Intra-Warp Compaction Techniques

Goal

• Compact threads in a warp to coalesce (and eliminate) idle cycles → improve utilization
References

- V. Narasiman, et. al., “Improving GPU Performance via Large Warps and Two-Level Scheduling,” MICRO 2011

Improving GPU Performance via Large Warps and Two-Level Scheduling
V. Narsiman et. al
MICRO 2011
Goals

- Improve performance of divergent code via compaction of threads within a warp
- Integrate warp scheduling optimization with intra-warp compaction

Resource Underutilization

32 warps, 32 threads per warp, single SM

Due to control divergence  Due to memory divergence
Warp Scheduling and Locality

- Opportunities for memory coalescing
- Potential for exposing memory stalls
- Degrades memory reference locality and row buffer locality
- Overlaps memory accesses
- Opportunities for memory coalescing
- Potential for exposing memory stalls
- Degrades memory reference locality and row buffer locality
- Overlaps memory accesses

Key Ideas

- Conventional design today → warp size = #SIMD lanes
- Use large warps → multi-cycle issue of sub-warps
  - Compact threads in a warp to form fully utilized sub-warps
  - 2-level scheduler to spread memory accesses in time
  - Reduce memory related stall cycles
Large Warps

Typical Operation
- Warp size = 4
- Warp size = SIMD width
- SIMD Width = 4

Proposed Approach
- Warp size = 16
- SIMD Width = 4
- sub-warp

• Large warps converted to a sequence of sub-warps

Sub-Warp Compaction

• Iteratively select one thread per column to create a packed sub-warp
• Dynamic generation of sub-warps
Impact on the Register File

Baseline Register File

Large Warp Active Mask Organization

Large Warp Register File

Need separate decoders per bank

Scheduling Constraints

- Next large warp cannot be scheduled until first sub-warp completes execution
- Scoreboard checks for issue dependencies
  - Thread not available for packing into a sub-warp unless previous issue (sub-warp) has completed → single bit status
  - Simple check
  - However on a branch, all sub-warps must complete before it is eligible for instruction fetch scheduling
- Re-fetch policy for conditional branches
  - Must wait till last sub-warp finishes
- Optimization for unconditional branch instructions
  - Don’t create multiple sub-warps
  - Sub-warping always completes in a single cycle
Effect of Control Divergence

- Note that divergence is unknown until all sub-warps execute
  - Divergence management just happens on large warp boundaries
  - Need to buffer sub-warp state, e.g., active masks

- The last warp effect
  - Cannot fetch the next instruction in a warp until all sub-warps issue
  - Trailing warp (warp divergence effect) can lead to many idle cycles

- Effect of the last thread
  - E.g., in data dependent loop iteration count across threads
  - Last thread can hold up reconvergence

A Round Robin Warp Scheduler

- Exploit inter-warp reference locality in the cache
- Exploit inter-warp reference locality in the DRAM row buffers
- However, need to maintain latency hiding
Two Level Round Robin Scheduler

Fetch Group 0

LW0
LW2
LW8
LW10
LW12

RR

Fetch Group 1

LW1
LW3
LW9
LW11
LW13

LW4
LW6
LW14

LW5
LW7
LW15

Fetch Group 2

Fetch Group 3

Scheduler Behavior

- Need to set fetch group size carefully → tune to fill the pipeline
- Timeout on switching fetch groups to mitigate the last warp effect
Summary

- Intra-warp compaction made feasible due to multi-cycle warp execution
  - Mismatch between warp size and SIMD width enables flexible intra-warp compaction
- Do not make warps too big → last thread effect begins to dominate

SIMD Divergence Optimization Through Intra-warp Compaction
A. S. Vaidya, et. al
ISCA 2013
Goals

- Improve utilization in divergent code via *intra-warp* compaction
- Become familiar with the architecture of Intel’s Gen integrated general purpose GPU architecture

Integrated GPUs: Intel HD Graphics

Integrated GPUs: Intel HD Graphics

- Shared physical memory

Gen graphics processor:
- 32-byte bidirectional ring
- Dedicated coherence signals
- Coherent distributed cache
- Shared with GPU
- Operates as a memory side cache

Inside the Gen9 EU

- Up to 7 threads
- 128, 256-bit registers/thread (8-way SIMD)
- Each thread executes a kernel
  - Threads may execute different kernels
  - Multi-instruction dispatch

Figure from The Computer Architecture of the Intel Processor Graphics Gen9, https://software.intel.com/en-us/articles/intel-graphics-developers-guides (21)
Operation (2)

- Dispatch 4 instructions from 4 threads
- Constraints between issue slots
- Support both FP and Integer operations
- 8, 16-bit integer operations
- 8, 16-bit FP operations
- MAD operations each cycle
- Support both FP and Integer operations
- 8, 16-bit integer operations
- 8, 16-bit FP operations
- MAD operations each cycle
- 96 bytes/cycle read BW
- 32 bytes/cycle write BW
- Divergence/reconvergence management

Figure from The Computer Architecture of the Intel Processor Graphics Gen9,

(23)
Mapping the BSP Model

- Map multiple threads to SIMD instance executed by a EU Thread
- All threads in a TB or workgroup mapped to same thread (shared memory access)

Subslice Organization

- #Eus * #threads/EU determines width of the slice
- Flexible data interface
  - Scatter/gather support
  - Memory request coalescing across 64-byte cache lines
  - Shared memory access
Slice Organization

Flexible partitioning
• SM
• Data cache
• Buffers for accelerators

Shared memory
• 64Kbyte/slice
• Not coherent with other structures

Product Organization

• Load balancing
• Honor barrier and shared memory constraints

• Shared virtual memory
  • Share pointer rich data structures between CPU and GPU
• Coherent shared memory between CPU and GPU

• Implemented shared atomics (with CPU)
Coherent Memory Hierarchy


Microarchitecture Operation

- Per-thread operation
  
  Per-thread scoreboard check

  Thread arbitration and dual issue/2-cycles

  Operand fetch/swizzle
  - Encode swizzle in RF access

  Instruction execution happens in waves of 4-wide operations
  - Note: variable width SIMD instructions
Divergence Assessment

SIMD Efficiency

Coherent applications

Divergent Applications

Basic Cycle Compression

RF for a Single Operand

Example:

- Actual operation depends on data types, execution cycles/op
- Note power/energy savings

(31)
Swizzle Cycle Compression

Can compact across Quads

RF for a Single Operand

- Note increased area, power/energy

128b

cycle i
cycle i+1
cycle i+2
cycle i+3

4 lanes
Compaction Opportunities

For $K$ active threads what is the maximum cycle savings for SIMD $N$ instructions?

Performance Savings

- Difference between saving cycles and saving time
  - When is $\#cycles \neq \text{time}$?
Summary

• Multi-cycle warp/SIMD/work_group execution

• Optimize #cycles/warp by compressing idle cycles
  ❖ Rearrange idle cycles via swizzling to create opportunity

• Sensitivities to the memory interface speeds
  ❖ Memory bound applications may experience limited benefit

Intra-Warp Compaction

• Scope limited to within a warp

• Increasing scope means increasing warp size, explicitly, or implicitly (treating multiple warps as a single warp)