GPU Initiated OpenSHMEM: Correct and Efficient Intra-Kernel Networking for dGPUs

Khaled Hamidouche and ***Michael LeBeane***
Michael.Lebeane@amd.com

FEBRUARY 26, 2020

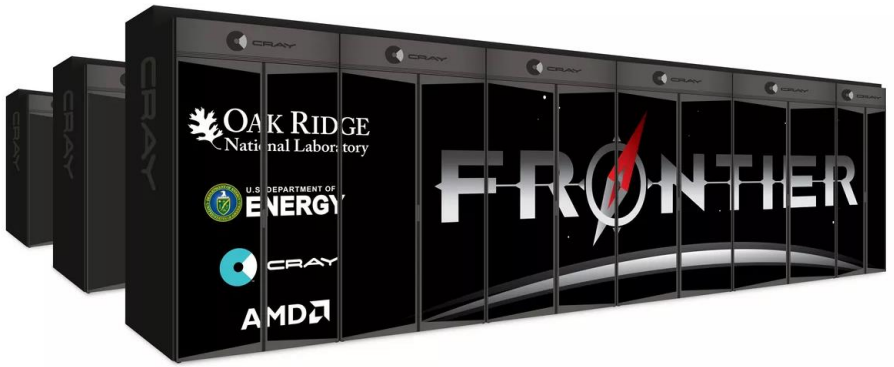
AMD

GPUs and Networks in the Wild

- GPUs are everywhere in HPC, Big Data, 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 need to talk to each other through Network Interface Controllers (NICs)

- Example: **Frontier** (expected world's fastest supercomputer in 2021)
  - **1.5** Exaflops peak compute performance
  - Per-Node Configuration [1]
    - **1** AMD EPYC CPU
    - **4** AMD Radeon Instinct GPUs
    - **100GB/s** Cray Slingshot Network



[1] Frontier Supercomputer. https://www.olcf.ornl.gov/frontier/

AMD

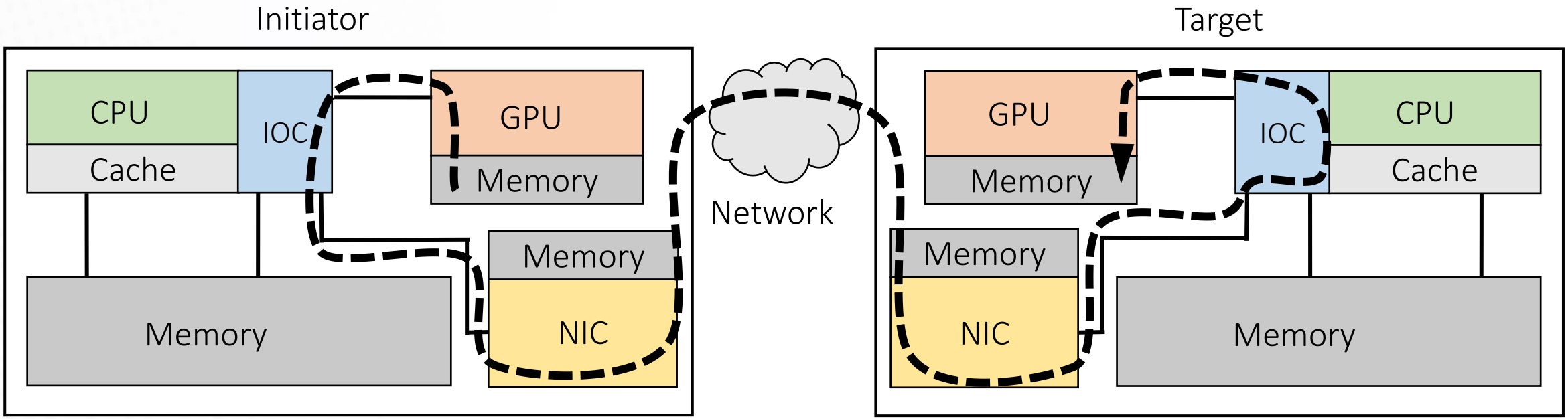
Data Movement on GPU Clusters

- GPU networking can largely be broken down into two pieces
  - **Data Path**
    - i.e., where the data that goes across the network flows
    - Important to maximize bandwidth
  - **Control Path**
    - i.e., who tells the NIC to move the data across the network
    - Important to minimize latency

- Much industry and academic work
  - Data path has been highly optimized
  - **Control path is an open research area**

Data Path Optimizations

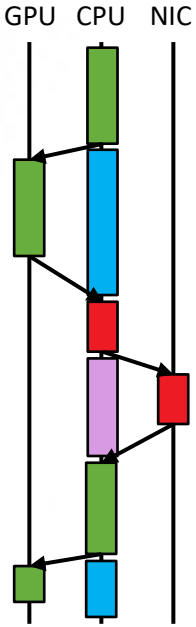
- Direct path from discrete GPU memory to NIC
  - No bounce buffers/host memory copies
  - Implemented in Mellanox's PeerDirect interface
  - Used by AMD's ROCn RDMA and Nvidia's GPUDirect RDMA



Control Path Optimizations

Host Driven Networking

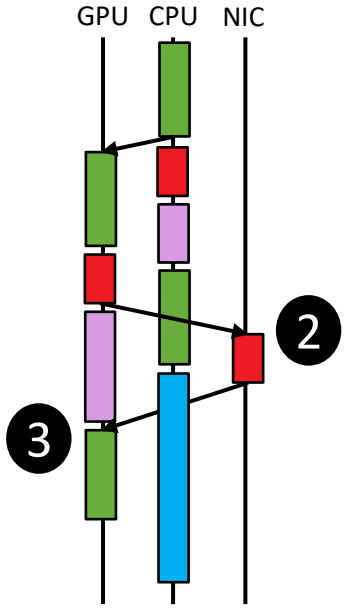
```
1  a_kernel<<<…, stream>>>(buf);
   cudaStreamSynchronize(stream);
2  netSend(buf);
   netWait();
3  b_kernel<<<…, stream>>>(buf);
   cudaStreamSynchronize(stream);
```



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)

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

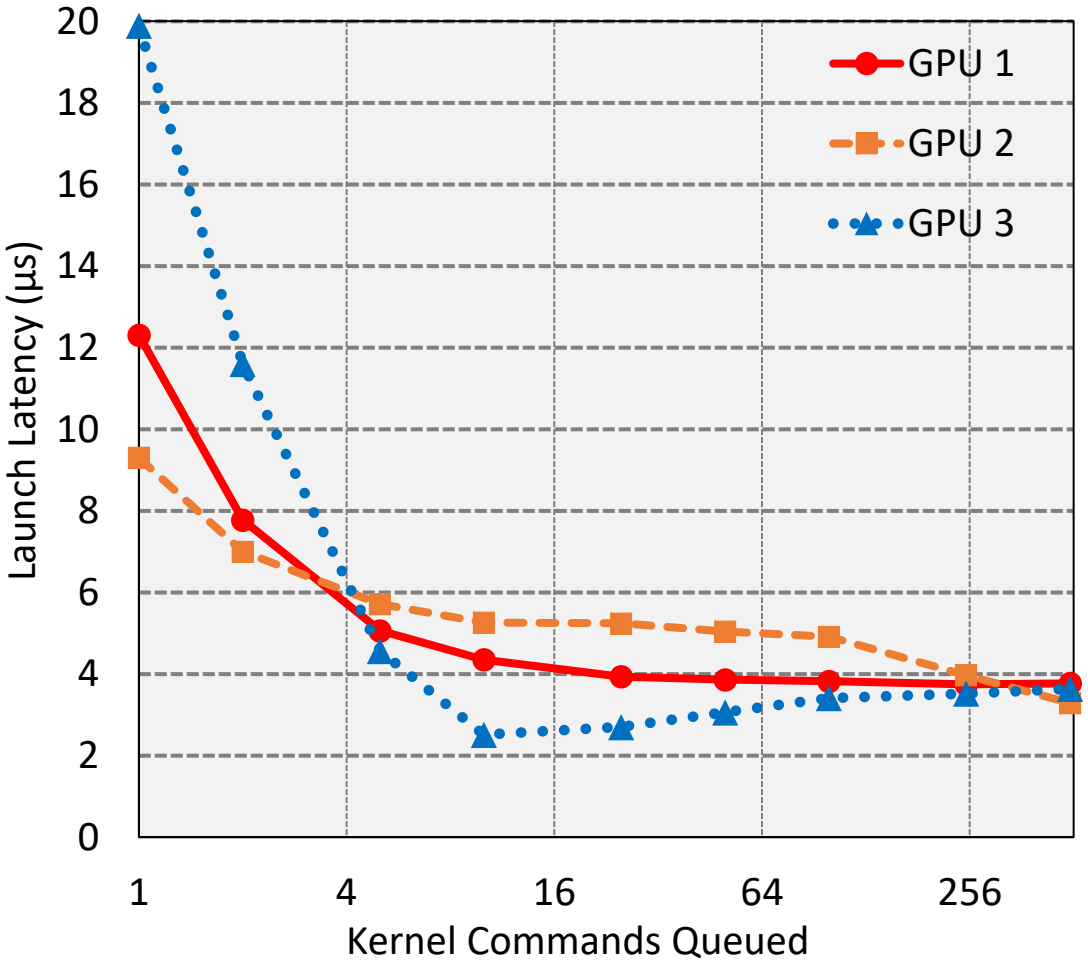


1. CPU schedules kernel, network operation, and final kernel
2. GPU triggers initiation of a network operation after kernel
3. GPU launches final kernel

- GDS removes the CPU from the critical path and avoids control flow switches
- Still kernel boundary…

Issues with Kernel Boundary Communication

- Communications restricted to kernel boundaries
- High overhead of starting/finishing a kernel
- Restricts networking to large messages to amortize overheads
- Poor for irregular or frequent communication patterns



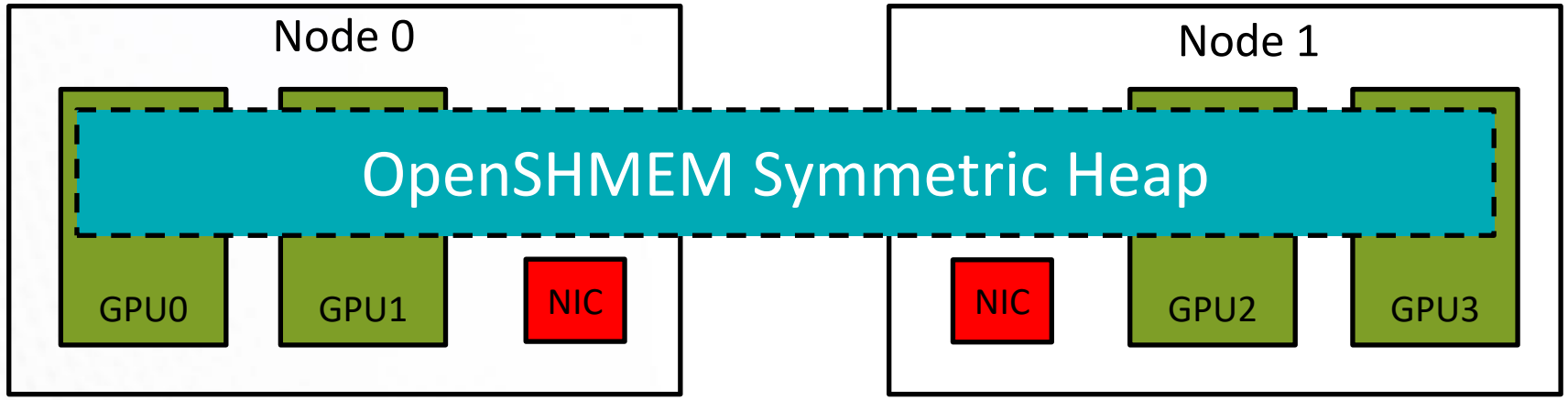
AMD

Introducing GPU Initiated OpenSHMEM (GIO)

- GIO enables intra-kernel networking from a GPU through an OpenSHMEM-like interface
  - GPU directly creates InfiniBand™ command packets
  - Removes GPU Stream + MPI programming abstraction
- Identifies and solves data visibility issues common to prior intra-kernel networking approaches
- Introduces novel packet templating design to overcome poor performance of network code on GPU
- Provides speedup vs. traditional kernel boundary communication
  - Up to 40% performance optimization for irregular Sparse Triangular Solver

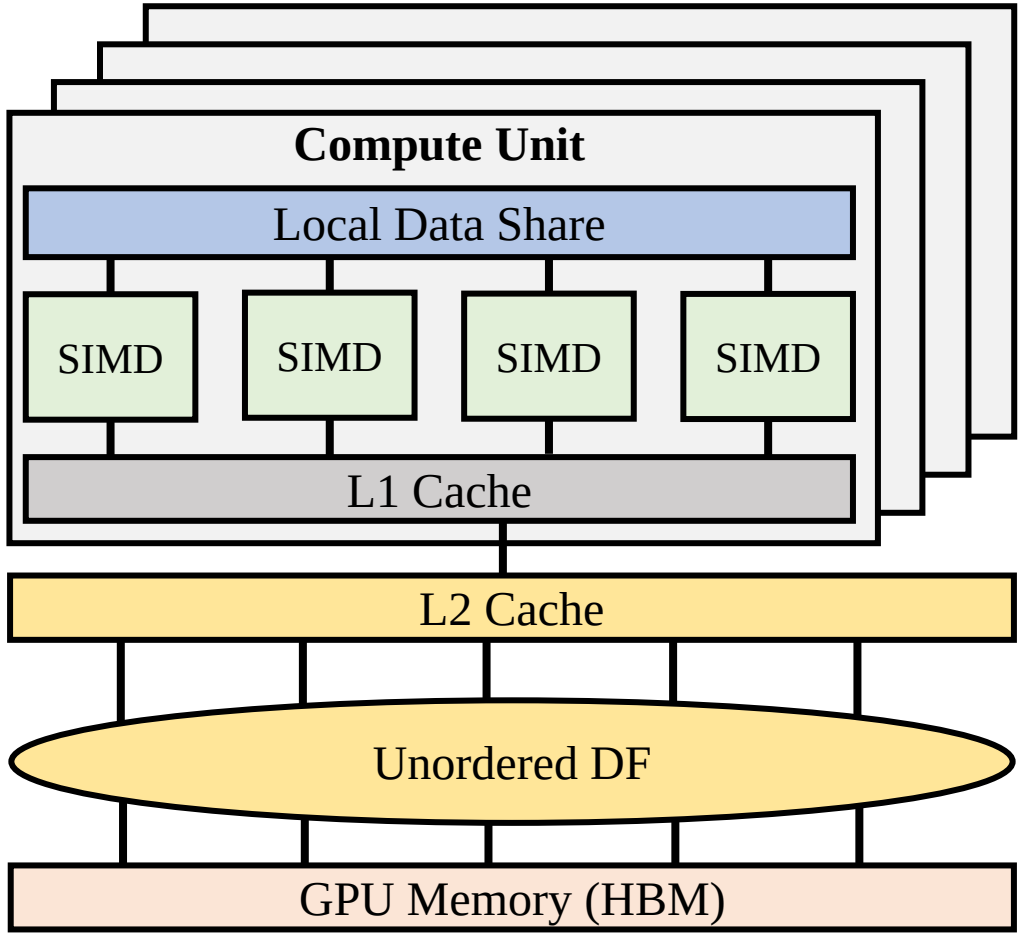
1 Slide OpenSHMEM Tutorial

- OpenSHMEM is a PGAS communications model implemented through a standardized runtime
- Network accessible memory allocated collectively on all nodes
  - Referred to as the “Symmetric Heap”
- Remote memory on the Symmetric Heap accessed using `Puts()` and `Gets()`
  - Analogous to local Stores and Loads, respectively
- Ordering and network completion controlled via `Fence()` and `Quiet()` calls



1 Slide GPU Tutorial

- Single Instruction-Multiple Thread (SIMT) Style
  - Bundles of threads scheduled across multiple SIMD units
  - Good for data parallel code, bad for branchy serial code
- Components of interest
  - Work-groups (Threadblocks): Groups of thread bundles on the same CU can share scratchpad memory and synchronize
  - LDS (Shared Memory): Scratchpad memory used by work-groups



GIO Programming Model

```
// Initialize GIO Runtime
gio_shmem_handle_t* gio_shmem_handle;
gio_shmem_init(&gio_shmem_handle);

// Allocate symmetric heap memory
int size = sizeof(char) * ELEMENTS;
char* src = gio_shmem_malloc(size);
char* dst = gio_shmem_malloc(size);

//Initiator/target launches kernel
pe = gio_shmem_my_pe(gio_shmem_handle);
if (pe == INITIATOR) {
    hipLaunchKernel(Ping, GRID_SZ,
                    GRID_SZ / WG_SZ, 0, 0,
                    gio_shmem_handle, src,
                    dst);
} else {
// Launch pong kernel (not shown)
}
```

Host Code

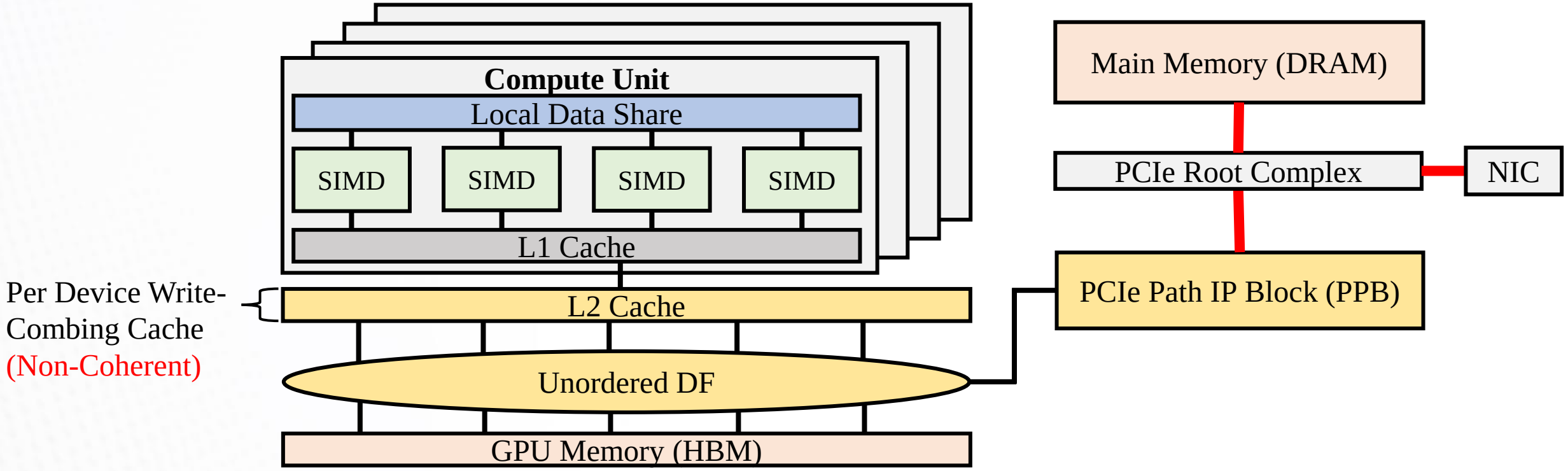
```
__device__ void
devicePing(gio_shmem_handle_t* gio_shmem_handle,
           char* src, char* dst)
{
    // Extract context from global handle
    __shared__ gio_shmem_ctx_t gio_shmem_ctx;
    gio_shmem_ctx_create(gio_shmem_handle,
                         &gio_shmem_ctx);

    // Each WG pings target
    gio_shmem_put_nbi(gio_shmem_ctx,
                      dst[hipBlockIdx_x],
                      src[hipBlockIdx_x],
                      sizeof(char), TARGET);

    // Wait on the network completion
    gio_shmem_quiet(gio_shmem_ctx);

    // Each WG waits for pong target
    gio_shmem_wait_until(dst[hipBlockIdx_x], 1);

    gio_shmem_ctx_destroy(gio_shmem_ctx);
}
```

GPU Code

GIO Runtime Challenges

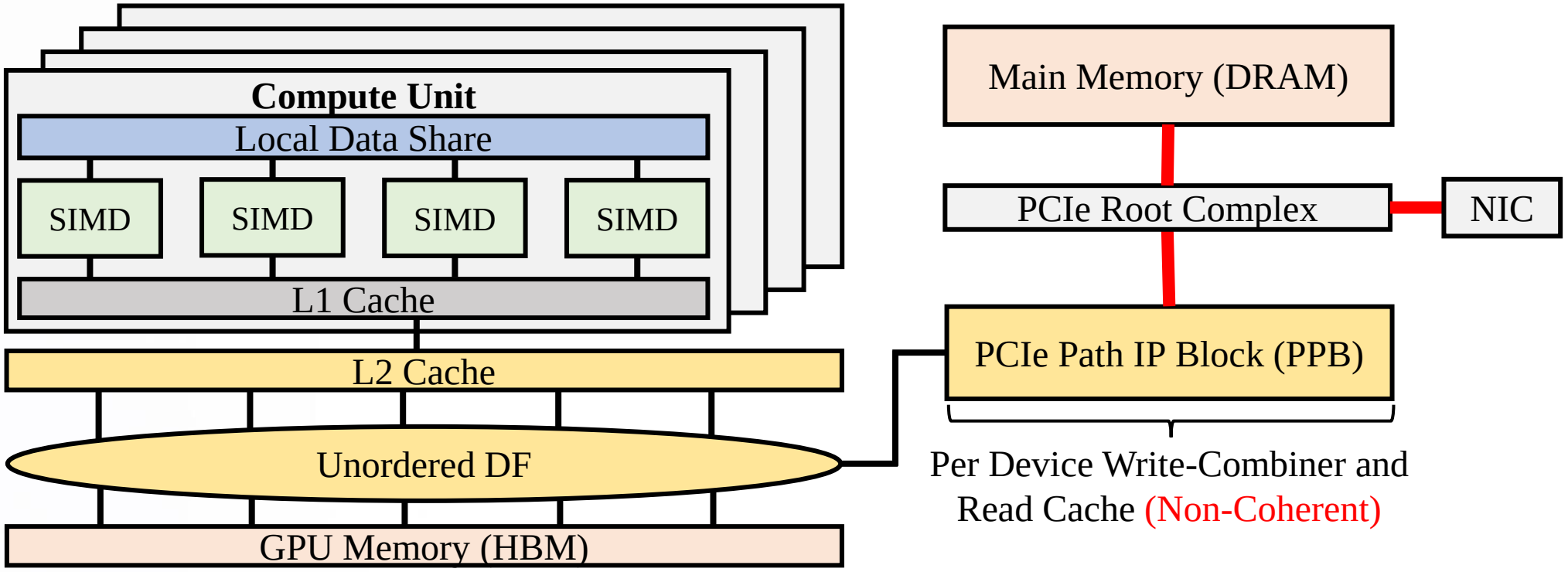
- Data visibility challenges
  - Memory model requires kernel termination for:
    - GPU to see data produced by others
    - Others to see data produced by GPU
- Performance challenges
  - GPU very bad at constructing network command packets
  - Long sequence of serial code performed by a single thread

Data Visibility Challenges



- **Problem**: L2 Cache is not probed when NIC writes to DRAM / updates can get stuck in L2
- **Solution**: Map NIC accessible pages as uncacheable on allocation

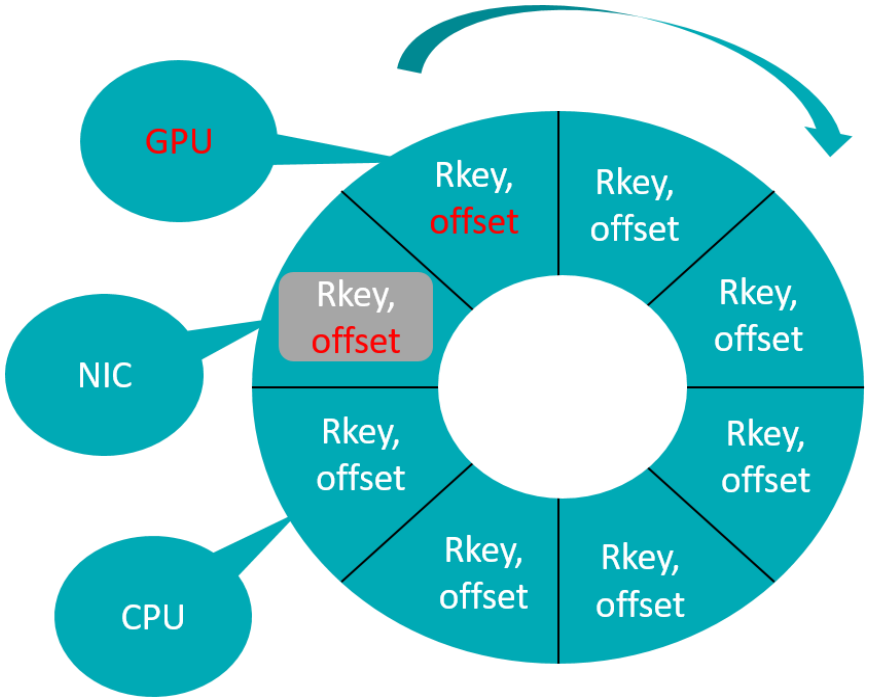
Data Visibility Challenges



- **Problem**: PCIe Path IP Block (PPB) cache not updated on GPU writes to HBM
- **Solution**: Memory-map PPB control registers to GPU address space - flush within GPU runtime as needed

Performance Challenges

- **Problem**: GPU is very poor at writing network packet data
  - Branchy condition checks and serial packing of control information leads to control flow divergence
- **Observation**: Most of the network packet data is static!
  - lkey, rkey, control, etc.
  - Few pieces of dynamic information (src, dst, size, type)
- **Solution**: CPU creates and posts network templated WQEs
  - CPU runs ahead of GPU and fills in static fields
  - GPU threads only update the dynamic information
  - GPU threads ring the NIC doorbell



Putting it All Together

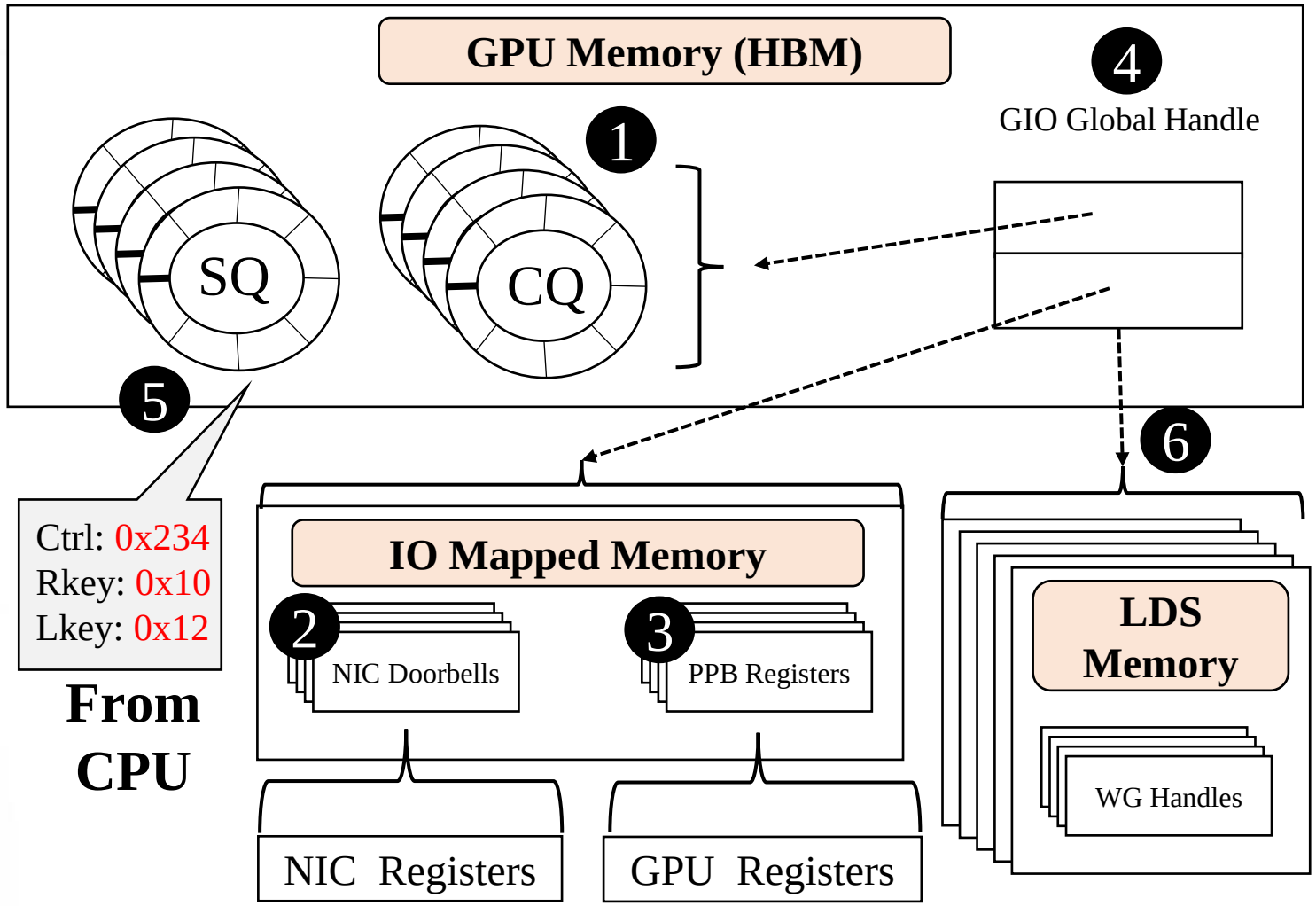
CPU Side `(gio_shmem_init())`

1. Create network CQ/SQ resources
2. Create doorbell mapping
3. Create PPB register mappings
4. Create GPU handle and populate with allocated structs
5. CPU begins templating common packet fields for GPU

GPU Side `(gio_shmem_ctx_create())`

6. Store local cache of GIO global handle into each work-group's shared memory

Ready for Networking!

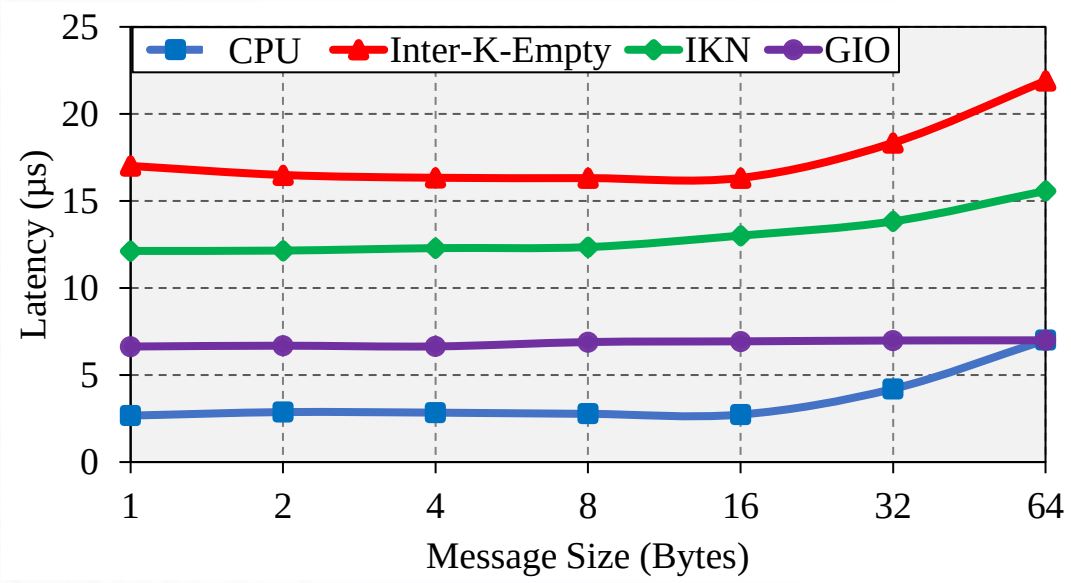


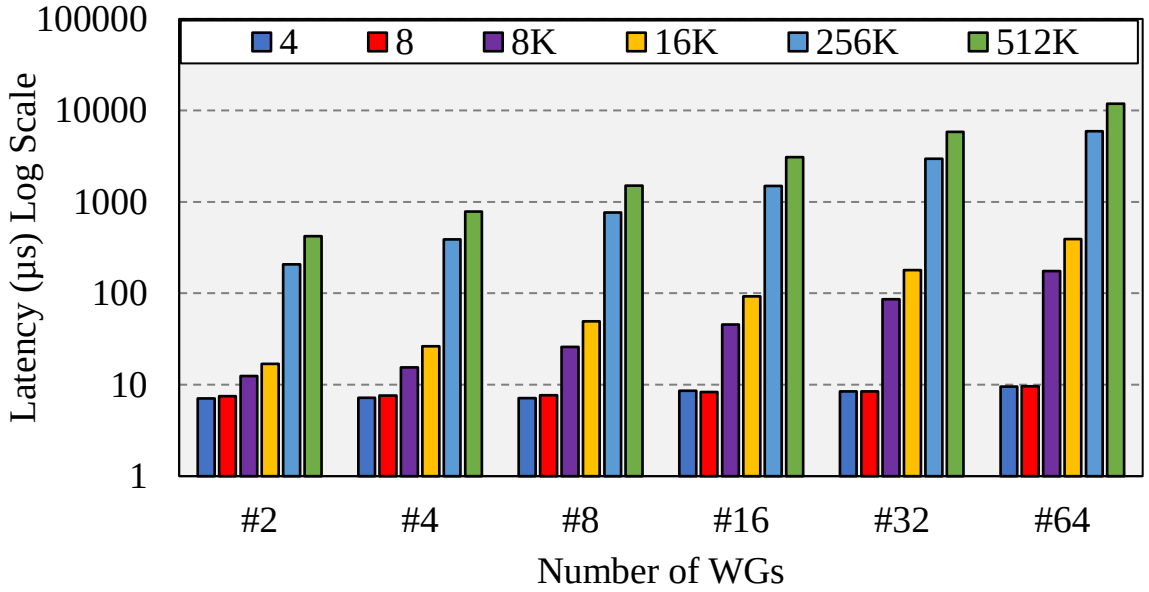
Experimental Setup

- Cluster configuration
  - Up to 4 nodes
  - X86 Server Processor
  - AMD Radeon™ MI-25 GPU
  - 56Gbps InfiniBand™ Network

- Baselines (not all are used in every experiment)
  - **CPU:** No GPU; communication and computation performed by the CPU
  - **Inter-K:** GPU performs computation and networking is routed through CPU-centric MPI calls at kernel boundaries
  - **Inter-K-Overlap:** Same as Inter-K but GPU kernels overlap with CPU MPI calls when allowed by the algorithm
  - **Intra-Kernel Networking (IKN):** Rely on CPU threads to perform network operations on behalf of the GPU

Microbenchmarks



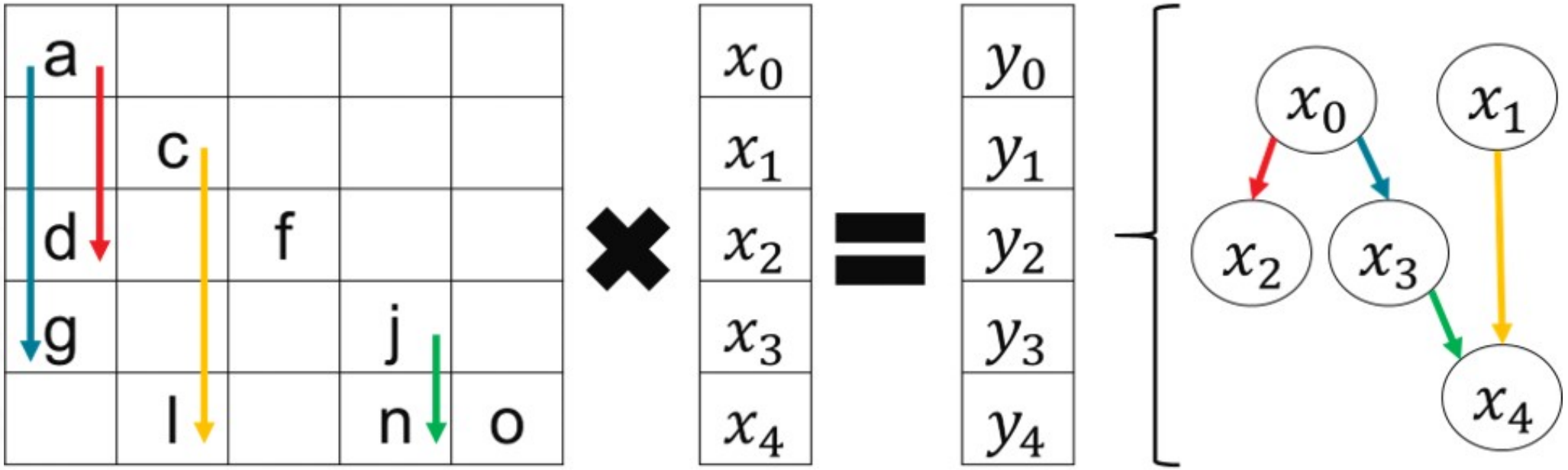


- Latency Analysis
  - Small Messages
    - CPU < GIO < IKN < Inter-K-Empty
  - Large Messages
    - Limited by network bandwidth
    - All trends the same

- Work-group scaling
  - Network not saturated
    - Latency independent of number of work-groups
  - Network saturated
    - Latency dependent on number of work-groups
    - Work-groups wait on network availability

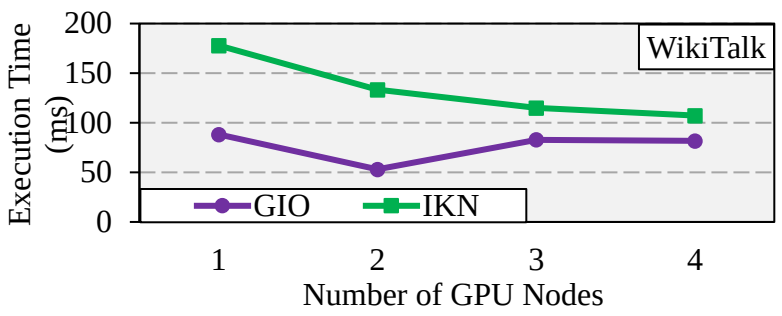
Sparse Triangular Solver

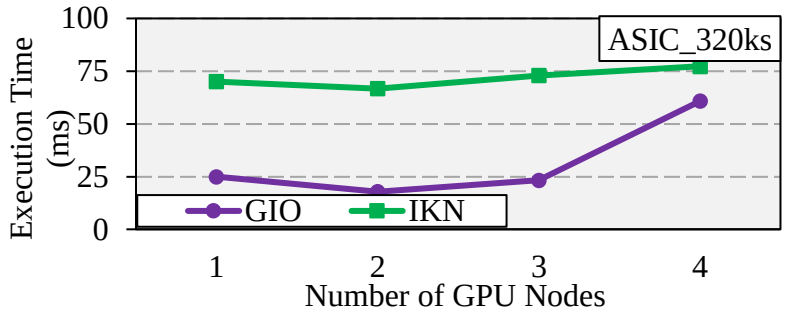
- **Goal**: Solve for the vector x in the equation $Ax = y$ where the matrix A and the vector y are inputs
  - Each row solves a single system of equations based on the previous rows' values
  - Dependencies flow downwards
  - More sparsity -> More independently solvable rows -> More parallelism

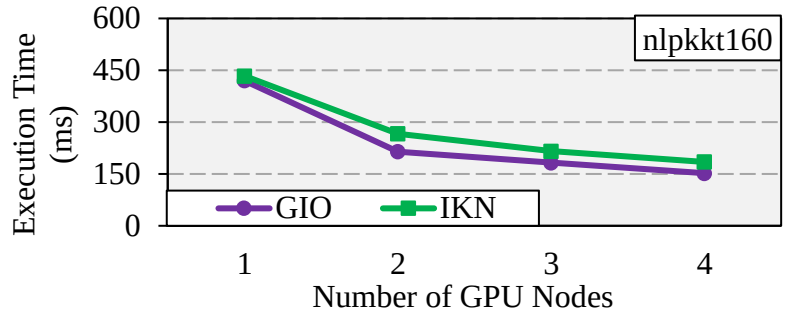


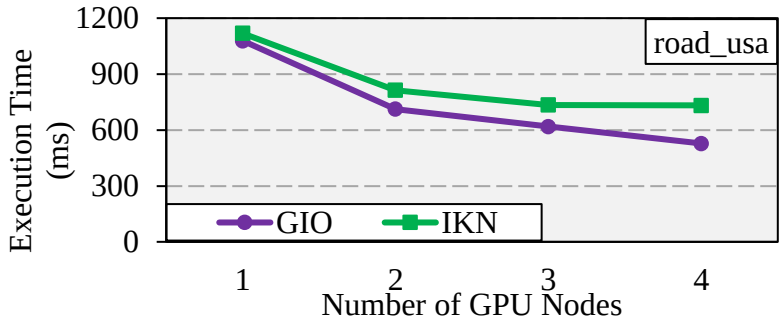
Sparse Triangular Solver

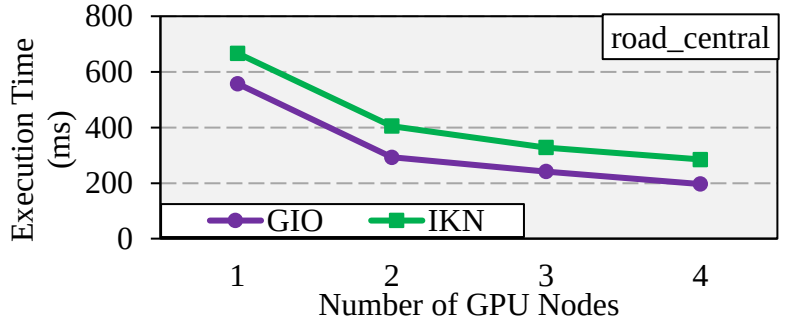
	WikiTalk	ASIC_320ks	nlpkkt160	road_usa	road_central
NNZ	3.072M	1.074M	118.9M	52.80M	31.02M
# Rows	2.394M	.321M	8.354M	23.95M	14.08M











- Strong scaling (same problem size, add more nodes)
  - Compare against IKN only (kernel boundary networking performs poorly here)
  - Performance uplift and scalability very input dependent
  - GIO performs better than IKN in all cases

Summary

- GIO enables intra-kernel networking from a GPU through an OpenSHMEM-like interface
  - GPU directly creates InfiniBand™ command packets
  - Removes GPU Stream + MPI programming abstraction
  - Large boost in productivity
- Identifies and solves data visibility issues common to prior intra-kernel networking approaches
- Introduces novel packet templating design to overcome poor performance of network code on GPU
- Provides speedup vs. traditional kernel boundary communication
  - Up to 40% performance optimization for irregular Sparse Triangular Solver

- **Planned open-source release mid-March on ROCm tools GitHub**

Thank you!

Michael.Lebeane@amd.com

Questions?

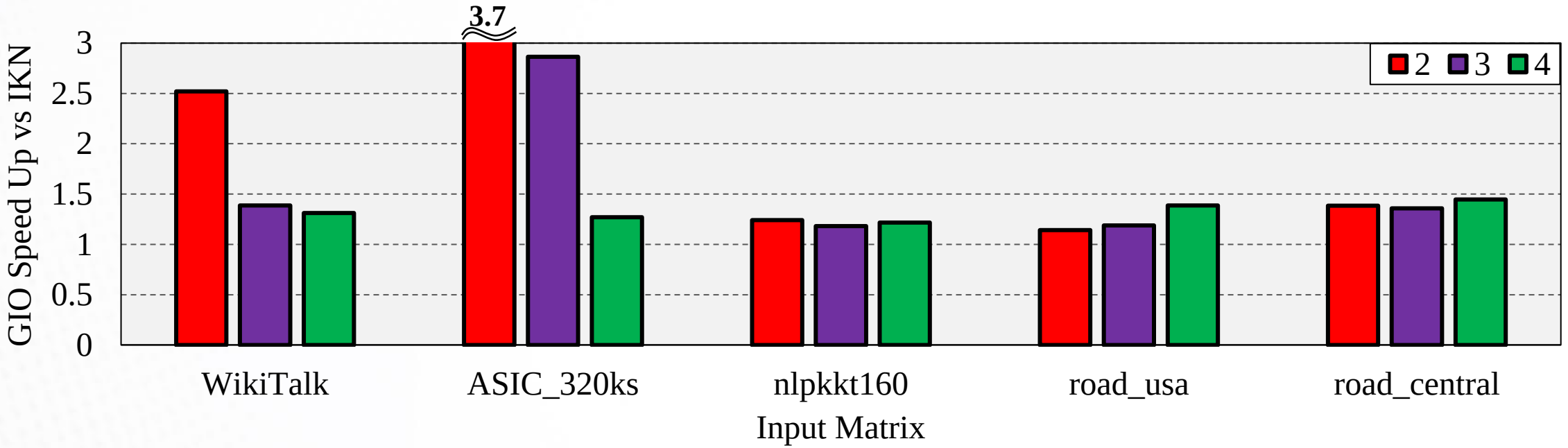
Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

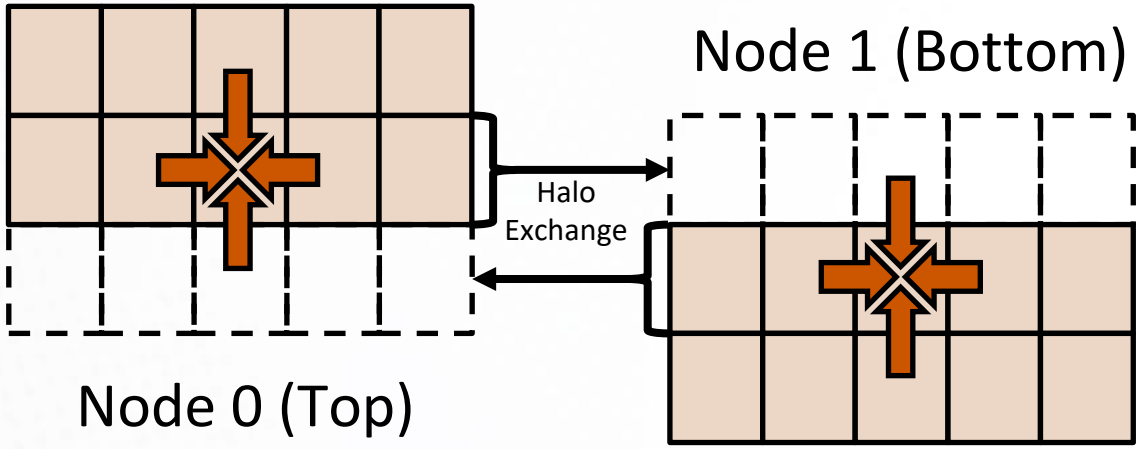
Sparse Triangular Solver

	WikiTalk	ASIC_320ks	nlpkkt160	road_usa	road_central
NNZ	3.072M	1.074M	118.9M	52.80M	31.02M
# Rows	2.394M	.321M	8.354M	23.95M	14.08M

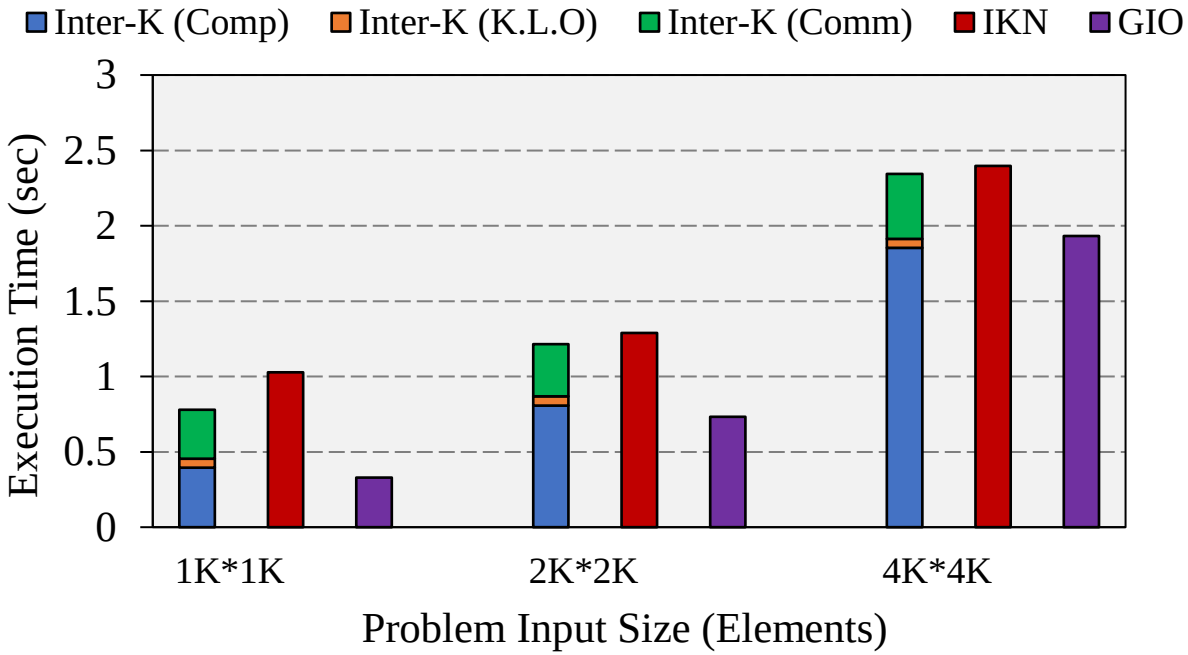


- For small matrices (e.g., ASIC_320ks) GIO shows up to 3.7X improvement on 2 nodes and more than 30% on 4 nodes
- For large matrices, (e.g., road_central) we show 38%, 35% and 44% improvement on 2, 3, and 4 GPU nodes, respectively

2D-Jacobi Stencil

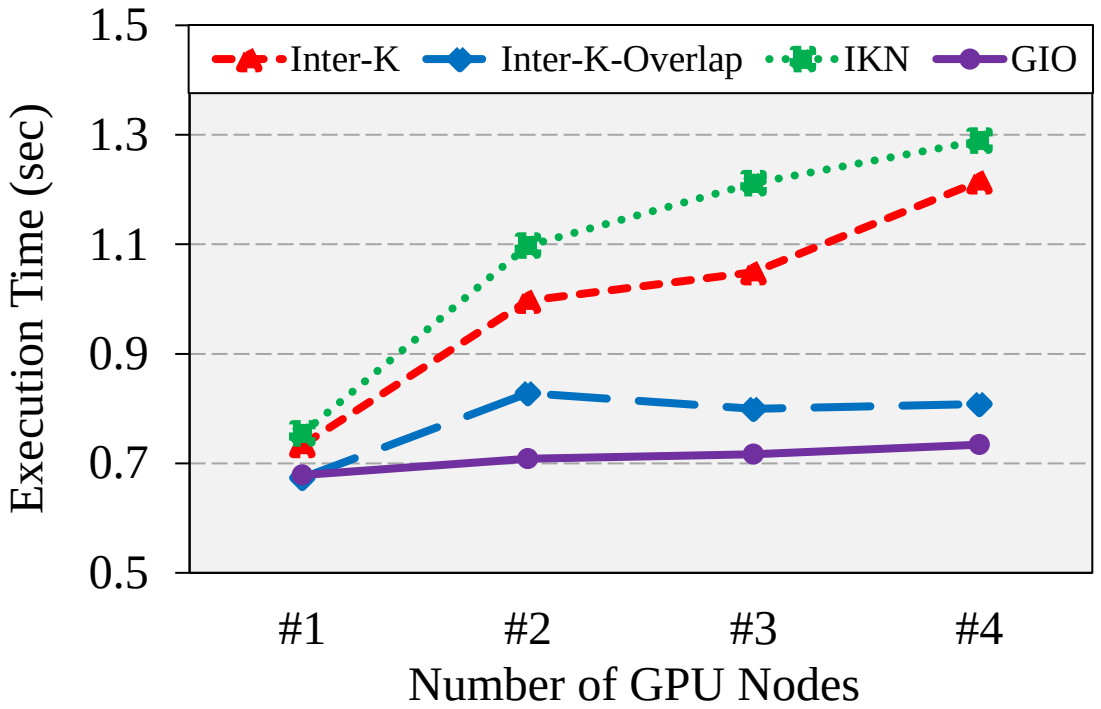


- Elements arranged on a grid and divided in 2 dimensions across all nodes
- Next element value dependent on immediate neighbor elements
- Elements on “halo” region must be exchanged with immediate neighbors
- Repeat compute and exchange until convergence

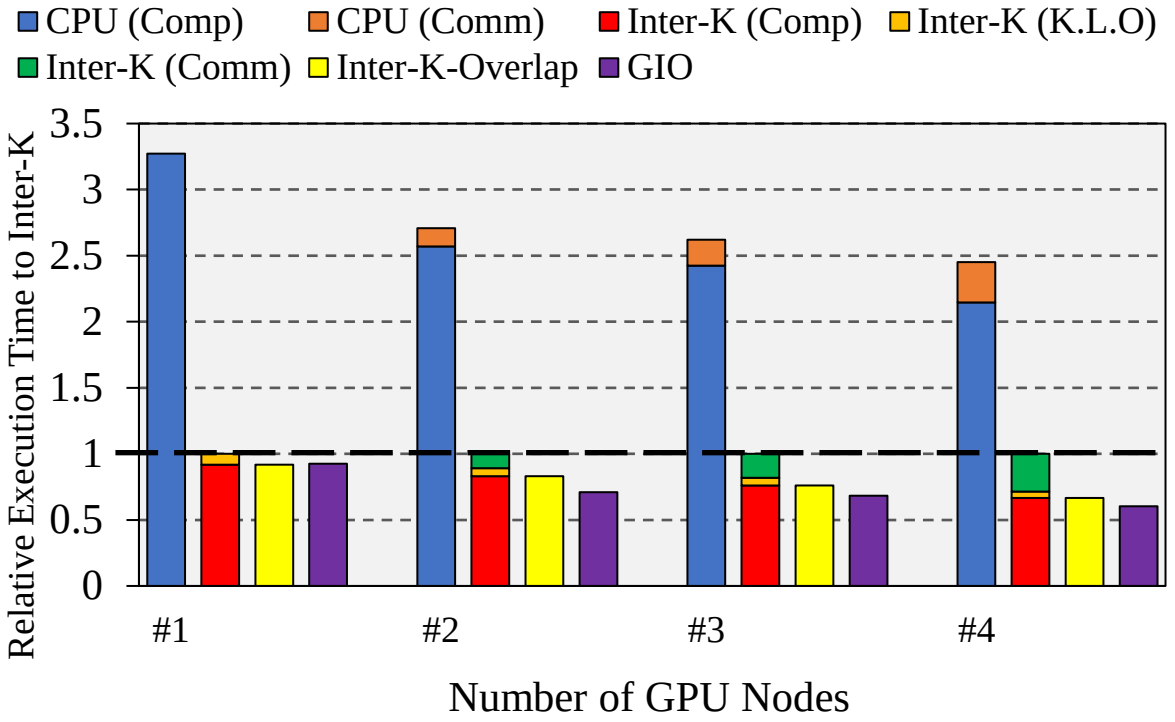


- Strong Scaling with Time Breakdown
  - GIO outperforms both Inter-K and IKN versions for all input sizes
  - Compared to Inter-K, GIO demonstrates 57%, 40% and 18% performance uplift
  - The GIO runtime has a negligible overhead in GPU compute time as it exhibits similar performance to the Inter-K computation phase

2D-Jacobi Stencil



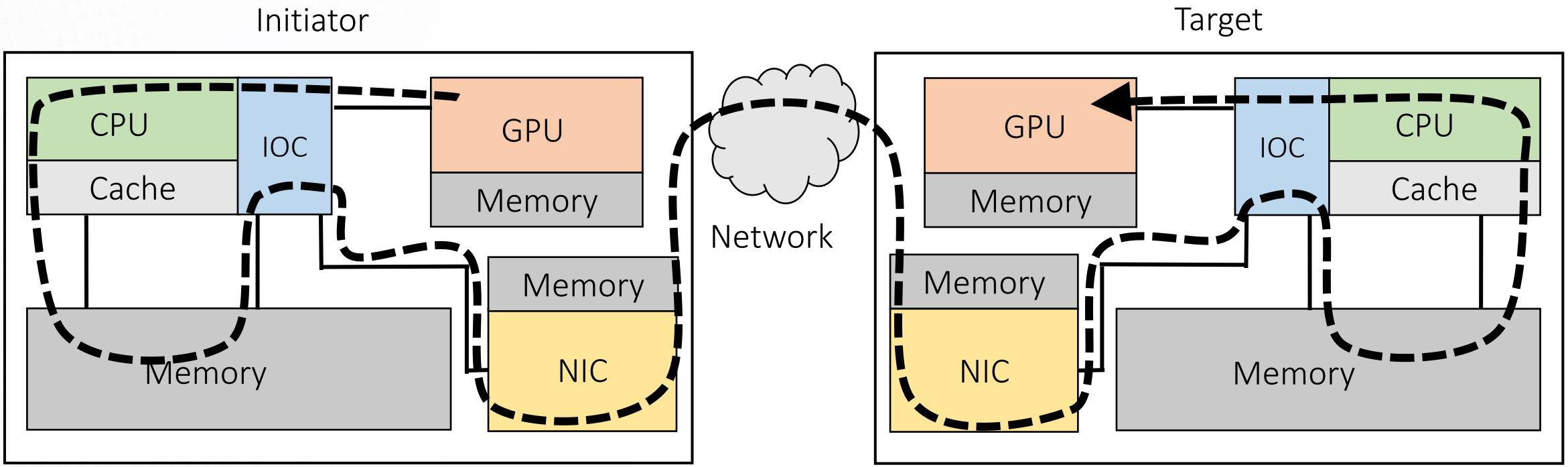
- Weak Scaling
  - 2K *2K (element) problem size per GPU
  - Flat line is perfect scaling
  - GIO scales near-optimally



- Weak Scaling with time breakdown
  - Normalized to Inter-K baseline
  - CPU performs poorly (means application is good for GPU)
  - GIO overlaps communication and Kernel Launch Overhead (K.L.O)

Challenges with Today's GPU Networks

- Control plane is unoptimized!
  - Focused on a host-centric model where only the CPU can coordinate network transfers
  - Very high latencies to perform networking from the GPU

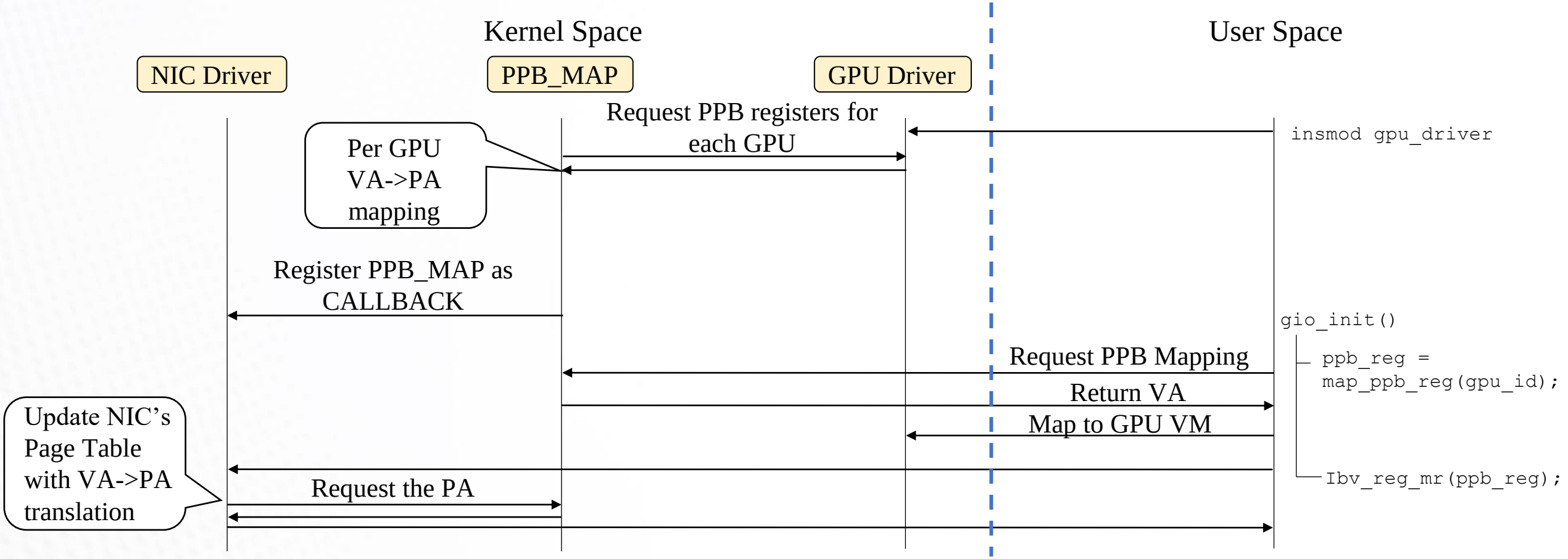


L2 Cache Impact on Performance

- L2 Cache must be disabled for network-accessible GPU memory
  - Else data can get stuck
- Disabling the L2 cache can have adverse affects on compute that is performed in-place on network data
- Up to 9.3% overhead on SPtS single GPU when disabling the cache
- **Recommendation**: Provide shader-visible L2 cache maintenance operations

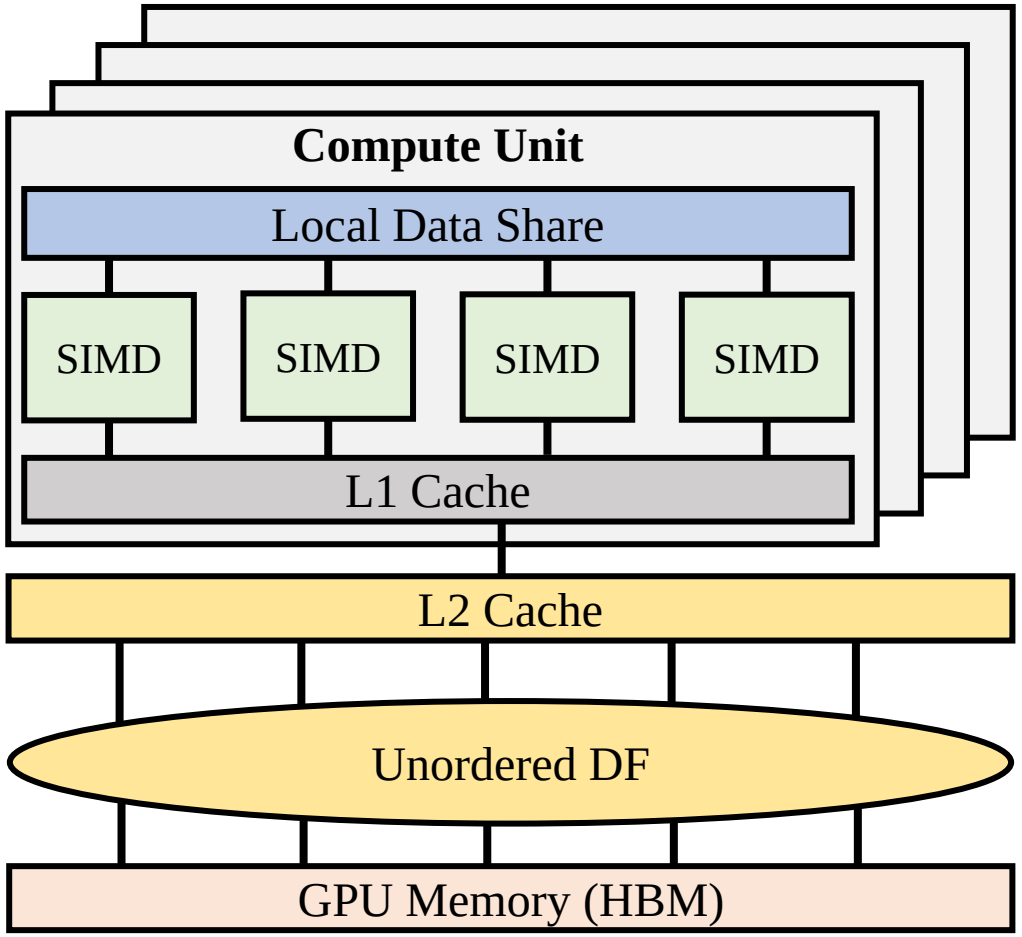
	WikiTalk	ASIC_320ks	nlpkkt160	road_usa	road_central
L2 OFF (ms)	88.03	24.94	419.9	1078.9	557.7
L2 On (ms)	87.14	23.56	380.7	1003.2	541.4
Overhead (%)	1	5.5	9.3	7	2.9

GIO System Software Interactions



1 Slide GPU Tutorial

- Single Instruction-Multiple Thread (SIMT) Style
  - Bundles of threads execute in lockstep (single Program Counter)
  - Groups of thread bundles on the same CU can share scratchpad memory and synchronize
  - Want to minimize memory and control flow divergence



- Common Terms
  - Work-item = Thread
  - Wavefront (64 Threads) = Warp (32 Threads)
    - Unit of thread dispatch
  - Work-group = Thread Block
    - Unit of synchronization and data sharing
  - Local Data Share (LDS) = Shared Memory
    - Work-group scratchpad
  - Private Memory = Local Memory
    - Thread local storage
  - Compute Unit (CU) = Streaming Multi-Processor (SM)
    - Collection of SIMD engines sharing LDS and L1 cache