CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration in GPGPU Workloads
S. –Y Lee, A. A. Kumar and C. J Wu
ISCA 2015

Goal

- Reduce warp divergence and hence increase throughput
- The key is the identification of critical (lagging) warps
- Manage resources and scheduling decisions to speed up the execution of critical warps thereby reducing divergence
**Review: Resource Limits on Occupancy**

- **Kernel Distributor**
  - **SM Scheduler**
  - **SM** (Stream Multiprocessor)
  - **DRAM**

- **Thread Block Control**
  - **TB 0**
  - **Warp Schedulers**
  - **Warp Context**
  - **Register File**
  - **L1/Shared Memory**

- **Locality effects**

**Evolution of Warps in TB**

- **Coupled lifetimes of warps in a TB**
  - Start at the same time
  - Synchronization barriers
  - Kernel exit (implicit synchronization barrier)

Figure from P. Xiang, Et. Al; "Warp Level Divergence: Characterization, Impact, and Mitigation"
Warp Execution Time Disparity

- Branch divergence, interference in the memory system, scheduling policy, and workload imbalance

Figure from S. Y. Lee, et. al, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration in GPGPU Workloads," ISCA 2015

Workload Imbalance

- Imbalance exists even without control divergence
Branch Divergence

Warp-to-warp variation in dynamic instruction count (w/o branch divergence)

Intra-warp branch divergence

Example: traversal over constant node degree graphs

Extent of serialization over a CFG traversal varies across warps
Impact of the Memory System

Executing Warps

- >60% of Cache Misses for Critical Warps
- Could have been re-used
- Too slow!

Intra-wavefront footprints in the cache

Scheduling gaps are greater than the reuse distance

Figure from S. Y. Lee, et. al, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration in GPGPU Workloads," ISCA 2015

Impact of Warp Scheduler

- Amplifies the critical warp effect

Figure from S. Y. Lee, et. al, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration in GPGPU Workloads," ISCA 2015
Criticality Predictor Logic

- Non-divergent branches can generate large differences in dynamic instruction counts across warps (e.g., $m >> n$)
- Update CPL counter on branch: estimate dynamic instruction count
- Update CPL counter on instruction commit

Warp Criticality Problem

Manage resources and schedules around Critical Warps
CPL Calculation

\[ n \text{Criticality} = n\text{Instr} \times w.CPI_{avg} + n\text{Stall} \]

- Average instruction
- Disparity between warps
- Average warp CPI
- Inter-instruction memory stall cycles

Scheduling Policy

- Select warp based on criticality
- Execute until no more instructions are available
  - A form of GTO
- Critical warps get higher priority and large time slice
Behavior of Critical Warp References

Criticality Aware Cache Prioritization

- Prediction: critical warp
- Prediction: re-reference interval
- Both used to manage the cache footprint

Figure from S. Y. Lee, et. al, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration in GPGPU Workloads," ISCA 2015
Integration into GPU Microarchitecture

Criticality Prediction  Cache Management

Figure from S. Y. Lee, et. al, "CAWA: Coordinated Warp Scheduling and Cache Prioritization for Critical Warp Acceleration in GPGPU Workloads," ISCA 2015

Performance

Periodic computation of accuracy for critical warps

Due in part to low miss rates

(17)

(18)
Summary

- Warp divergence leads to some lagging warps \( \rightarrow \) critical warps
- Expose the performance impact of critical warps \( \rightarrow \) throughput reduction
- Coordinate scheduler and cache management to reduce warp divergence

Cache Conscious Wavefront Scheduling
T. Rogers, M O’Conner, and T. Aamodt
MICRO 2012
Goal

- Understand the relationship between schedulers (warp/wavefront) and locality behaviors
  - Distinguish between inter-wavefront and intra-wavefront locality
- Design a scheduler to match #scheduled wavefronts with the L1 cache size
  - Working set of the wavefronts fits in the cache
  - Emphasis on intra-wavefront locality

Reference Locality

- Scheduler decisions can affect locality
- Need non-oblivious (to locality) schedulers

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Cache Conscious Wavefront Scheduling," MICRO 2012
Key Idea

- Impact of issuing new wavefronts on the intra-warp locality of executing wavefronts
  - Footprint in the cache
  - When will a new wavefront cause thrashing?

Concurrency vs. Cache Behavior

- Adding more wavefronts to hide latency traded off against creating more long latency memory references due to thrashing

Figure from T. Rogers, M. O’Connor, T. Aamodt, “Cache Conscious Wavefront Scheduling,” MICRO 2012
Additions to the GPU Microarchitecture

Control issue of wavefronts

Feedback

Keep track of locality behavior on a per wavefront basis

Intra-Wavefront Locality

Example 1: Example graph algorithm kernel run by each scalar thread.

```c
int node_degree = nodes[thread_id].degree;
int thread_first_edge = nodes[thread_id].starting_edge;
for (int i = 0; i < node_degree; i++) {
    edge_attributes = edges[thread_first_edge + i];
    int neighbour_node_id = edge_attributes.node;
    int edge_weight = edge_attributes.weight;
    ...
}
```

- Intra-thread locality leads to intra-wavefront locality
- #wavefronts determined by its reference footprint and cache size
- Scheduler attempts to find the right #wavefronts
  - Scalar thread traverses edges
  - Edges stored in successive memory locations
  - One thread vs. many
Static Wavefront Limiting

- Limit the number of wavefronts based on working set and cache size
  - Based on profiling?
- Current schemes allocate wavefronts based on resource consumption not effectiveness of utilization
- Seek to shorten the re-reference interval

Cache Conscious Wavefront Scheduling

- Basic Steps
  - Keep track of lost locality of a wavefront
  - Control number of wavefronts that can be issued
- Approach
  - List of wavefronts sorted by lost locality score
  - Stop issuing wavefronts with least lost locality
Keeping Track of Locality

1. Fetch
2. Decode
3. I-Buffer
4. Issue
5. PRF
6. RF
7. Pipeline
8. D-Cache
9. All Hit?
10. Writeback

Keep track of associated Wavefronts

Indexed by WID
Victims indicate "lost locality"

Update lost locality score

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Cache Conscious Wavefront Scheduling," MICRO 2012 (29)

Updating Estimates of Locality

Waves Ready [1:N]
Baseline Priority Logic

Prioritized Waves [1:N]
Intersection

Wavefront Issue Arbiter

LSS

Can Issue [1:N]

Score [1:N]

To Exec Inst. (WID)

Update wavefront lost locality score on victim hit

Changes based on #wavefronts

Wavefronts sorted by lost locality score

Note which are allowed to issue

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Cache Conscious Wavefront Scheduling," MICRO 2012 (30)
Estimating the Feedback Gain

- Prevent issue
- VYA hit – increase score
- Drop the scores – no VTA hits

\[ LLDS = \frac{VTA\text{Hits}\_{\text{total}}}{\text{InsIssued}\_{\text{total}}} \cdot K_{\text{throttle}} \cdot \text{CumLLSCutoff} \]

Scale gain based on percentage of cutoff

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Cache Conscious Wavefront Scheduling," MICRO 2012

Schedulers

- Loose Round Robin
- Greedy-Then-Oldest (GTO)
  - Execute one wavefront until stall and then oldest
- 2LVL-GTO
  - Two level scheduler with GTO instead of RR
- Best SWL
- CCWS
Some General Observations

- Performance largely determined by
  - Emphasis on oldest wavefronts
  - Distribution of references across cache lines – extent of lack of coalescing

- GTO works well for this reason \( \rightarrow \) prioritizes older wavefronts

- LRR touches too much data (too little reuse) to fit in the cache
Summary

- Dynamic tracking of relationship between wavefronts and working sets in the cache
- Modify scheduling decisions to minimize interference in the cache
- Tunables: need profile information to create stable operation of the feedback control
Divergence Aware Warp Scheduling
T. Rogers, M O’Conner, and T. Aamodt
MICRO 2013

Goal

• Understand the relationship between schedulers (warp/wavefront) and control & memory locality behaviors
  ❖ Distinguish between inter-wavefront and intra-wavefront locality

• Design a scheduler to match #scheduled wavefronts with the L1 cache size
  ❖ Working set of the wavefronts fits in the cache
  ❖ Emphasis on intra-wavefront locality
  ❖ Couple effects of control flow divergence

• Differs from CCWS in being proactive
  ❖ Deeper look at what happens inside loops
Key Idea

- Manage the relationship between control divergence, memory divergence and scheduling

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013

Key Idea (2)

- Coupling control divergence and memory divergence
  - The former indicates reduced cache capacity
  - “learning” eviction patterns
**Key Idea (2)**

while \((i < C[tid+1])\)

Fill the cache with 4 references – delay warp 1

Available room in the cache due to divergence in warp 0 – schedule warp 1

Use warp 0 behavior to predict interference due to warp 1

---

**Goal**

- Each thread computes a row
- Structured similar to a multithreaded CPU version
  - Desirable

---

**Simpler portable version**

Example 1 Highly Divergent SPMV-Scalar Kernel

```c
#include <iostream>

__global__ void
spmv_cost_scalar_kernel(const float* val,
const int* cols,
const int* rowStarts,
const int dim,
float* out)
{
    int myRow = blockIdx.y * blockDim.y + threadIdx.y;
    texReader vecTexReader;
    if (myRow < dim)
    {
        float t = 0.0f;
        int start = rowStarts[myRow];
        int end = rowStarts[myRow + 1];
        // Divergent Branch
        for (int j = start; j < end; j++)
        {
            // Uncoalesced Loads
            int col = cols[j];
            t += vecTexReader[col];
        }
        out[myRow] = t;
    }
}
```

---

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013
Goal

- Each warp handles a row
- Renumber threads modulo warp size
- Row element index for each thread
- Warps traverse the whole row
- Each thread handles a few row (column elements)
- Need to get sum the partial sums computed by each thread in a row

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013

Optimizing the Vector Version

- What do need to know?
- TB & warp sizes, grid mappings
- Tuning TB sizes as a function of machine size
- Orchestrate the partial product summation

Simplicity (productivity) with no sacrifice in performance
**Goal**

**Simpler portable version**

Example 1 Highly Divergent SPMV-Scalar Kernel

```c
__global__ void
spmv_csc_scalar_kernel(const float* val,
const int* cols, const int* rowdlen,
int n, int nBW)
{
    int myRow = blockIdx.x * blockDim.x + threadIdx.x;
    if (myRow < n)
    {
        float t = 0.0f;
        int start = rowdlen[myRow];
        int end = rowdlen[myRow+1];
        for (int j = start; j < end; j++)
        {
            // Uncoalesced Loads
            int col = cols[j];
            t += val[j] * vec4x4srs(file[col]);
        }
        out[myRow] = t;
    }
}
```

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013

**GPU-Optimized Version**

Example 2 GPU-Optimized SPMV-Vector Kernel

```c
__global__ void
spmv_csc_scalar_kernel(const float* val,
const int* cols, const int* rowdlen,
int n, int nBW)
{
    int myRow = blockIdx.x * blockDim.x + threadIdx.x;
    int myRow = threadIdx.x;
    for (int j = threadIdx.x; j < blockDim.x; j += blockDim.x)
    {
        // Uncoalesced Loads
        int col = cols[j];
        t += val[j] * vec4x4srs(file[col]);
    }
    out[myRow] = t;
}
```

**Additions to the GPU Microarchitecture**

- Control issue of warps
- Implements PDOM for control divergence

![Diagram of GPU microarchitecture](image)
Observation

- Bulk of the accesses in a loop come from a few static load instructions
- Bulk of the locality in (these) applications is intra-loop

Distribution of Locality

- Bulk of the locality comes from a few static loads in loops
- Find temporal reuse

Hint: Can we keep data from last iteration?
A Solution

- **Prediction** mechanisms for locality across iterations of a loop
- **Schedule** such that data fetched in one iteration is still present at next iteration
- Combine with control flow divergence (how much of the footprint needs to be in the cache?)

![Example 1 Highly Divergent SPMV-Scalar Kernel](Image)

Classification of Dynamic Loads

- Group static loads into equivalence classes → reference the same cache line
- Identify these groups by repetition ID
- Prediction for each load by compiler or hardware

![Classification of Dynamic Loads](Image)
Coupling Divergence Effects

- Can we improve performance with branch prediction?
  - Diverged warps reduce cache footprint

Predicting a Warp’s Cache Footprint

- Entering loop body
  - Create footprint prediction

- Exit loop
  - Reinitialize prediction

- Some threads exit the loop
  - Predicted footprint drops

- Predict locality usage of static loads
  - Not all loads increase the footprint

- Combine with control divergence to predict footprint
  - Use footprint to throttle/not-throttle warp issue

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013
Predicting a Warp’s Active Thread Count

- Modulate footprint based on predicted cache usage
  - Predictions include control divergence effects
- Classify each loop static load based on divergence and repetition ID → profiled DAWs

Principles of Operation

- Prefix sum of each warp’s cache footprint used to select warps that can be issued
  \[ \text{EffCacheSize} = k \text{AssocFactor} \cdot \text{TotalNumLines} \]
  - Scaling back from a fully associative cache
  - Empirically determined

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013
Principles of Operation (2)

Example 1: Highly Divergent SPMV-Scalar Kernel

```c
__global__ void
spmv_sos_scalar_kernel(const float* v, const int* cols, const int* rowDelimiters, const int* dim, float* out)
{
    int myRow = blockIdx.x * blockDim.x + threadIdx.x;
    texIndex texIndex = threadIdx.x;
    if (myRow < dim)
    {
        float t = 0.0f;
        int start = rowDelimiters[myRow];
        int end = rowDelimiters[myRow + 1];
        // Divergent Branch
        for (int j = start; j < end; j++)
        {
            // Misaligned loads
            int col = cols[j];
            t = v[base + texIndex * blockDim.y];
            out[myRow] = t;
        }
    }
}
```

Profile static load instructions
- Are they divergent?
- Loop repetition ID
  - Assume all loads with same base address and offset within cache line access are repeated each iteration

Prediction Mechanisms

- Profiled Divergence Aware Scheduling (DAWS)
  - Used offline profile results to dynamically determine de-scheduling decisions

- Detected Divergence Aware Scheduling (DAWS)
  - Behaviors derived at run-time to drive de-scheduling decisions
    - Loops that exhibit intra-warp locality
    - Static loads are characterized as divergent or convergent

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013 (55)
Extensions for DAWS

- I-Fetch
- Decode
- I-Buffer
- Issue
- PRF
- RF
- Backing
- D-Cache

All Hit? Data
Writeback

Operation: Tracking

- Warp Issue Arbiter (WIA)
- Divergence Aware Scheduler (Profiled- and Detected-DAWS)
- Cache Footprint Prediction Table
- Loop Start/End
- D-Cache
- Static Loop Classification Table
- Issue
- WID
- HasLocality

- Intra-Loop Repetition Detector
- Memory Divergence Detector
- PC_WID

- Dynamic Load Classifier (Detected-DAWS Only)
- Sampling Warden Table
- (WID/Tag/HasLocality) on load

- # Active Lanes
- Basis for throttling
- Profile-based information

Figure from T. Rogers, M. O'Conner, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013

#Active Lanes

Created/re moved at loop begin/end

One entry per warp issue slot
Operation: Prediction

- Sum results from static loads in this loop
  - Add #active-lanes of cache lines for divergent loads
  - Add 2 for converged loads
  - Count loads in the same equivalence class only once (unless divergent)

- Generally only considering de-scheduling warps in loops
  - Since most of the activity is here
  - Can be extended to non-loop regions by associating non-loop code with next loop

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013 (59)

Operation: Nested Loops

```
for (i=0; i<limitx; i++){
  ..
  ..
  for (j=0; j<limity; j++){
    ..
    ..
  }
  ..
  ..
}
```

- On-entry update prediction to that of inner loop
- On re-entry predict based inner loop predictions
- De-scheduling of warps determined by inner loop behaviors!
- On-exit, do not clear prediction

- Re-used predictions based on inner-most loops which is where most of the date re-use is found
- Note the attempt at tracking footprints to code segments

(60)
**Detected DAWS: Prediction**

- Detect both memory divergence and intra-loop repetition at run time
- Fill PC<sub>Load</sub> entries based on run time information
- Use profile information to start

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013

**Detected DAWS: Classification**

Create equivalence classes of loads (checking the PCs) Increment or decrement the counter depending on #memory accesses for a load

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013
**Performance**

![Graph showing performance comparison of different scheduling algorithms.]

Little to no degradation

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013

---

**Performance**

![Graph showing significant intra-warp locality.]

Significant intra-warp locality

SPMV-scalar normalized to best SPMV-vector

Figure from T. Rogers, M. O'Connor, T. Aamodt, "Divergence-Aware Warp Scheduling," MICRO 2013

---
Summary

• If we can characterize warp level memory reference locality, we can use this information to minimize interference in the cache through scheduling constraints

• Proactive scheme outperforms reactive management

• Understand interactions between memory divergence and control divergence

---

OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance
A. Jog et. al ASPLOS 2013
Goal

- Understand memory effects of scheduling from deeper within the memory hierarchy
- Minimize idle cycles induced by stalling warps waiting on memory references

Off-chip Bandwidth is Critical!

Percentage of total execution cycles wasted waiting for the data to come back from DRAM

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013
Source of Idle Cycles

- Warps stalled on waiting for memory reference
  - Cache miss
  - Service at the memory controller
  - Row buffer miss in DRAM
  - Latency in the network (not addressed in this paper)
- The last warp effect
- The last CTA effect
- Lack of multiprogrammed execution
  - One (small) kernel at a time

Impact of Idle Cycles

<table>
<thead>
<tr>
<th>#</th>
<th>App. Suite</th>
<th>Type-1 Applications</th>
<th>Abbr.</th>
<th>PMEM</th>
<th>CINV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parboil</td>
<td>Sum of Abs. Differences</td>
<td>SAD</td>
<td>H (6.39x)</td>
<td>91%</td>
</tr>
<tr>
<td>2</td>
<td>MapReduce</td>
<td>PageViewCount</td>
<td>PVC</td>
<td>H (4.99x)</td>
<td>93%</td>
</tr>
<tr>
<td>3</td>
<td>MapReduce</td>
<td>SimilarityScore</td>
<td>SSC</td>
<td>H (4.60x)</td>
<td>85%</td>
</tr>
<tr>
<td>4</td>
<td>CUDA SDK</td>
<td>Breadth First Search</td>
<td>BFS</td>
<td>H (2.77x)</td>
<td>81%</td>
</tr>
<tr>
<td>5</td>
<td>CUDA SDK</td>
<td>MUMerGPU</td>
<td>MUM</td>
<td>H (2.66x)</td>
<td>72%</td>
</tr>
<tr>
<td>6</td>
<td>Rodinia</td>
<td>CFD Solver</td>
<td>CFD</td>
<td>H (2.46x)</td>
<td>66%</td>
</tr>
<tr>
<td>7</td>
<td>Rodinia</td>
<td>Kmeans Clustering</td>
<td>KMN</td>
<td>H (2.43x)</td>
<td>65%</td>
</tr>
<tr>
<td>8</td>
<td>CUDA SDK</td>
<td>Scalar Product</td>
<td>SCP</td>
<td>H (2.37x)</td>
<td>58%</td>
</tr>
<tr>
<td>9</td>
<td>CUDA SDK</td>
<td>Fast Walsh Transform</td>
<td>FWT</td>
<td>H (2.29x)</td>
<td>58%</td>
</tr>
<tr>
<td>10</td>
<td>MapReduce</td>
<td>InvertedIndex</td>
<td>IIX</td>
<td>H (2.29x)</td>
<td>65%</td>
</tr>
<tr>
<td>11</td>
<td>Parboil</td>
<td>Sparse-Matrix-Mul.</td>
<td>SPMV</td>
<td>H (2.19x)</td>
<td>65%</td>
</tr>
<tr>
<td>12</td>
<td>3rd Party</td>
<td>JPEG Decoding</td>
<td>JPEG</td>
<td>H (2.12x)</td>
<td>54%</td>
</tr>
<tr>
<td>13</td>
<td>Rodinia</td>
<td>Breadth First Search</td>
<td>BFSR</td>
<td>H (2.09x)</td>
<td>64%</td>
</tr>
<tr>
<td>14</td>
<td>Rodinia</td>
<td>Streamcluster</td>
<td>SC</td>
<td>H (1.94x)</td>
<td>52%</td>
</tr>
<tr>
<td>15</td>
<td>Parboil</td>
<td>FFT Algorithm</td>
<td>FFT</td>
<td>H (1.56x)</td>
<td>37%</td>
</tr>
<tr>
<td>16</td>
<td>Rodinia</td>
<td>SRAD2</td>
<td>SD2</td>
<td>H (1.53x)</td>
<td>36%</td>
</tr>
<tr>
<td>17</td>
<td>CUDA SDK</td>
<td>Weather Prediction</td>
<td>WP</td>
<td>H (1.50x)</td>
<td>54%</td>
</tr>
</tbody>
</table>

Figure from A. Jog et al., "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013
High-Level View of a GPU

SIMT Cores

Threads
Warps

Cooperative Thread Arrays (CTAs)

Interconnect

L2 cache

DRAM

CTA-Assignment Policy (Example)

Multi-threaded CUDA Kernel

CTA-1  CTA-2  CTA-3  CTA-4

SIMT Core-1

CTA-1  CTA-3
Warp Scheduler
L1 Caches  ALUs

SIMT Core-2

CTA-2  CTA-4
Warp Scheduler
L1 Caches  ALUs

Courtesy A. Jog, “OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance,” ASPLOS 2013
Organizing CTAs Into Groups

- Set minimum number of warps equal to #pipeline stages
  - Same philosophy as the two-level warp scheduler
- Use same CTA grouping/numbering across SMs?

Warp Scheduling Policy

- All launched warps on a SIMT core have equal priority
  - Round-Robin execution

- **Problem**: Many warps stall at long latency operations roughly at the same time

Send Memory Requests

All warps have equal priority

All warps compute

SIMT Core Stalls

All warps have equal priority

All warps compute

Time

Figure from A. Jog et al., "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013

(73)
Two Level Round Robin Scheduler

Group 0
- CTA0
- CTA1
- CTA3
- CTA2

RR

Group 1
- CTA4
- CTA5
- CTA7
- CTA8

RR

Group 2

Group 3
- CTA12
- CTA13
- CTA15
- CTA14

Agnostic to when pending misses are satisfied

All warps compute
All warps have equal priority

• Form Warp-Groups
  (Narasiman MICRO’11)
• CTA-Aware grouping
• Group Switch is Round-Robin

Send Memory Requests

SIMT Core Stalls

Saved Cycles

Time

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013
Key Idea

Executing Warps (EW)  Stalled Warps (SW)  Cache Resident Warps (CW)

- Relationship between EW, SW and CW?
- When EW ≠ CW, we get interference
- Principle – optimize reuse: Seek to make EW = CW

Objective 1: Improve Cache Hit Rates

Data for CTA1 arrives.
No switching.

Data for CTA1 arrives.
Switch to CTA1.

Fewer CTAs accessing the cache concurrently → Less cache contention

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013
Reduction in L1 Miss Rates

- Limited benefits for cache insensitive applications
- What is happening deeper in the memory system?

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013

The Off-Chip Memory Path

Access patterns?

Ordering and buffering?
Inter-CTA Locality

How do CTAs Interact at the MC and in DRAM?

Impact of the Memory Controller

- Memory scheduling policies
  - Optimize BW vs. memory latency
- Impact of row buffer access locality
- Cache lines?
Row Buffer Locality

Multiple Banks

Row Buffers

4 Cache Blocks

CTA Data Layout (A Simple Example)

Data Matrix

A(0,0) | A(0,1) | A(0,2) | A(0,3)
A(1,0) | A(1,1) | A(1,2) | A(1,3)
A(2,0) | A(2,1) | A(2,2) | A(2,3)
A(3,0) | A(3,1) | A(3,2) | A(3,3)

mapped to Bank 1
mapped to Bank 2
mapped to Bank 3
mapped to Bank 4

Average percentage of consecutive CTAs (out of total CTAs) accessing the same row = 64%

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013

(83)
Implications of high CTA-row sharing

![Diagram showing CTA prioritization order and implications of high CTA-row sharing.](Diagram)

Lower Row Locality
Higher Bank Level Parallelism

High Row Locality
Low Bank Level Parallelism

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013
Some Additional Details

- Spread reference from multiple CTAs (on multiple SMs) across row buffers in the distinct banks
- Do not use same CTA group prioritization across SMs
  - Play the odds
- What happens with applications with unstructured, irregular memory access patterns?

Objective 2: Improving Bank Level Parallelism

- 11% increase in bank-level parallelism
- 14% decrease in row buffer locality

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013
Objective 3: Recovering Row Locality

Memory Side Prefetching

- Prefetch the so-far-unfetched cache lines in an already open row into the L2 cache, just before it is closed

- What to prefetch?
  - Sequentially prefetches the cache lines that were not accessed by demand requests
  - Sophisticated schemes are left as future work

- When to prefetch?
  - Opportunistic in Nature
  - **Option 1:** Prefetching stops as soon as demand request comes for another row. (Demands are always critical)
  - **Option 2:** Give more time for prefetching, make demands wait if there are not many. (Demands are **NOT** always critical)

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013
IPC results (Normalized to Round-Robin)

- 11% within Perfect L2

Summary

- Coordinated scheduling across SMs, CTAs, and warps
- Consideration of effects deeper in the memory system
- Coordinating warp residence in the core with the presence of corresponding lines in the cache

Courtesy A. Jog, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," ASPLOS 2013