Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance

Jiayuan Meng, David Tarjan, Kevin Skadron
University of Virginia
37th International Symposium on Computer Architecture (ISCA 2010)
SIMD Warp Execution

• SIMD execution unit – warp
• If warp stalled? – Warps interleaving
• If no warps are ready to execute? – Idle cycles; Throughput reduced
• Warps interleaving disadvantages
  – Cache contention
  – Increases cost of register file
• Proposed solution – Dynamic warp subdivision
Why DWS?

- Intra-warp latency hiding approach
- "Warp-splits" – independent Scheduling entities
- Exploited in two cases
  - Branch divergence
  - Memory latency divergence
- No overhead on registers and cache
- Improved memory level parallelism and latency hiding

Memory divergence – Warp-splits
DWS Upon Branch divergence

(a) The example program

(b) Branch divergence and re-convergence

(c) Initial State

(d) After divergence and branch to B

(e) After completion of one branch

(f) After reconvergence

Conventional mechanism – Branch divergence and re-convergence
DWS Upon Branch divergence

Conventional - Only one active branch path at a time
DWS Upon Branch divergence

• Delayed re-convergence – Advantages
  – Memory request issued earlier; Prefetching for others
Warp-split subdivision

• Aggressive subdivision – narrow warp-split
• Which branches allowed to subdivide?
• Heuristic approach – subdivide upon branches whose post-dominator is followed by basic block of considerable length
• Advantages of Heuristic approach
  – Run-ahead threads not too far ahead
  – Early memory request, prefetching with delayed re-convergence
Stack-based and PC-based Reconvergence

Results

- Stack-based re-convergence
- PC-based re-convergence

Speedup vs. Application:
- h-mean
- FFT
- Filter
- HotSpot
- LU
- Merge
- Short
- KMeans
- SVM
DWS Upon Memory Divergence

Initiating misses earlier

Initiating misses earlier + Data prefetching
Preventing over-subdivision

- Aggressive split
- Lazy split
- Revive split
Re-converge or run-ahead?

- If not re-converged early – Same instruction sequence executed by warp-splits
- If re-converged early – Run-ahead warp-split stalls – Can’t issue outgoing memory request
- When to re-converge? – Need knowledge on future cache miss
- Results
  - Only based on memory divergence - poor performance
  - Branch limited re-convergence – a little performance gain

Results
Implementation – DWS Upon Memory Divergence

(a) Example program

(b) The re-convergence stack after completion of branch C

(c) The initial warp-split before subdivision

(d) After memory divergence, two warp-splits are created

(e) After the two warp-splits are re-united

(f) After the warp-split hits the post-dominator on the TOS of the re-convergence stack.
Results

• Compared with adaptive slip
• Influencing Factors – Frequency of branch and memory divergence, length of memory latencies, ability of WPU to hide latency with existing warps.
Conclusion

• Drawback: Doubles complexity and hardware cost in scheduling (WST)

• Future work: We can speculate cache miss frequency and miss latency to decide when to subdivide warp