Cooperative Caching for GPUs

Saumay Dublish, Vijay Nagarajan, Nigel Topham
The University of Edinburgh

at

HiPEAC
Stockholm, Sweden
24th January, 2017
Multithreading on GPUs

- Kernel
- Hardware Scheduler
- DRAM

Host CPU to GPU

Hide memory latencies with execution
Multithreading on GPUs

Kernel

Core

Core

Core

Core

Hardware Scheduler

Host CPU to GPU

Memory-intensive applications

Latencies grow

Bandwidth Bottleneck

Appear in critical path

DRAM
Deeper Memory Hierarchy

Small L1s
High multithreading

High L1 miss rates

Bandwidth Bottleneck
Deeper Memory Hierarchy

- Core
- Core
- Core
- Core

Bandwidth filtering

L1
L1
L1
L1

L2

L2 roundtrip latency
\sim 300 \text{ cycles}
\text{(2-3x higher)}

Small L1s
High multithreading
High L1 miss rates
Bandwidth Bottleneck

DRAM
Deeper Memory Hierarchy

Core → L1 → L2 → DRAM

- Bandwidth filtering
- L2 roundtrip latency: ~200 cycles
- Small L1s
- High multithreading
- High L1 miss rates
- Reduce congestion
Outline

- **Observations**: Inter-core reuse
- **Proposed Architecture**: Cooperative Caching Network
- **Results**: Reduced L2 Congestion, Lower Latencies, Speedup
- **Comparative study**: Increasing L2 banks, Clustered sharing
- **Conclusion**
Observations

Graphics applications

• Kernels work on independent data
• Thread-blocks execute in considerable isolation

General-purpose applications

• Thread-blocks tend to share data
• Thread-blocks scheduled on different cores lead to inter-core reuse
Inter-core Reuse

Benchmark: Coulombic Potential (*cutcp*)

Compute electrostatic potential from neighbours
Inter-core Reuse

Benchmark: Coulombic Potential (*cutcp*)

Intra Thread Block Reuse

Inter Thread Block Reuse

Compute electrostatic potential from neighbours
Inter-core Reuse

Benchmark: Coulombic Potential (cutep)

Results in Inter-Core Reuse
Duplicate Requests to L2
Reuse patterns

Core where the L1 miss occurs

Remote sharers for L1 miss

Strong sharing at core distance of 4

Random sharing

Strong sharing with immediate neighbour

Sharing with all cores

Reuse Score
Efficacy of Reuse

- **L1 misses cached in remote L1s**
  - btree: 18%
  - pd: 78%
  - hotspot: 18%
  - lud: 0.19%
  - ss: 33%
  - cutgc: 46%
  - ipact: 22%

- **Speedup with no reuse overhead**
Sensitivity to Reuse Overhead

(Performance as a function of remote L1 access latencies)
Sensitivity to Reuse Overhead

(Performance as a function of remote L1 access latencies)

Stable reuse latency range

Permissible range for each remote L1 access: 0-80 cycles

Mean Speedup

22%  20%

Remote L1 access latency

IPC improvement (%)
Sensitivity to Reuse Overhead

(Performance as a function of remote L1 access latencies)

Stable reuse latency range

L2 access latency range

Mean Speedup

22%  20%  13%  2%

Permissible range for each remote L1 access: 0-80 cycles
Cooperative Caching Network

- Lightweight \textit{ring-based} network for inter-core communication
- All core-to-core connections are \textit{near-neighbour}
- Fewest number of inter-core connections compared to other topologies
- Routers are simple \textit{multiplexers}
- Leverages latency tolerance of up to 80 cycles

\textbf{Key design point}

\textit{Trade-off higher latencies for simplicity and short wires}
Baseline GPU

Representation of GTX480 die shot
Cooperative Caching Network (CCN)
Cooperative Caching Network (CCN)

- Two unidirectional rings: Request and Response
- Shadow Tags at each L1 cache
Cooperative Caching Network (CCN)
Cooperative Caching Network (CCN)

MISS

Core-0
Core-1
Core-2

 ReqQ-0
RespQ-0

 ReqQ-1
RespQ-1

 ReqQ-2
RespQ-2
Cooperative Caching Network (CCN)
Cooperative Caching Network (CCN)
Cooperative Caching Network (CCN)

Cooperative Caching for GPUs
Cooperative Caching Network (CCN)
Cooperative Caching Network (CCN)
Source of Speedup?

- Reduced latency due to less congestion in L2 cache outstrips the overhead of traversing the ring

- Average memory latencies, therefore, are lower with CCN

- Detailed analytical model in the paper
Request Throttling (CCN-RT)

- Record statistics about the presence of inter-core reuse
- When no reuse is detected, throttles requests directly to L2
- Maintains low congestion in the ring and minimizes ring overhead

Memory Consistency

- GPUs employ weak memory consistency model
- CCN does not further weaken the existing memory model
Evaluation

• **Platform**
  - GPGPU-Sim (v3.2.2)
  - GPUWattch (McPAT)

• **Benchmarks**
  - Rodinia
  - Parboil
  - MapReduce
Results

Cooperative Caching for GPUs

Memory-intensive: 14.7% Speedup
Non-memory-intensive
Results

Overheads

- Area: 1.3%
- Energy: 2.5%
Comparative Study

Clustering*

*Keshtegar et al.* Cluster-based approach for improving graphics processing unit performance by inter streaming multiprocessors locality, IET-CDT 2015
Comparative Study

2x L2 Bandwidth

<table>
<thead>
<tr>
<th>IPC (normalized)</th>
<th>b-tree</th>
<th>ddl</th>
<th>hotspot</th>
<th>lud</th>
<th>spec</th>
<th>culcp</th>
<th>ipactf</th>
<th>km</th>
<th>pvr</th>
<th>ss</th>
<th>wc</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank12</td>
<td>1.2</td>
<td>1.3</td>
<td>1.4</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.3</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>bank24</td>
<td>1.1</td>
<td>1.2</td>
<td>1.5</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>bank12/CCN</td>
<td>1.1</td>
<td>1.2</td>
<td>1.4</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.3</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
<tr>
<td>bank24/CCN</td>
<td>1.1</td>
<td>1.2</td>
<td>1.5</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.2</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
</tr>
</tbody>
</table>

- 10% increase
- 14.7% increase
- 24% increase
Conclusion

• Problem:
  • High congestion between L1 and L2
  • Congestion leads to high memory latencies
  • High latencies appear in the critical path for memory-intensive applications

• Observation:
  • Considerable inter-core reuse in GPGPU applications
  • GPUs can tolerate reuse latencies gracefully up to 80 cycles
  • An aggressive policy/network to fetch sharers is an overkill

• Proposal:
  • Propose a ring-based Cooperative Caching Network
  • Trades-off higher latencies for simplicity and cost
  • Reduces congestion in the L1-L2 access path
  • Lower average memory latencies
Questions?

Saumay Dublish
saumay.dublish@ed.ac.uk
http://homepages.inf.ed.ac.uk/s1433370/