Reducing Branch Divergence in GPU Programs
T. D. Han and T. Abdelrahman
GPGPU 2011

Reading

• T. D. Han and T. Abdelrahman, “Reducing Branch Divergence in GPGPU Programs,” GPGPU 2011
Goal

- Improve the utilization of the SIMD core
- Temporal compaction of control flow paths
  - Across successive loop iterations
  - Re-align iteration across threads in a warp so that we increase the number of threads taking the same control flow path

Basic Idea

- Delay iterations in a thread so that majority of threads in a warp will take the same control flow path
- Improve utilization and reduce dynamic instruction count
- Local decision on when to delay an iteration?

Figure from T. D. Han and T. Abdelrahman, "Reducing Branch Divergence in GPGPU Programs," GPGPU 2011
Basic Idea (2): 2 Warps

Basic Idea (3)

Figure from T. D. Han and T. Abdelrahman, "Reducing Branch Divergence in GPGPU Programs," GPGPU 2011
Operation

Thread specific local variable

Collect branch conditions in a 32-bit register

Count number of 1's in a 32-bit pattern

Compute this iteration?

Iteration count adjusted on a per thread basis

Figure from T. D. Han and T. Abdelrahman, “Reducing Branch Divergence in GPGPU Programs,” GPGPU 2011

Example

Loops must be barrier free

Figure from T. D. Han and T. Abdelrahman, “Reducing Branch Divergence in GPGPU Programs,” GPGPU 2011
Optimization

- Starvation effects of threads that take the less traveled path
  - Periodically reverse the decision → complement the branch strategy

```c
// branch direction period: 0,0,0,1
// 1 means the IF branch path
// 0 means the ELSE branch path
int num_zeros = 3, num_ones = 1;
int period = num_zeros + num_ones;
int counter = -1;
...
// update the direction
if (++counter == period) counter = 0;
cond_for_all = (counter >= num_zeros);
// remove idle iteration
cond_for_all = cond_for_all ? __any(cond) : __all(cond);
```

Figure from T. D. Han and T. Abdelrahman, “Reducing Branch Divergence in GPGPU Programs,” GPGPU 2011

Properties

- Applicable to other forms of control flow in a loop, e.g., `switch`

- Performance is dependent on
  - Code size in branch paths relative to fixed instruction overhead of the delayed branching
  - Code size outside of the branch paths
  - Branching pattern in thread → high degree of branching can be effectively exploited

- Impact on memory coalescing
  - No worse than without delaying iterations?
Branch Distribution

- Code transformation to reduce impact of branch divergence
- Aggressively hoist common code structure, rather than common subexpression elimination
- Produce smaller divergent code blocks

Example

Greater register pressure

Smaller divergent code body

Figure from T. D. Han and T. Abdelrahman, "Reducing Branch Divergence in GPGPU Programs," GPGPU 2011
Summary

- Up to 20% improvement in performance indicated when using both techniques
- Does not require any special purpose hardware support
- Compatible with other compiler optimizations

Dynamic Warp Subdivision
Meng, Tarjan, and Skadron
ISCA 2010
Goals

- Understand the interactions between divergence behaviors and memory and control flow behaviors
- Integrated handling of branch and memory divergence behaviors
- Application of warp splitting and re-combining (re-convergence) to counteract disadvantages of lock-step, synchronous execution of thread bundles

Reading

Memory Divergence Behavior

- Wider warps \( \rightarrow \) more likely to exhibit memory divergence
  - Less total number of warps but greater need for more warps for latency hiding
  - Single thread miss can hold up an entire warp
- Effective throughput depends on footprint in the L1
  - Think of the number of warps suspended on cache misses
  - Cache contention

Figure from J. Meng, D. Tarjan, and K. Skadron, “Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance,” ISCA 2010

Memory Divergence and Control Flow

- PDOM reconvergence permits only one branch path execution at a time \( \rightarrow \) limiting latency hiding?
- What happens on cache misses for some threads?
  - Misses on code block C but not D?

Figure from J. Meng, D. Tarjan, and K. Skadron, “Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance,” ISCA 2010
Memory Divergence

- How do we hide latency between threads in a warp?
- The "last thread" effect due to implicit lock step progress
- Dynamic Warp Subdivision (DWS) as a solution

Dynamic Warp Subdivision

- Dynamically split warps into smaller, independently schedulable units
  - Overlap miss behavior as much as possible without adding new threads
  - Allow the active threads to run-ahead
- When to split and reconverge?
- How do we orchestrate (schedule) better memory behaviors?
MLP for Divergent Branches

- Overlap memory access in different branch paths

Run-Ahead Execution

- Early issue of memory instructions
Intra-Warp Latency Hiding

- Threads in the same warp can progress overlapping memory access latency

Figure from J. Meng, D. Tarjan, and K. Skadron, “Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance,” ISCA 2010

Prefetching Behavior of Warp Splits

Figure from J. Meng, D. Tarjan, and K. Skadron, “Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance,” ISCA 2010
Reconvergence Behaviors

• Over-subdivision → when?
  ➢ Over-splitting degrades benefits of warps
  ➢ Judicious splits → e.g., when throughput has dropped due to a large number of suspended warps

• Reconvergence → ? when
  ➢ Still can take place at the TOS of the reconvergence stack
  ➢ Keep track of warp split and use PC based reconvergence
    o Check every cycle?

---

Reconvergence

Re-convergence using PCs

Figure from J. Meng, D. Tarjan, and K. Skadron, “Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance,” ISCA 2010

(25)
**PC vs. Stack Based Reconvergence**

- PC-based reconvergence can reconverge across loop iterations
- Stack based causes reconvergence at the PDOM

Figure from J. Meng, D. Tarjan, and K. Skadron, “Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance,” ISCA 2010

---

**Summary**

- Exploit memory level parallelism within a warp
  - Optimization of memory behavior drives the management of threads/warps rather than computation
- Balancing warp size, warp count, and memory latency
- Integrated solution leads upto 1.7X speedup
- Low area overhead
The Dual Path Execution Model for Efficient GPU Control Flow
M. Rhu and M. Erez
HPCA 2013

Goal

- Reconvergence-based techniques permit only a single divergent path to be active at any point in time
  - Elegant, but serializes execution
- Support parallel execution using stack-based reconvergence
  - Combine principals of dynamic subdivision and stack-based reconvergence
Baseline Execution Profile

- Only one active path at a time for a warp
- Serialization of path execution

Figure from M. Rhu and M. Erez, "The Dual Path Execution Model for Efficient GPU Control Flow," HPCA 2013

PDOM Execution

(a) Initial status of the stack. The current TOS designates the fact that basic block A is being executed.
(b) Two entries of block B and C are pushed into the stack when the warp executes $S_{B, A}$.
(c) The stack entry, corresponding to block $b$ at TOS, is popped out when again when the stack entry for block $f$ is popped out.
(d) Two more entries for block $D$ and $E$ are pushed into the stack when the warp executes $S_{D, E}$.
(e) Threads are recombined back at block $F$ when both entries for block $D$ and $E$ are popped out.
(f) All four threads become active to block $G$ at TOS, is popped out when PC matches $R_{G}$ value of $G$.

Figure from M. Rhu and M. Erez, "The Dual Path Execution Model for Efficient GPU Control Flow," HPCA 2013
Dual Path Execution

Store state of both paths

Concurrency Behavior

• Note only two active paths at a time in a single stack (nested control flow segment of code)

• Top level divergent set of threads are not interleaved

Figure from M. Rhu and M. Erez, "The Dual Path Execution Model for Efficient GPU Control Flow," HPCA 2013 (33)
Handling Dependencies in a Warp

- What about data dependencies?
  - Dual instruction issue from a warp

- Need a per warp scoreboard

---

Dual Path Scoreboard

- Replicate Scoreboard for each path

- How do we distinguish cross-path vs. in-path dependencies?

---

Figure from M. Rhu and M. Erez, "The Dual Path Execution Model for Efficient GPU Control Flow," HPCA 2013
Dependency Management

Case I
- Incorrect execution
- Deadlock

Case II
- Pending writes before reconvergence

Case III
- False dependencies

Case IV
- Distinct registers

Figure from M. Rhu and M. Erez, “The Dual Path Execution Model for Efficient GPU Control Flow,” HPCA 2013

Dependency Management (2)

- On divergence
  - Shadow bits set
  - Actually write (B) unrelated to reads (D&E)

- These false dependencies avoided

Figure from M. Rhu and M. Erez, “The Dual Path Execution Model for Efficient GPU Control Flow,” HPCA 2013
Updating the Scoreboard

- Indicates cross-path dependency due to pending write before divergence
- Copy P→S at divergence

- When checking scoreboard during issue, check shadow bits in the other scoreboard
  - E.g., make sure Path A write has completed for Path C
- Writes need single bit to indicate which scoreboard to clear

Figure from M. Rhu and M. Erez, “The Dual Path Execution Model for Efficient GPU Control Flow,” HPCA 2013

Warp Scheduler Impact

- Typically multiple warp schedulers per SM
- Dual path execution can double the number of schedulable entries
- Expect it can be more sensitive to the warp scheduling policy
Opportunity

- Non-interleavable vs. interleavable branches
- How do we assess opportunity?

\[ \text{Avg}_{\text{path}} = \frac{1}{N} \sum_{i=1}^{N} \text{NumPath}_i \]

Number of paths at instruction \( i \)

Number dynamic instructions in a warp

Figure from M. Rhu and M. Erez, “The Dual Path Execution Model for Efficient GPU Control Flow,” HPCA 2013

Application Profiles

Figure from M. Rhu and M. Erez, “The Dual Path Execution Model for Efficient GPU Control Flow,” HPCA 2013
Some Points to Note

- Achieve reduction in idle cycles → expected
- Impact on L1 cache misses → can increase
- Use for handling memory divergence
- Path forwarding to enable concurrency across stack levels (not just at the TOS) → not clear this is worth the cost

Summary

- Looking for structured concurrency → living with control divergence handling solutions
- Low cost overhead for applications that have high percentage of interleavable divergent branches
- Extends the performance of PDOM