Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance

Rachata Ausavarungnirun

Saugata Ghose, Onur Kayiran, Gabriel H. Loh
Chita Das, Mahmut Kandemir, Onur Mutlu

SAFARI

Carnegie Mellon

AMD

PENNSTATE
Overview of This Talk

• Problem: memory divergence
  – Threads execute in lockstep, but not all threads hit in the cache
  – A single long latency thread can stall an entire warp

• Key Observations:
  – Memory divergence characteristic differs across warps
  – Some warps mostly hit in the cache, some mostly miss
  – Divergence characteristic is stable over time
  – L2 queuing exacerbates memory divergence problem

• Our Solution: Memory Divergence Correction
  – Uses cache bypassing, cache insertion and memory scheduling to prioritize mostly-hit warps and deprioritize mostly-miss warps

• Key Results:
  – 21.8% better performance and 20.1% better energy efficiency compared to state-of-the-art caching policy on GPU
Outline

• Background
• Key Observations
• Memory Divergence Correction (MeDiC)
• Results
Latency Hiding in GPGPU Execution

**GPU Core Status**
- **Active**
- **Stall**
- **Active**

**Time**

**Warp A**

**Warp B**

**Warp C**

**Warp D**

**Lockstep Execution**

**Thread**

SAFARI
Problem: Memory Divergence

Warp A

Cache Hit

Stall Time

Main Memory

Time

Cache Miss

Cache Hit
Outline

• Background
• **Key Observations**
• **Memory Divergence Correction (MeDiC)**
• Results
Observation 1: Divergence Heterogeneity

Key Idea:
- Convert mostly-hit warps to all-hit warps
- Convert mostly-miss warps to all-miss warps
Observation 2: Stable Divergence Char.

• Warp retains its hit ratio during a program phase
  – Hit ratio $\rightarrow$ number of hits / number of access
Observation 2: Stable Divergence Char.

- Warp retains its hit ratio during a program phase
Observation 3: Queuing at L2 Banks

45% of requests stall 20+ cycles at the L2 queue. Long queuing delays exacerbate the effect of memory divergence.
Outline

• Background
• Key Observations
• **Memory Divergence Correction (MeDiC)**
• Results
Our Solution: MeDiC

• Key Ideas:
  – Convert **mostly-hit** warps to **all-hit** warps
  – Convert **mostly-miss** warps to **all-miss** warps
  – Reduce L2 queuing latency
  – Prioritize mostly-hit warps at the memory
  – Maintain memory bandwidth
Memory Divergence Correction

Warp-type-aware Cache Bypassing

Memory Type Identification Logic

Bypassing Logic

Shared L2 Cache

Bank 0
Bank 1
Bank 2
Bank n

Low Priority
High Priority

N
Y

Any Requests in High Priority

Memory Scheduler

Warp-type-aware Cache Insertion Policy

SAFARI
Mechanism to Identify Warp Type

- Profile hit ratio for each warp
- Group warp into one of five categories

- Higher Priority
  - All-hit
  - Mostly-hit
  - Balanced

- Lower Priority
  - Mostly-miss
  - All-miss

- Periodically reset warp-type
Warp-type-aware Cache Bypassing

Warp-type-aware Cache Bypassing

Warp-type-aware Cache Insertion Policy

Warp Type Identification Logic

Bypassing Logic

Shared L2 Cache

Bank 0

Bank 1

Bank 2

Bank n

Warp-type-aware Memory Scheduler

Low Priority

High Priority

Any Requests in High Priority

N

Y

To DRAM

Memory Request

Mostly-miss, All-miss
Warp-type-aware Cache Bypassing

• Goal:
  – Convert mostly-hit warps to all-hit warps
  – Convert mostly-miss warps to all-miss warps

• Our Solution:
  – All-miss and mostly-miss warps → Bypass L2
  – Adjust how we identify warps to maintain miss rate

• Key Benefits:
  – More all-hit warps
  – Reduce queuing latency for all warps
Warp-type-aware Cache Insertion

Warp-type-aware Cache Bypassing

Mostly-miss, All-miss

Memory Request → Warp Type Identification Logic → Bypassing Logic → Warp-time-aware Cache Bypassing Logic → Shared L2 Cache

Bank 0 → Bank 1 → Bank 2 → Bank n

Warp-type-aware Memory Scheduler

Low Priority

High Priority

Any Requests in High Priority

N → Y

To DRAM

Warp-type-aware Cache Insertion Policy
Warp-type-aware Cache Insertion

• **Goal:** Utilize the cache well
  – Prioritize mostly-hit warps
  – Maintain blocks with high reuse

• **Our Solution:**
  – All-miss and mostly-miss \(\rightarrow\) Insert at LRU
  – All-hit, mostly-hit and balanced \(\rightarrow\) Insert at MRU

• **Benefits:**
  – All-hit and mostly-hit are **less likely** to be evicted
  – **Heavily reused cache blocks** from mostly-miss are likely to **remain in the cache**
Warp-type-aware Memory Sched.

Warp-type-aware Cache Bypassing

Mostly-miss, All-miss

Memory Request

Warp Type Identification Logic

Bypassing Logic

Shared L2 Cache

Bank 0

Bank 1

Bank 2

Bank n

Low Priority

High Priority

Any Requests in High Priority

Warp-type-aware Memory Scheduler

Warp-type-aware Cache Insertion Policy

SAFARI
Not All Blocks Can Be Cached

• Despite best efforts, accesses from mostly-hit warps *can still miss* in the cache
  – Compulsory misses
  – Cache thrashing

• **Solution:** Warp-type-aware memory scheduler
Warp-type-aware Memory Sched.

• Goal: Prioritize mostly-hit over mostly-miss
• Mechanism: Two memory request queues
  – High-priority → all-hit and mostly-hit
  – Low-priority → balanced, mostly-miss and all-miss
• Benefits:
  – Mostly-hit warps stall less
MeDiC: Example

- Mostly-miss Warp
  - Cache Queuing Latency
  - DRAM Queuing Latency
  - Cache/Mem Latency
  - Bypass Cache
  - Insert at LRU

- All-miss Warp
  - Lower queuing latency

- Mostly-hit Warp
  - High Priority
  - Insert at MRU

- All-hit Warp
  - Lower stall time
Outline

• Background
• Key Observations
• Memory Divergence Correction (MeDiC)
• Results
Methodology

• Modified GPGPU-Sim modeling GTX480
  – 15 GPU cores
  – 6 memory partition
  – 16KB 4-way L1, 768KB 16-way L2
  – Model L2 queue and L2 queuing latency
  – 1674 MHz GDDR5

• Workloads from CUDA-SDK, Rodinia, Mars and Lonestar suites
Comparison Points

• **FR-FCFS baseline** [Rixner+, ISCA’00]

• **Cache Insertion:**
  – **EAF** [Seshadri+, PACT’12]
    • Tracks blocks that are recently evicted to detect high reuse and inserts them at the MRU position
    • Does not take divergence heterogeneity into account
    • Does not lower queuing latency

• **Cache Bypassing:**
  – **PCAL** [Li+, HPCA’15]
    • Uses tokens to limit number of warps that gets to access the L2 cache → Lower cache thrashing
    • Warps with highly reuse access gets more priority
    • Does not take divergence heterogeneity into account
  – PC-based and Random bypassing policy
Results: Performance of MeDiC

MeDiC is effective in identifying warp-type and taking advantage of divergence heterogeneity.
Results: Energy Efficiency of MeDiC

Performance improvement outweighs the additional energy from extra cache misses

SAFARI
Other Results in the Paper

• Breakdowns of each component of MeDiC
  – Each component is effective
• Comparison against PC-based and random cache bypassing policy
  – MeDiC provides better performance
• Analysis of combining MeDiC+reuse mechanism
  – MeDiC is effective in caching highly-reused blocks
• Sensitivity analysis of each individual components
  – Minimal impact on L2 miss rate
  – Minimal impact on row buffer locality
  – Improved L2 queuing latency
Conclusion

• **Problem: memory divergence**
  – Threads execute in lockstep, but not all threads hit in the cache
  – A single long latency thread can stall an entire warp

• **Key Observations:**
  – Memory divergence characteristic differs across warps
  – Some warps mostly hit in the cache, some mostly miss
  – Divergence characteristic is stable over time
  – L2 queuing exacerbates memory divergence problem

• **Our Solution: Memory Divergence Correction**
  – Uses cache bypassing, cache insertion and memory scheduling to prioritize mostly-hit warps and de prioritize mostly-miss warps

• **Key Results:**
  – 21.8% better performance and 20.1% better energy efficiency compared to state-of-the-art caching policy on GPU
Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance

Rachata Ausavarungnirun

Saugata Ghose, Onur Kayiran, Gabriel H. Loh
Chita Das, Mahmut Kandemir, Onur Mutlu
Backup Slides
Queuing at L2 Banks: Real Workloads

Fract. of L2 Requests vs. Queuing Time (cycles)

- Fract. of L2 Requests
- Queuing Time (cycles)

- 53.8% for 0-19 cycles
- 16% for 20-39 cycles
- 14% for 40-59 cycles
- 12% for 60-79 cycles
- 10% for 80-99 cycles
- 8% for 100-119 cycles
- 6% for 120-139 cycles
- 4% for 140-159 cycles
- 2% for 160-179 cycles
- 1% for 180-199 cycles
- 0.5% for 200-219 cycles
- 0.25% for 220-239 cycles
- 0.1% for 240-259 cycles
- 0.05% for 260-279 cycles
- 0.025% for 280-299 cycles
- 0.01% for 300+ cycles
Adding More Banks

Normalized Performance

- 12 Banks 2 Ports
- 24 Banks 2 Ports
- 24 Banks 4 Ports
- 48 Banks 2 Ports

5%
Queuing Latency Reduction

![Graph showing queuing latency reduction for different algorithms: Baseline, WByp, and MeDiC. The graph indicates a reduction of 69.8% on average.](Image)
MeDiC: Performance Breakdown
MeDiC: Row Buffer Hit Rate

![Graph showing row buffer hit rate for different algorithms and workloads. The graph compares Baseline, WMS, and MeDiC. The x-axis represents the workloads, and the y-axis represents the row buffer hit rate. The average hit rate is also shown.]
MeDiC-Reuse

![Graph showing speedup over baseline for MeDiC and MeDiC-reuse across various benchmarks. The graph includes benchmarks such as NN, CONS, SCP, BP, HS, SC, IIX, PVC, PVR, SS, BFS, BH, DMR, MST, SSSP, and Average.]
L2 Queuing Penalty
## Divergence Distribution

<table>
<thead>
<tr>
<th>#</th>
<th>Application</th>
<th>AH</th>
<th>MH</th>
<th>BL</th>
<th>MM</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nearest Neighbor (NN) [48]</td>
<td>19%</td>
<td>79%</td>
<td>1%</td>
<td>0.9%</td>
<td>0.1%</td>
</tr>
<tr>
<td>2</td>
<td>Convolution Separable (CONS) [48]</td>
<td>9%</td>
<td>1%</td>
<td>82%</td>
<td>1%</td>
<td>7%</td>
</tr>
<tr>
<td>3</td>
<td>Scalar Product (SCP) [48]</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.7%</td>
<td>99%</td>
</tr>
<tr>
<td>4</td>
<td>Back Propagation (BP) [6]</td>
<td>10%</td>
<td>27%</td>
<td>48%</td>
<td>6%</td>
<td>9%</td>
</tr>
<tr>
<td>5</td>
<td>Hotspot (HS) [6]</td>
<td>1%</td>
<td>29%</td>
<td>69%</td>
<td>0.5%</td>
<td>0.5%</td>
</tr>
<tr>
<td>6</td>
<td>Streamcluster (SC) [6]</td>
<td>6%</td>
<td>0.2%</td>
<td>0.5%</td>
<td>0.3%</td>
<td>93%</td>
</tr>
<tr>
<td>7</td>
<td>Inverted Index (IIX) [17]</td>
<td>71%</td>
<td>5%</td>
<td>8%</td>
<td>1%</td>
<td>15%</td>
</tr>
<tr>
<td>8</td>
<td>Page View Count (PVC) [17]</td>
<td>4%</td>
<td>1%</td>
<td>42%</td>
<td>20%</td>
<td>33%</td>
</tr>
<tr>
<td>9</td>
<td>Page View Rank (PVR) [17]</td>
<td>18%</td>
<td>3%</td>
<td>28%</td>
<td>4%</td>
<td>47%</td>
</tr>
</tbody>
</table>
## Divergence Distribution

<table>
<thead>
<tr>
<th>#</th>
<th>Application</th>
<th>AH</th>
<th>MH</th>
<th>BL</th>
<th>MM</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Similarity Score (SS) [17]</td>
<td>67%</td>
<td>1%</td>
<td>11%</td>
<td>1%</td>
<td>20%</td>
</tr>
<tr>
<td>11</td>
<td>Breadth-First Search (BFS) [4]</td>
<td>40%</td>
<td>1%</td>
<td>20%</td>
<td>13%</td>
<td>26%</td>
</tr>
<tr>
<td>12</td>
<td>Barnes-Hut N-body Simulation (BH) [4]</td>
<td>84%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td>15%</td>
</tr>
<tr>
<td>13</td>
<td>Delaunay Mesh Refinement (DMR) [4]</td>
<td>81%</td>
<td>3%</td>
<td>3%</td>
<td>1%</td>
<td>12%</td>
</tr>
<tr>
<td>14</td>
<td>Minimum Spanning Tree (MST) [4]</td>
<td>53%</td>
<td>12%</td>
<td>18%</td>
<td>2%</td>
<td>15%</td>
</tr>
<tr>
<td>15</td>
<td>Survey Propagation (SP) [4]</td>
<td>41%</td>
<td>1%</td>
<td>20%</td>
<td>14%</td>
<td>24%</td>
</tr>
</tbody>
</table>
Stable Divergence Characteristics

• Warp retains its hit ratio during a program phase

• Heterogeneity
  – Control Divergence
  – Memory Divergence
  – Edge cases on the data the program is operating on
  – Coalescing
  – Affinity to different memory partition

• Stability
  – Temporal + spatial locality
Warps Can Fetch Data for Others

• **All-miss and mostly-miss** warps can fetch cache blocks for other warps
  – Blocks with high reuse
  – Shared address with all-hit and mostly-hit warps

• **Solution:** Warp-type aware cache insertion
Warp-type Aware Cache Insertion

Future Cache Requests

L2 Cache

SAFARI
Warp-type Aware Memory Sched.

Memory Request

Warp Type

All-hit, Mostly-hit

Balanced, Mostly-miss, All-miss

High Priority Queue

Memory Request Queue

FR-FCFS Scheduler

Memory Schedule

FR-FCFS Scheduler

Main Memory