Divergence-Aware Warp Scheduling

Timothy G. Rogers\textsuperscript{1}, Mike O’Connor\textsuperscript{2}, Tor M. Aamodt\textsuperscript{1}

\textsuperscript{1}The University of British Columbia
\textsuperscript{2}NVIDIA Research

MICRO 2013 Davis, CA
GPU

- 10000’s concurrent threads
- Grouped into warps
- Scheduler picks warp to issue each cycle
2 Types of Divergence

Branch Divergence

if(...) {
  ...
} 

Memory Divergence

Can waste memory bandwidth

Effects functional unit utilization

Aware of branch divergence

Aware of memory divergence

Focus on improving performance
Motivation

• **Improve performance of programs with memory divergence**
  • Parallel irregular applications
  • Economically important (server computing, big data)

• **Transfer locality management from SW to HW**
  • Software solutions:
    • Complicate programming
    • Not always performance portable
    • Not guaranteed to improve performance
    • Sometimes impossible
Programmability Case Study
Sparse Vector-Matrix Multiply
2 versions from SHOC

Divergent Version

Divergence

Each thread has locality

GPU-Optimized Version

Explicit Scratchpad Use

Dependent on Warp Size

Added Complication

Parallel Reduction

Example 1 Highly Divergent SPMV-Scalar Kernel

```c
__global__ void
spmv_car_scalar_kernel(const float* val,
const int* cols,
const int* rowDelimiters,
const int dim,
float* out)
{
    int t = threadIdx.x;
    int id = t % (warpSize-1);
    int warpsPerBlock = blockDim.x / warpSize;

    __shared__ volatile
    float partialSums[ BLOCK_SIZE ];

    // Divergent Branch
    for (int j = start; j < end; j++)
    {
        // Uncoalesced Load
        int col = cols[j];
        t = val[j] * texReader(col);
        out[myRow] = t;
    }
}
```
Previous Work

- Scheduling used to capture intra-thread locality (MICRO 2012)

Reactive
- Detects interference then throttles

Unaware of branch divergence
- All warps treated equally

Outperformed by profiled static throttling

Case Study: Divergent code 50% slowdown

Divergence-Aware Warp Scheduling

Predict and be Proactive

Adapt to branch divergence

Outperform static solution

Case Study: Divergent code <4% slowdown
Divergence-Aware Warp Scheduling

How to be proactive

- Identify where locality exists
- Limit the number of warps executing in high locality regions

Adapt to branch divergence

- Create cache footprint prediction in high locality regions
- Account for number of active lanes to create *per-warp footprint prediction*.
- Change the prediction as branch divergence occurs.
Where is the locality?

- Examine every load instruction in program

Static Load Instructions in GC workload

Locality Concentrated in Loops
Locality In Loops Limit Study

How much data should we keep around?

- Hits on data accessed in immediately previous trip
- Line accessed last iteration
- Line accessed this iteration
- Other

Average

Fraction cache hits in loops

0 0.2 0.4 0.6 0.8 1

How much data should we keep around?
DAWS Objectives

1. Predict the amount of data accessed by each warp in a loop iteration.

2. Schedule warps in loops so that aggregate predicted footprint does not exceed L1D.
Observations that enable prediction

- Memory divergence in static instructions is predictable.
- Data touched by divergent loads dependent on active mask.

Both Used To Create Cache Footprint Prediction
Online characterization to create cache footprint prediction

1. Detect loops with locality
   - Some loops have locality
   - Some don’t

2. Classify loads in the loop
   - Limit multithreading here
   - Loop with locality
     ```
     while(...) {
       load 1  Diverged
       ...
       load 2  Not Diverged
     }
     ```

3. Compute footprint from active mask
   - Warp 0’s Footprint = 5 cache lines
     - Warp 0
     ```
     while(...) {
       load 1  Diverged
       ...
       load 2  Not Diverged
     }
     ```
     - 4 accesses + 1 access = 5 cache lines
```c
int C[]={0,64,96,128,160,160,192,224,256};
void sum_row_csr(float* A, ...) {
  float sum = 0;
  int i = C[tid];
  while(i < C[tid+1]) {
    sum += A[i];
    ++i;
  }
}
```

**Example Compressed Sparse Row Kernel**

**DAWS Operation Example**

**Cache Footprint**
- Warp 0: No Footprint
- Warp 1: Footprint = 4X1

**Memory Divergence**
- Warp 0 has branch divergence in the loop for later warps
- Both warps capture spatial locality together
- Footprint decreased
Methodology

GPGPU-Sim (version 3.1.0)
- 30 Streaming Multiprocessors
  - 32 warp contexts (1024 threads total)
- 32k L1D per streaming multiprocessor
- 1M L2 unified cache

Compared Schedulers
- Cache-Conscious Wavefront Scheduling (CCWS)
- Profile based Best-SWL
- Divergence-Aware Warp Scheduling (DAWS)

More schedulers in paper
Sparse MM Case Study Results

- Performance (normalized to optimized version)

Within 4% of optimized with no programmer input
Sparse MM Case Study Results

- Properties (normalized to optimized version)

- <20% increase in off-chip accesses

- Divergent code now has potential energy advantages

- Divergent code issues 2.8x less instructions
Cache-Sensitive Applications

- Breadth First Search (BFS)
- Memcached-GPU (MEMC)
- Sparse Matrix-Vector Multiply (SPMV-Scalar)
- Garbage Collector (GC)
- K-Means Clustering (KMN)

Cache-Insensitive Applications in paper
Results

Outperform Best-SWL in highly branch divergent

Overall 26% improvement over CCWS
Summary

Problem
- Divergent loads in GPU programs.
  - Software solutions complicate programming

Solution
- DAWS
  - Captures opportunities by accounting for divergence

Result
- Overall 26% performance improvement over CCWS
- Case Study: Divergent code performs within 4% code optimized to minimize divergence