Introduction to Control Divergence

Objective

- Understand the occurrence of control divergence and the concept of thread reconvergence
  - Also described as branch divergence and thread divergence

- Cover a basic thread reconvergence mechanism – PDOM
  - Set up discussion of further optimizations and advanced techniques

- Dynamic Warp formation
  - Increase lane utilization by merging warps
Reading


Handling Branches

- CUDA Code:

  ```
  if(…) … (True for some threads)  
  else … (True for others)  
  ```

- What if threads takes different branches?

- Branch Divergence!
Branch Divergence

- Occurs within a warp
- Branches lead serialization of branch dependent code
  - Performance issue: low warp utilization

```
if(...) {
...
}
else {
...
}
```

Idle threads

Reconvergence!

- Different threads follow different control flow paths through the kernel code
- Thread execution is (partially) serialized
  - Subset of threads that follow the same path execute in parallel

Example:

Courtesy of Wilson Fung, Ivan Sham, George Yuan, Tor Aamodt
Basic Idea

• **Split**: partition a warp
  - Two mutually exclusive thread subsets, each branching to a different target
  - Identify subsets with two activity masks \( \rightarrow \text{effectively} \) two warps

• **Join**: merge two subsets of a previously split warp
  - Reconverge the mutually exclusive sets of threads

• **Orchestrate** the correct execution for nested branches

• **Note** the long history of techniques in SIMD processors (see background in Fung et. al.)

Thread Warp

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**activity mask**

Thread Reconvergence

• **Fundamental problem**: 
  - Merge threads with the same PC
  - How do we sequence execution of threads? Since this can effect the ability to reconverge

• **Question**: When can threads productively reconverge?

• **Question**: When is the best time to reconverge?

*Must be at same PC*
**Dominator**

- Node \( d \) dominates node \( n \) if every path from the entry node to \( n \) must go through \( d \)

**Immediate Dominator**

- Node \( d \) immediate dominates node \( n \) if every path from the entry node to \( n \) must go through \( d \) and no other nodes dominate \( n \) between \( d \) and \( n \)
Post Dominator

- Node $d$ post dominates node $n$ if every path from the node $n$ to the exit node must go through $d$.

Immediate Post Dominator

- Node $d$ immediate post dominates node $n$ if every path from node $n$ to the exit node must go through $d$ and no other nodes post dominate $n$ between $d$ and $n$. 
Baseline: PDOM

The Stack-Based Structure

- A stack entry is a specification of a group of active threads that will execute that basic block
- The natural nested structure of control exposes the use of stack-based serialization

Example:
More Complex Example

- Stack based implementation for nested control flow
  - Stack entry RPC set to IPDOM
- Re-convergence at the immediate post-dominator of the branch


Implementation

- GPGPUSim model:
  - Implement per warp stack at issue stage
  - Acquire the active mask and PC from the TOS
  - Scoreboard check prior to issue
  - Register writeback updates scoreboard and ready bit in instruction buffer
  - When RPC = Next PC, pop the stack
  - Implications for instruction fetch?

Implementation (2)

• warpPC (next instruction) compared to reconvergence PC

• On a branch
  ❖ Can store the reconvergence PC as part of the branch instruction
  ❖ Branch unit has NextPC, TargetPC and reconvergence PC to update the stack

• On reaching a reconvergence point
  ❖ Pop the stack
  ❖ Continue fetching from the NextPC of the next entry on the stack

Can We Do Better?

• Warps are formed statically

• Key idea of dynamic warp formation
  ❖ Show a pool of warps and how they can be merged

• At a high level what are the requirements?

(17)
Compaction Techniques

• Can we reform warps so as to increase utilization?

• Basic idea: Compaction
  - Reform warps with threads that follow the same control flow path
  - Increase utilization of warps

• Two basic types of compaction techniques

• Inter-warp compaction
  - Group threads from different warps
  - Group threads within a warp
    - Changing the effective warp size

Inter-Warp Thread Compaction:
Dynamic Warp Formation
Goal

Warp 0

Warp 1

Warp 2

if(...) 

{...  }

else {

...

}

merge threads?

Warp 3

Warp 4

Warp 5

Reading

• W. Fung, I. Sham, G. Yuan, and T. Aamodt, "Dynamic Warp
  Formation: Efficient MIMD Control Flow on SIMD Graphics
  Hardware," ACM TACO, June 2009, Section 4.
DWF: Example

How Does This Work?

- Criteria for merging
  - Same PC
  - Complements of active threads in each warp
  - Recall: many warps/TB all executing the same code

- What information do we need to merge two warps
  - Need thread IDs and PCs

- Ideally how would you find/merge warps?
Thread Scheduler

- Warps formed dynamically in the warp pool
- After commit check PC-Warp LUT and merge or allocated newly forming warp

Lane aware

No Lane Conflict

DWF: Microarchitecture Implementation

Courtesy of Wilson Fung, Ivan Sham, George Yuan, Tor Aamodt
Resource Usage

- Ideally would like a small number of unique PCs in progress at a time → minimize overhead
- Warp divergence will increase the number of unique PCs
  - Mitigate via warp scheduling
- Scheduling policies
  - FIFO
  - Program counter – address variation measure of divergence
  - Majority/Minority - most common vs. helping stragglers
  - Post dominator (catch up)

Hardware Consequences

- Expose the implications that warps have in the base design
  - Implications for register file access → lane aware DWF
- Register bank conflicts

From Fung, et. al., "Dynamic Warp Formation: Efficient MIMD Control Flow in SIMD Graphics Hardware, ACM TACO, June 2009"
Relaxing Implications of Warps

- Thread swizzling
  - Essentially remap work to threads so as to create more opportunities for DWF → requires deep understanding of algorithm behavior and data sets

- Lane swizzling in hardware
  - Provide limited connectivity between register banks and lanes → avoiding full crossbars

![Diagram of lane swizzling](image)

Summary

- Control flow divergence is a fundamental performance limiter for SIMT execution

- Dynamic warp formation is one way to mitigate these effects
  - We will look at several others

- Must balance a complex set of effects
  - Memory behaviors
  - Synchronization behaviors
  - Scheduler behaviors