# GPU TRIGGERED NETWORKING FOR INTRA-KERNEL COMMUNICATIONS

**Michael LeBeane**, Khaled Hamidouche, Brad Benton, Mauricio Breternitz, Steven K. Reinhardt, Lizy K. John

ADVANCED MICRO DEVICES INC., THE UNIVERSITY OF TEXAS AT AUSTIN, MICROSOFT CORPORATION, INSTITUTO UNIVERSITARIO DE LISBOA

# GPUS EVERYWHERE!

## INTRODUCTION

- GPUs are everywhere in modern HPC
- Over 70 of the Top 500 supercomputers use accelerators[1]
- 100's of applications designed to leverage GPU compute[2]
- High performance and energy efficiency for many data-parallel applications
- All-in-one solutions with ***both*** GPUs and NICs
  - Example: AMD's Project 47
    - **80** Radeon Instinct GPUs
    - **20** Mellanox 100G NICs
    - **20** EPYC 7601 32-Core CPUs




[1] TOP500.org, "Highlights – June 2017," http://www.top500.org/lists/2017/06/highlights, 2017.
[2] Nvidia, "GPU-Accelerated Applications," http://www.nvidia.com/content/gpu-applications/pdf/gpu-apps-catalog-mar2015.pdf, 2016.

# CHALLENGES IN GPU-NETWORKING

## INTRODUCTION

- What is the current state-of-the-art in GPU Networking?
- Where are we going?
- What are the challenges?

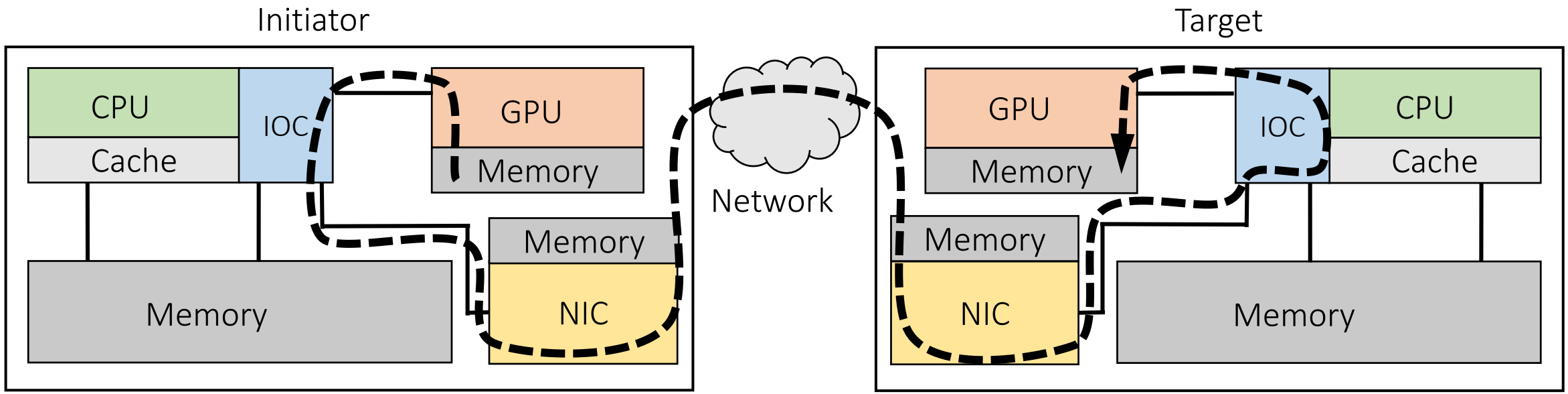
- Let's break this down into two parts:
  - ***Data Path***
    - i.e., where the data that goes across the network flows
  - ***Control Path***
    - i.e., who tells the NIC to move the data across the network

# DATA PATH OPTIMIZATIONS

## INTRODUCTION

IOC = IO Controller
RDMA = Remote Direct Memory Access
NIC = Network Interface Controller
HCA = Host Channel Adaptor (same as NIC)

- Network data path is highly optimized
- Direct path from discrete GPU memory to NIC
  - No bounce buffers/host memory copies
  - Implemented in Mellanox's PeerDirect interface for their HCA
  - Used by AMD's ROCn RDMA[3] and Nvidia's GPUDirect RDMA[4]





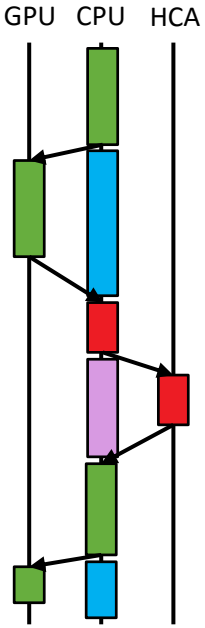

[3] AMD. 2017. ROCn RDMA. https://github.com/RadeonOpenCompute/ROCnRDMA
[4] Mellanox. 2017. Mellanox OFED GPUDirect RDMA. http://www.mellanox.com/page/products_dyn?product_family=116

# CONTROL PATH OPTIMIZATIONS

## INTRODUCTION

### Host Driven Networking (HDN)

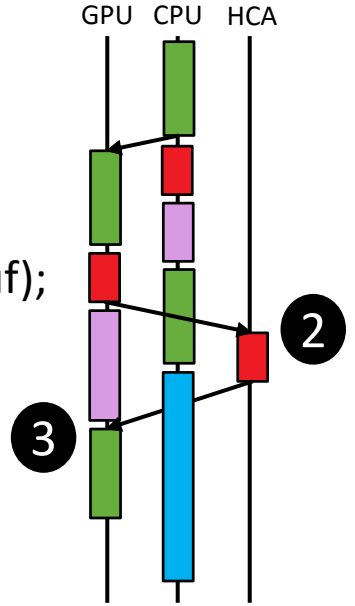
```
1  a_kernel<<<…, stream>>>(buf);
   cudaStreamSynchronize(stream);
2  ibv_post_send(buf);
   while (!done) ibv_poll_cq(txcq);
3  b_kernel<<<…, stream>>>(buf);
   cudaStreamSynchronize(stream);
```




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)[5]

```
1  a_kernel<<<…, stream>>>(buf);
   gds_stream_queue_send(stream, qp, buf);
   gds_stream_wait_cq(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 restricted to kernel boundary…… ***but***….. Is that really an overhead?



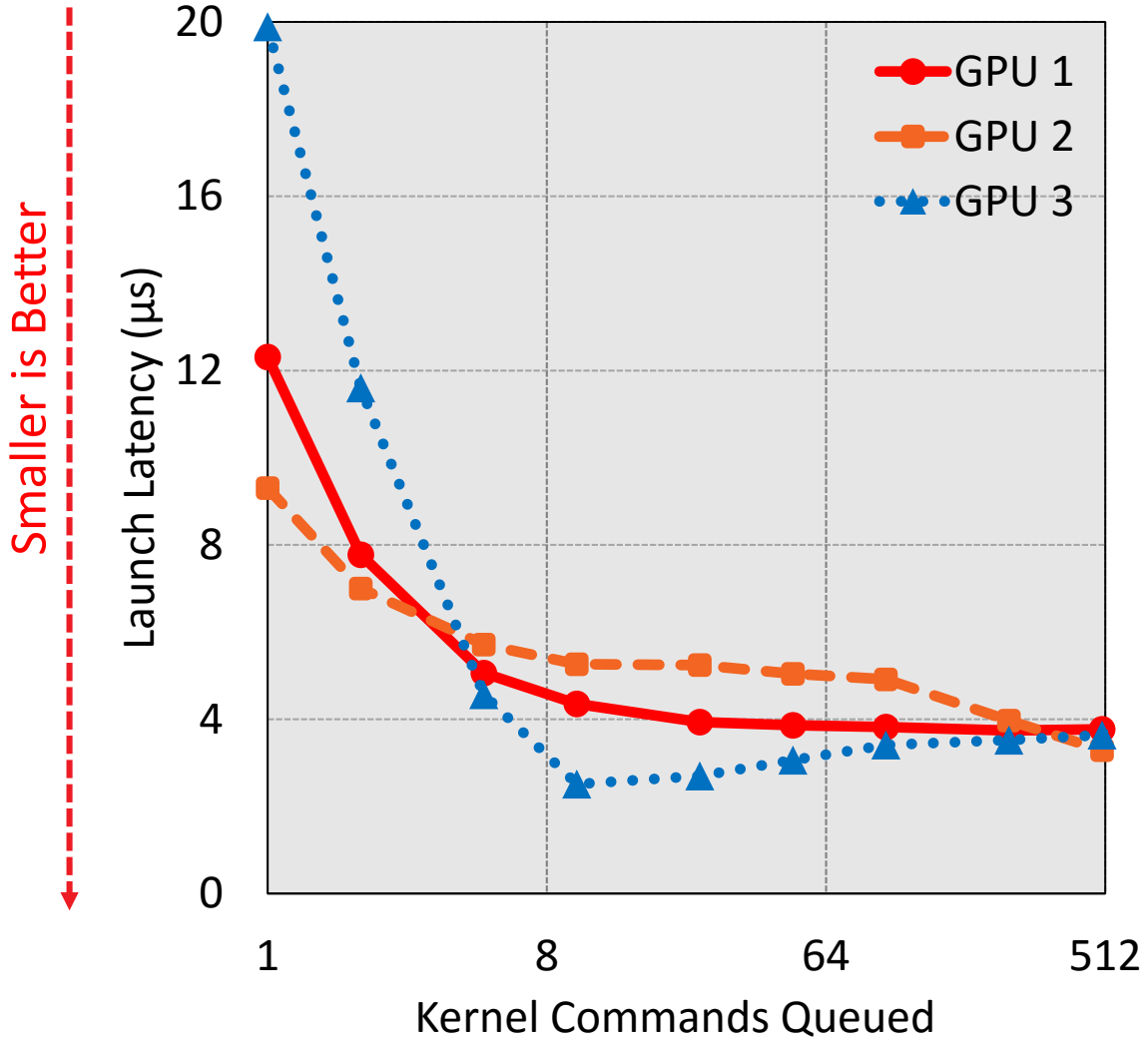
[5] Elena Agostini, Davide Rossetti, and Sreeram Potluri. 2017. Offloading communication control logic in GPU accelerated applications. In Intl. Symp. on Cluster, Cloud and Grid Computing (CCGrid). DOI:https://doi.org/10.1109/CCGRID.2017.

# OVERHEAD OF KERNEL BOUNDARY COMMUNICATION

## INTRODUCTION

- **Launch latencies much higher than HPC network overheads!**
  - Can be up to 20µs for a kernel launch!
  - Compare that to the 1-2µs it takes to get to another node over the network

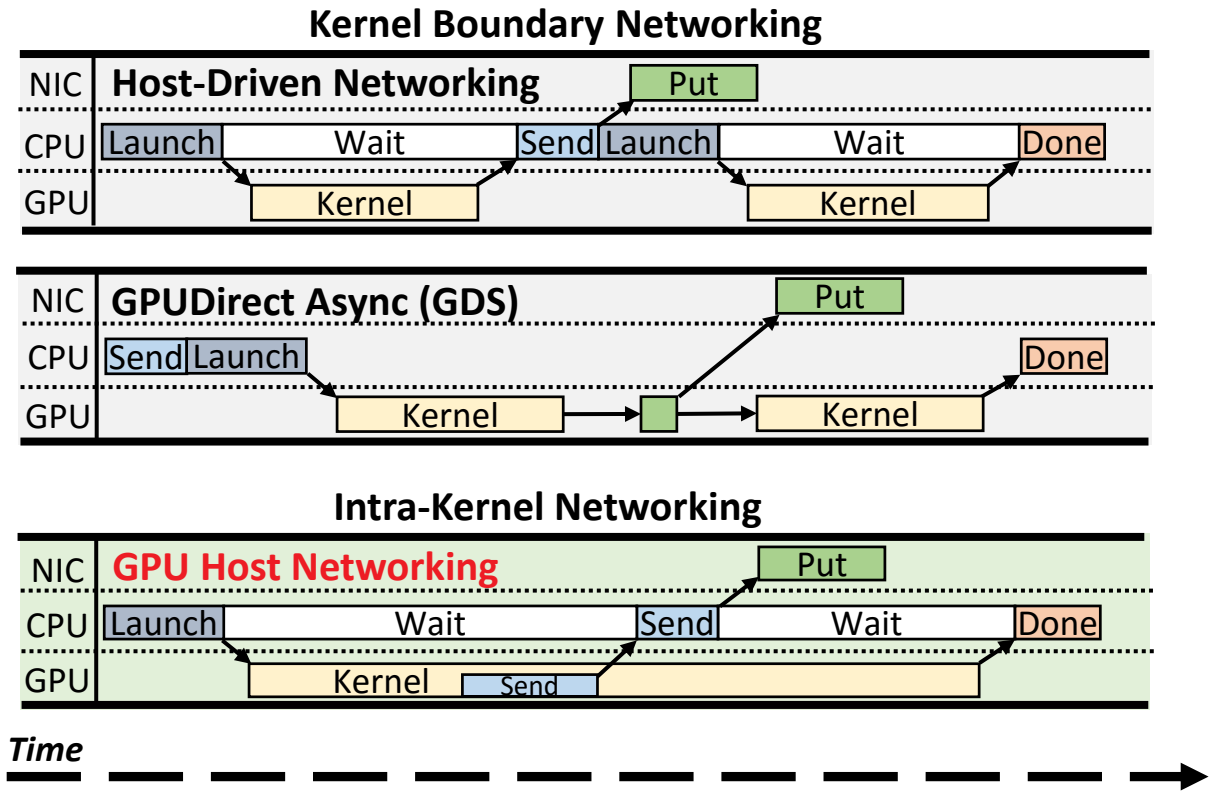
- **Obvious Solution**: Can you do networking from within a kernel?
  - Absolutely!
  - Two main schools of thought here…



# GPU HOST NETWORKING[6,7,8]

## 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 and more GPUs





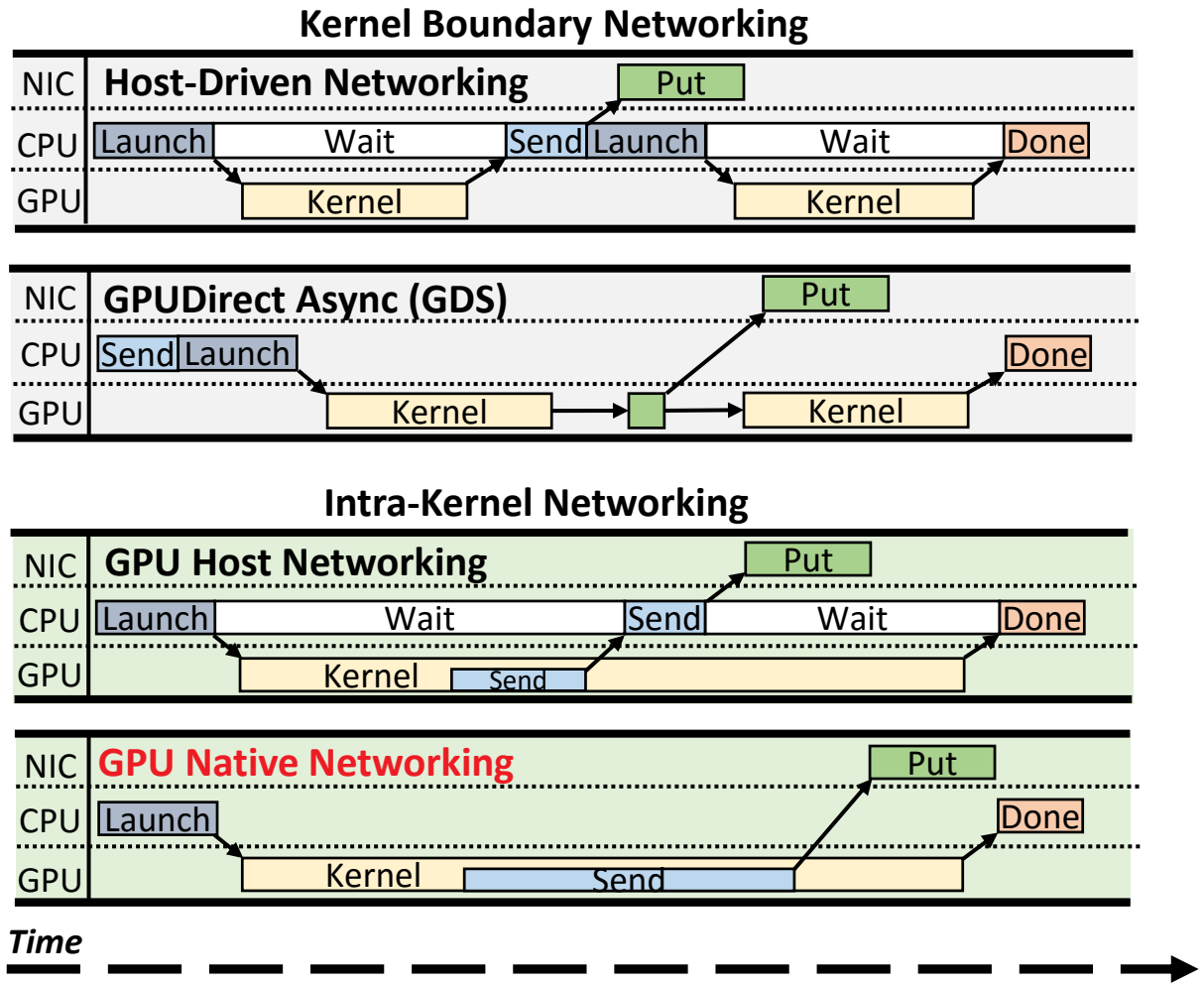
[6] Jeff A. Stuart and John D. Owens. 2009. Message passing on data-parallel architectures. In Intl. Symp. on Parallel Distributed Processing (IPDPS). 1–12.
[7] Sangman Kim, Seonggu Huh, Yige Hu, Xinya Zhang, Emmett Witchel, Amir Wated, and Mark Silberstein. 2014. GPUnet: Networking Abstractions for GPU Programs. In USENIX Conf. on Operating Systems Design and Implementation (OSDI). 201–216.
[8] 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.

# GPU NATIVE NETWORKING[9,10,11,12]

## INTRODUCTION

- Run a networking stack on the GPU
- Allow the GPU work-items themselves 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 divergence and register pressure

**Is there another option?**



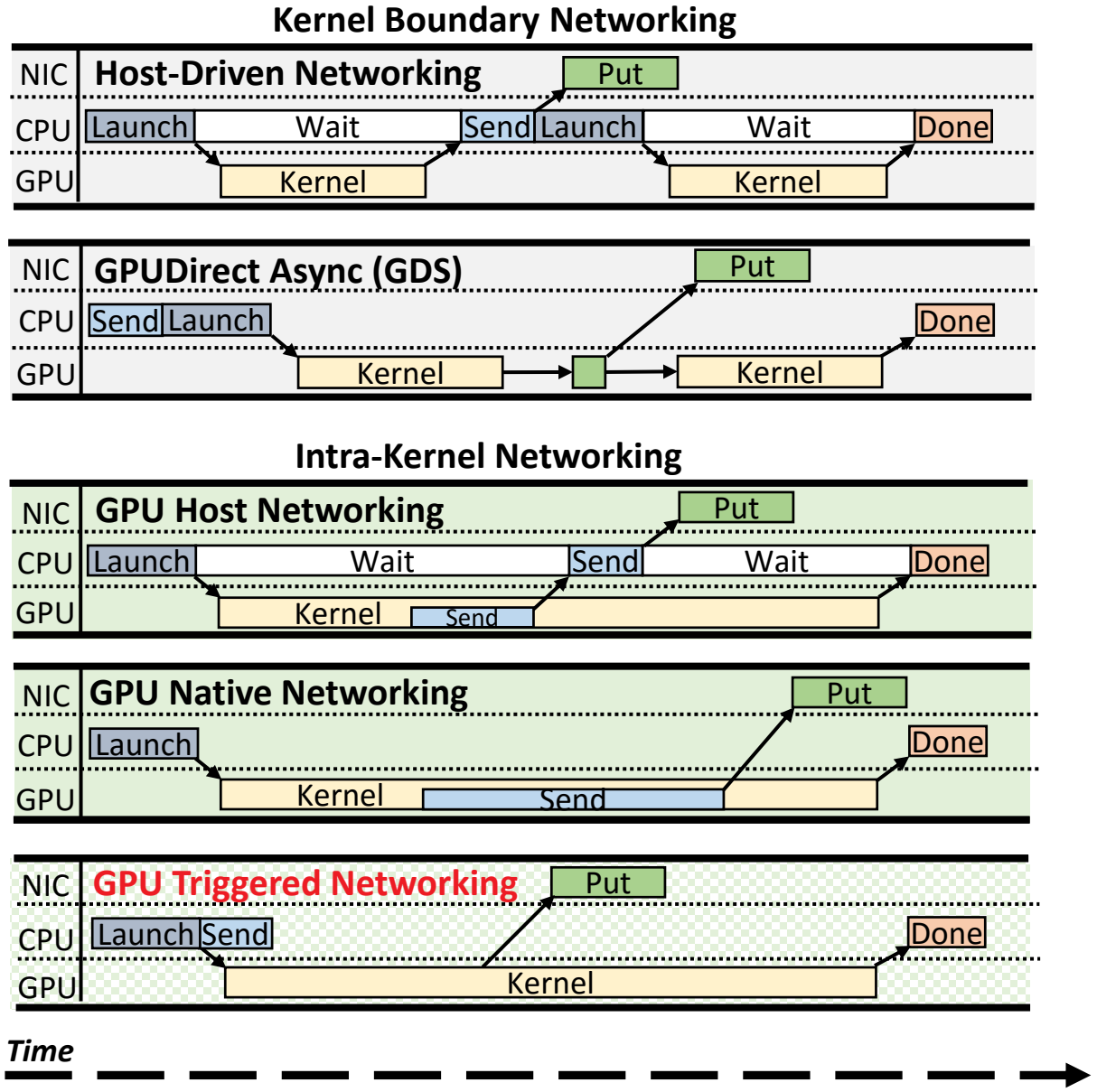


[9] Lena Oden, Holger Froning, and Franz-Joseph Pfreundt. 2014. Infiniband-Verbs on GPU: A Case Study of Controlling an Infiniband Network Device from the GPU. In Intl. Conf. on Parallel Distributed Processing Symposium Workshops (IPDPSW). 976–983.
[10] Benjamin Klenk, Lena Oden, and Holger Froning. 2014. Analyzing Put/Get APIs for Thread-Collaborative Processors. In Intl. Conf. on Parallel Processing (ICPP) Workshops.
[11] 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).
[12] 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.

# INTRODUCING GPU TRIGGERED NETWORKING

## GPU-TN ARCHITECTURE

- GPU Triggered Networking (GPU-TN)
  - Control path optimization
  - CPU prepares network operations and registers them with the GPU
  - GPU triggers the operation from ***within*** a kernel
  - Inspired by triggered operations from the Portals 4 networking API
- Similar concept to GDS with a few key differences:
  - Can be triggered from inside the GPU at **different granularities**
  - Complexity managed inside the **NIC itself**

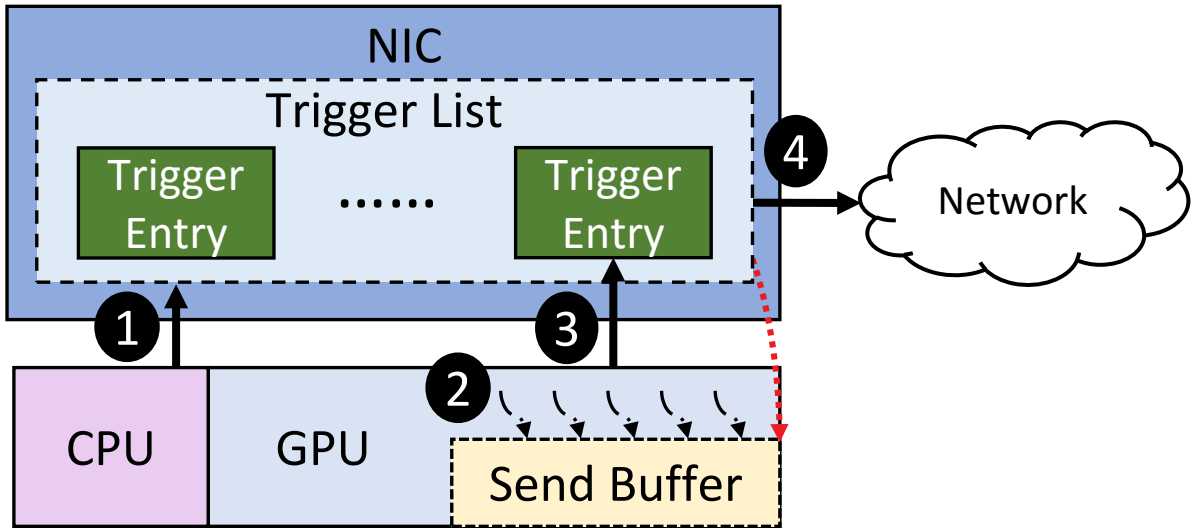


# OVERVIEW

## GPU-TN ARCHITECTURE

- Steps in a GPU-TN operation:
  1. Register the network operation with the NIC using code on the CPU
  2. GPU populates the network buffer with data it wants to send to another node
  3. GPU triggers the operation from within a kernel
     - GPU-TN supports several different granularities
     - More details to follow here
  4. NIC performs the requested network operation (Put, Get, etc.)

```
...
// Initialize RDMA comm layer
int rank = RdmaInit();
void * buf = malloc(BUFFER_SIZE);
//Register operations with the NIC
for (int i = 0; i < N_MSGS; i++)
   TrigPut(TAG + i, buf, target, thresh,
           ...);
//Request trigger address from NIC
char *trigAddr = GetTriggerAddr();
//Launch GPU Kernel
LaunchKern(trigAddr, TAG, N_MSGS, buf, ...);
// Cleanup, do more compute, etc.
...
```



# PROGRAMMING INTERFACE

## GPU-TN ARCHITECTURE

### Work-item

A thread on the GPU

```
__kernel
void kern1(__global char *trigAddr,
           const int tagBase,
           __global void *buffer)
{
   // do work
   ...
   buffer = ...;
   atomic_work_item_fence(...);
   int id = get_global_id(...);
   atomic_store_explicit(trigAddr,
                         tagBase + id,
                         ...);
   // do additional work
   ...
}
```

### Work-group

A collection of threads on the GPU

```
__kernel
void kern2(__global char *trigAddr,
           const int tagBase,
           __global void *buffer)
{
   // do work
   ...
   buffer = ...;
   work_group_barrier(...);
   if (!get_local_id(...)) {
      int id = get_group_id(...);
      atomic_store_explicit(trigAddr,
                            tagBase + id,
                            ...);
   }
   // do additional work
   ...
}
```

### Kernel

All threads in a GPU program

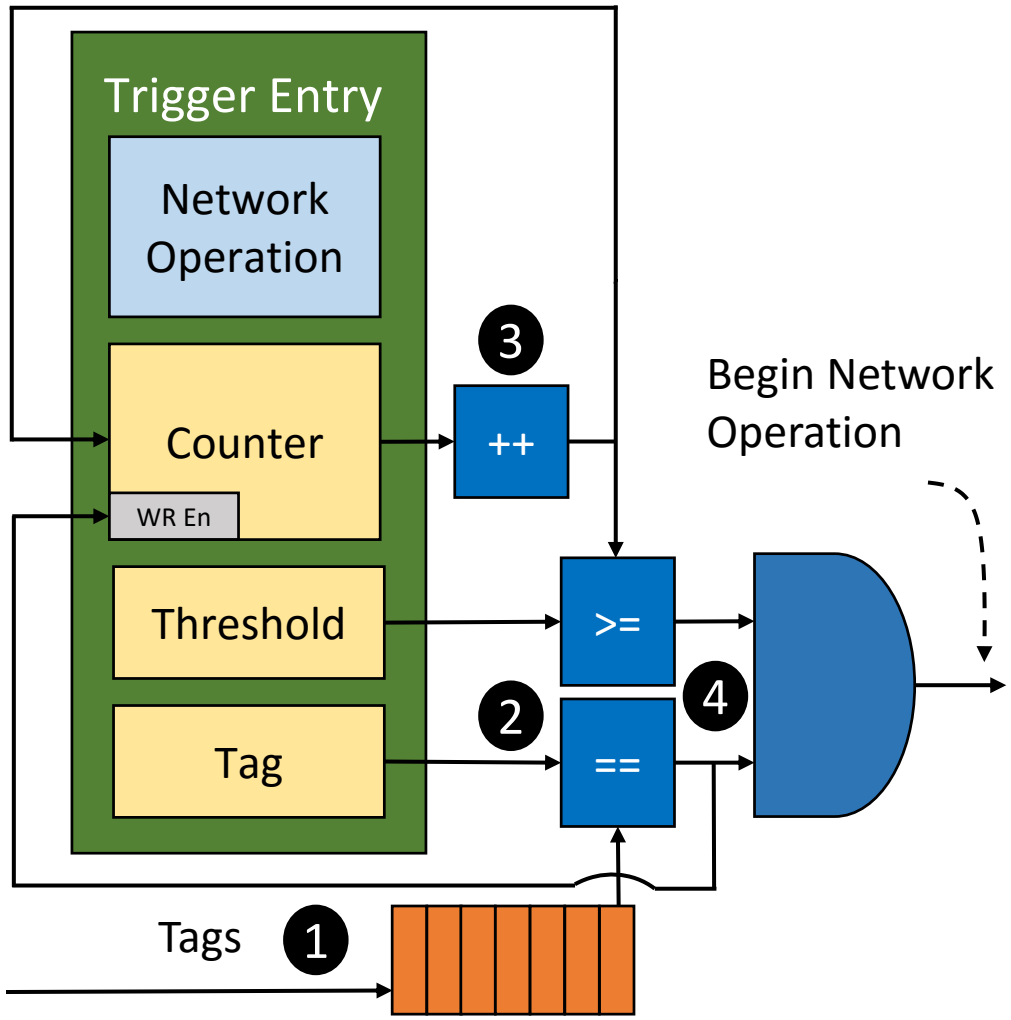
```
__kernel
void kern3(__global char *trigAddr,
           const int tag,
           __global void *buffer)
{
   // do work
   ...
   buffer = ...;
   work_group_barrier(...);
   if (!get_local_id(...)) {
      atomic_store_explicit(trigAddr,
                            tag,
                            ...);
   }
   // do additional work
   ...
}
```

- Challenges/ Caveats
  - GPU's relaxed memory consistency model
  - GPU's lack of forward progress guarantees
- Take away messages
  - Can be triggered at different granularities
  - Triggers with multiple granularities combined on the NIC

# HARDWARE COMPLEXITY

## GPU-TN ARCHITECTURE

- Steps on the NIC side
  1. GPU writes tag
  2. Tag matched to trigger entries
  3. On tag match, increment counter
  4. When counter >= threshold, perform the network operation
- Logic implementable in software or using hardware similar to the figure
- Synchronization/aggregation among messages done on the NIC



# SIMULATION ENVIRONMENT

## RESULTS

- All data collected in gem5[13]
  - AMD GPU model[14]
  - Cache-coherent, APU system architecture
  - No dedicated GPU memory
  - Static launch latency model calibrated from most optimistic real system data

- Portals 4-based NIC model[15]
  - Low-level RDMA network programming API
  - Currently supported by:
    - MPICH, Open MPI, GASNet, Berkeley UPC, and others

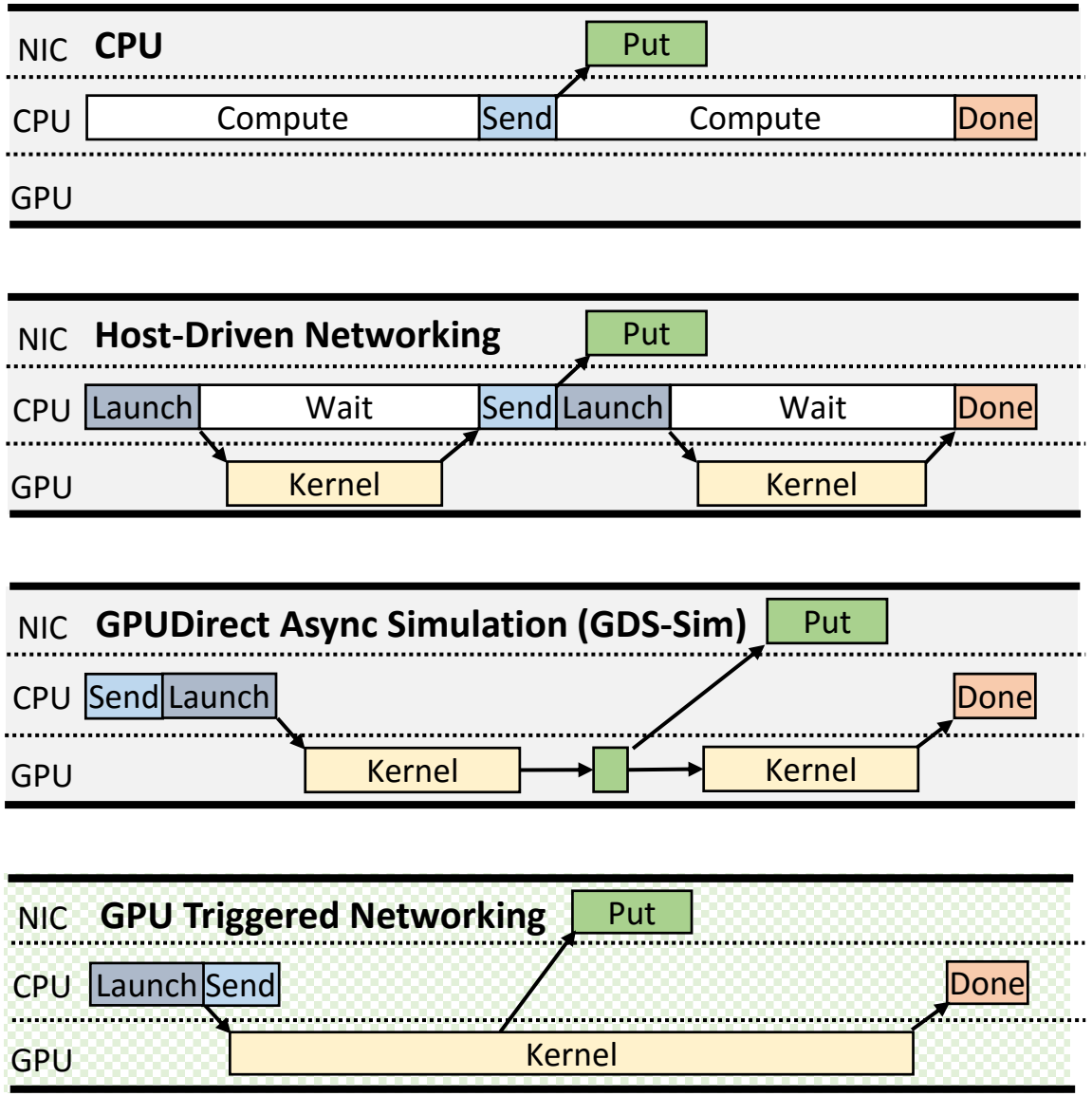
| CPU and Memory Configuration | |
|---|---|
| CPU Type | 8-wide OOO, 4Ghz, 8 cores |
| I,D-Cache | 64K, 2-way, 2 cycles |
| L2-Cache | 2MB, 8-way, 4 cycles |
| L3-Cache | 16MB, 16-way, 20 cycles |
| DRAM | DDR4, 8 Channels, 2133MHz |
| **GPU Configuration** | |
| GPU Type | 1 Ghz, 24 Compute Units |
| D-Cache | 16kB, 64B line, 16-way, 25 cycles |
| I-Cache | 32kB, 64B line, 8-way, 25 cycles |
| L2-Cache | 768kB, 64B line, 16-way, 150 cycles |
| Kernel Latency | 1.5µs launch / 1.5µs teardown |
| **NIC Configuration** | |
| Link Speed | 100ns Link, 100ns Switch |
| Bandwidth | 100 Gbps |
| Topology | Star (single switch) |


[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," SIGARCH Comput. Archit. News, pp. 1–7, 2011.
[14] AMD. (2015) The AMD gem5 APU simulator: Modeling heterogeneous systems in gem5. http://gem5.org/GPU_Models.
[15] Sandia National Laboratories, "The Portals 4.0.2 network programming interface," http://www.cs.sandia.gov/Portals/portals402.pdf, 2014.

# SYSTEMS UNDER TEST

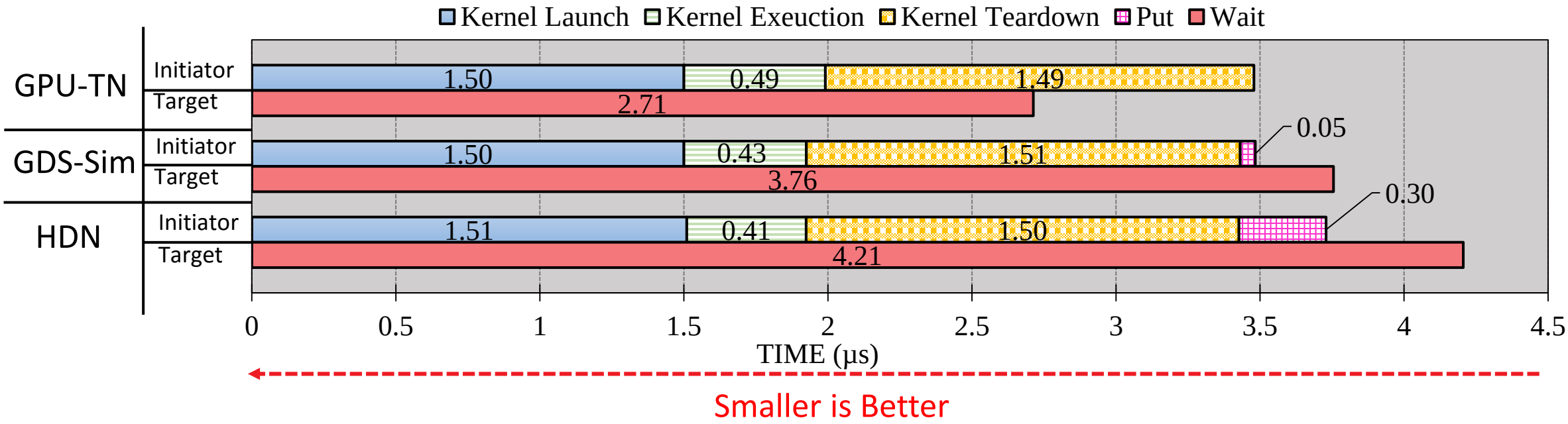
RESULTS

- CPU
  - No GPU, just for sanity check
- HDN (Host driven networking)
  - Kernel boundary, GPU<->CPU<->NIC control path
- GDS-Sim (GPUDirect Async Simulation)
  - Kernel boundary, GPU<->NIC control path
  - Simulator implementation…not necessarily representative of real GDS numbers!
- GPU-TN (GPU Triggered Networking)
  - Our scheme



# GPU-TN LATENCY

RESULTS

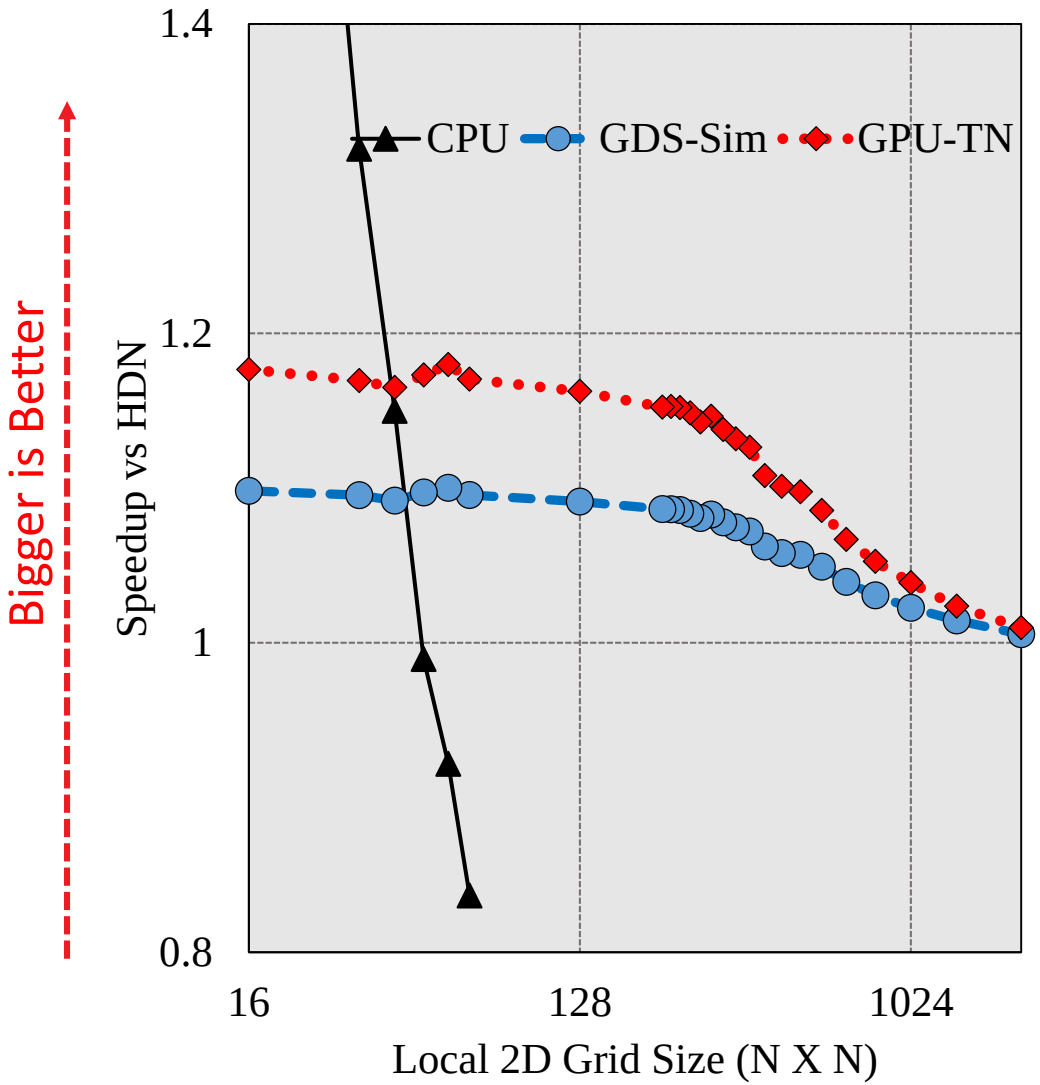



- One-sided put latency benchmark
  - Initiator launches dummy kernel, executes network command, and terminates
  - Target polls on put location
- Take-away messages
  - HDN < GDS-Sim < GPU-TN
  - GPU-TN actually overlaps kernel teardown with network transfer!

# STENCIL
## RESULTS

- 2D Jacobi stencil example
  - Computation blocks on communication
  - Communication is a simple halo exchange

- Take-away messages
  - For very small stencils
    - GPU is useless
  - For very large stencils
    - All GPU network strategies are similar
    - control path optimizations do not matter
  - For medium size stencils
    - CPU <<< HDS < GDS-Sim < GPU-TN

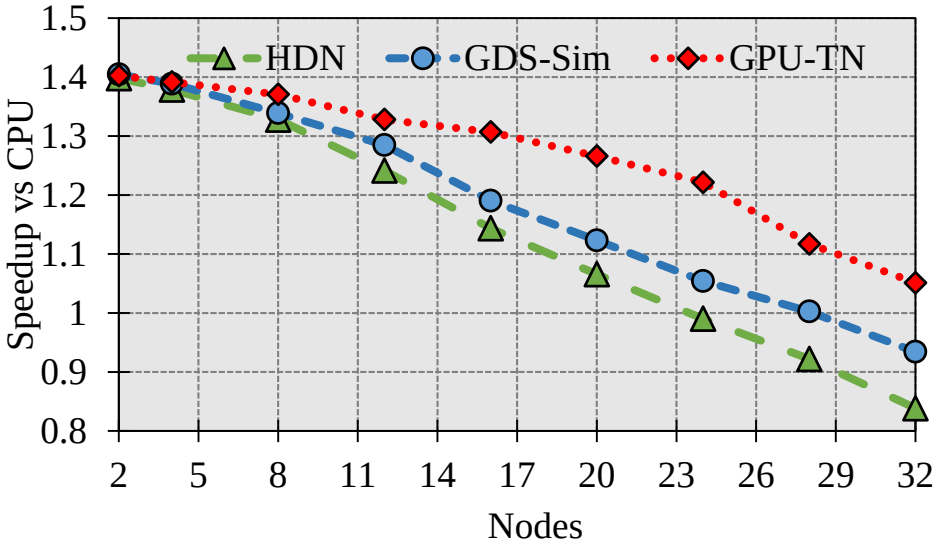


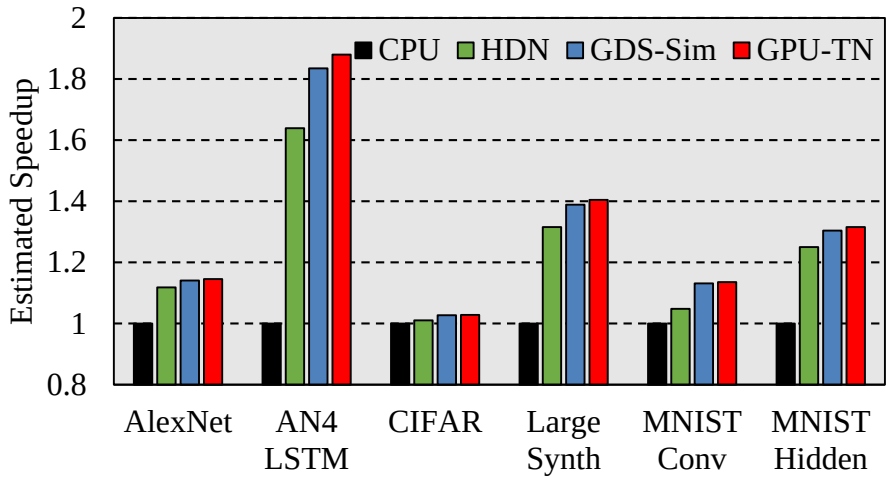
# MACHINE LEARNING (ML) TRAINING

## RESULTS

- Main communication step in ML training is AllReduce
- AllReduce microbenchmark
  - For a small number of nodes
    - all For a big numGPUs have similar speedup
  - For a large number of nodes
    - little benefit using GPUs
  - Sweet spot is, once again, in the middle
- ML training acceleration
  - Benchmarks from the Microsoft Cognitive Toolkit (CNTK)[16]
  - Results are projected using real runs augmented with Allreduce() speed-up numbers from the simulator
  - Almost nothing in CIFAR to 5-10% in AN4 LSTM
    - Speed-ups vary depending on communication to computation ratio for each workload

Bigger is Better



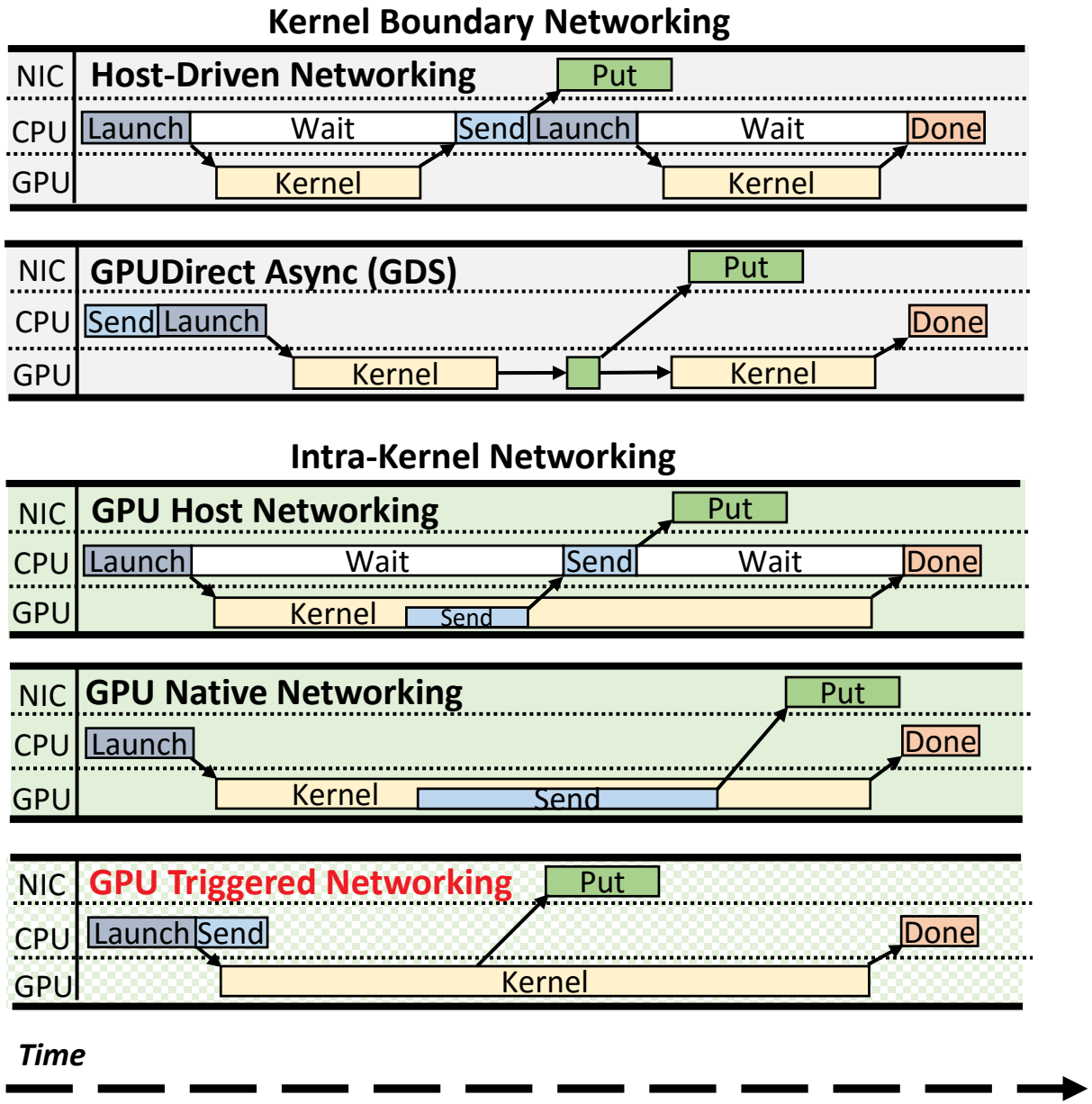



[16] Agarwal, et.al., "An introduction to computational networks and the Computational Network Toolkit," Microsoft, Technical Report, 2014.

# SUMMARY

## CONCLUSION

- Intra-kernel networking avoids costly overheads present in kernel boundary networking implementations
- GPU-TN combines intra-kernel networking with message preregistration by the CPU
- Multiple granularities of messages are supported using a small amount of hardware on the NIC
- Offers up to ~35% improvement over HDN and up to ~25% improvement vs GDS enabled solutions



# THANK YOU!

Michael.Lebeane@amd.com

mlebeane@utexas.edu

# QUESTIONS?

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

# REFERENCES


[1] TOP500.org, “Highlights – June 2017,” http://www.top500.org/lists/2017/06/highlights, 2017.
[2] Nvidia, “GPU-Accelerated Applications,” http://www.nvidia.com/content/gpu-applications/pdf/gpu-apps-catalog-mar2015.pdf, 2016.
[3] AMD. 2017. ROCn RDMA. https://github.com/RadeonOpenCompute/ROCnRDMA
[4] Mellanox. 2017. Mellanox OFED GPUDirect RDMA. http://www.mellanox.com/page/products_dyn?product_family=116
[5] 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.
[6] Jeff A. Stuart and John D. Owens. 2009. Message passing on data-parallel architectures. In Intl. Symp. on Parallel Distributed Processing (IPDPS). 1–12.
[7] 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.
[8] 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.
[9] 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.
[10] Benjamin Klenk, Lena Oden, and Holger Froning. 2014. Analyzing Put/Get APIs for Thread-Collaborative Processors. In Intl. Conf. on Parallel Processing (ICPP) Workshops.
[11] 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).
[12] 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.
[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The gem5 simulator,” SIGARCH Comput. Archit. News, pp. 1–7, 2011.
[14] AMD. (2015) The AMD gem5 APU simulator: Modeling heterogeneous systems in gem5. http://gem5.org/GPU_Models.
[15] Sandia National Laboratories, “The Portals 4.0.2 network programming interface,” http://www.cs.sandia.gov/Portals/portals402.pdf, 2014.
[16] Agarwal, et.al., An introduction to computational networks and the Computational Network Toolkit,” Microsoft, Technical Report, 2014.

# MOTIVATING EXAMPLE: MACHINE LEARNING

## INTRODUCTION

- Networks of accelerators are especially important for **machine learning**
- Almost frameworks and machines support multi-GPU, multi-node learning solutions
- Combine results through **Allreduce**() operation: