ComP-Net

Command Processor Networking for Efficient Intra-kernel Communications on GPUs

Michael LeBeane¹,², Khaled Hamidouche¹, Brad Benton¹, Mauricio Breternitz³, Steven K. Reinhardt⁴, Lizy K. John²

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

Presented by Sooraj Puthoor¹
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]
CONTROL PATH OPTIMIZATIONS

Introduction

Host Driven Networking

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

```
a_kernel<<<... , stream>>>(buf);
cudaStreamSynchronize(stream);
netSend(buf);
netWait();
b_kernel<<<... , stream>>>(buf);
cudaStreamSynchronize(stream);
```

GPU Direct Async (GDS)[3]

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

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

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

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…

![Graph showing launch latency vs. kernel commands queued for different GPUs. The x-axis represents the number of kernel commands queued, ranging from 1 to 512. The y-axis represents launch latency (µs), ranging from 0 to 20. Each GPU has a different line indicating its performance, with GPU 1 having the lowest latency. Smaller is better.](image_url)
GPU NATIVE NETWORKING\textsuperscript{[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

**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

---

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

<table>
<thead>
<tr>
<th>Work-Group</th>
<th>Command Processor Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDS / Non Coherent Cache</td>
<td>Registers / Non Coherent Cache</td>
</tr>
<tr>
<td>CP-Net GPU Context</td>
<td>CP-Net GPU Context</td>
</tr>
<tr>
<td>Write Idx</td>
<td>Base Ptr</td>
</tr>
<tr>
<td>Base Ptr</td>
<td>Local Read Idx</td>
</tr>
<tr>
<td>Read Idx Ptr</td>
<td></td>
</tr>
<tr>
<td>Local Read Idx</td>
<td></td>
</tr>
</tbody>
</table>

Cache/Memory/GPU Coherence Point

- Read Idx
- Status
- Queue Entry

- Registers / Non Coherent Cache
- CP-Net GPU Context
- Base Ptr
- Local Read Idx

- Work-Group
- Command Processor Thread
**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

**Cache/Memory/GPU Coherence Point**
COMP-NET PRODUCER/CONSUMER QUEUE
ComP-Net

2) Fill *Queue Entry* with networking metadata
- Or Inline small payloads in the *Queue Entry* itself
3) Set STATUS flag with release marker to notify CP
• 4) Increment local Write Idx
COMP-NET PRODUCER/CONSUMER QUEUE

ComP-Net

- **Cache/Memory/GPU Coherence Point**
  - Work-Group LDS / Non Coherent Cache
    - CP-Net GPU Context
      - Write Idx
      - Base Ptr
      - Read Idx Ptr
      - Local Read Idx

- Command Processor Thread

- **1)** Poll on next **Queue Entry** based on local **Read Idx** with acquire marker
2) Read data from **Queue Entry**
• 3) Perform network operation and set *Status* flag to 0 when complete with release marker
COMP-NET PRODUCER/CONSUMER QUEUE
ComP-Net

Cache/Memory/GPU Coherence Point

- **4a)** Update global *Read Idx* with release marker
- **4b)** Update local *Read Idx*
COMP-NET PRODUCER/CONSUMER QUEUE

- Work-Group
  - LDS / Non Coherent Cache
  - CP-Net GPU Context
    - Write Idx
    - Base Ptr
    - Read Idx Ptr
    - Local Read Idx

- Command Processor Thread
  - Registers / Non Coherent Cache
  - CP-Net GPU Context
    - Base Ptr
    - Local Read Idx

- Cache/Memory/GPU Coherence Point

- 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

![Graph showing L2 Hit Rate for CP with Networking Wavefronts to Streaming Wavefronts comparison between Baseline and LLC Locking]
SIMULATION ENVIRONMENT

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

\textsuperscript{[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**

<table>
<thead>
<tr>
<th>Type</th>
<th>8-wide OOO, x86, 8 cores @ 4GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>I,D-Cache</td>
<td>64KB, 2-way, 2 cycles</td>
</tr>
<tr>
<td>L2-Cache</td>
<td>2MB, 8-way, 8 cycles</td>
</tr>
<tr>
<td>L3-Cache</td>
<td>16MB, 16-way, 20 cycles</td>
</tr>
<tr>
<td>DRAM</td>
<td>DDR4, 8 Channels, 2133MHz</td>
</tr>
</tbody>
</table>

**GPU Configuration**

<table>
<thead>
<tr>
<th>Type</th>
<th>AMD GCN3 @ 1.5GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>CU Config</td>
<td>12 CUs with 4 SIMD-16 engines</td>
</tr>
<tr>
<td>Wavefronts</td>
<td>40 Waves per SIMD (64 lanes)</td>
</tr>
<tr>
<td>V-Cache</td>
<td>32KB, 16-way, 12 cycles, per CU</td>
</tr>
<tr>
<td>K-Cache</td>
<td>32KB, 8-way, 12 cycles, per 4 CU</td>
</tr>
<tr>
<td>I-Cache</td>
<td>64KB, 8-way, 12 cycles, per 4 CU</td>
</tr>
<tr>
<td>L2-Cache</td>
<td>1MB, 16-way, 8 banks, 100 cycles</td>
</tr>
</tbody>
</table>

**CP Configuration**

<table>
<thead>
<tr>
<th>Type</th>
<th>2-wide OOO, x86, 2 cores @ 2GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-Cache</td>
<td>32KB, 8-way, 4 cycles</td>
</tr>
<tr>
<td>I-Cache</td>
<td>16KB, 8-way, 4 cycles</td>
</tr>
</tbody>
</table>
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

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

Results

Node 0 (Top) → Halo Exchange → Node 1 (Bottom)

• 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

- 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

<table>
<thead>
<tr>
<th>Workload Name</th>
<th>Domain</th>
<th>%Blocked</th>
<th>Reductions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex Net</td>
<td>Classification</td>
<td>14%</td>
<td>4672</td>
</tr>
<tr>
<td>AN4 LSTM</td>
<td>Speech</td>
<td>50%</td>
<td>131192</td>
</tr>
<tr>
<td>CIFAR</td>
<td>Classification</td>
<td>4%</td>
<td>939820</td>
</tr>
<tr>
<td>Large Synth</td>
<td>Synthetic</td>
<td>28%</td>
<td>52800</td>
</tr>
<tr>
<td>MNIST Conv</td>
<td>Text Recognition</td>
<td>12%</td>
<td>900000</td>
</tr>
<tr>
<td>MNIST Hidden</td>
<td>Text Recognition</td>
<td>29%</td>
<td>900000</td>
</tr>
</tbody>
</table>
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

ComP-Net

GPU Host Networking
Thank You!

Sooraj.Puthoor@amd.com
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. 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.

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 MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY 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.

ATTRIBUTION

© 2017 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

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