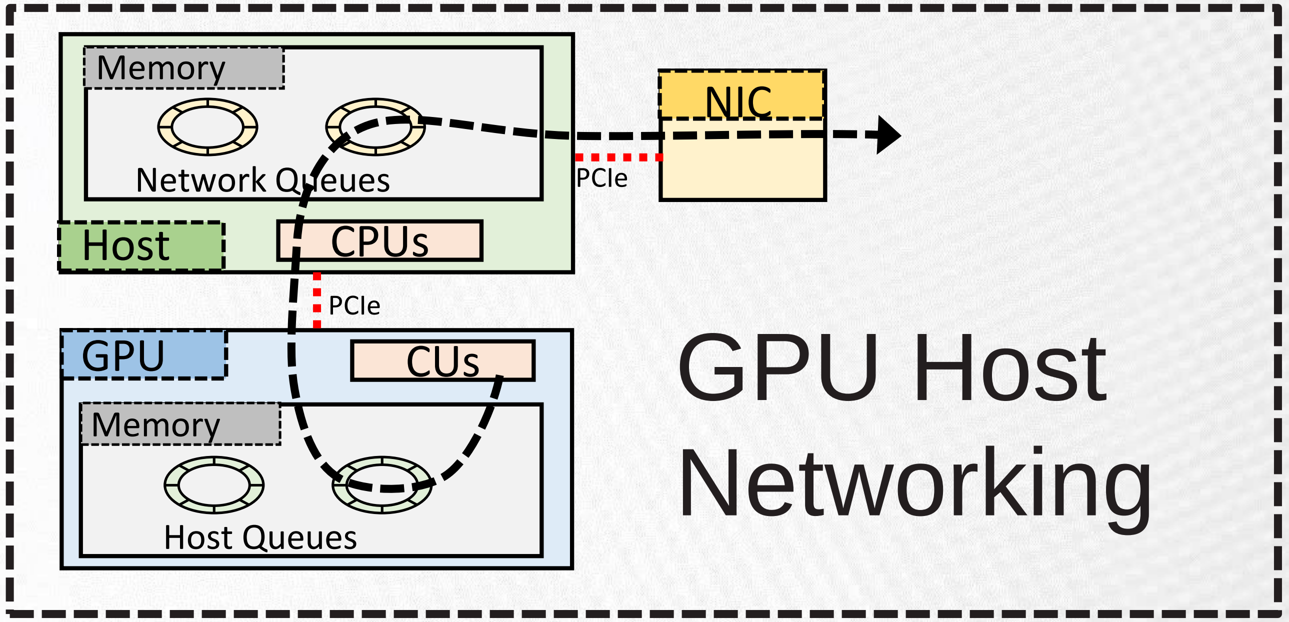


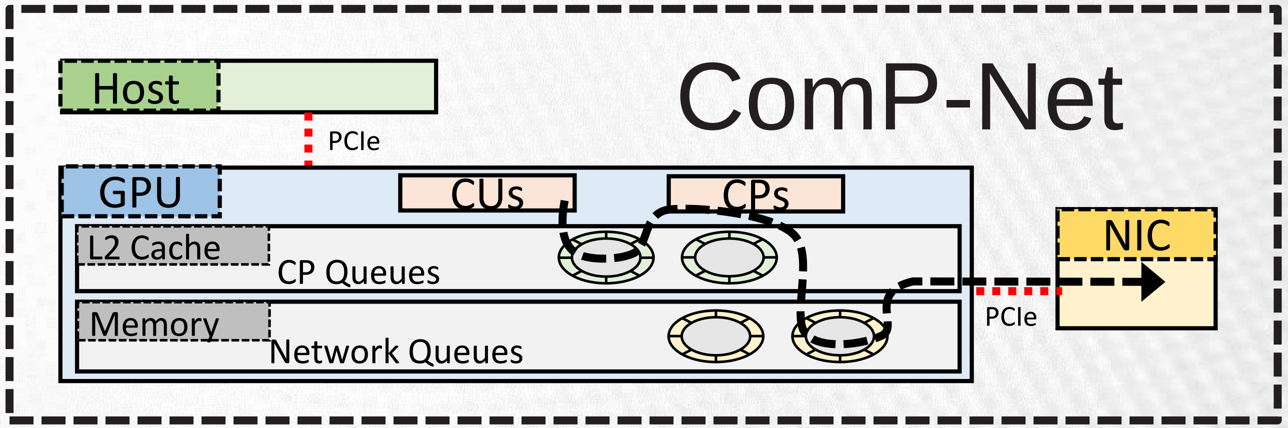


COMMAND PROCESSOR NETWORKING (COMP-NET) OVERVIEW

ComP-Net

- Uses built-in CP to support network operations
- CP/GPU communicate over shared L2 cache instead of PCIe
- Potentially much faster (lower latency) than other GPU Host Networking designs
- CP resources can scale with other GPU resources



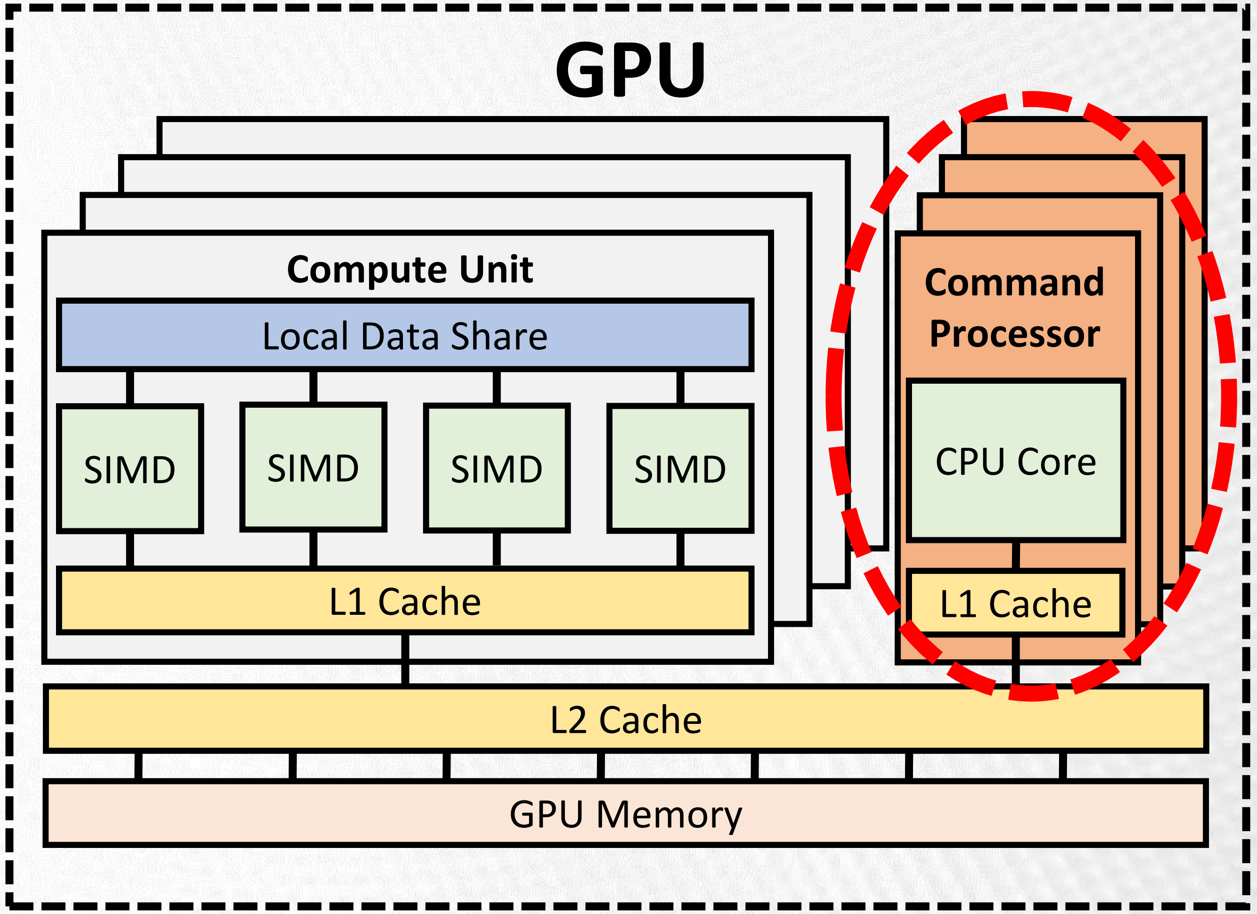


COMMAND PROCESSOR OVERVIEW

Introduction

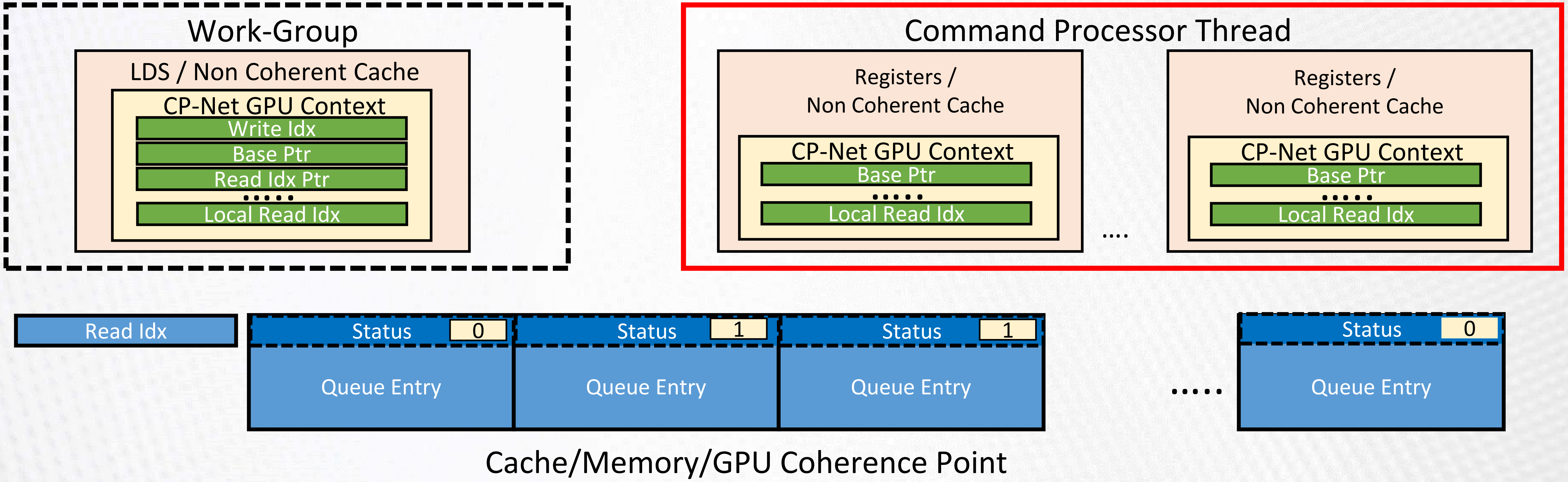
- **GPUs have built-in CPUs called Command Processors (CPs)**
  - Scalar cores == good at running network runtime code
  - Can connect to GPU CUs through a shared LLC
- **Traditionally used to launch kernels**
  - But intra-kernel networking encourages fewer kernels…..

Can we leverage CPs for intra-kernel networking?



COMP-NET PRODUCER/CONSUMER QUEUE

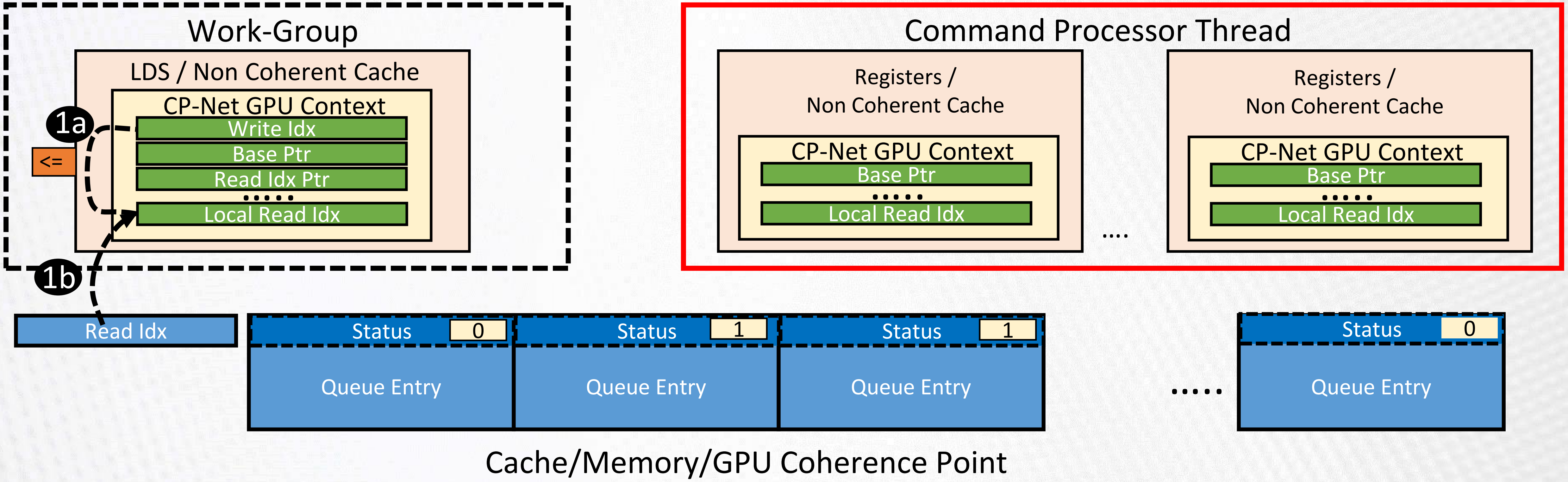
ComP-Net



- Main component of ComP-Net Runtime is CP/GPU producer/consumer queue
- Most steps are straightforward

COMP-NET PRODUCER/CONSUMER QUEUE

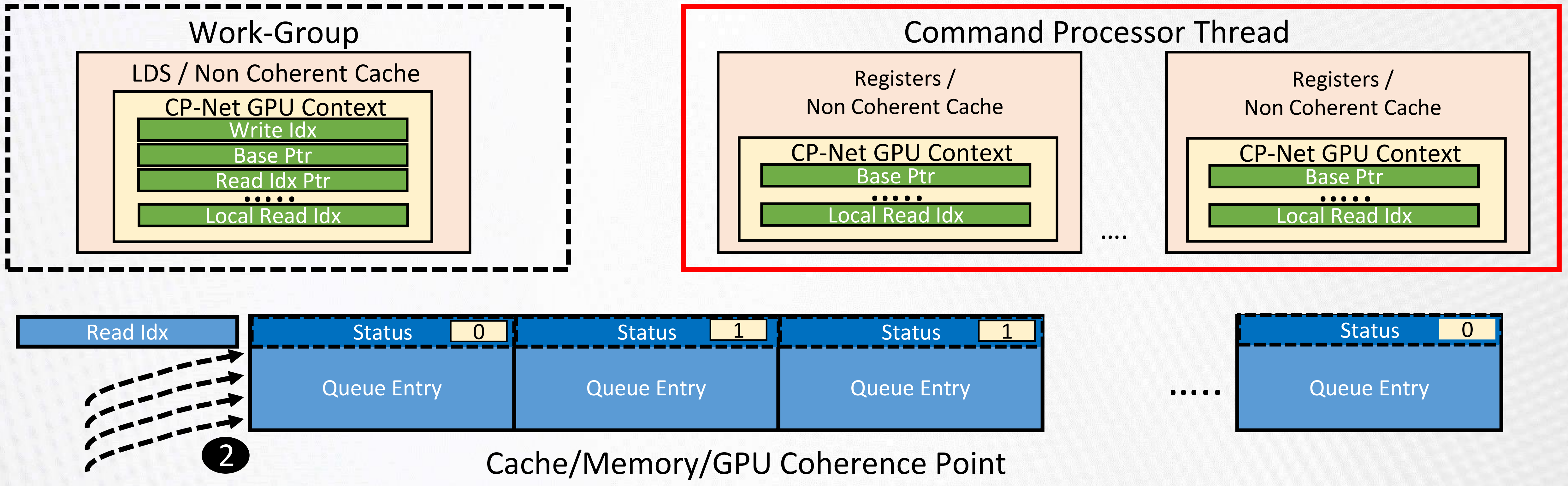
ComP-Net



- **1a)** Check if queue is full (using local *Read Idx*)
- **1b)** If full, update *Read Idx* and loop till not full

COMP-NET PRODUCER/CONSUMER QUEUE

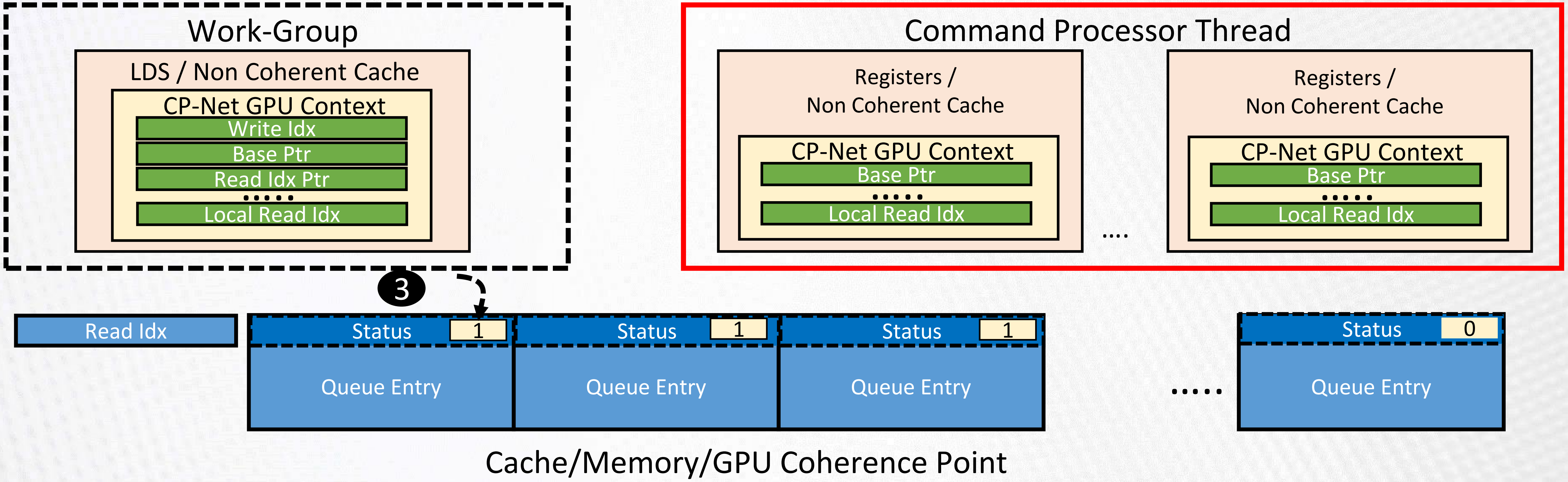
ComP-Net



- **2)** Fill *Queue Entry* with networking metadata
  - Or Inline small payloads in the *Queue Entry* itself

COMP-NET PRODUCER/CONSUMER QUEUE

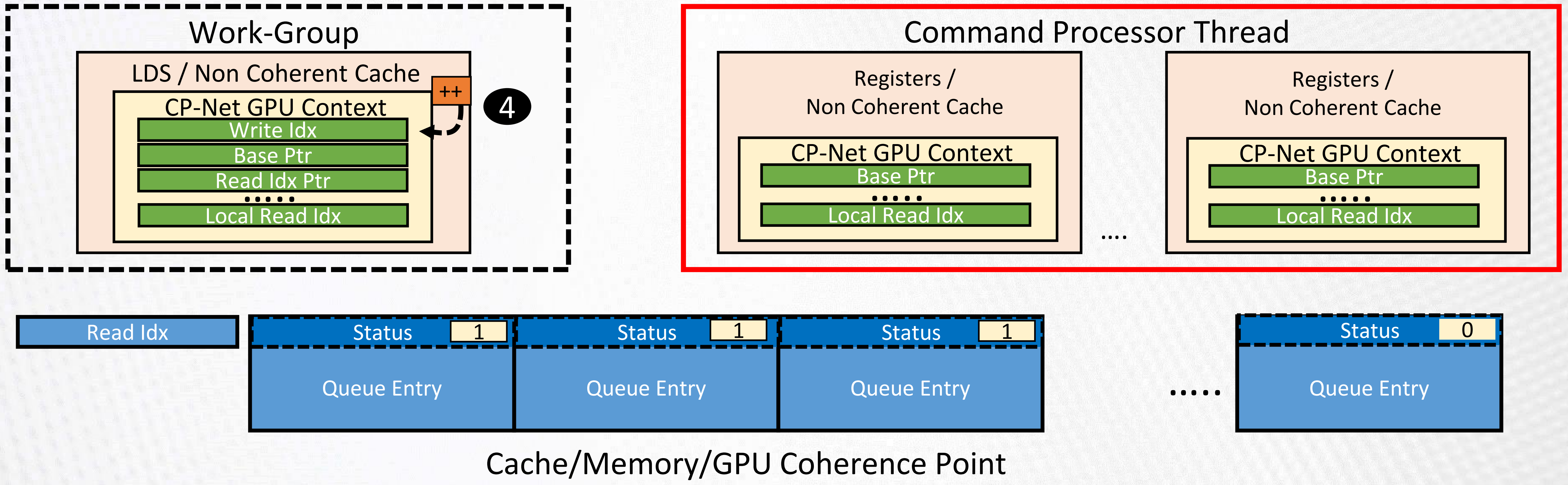
ComP-Net



- **3)** Set *Status* flag with release marker to notify CP

COMP-NET PRODUCER/CONSUMER QUEUE

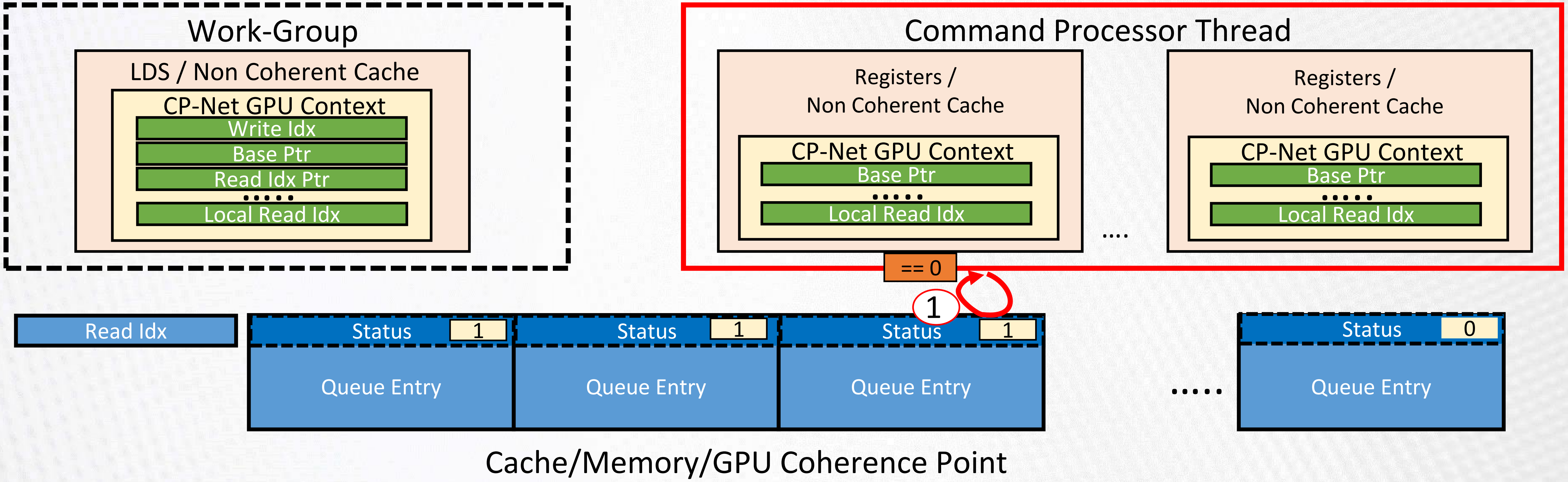
ComP-Net



- **4)** Increment local *Write Idx*

COMP-NET PRODUCER/CONSUMER QUEUE

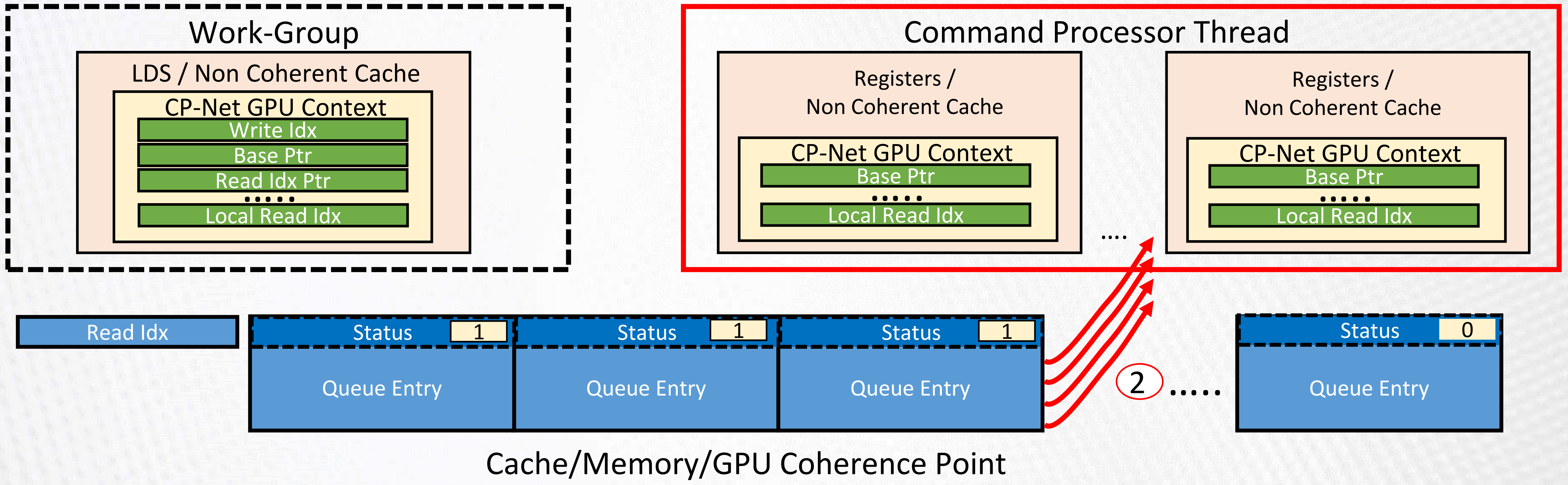
ComP-Net



- **1)** Poll on next *Queue Entry* based on local *Read Idx* with acquire marker

COMP-NET PRODUCER/CONSUMER QUEUE

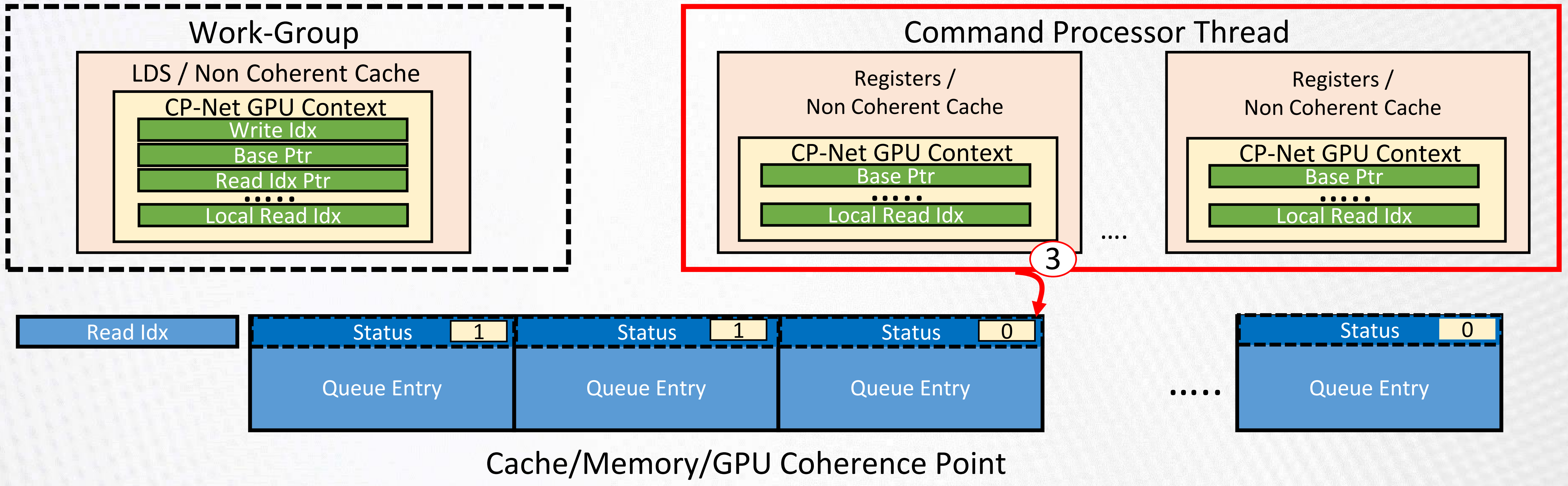
ComP-Net



- **2)** Read data from *Queue Entry*

COMP-NET PRODUCER/CONSUMER QUEUE

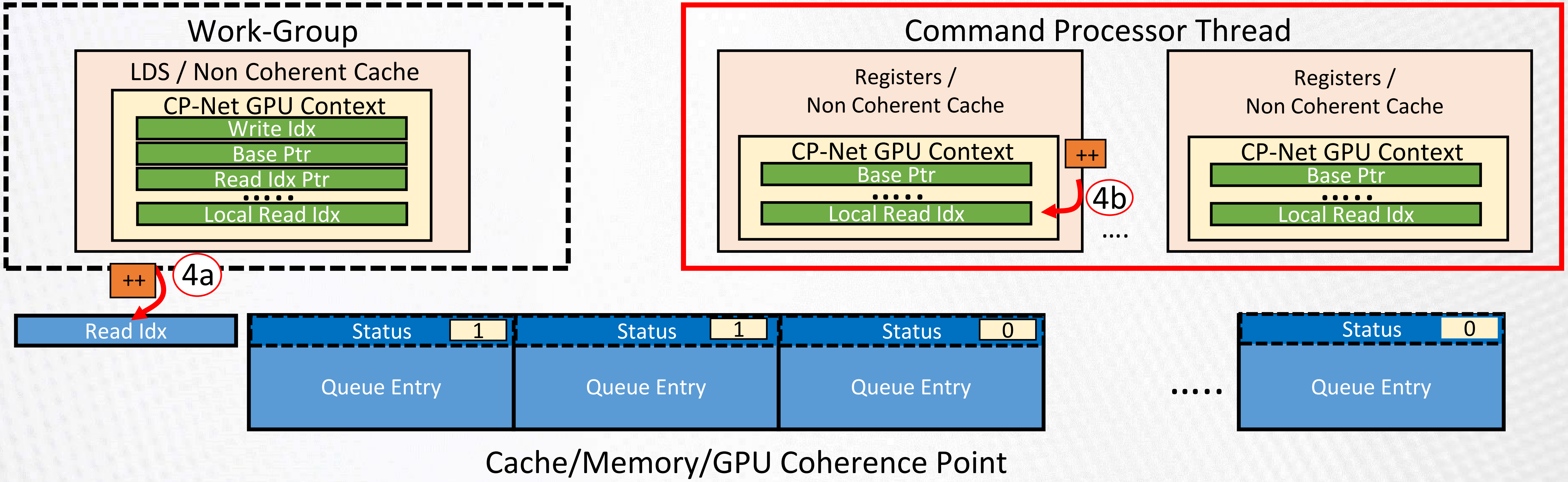
ComP-Net



- **3)** Perform network operation and set *Status* flag to 0 when complete with release marker

COMP-NET PRODUCER/CONSUMER QUEUE

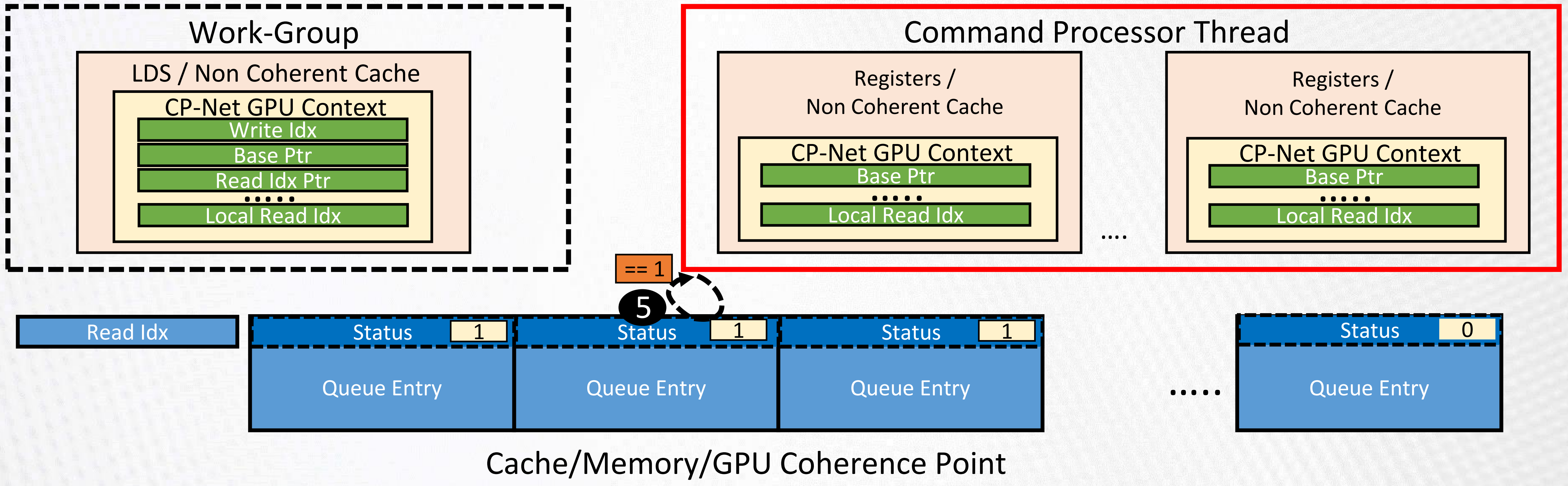
ComP-Net



- **4a)** Update global *Read Idx* with release marker
- **4b)** Update local *Read Idx*

COMP-NET PRODUCER/CONSUMER QUEUE

ComP-Net

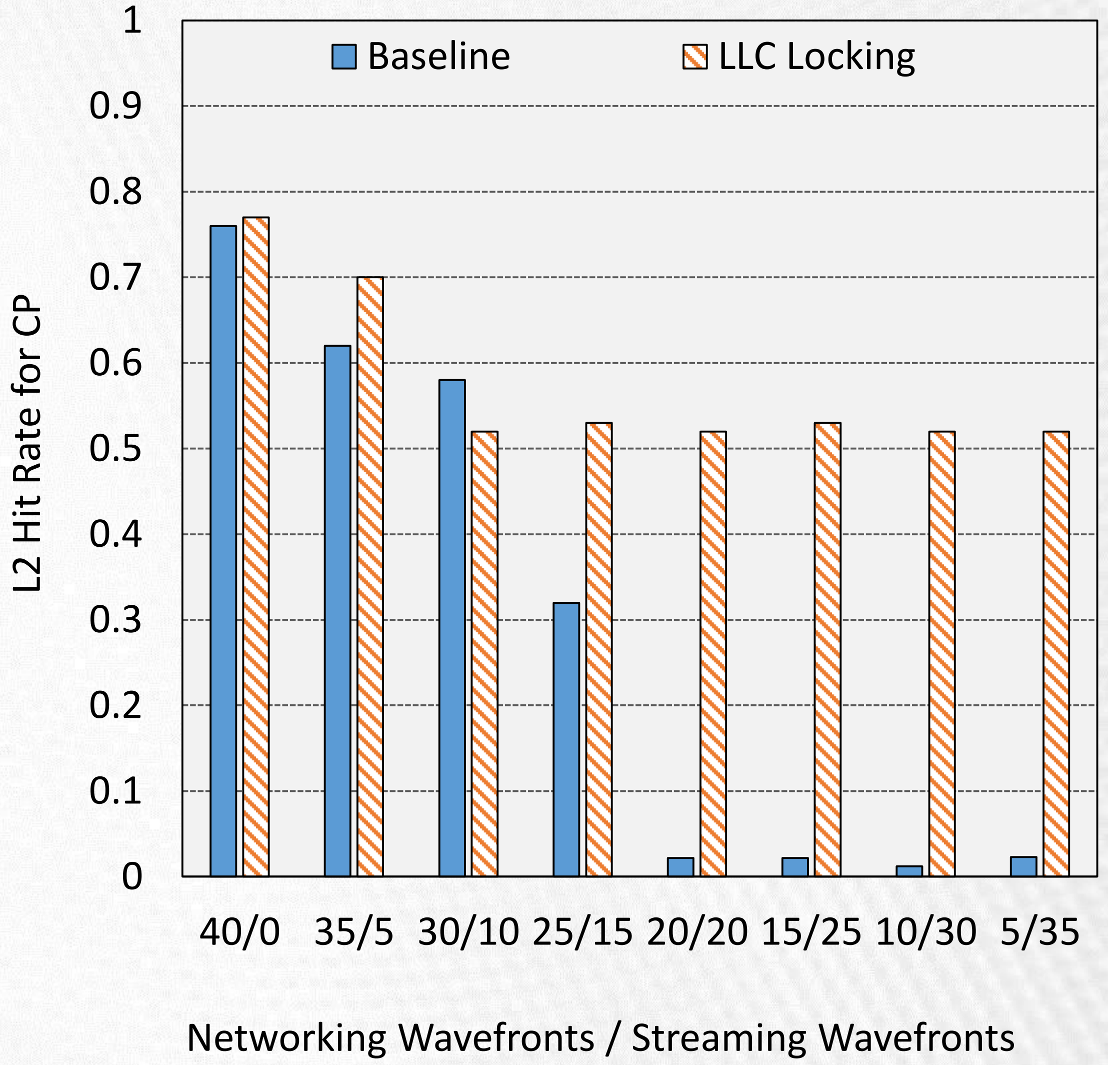


- **5)** Check *Status* bit to determine when CP completes operation

TACKLING GPU CACHE THRASHING

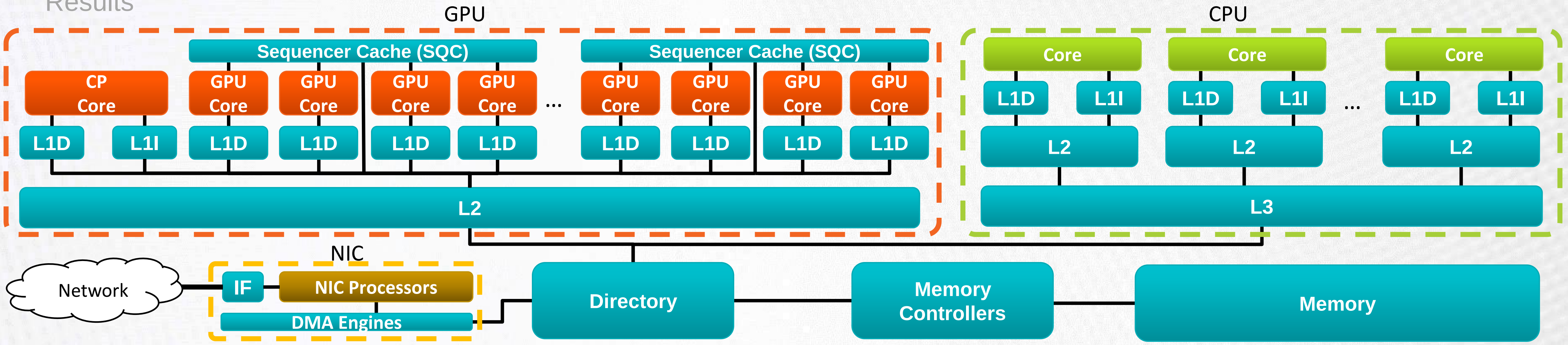
ComP-Net

- Residency of data in GPU L2 cache is very short
- Work-group data produced for CP is evicted when other work-groups perform streaming memory accesses
- Can be solved through cache line locking
  - Preliminary results are promising
  - Still much to explore here



SIMULATION ENVIRONMENT

Results



- **gem5 [11] + AMD GCN3 GPU model [12] + Internal Portals4 NIC model**
  - CPU power model with McPAT [13]
  - Baseline model is coherent APU
    - dGPU modeled with extra delay for I/O bus and by disabling coherence probes

[11] N. Binkert, et. al. 2011. The gem5 simulator. SIGARCH Comput. Archit. News 2001.
[12] A. Gutierrez *et al*., "Lost in Abstraction: Pitfalls of Analyzing GPUs at the Intermediate Language Level," *HPCA 2018.*
[13] S. Li. et. al. , "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," *MICRO 2009*

SIMULATOR CONFIGURATION

Results

- CPU: Standard CPU-only systems
  - Baseline non-accelerated system
- HDN: Host Driven Networking
  - Kernel boundary networking (host MPI + HIP)

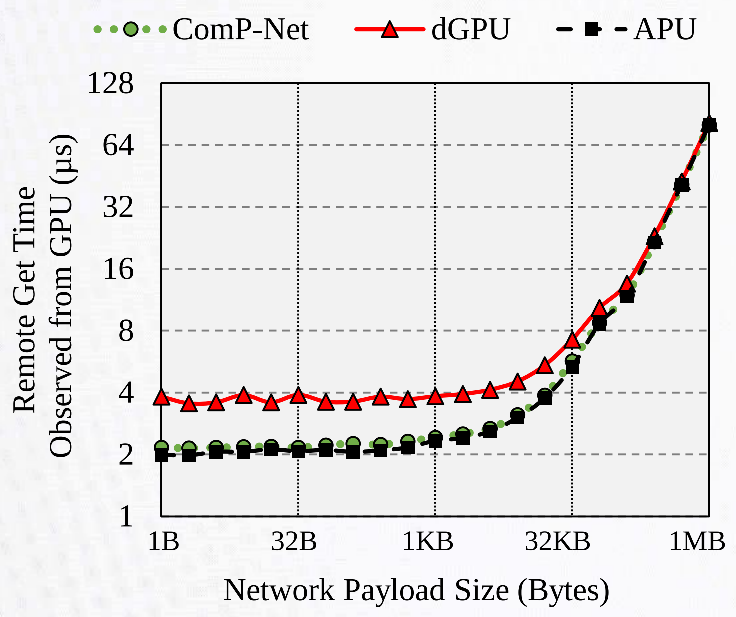
__Intra-kernel Networking Schemes:__

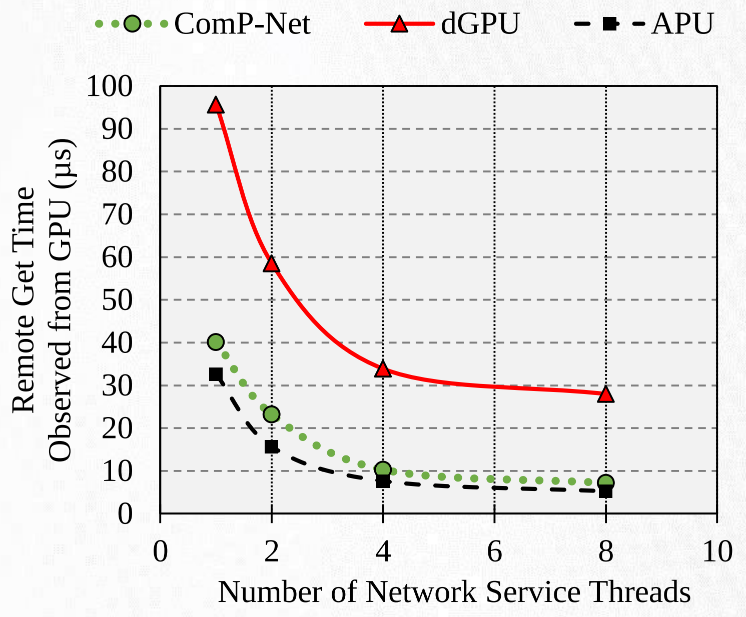
- APU: CPU/GPU on the same die
  - Intra-kernel networking through host threads on an APU
- dGPU: GPU Host Networking
  - Intra-kernel networking on a dGPU via host threads
- ComP-Net: Command Processor Networking
  - Intra-kernel networking through command processor

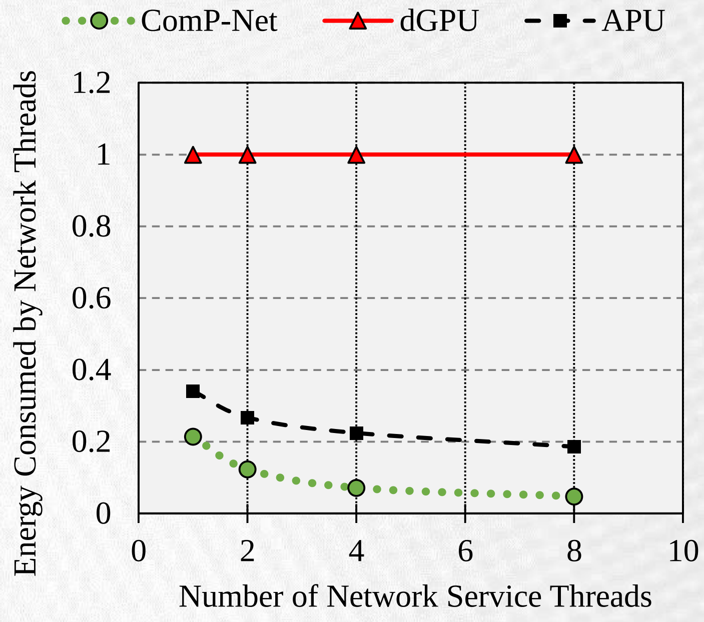
CPU and Memory Configuration	
Type	8-wide OOO, x86, 8 cores @ 4GHz
I,D-Cache	64KB, 2-way, 2 cycles
L2-Cache	2MB, 8-way, 8 cycles
L3-Cache	16MB, 16-way, 20 cycles
DRAM	DDR4, 8 Channels, 2133MHz
GPU Configuration	
Type	AMD GCN3 @ 1.5GHz
CU Config	12 CUs with 4 SIMD-16 engines
Wavefronts	40 Waves per SIMD (64 lanes)
V-Cache	32KB, 16-way, 12 cycles, per CU
K-Cache	32KB, 8-way, 12 cycles, per 4 CU
I-Cache	64KB, 8-way, 12 cycles, per 4 CU
L2-Cache	1MB, 16-way, 8 banks, 100 cycles
CP Configuration	
Type	2-wide OOO, x86, 2 cores @ 2GHz
D-Cache	32KB, 8-way, 4 cycles
I-Cache	16KB, 8-way, 4 cycles

MICROBENCHMARKS

Results







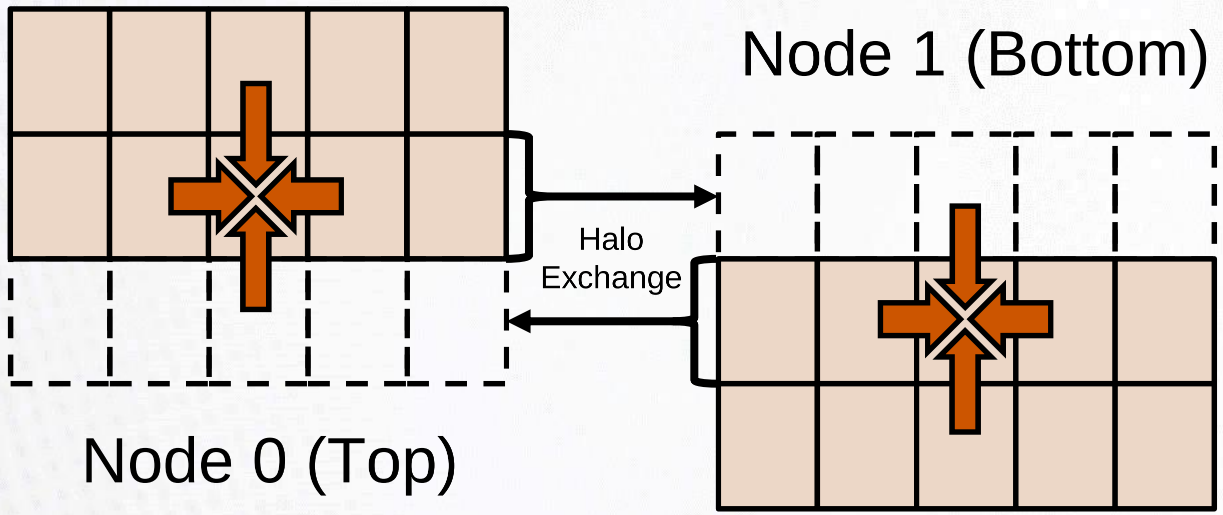
- Sweep of message size (single networking WG)
- Round trip network latency

- Sweep of number of network service threads (many networking WGs)
- Round trip network latency

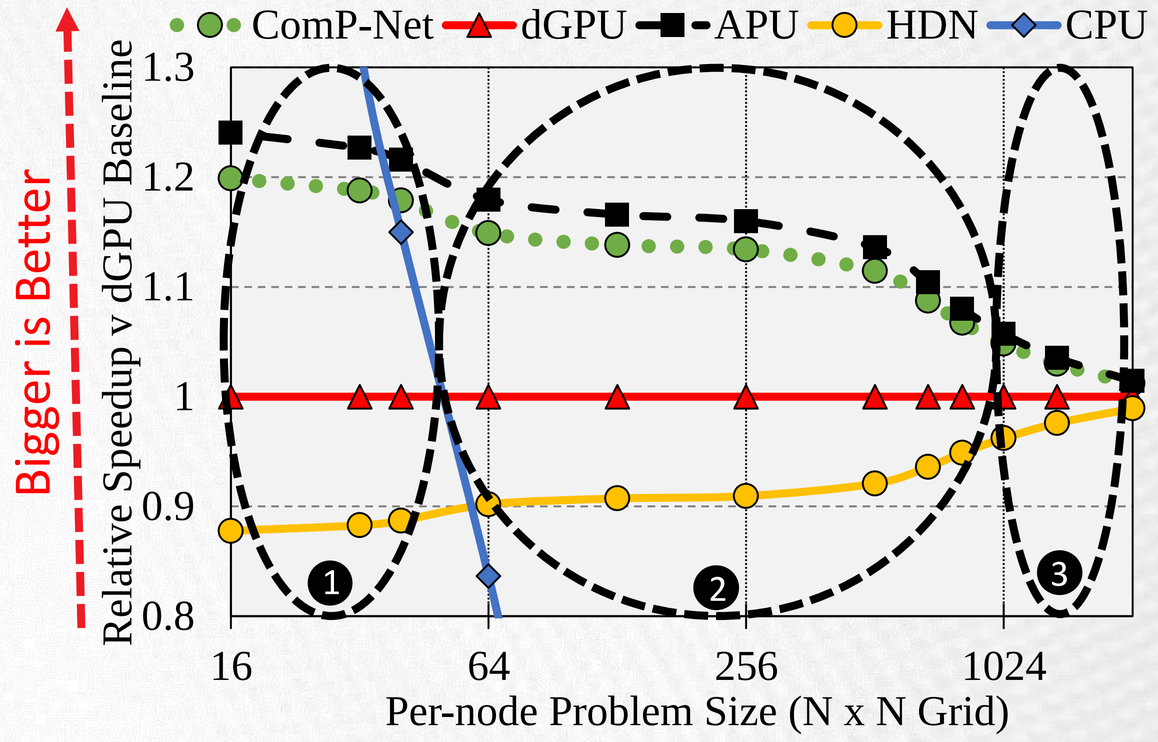
- Sweep of number of network service threads (many networking WGs)
- Energy consumption of CP/CPU threads

2D JACOBI STENCIL

Results



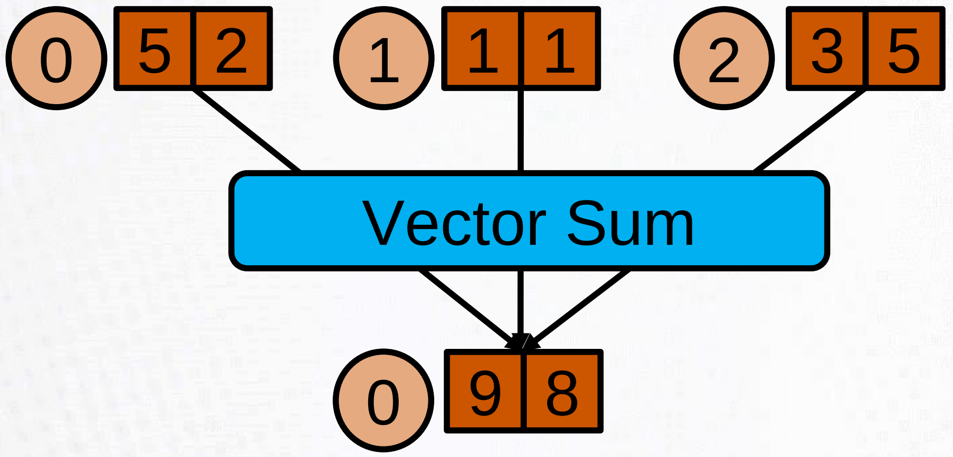
- 1D data decomposition
- Iterative computation and halo exchange
- Allreduce for residual calculation



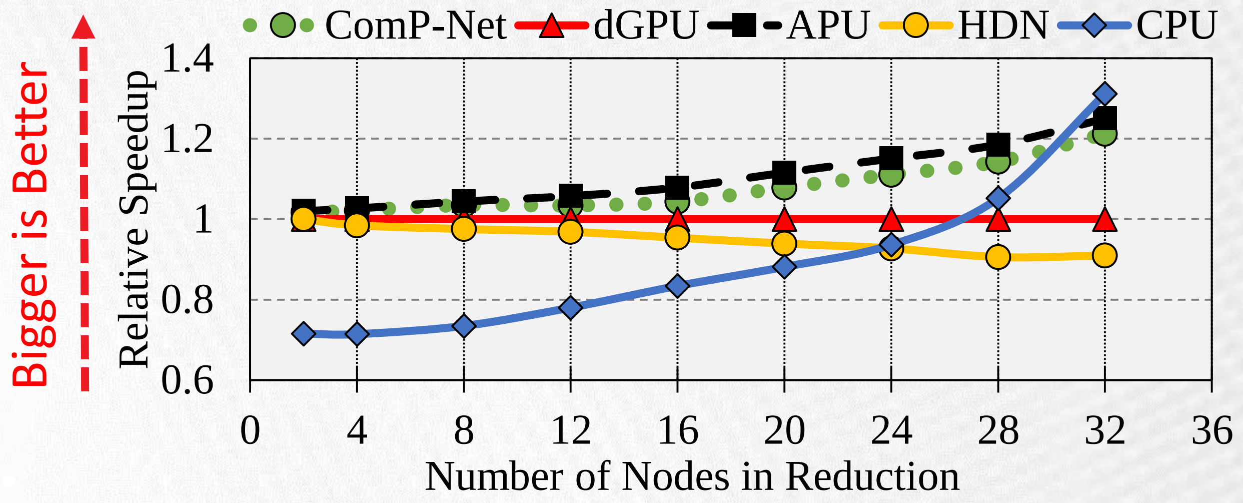
- Three regions of interest
  1. CPU is best
  2. Intra-kernel networking is best
  3. Any GPU solution is acceptable

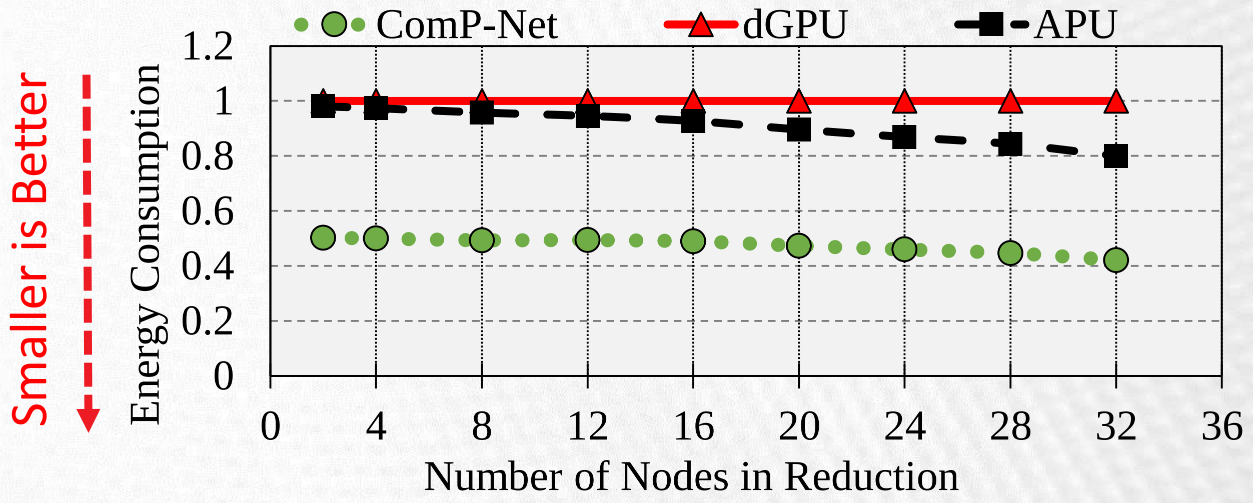
REDUCTION

Results



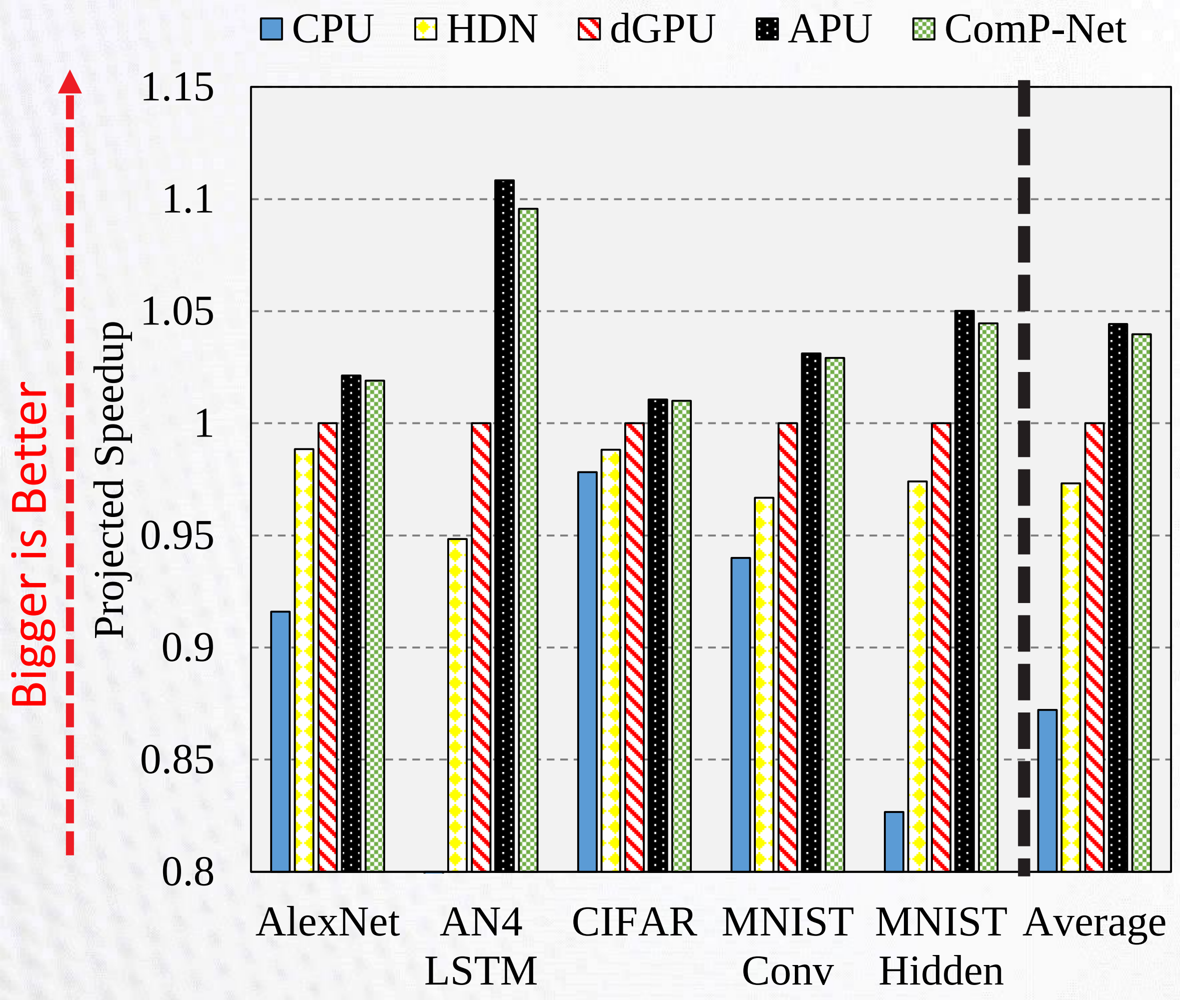
- 64MB strong scaling study
  - Fix problem size, sweep node count
  - APU performs better than ComP-Net
  - ComP-Net is much more energy efficient





DEEP LEARNING TRAINING

Results



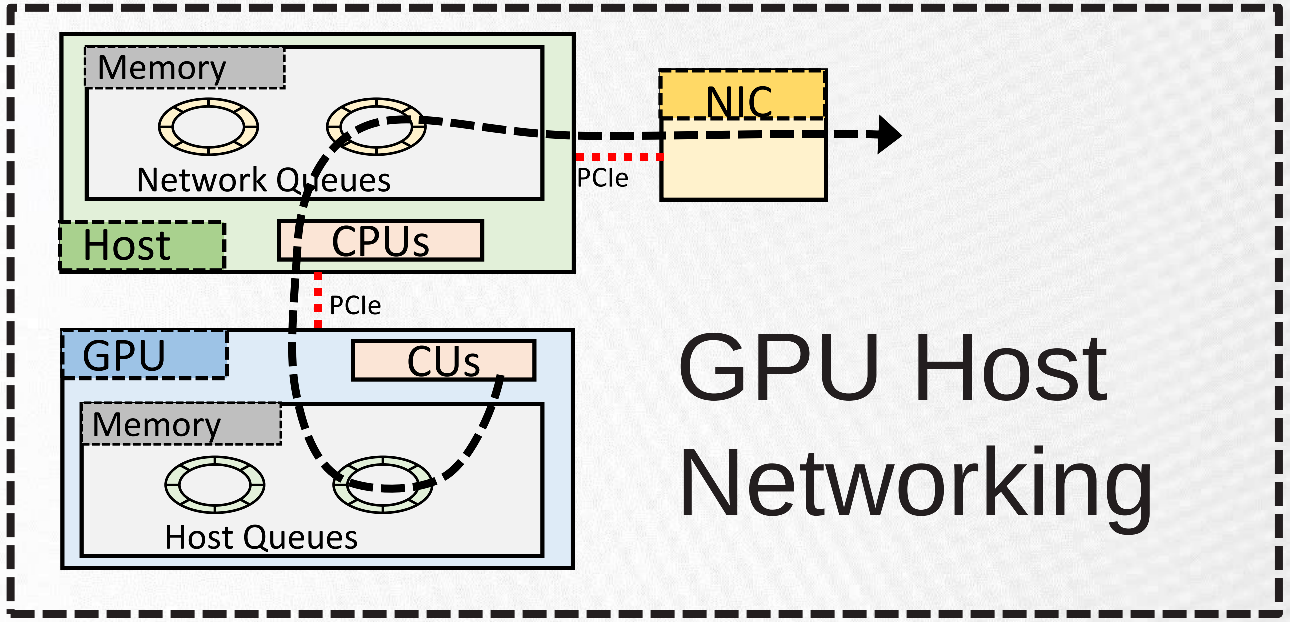
Workload Name	Domain	%Blocked	Reductions
Alex Net	Classification	14%	4672
AN4 LSTM	Speech	50%	131192
CIFAR	Classification	4%	939820
Large Synth	Synthetic	28%	52800
MNIST Conv	Text Recognition	12%	900000
MNIST Hidden	Text Recognition	29%	900000

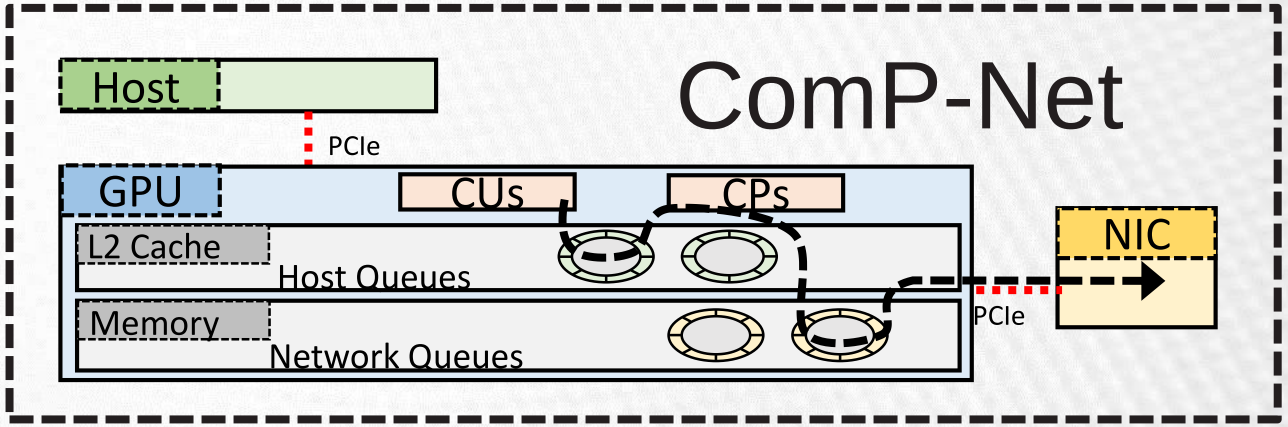
- Use Microsoft's Cognitive Toolkit and sample workloads
- Projected using simulation results + profiling data from TACC's Stampede supercomputer
- Speedups bound by % time application blocked on network data

SUMMARY

Conclusion

- Uses built-in CP to support network operations
- CP/GPU communicate over shared L2 cache instead of PCIe
- CP resources can scale with other GPU resources
- Up to 15% performance improvement and 2x energy reduction vs. GPU Host Networking





Thank You!

Sooraj.Puthoor@amd.com
Michael.LeBeane@amd.com

Questions?

Disclaimer

ATTRIBUTION

The work described in this presentation was made with Government support awarded by the DOE. The Government may have certain rights in this work.