Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow

Wilson W. L. Fung,Ivan Sham,George Yuan
Tor M. Aamodt
Department of Electrical and Computer Engineering
University of British Columbia

Motivation

GPGPU - Using a GPU for general purpose computation by efficiently handling control flow

GPU Programming Model

GPUs typically use Single Instruction Multiple Data - (SIMD) pipelines to achieve high performance with minimal overhead.

Programmers express parallelism using threads.

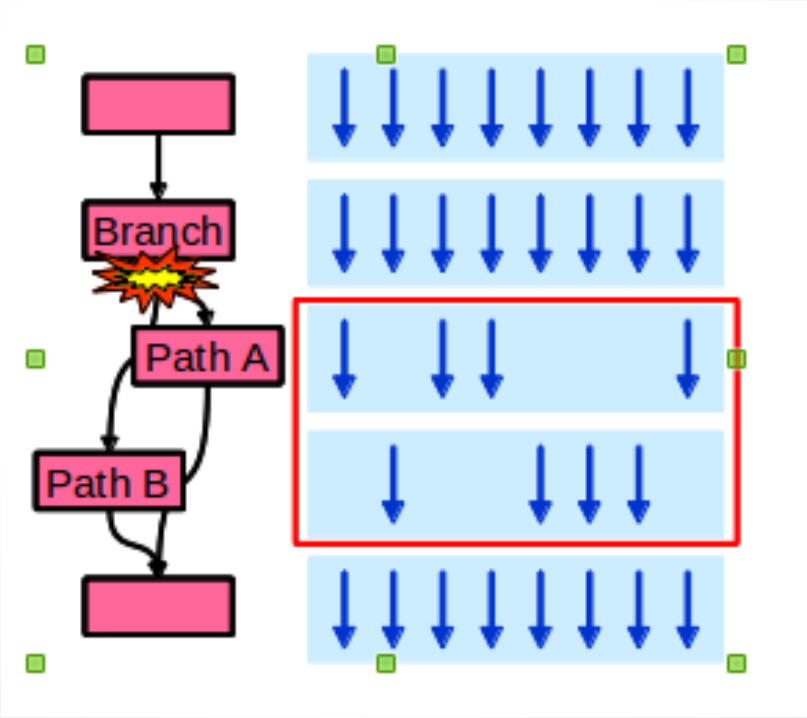
WARPS

Modern GPUs typically batch together groups of individual threads execute them together in lock step on a SIMD pipeline.

These thread batches are referred to as “Warps” by NVIDIA.

Challenges to GPGPU - Control Flow

Branch divergence occurs when threads inside warps branches to different execution paths.



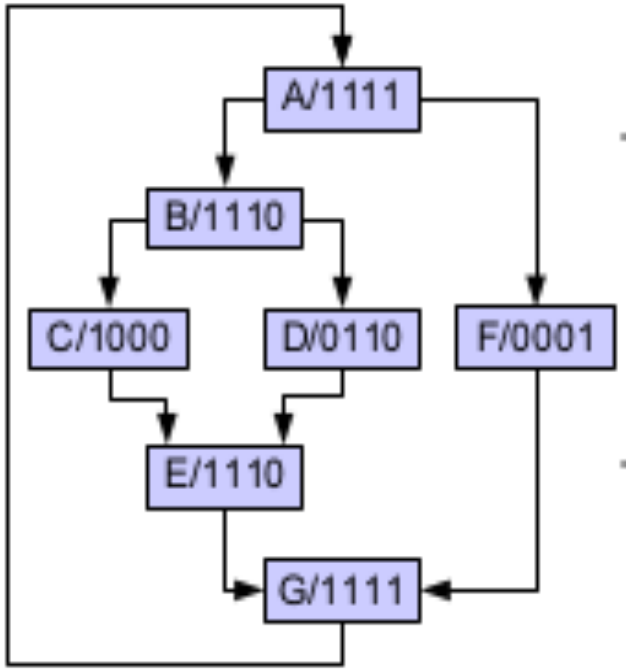
50.0% performance loss with SIMD width = 16

Existing solutions to Control Flow Divergence

1. Naive - serialize the threads within a warp as soon as the program counters diverge.
2. Predication - introduces overhead
3. Reconvergence of Control Flow
    3.1) Immediate Post-Dominator Reconvergence
    3.2) Dynamic Warp Formation and Scheduling

Immediate Post Dominator Reconvergence

A post-dominator is defined as follows: A basic block X post-dominates basic block Y (written as “X pdom Y”) if and only if all paths from Y to the exit node go through X, where a basic block is a piece of code with a single entry and exit point.



(a) Example Program

	Ret./Reconv. PC	Next PC	Active Mask
	-	G	1111
	G	F	0001
TOS →	G	B	1110

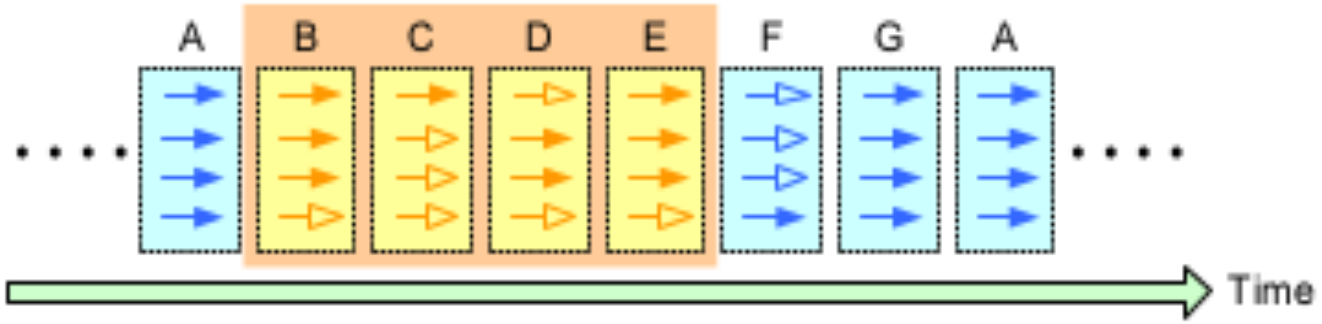
(c) Initial State

	Ret./Reconv. PC	Next PC	Active Mask	
	-	G	1111	
	G	F	0001	
	G	E	1110	(i)
	E	D	0110	(ii)
TOS →	E	C	1000	(iii)

(d) After Divergent Branch

	Ret./Reconv. PC	Next PC	Active Mask
	-	G	1111
	G	F	0001
TOS →	G	E	1110

(e) After Reconvergence



(b) Re-convergence at Immediate Post-Dominator of B

Execution resource utilization decreased from 82.7% for 4-wide, to 71.8% for 8-wide down to 49% for 16-wide. Benchmarks are SPLASH, SPECCPU-06 and CUDA

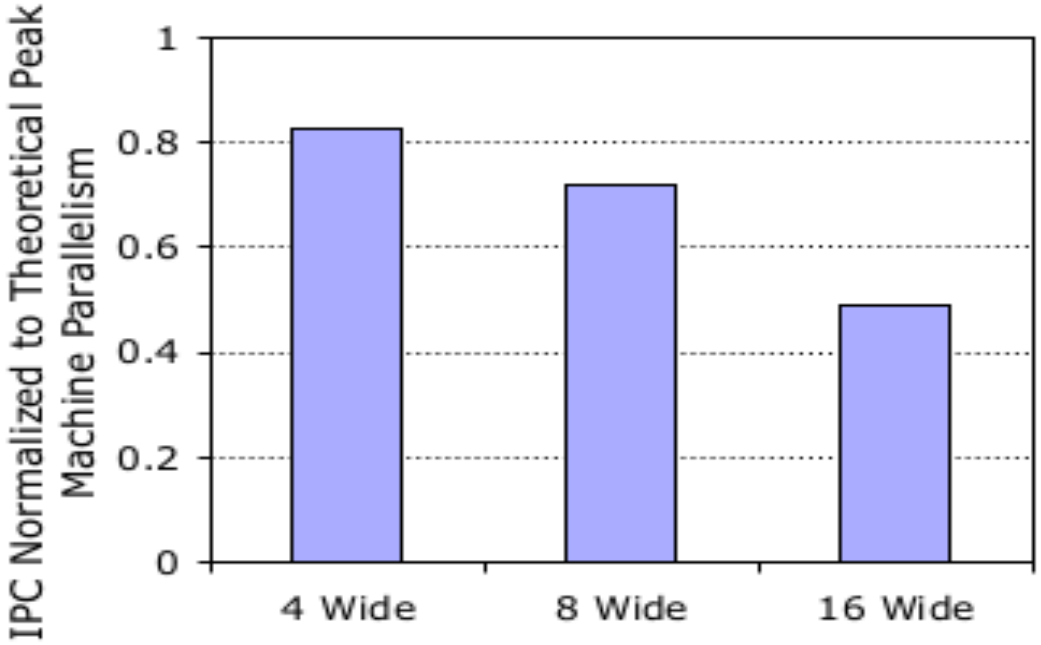


Figure 6. Performance loss for PDOM versus SIMD warp size (idealized memory system).

Dynamic Warp Formation & Scheduling

New warps formed after every divergence point.

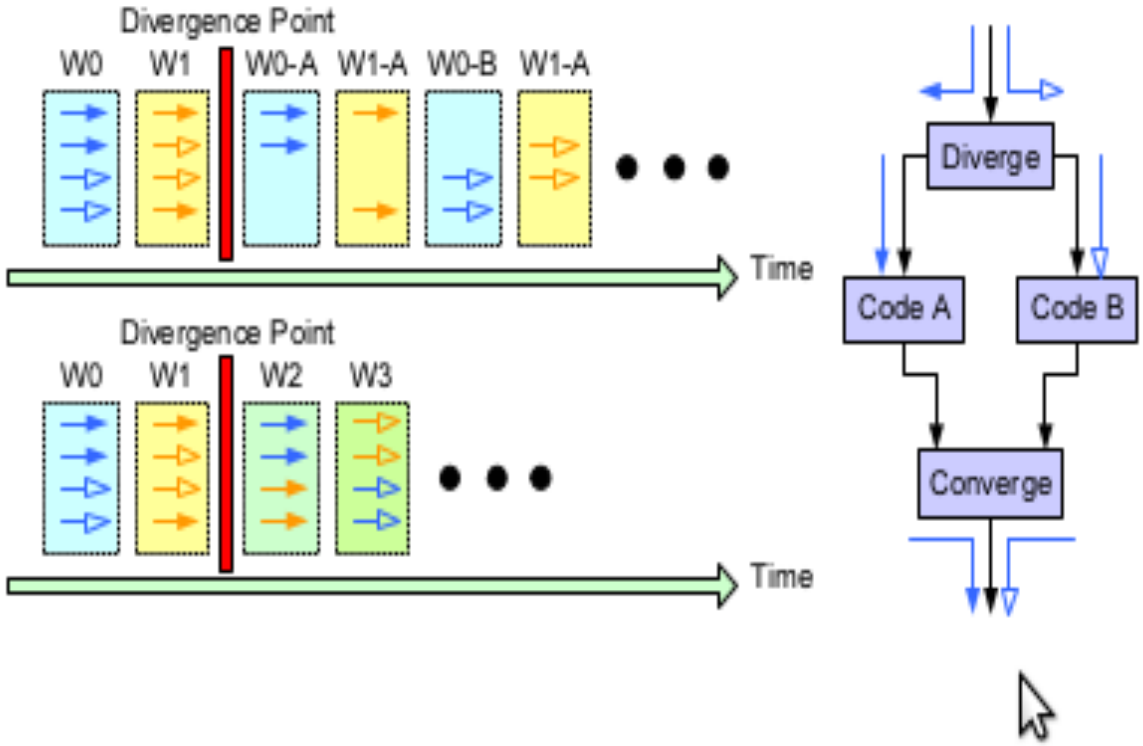


Figure 9. Dynamic warp formation example.

Implementing dynamic warp formation requires changes to the register file.

Register file must be organized in banks with all pipelines having equal access to it with uniform latency.

We must have “**lane aware** ” warp formation or the register file must have a crossbar.

This results in an increase in the size of the register file.

Issue Heuristics

Majority (DMaj): choosing the most common PC among all the existing warps and issuing all warps at this PC.

Minority (DMin): warps with the least frequent PCs are given priority so that they can catch up with the other threads.

Time Stamp (DTime): The oldest warp will be issued first.

Post-Dominator Priority (DPdPri): Issue priority of the warp is relative to it's distance from the PDOM.

Program Counter (DPC): Warps with smaller PC's are scheduled ahead.

Area Estimation

The overall area consumption of dynamic warp formation and scheduling for each core is 2.799 mm^2 .

With 8 cores per chip as per our initial configuration, this becomes 22.39 mm2 , which is 4.7% of the total area of the GeForce 8800GTX

Experimental Results

PDOM achieves a speedup of 93.4% versus not reconverging (Naive).

Dynamic warp formation (DYNB) achieves a further speedup of 22.5% using the Majority heuristic.

The difference between average DYNB and MIMD performance is only 4.6%.

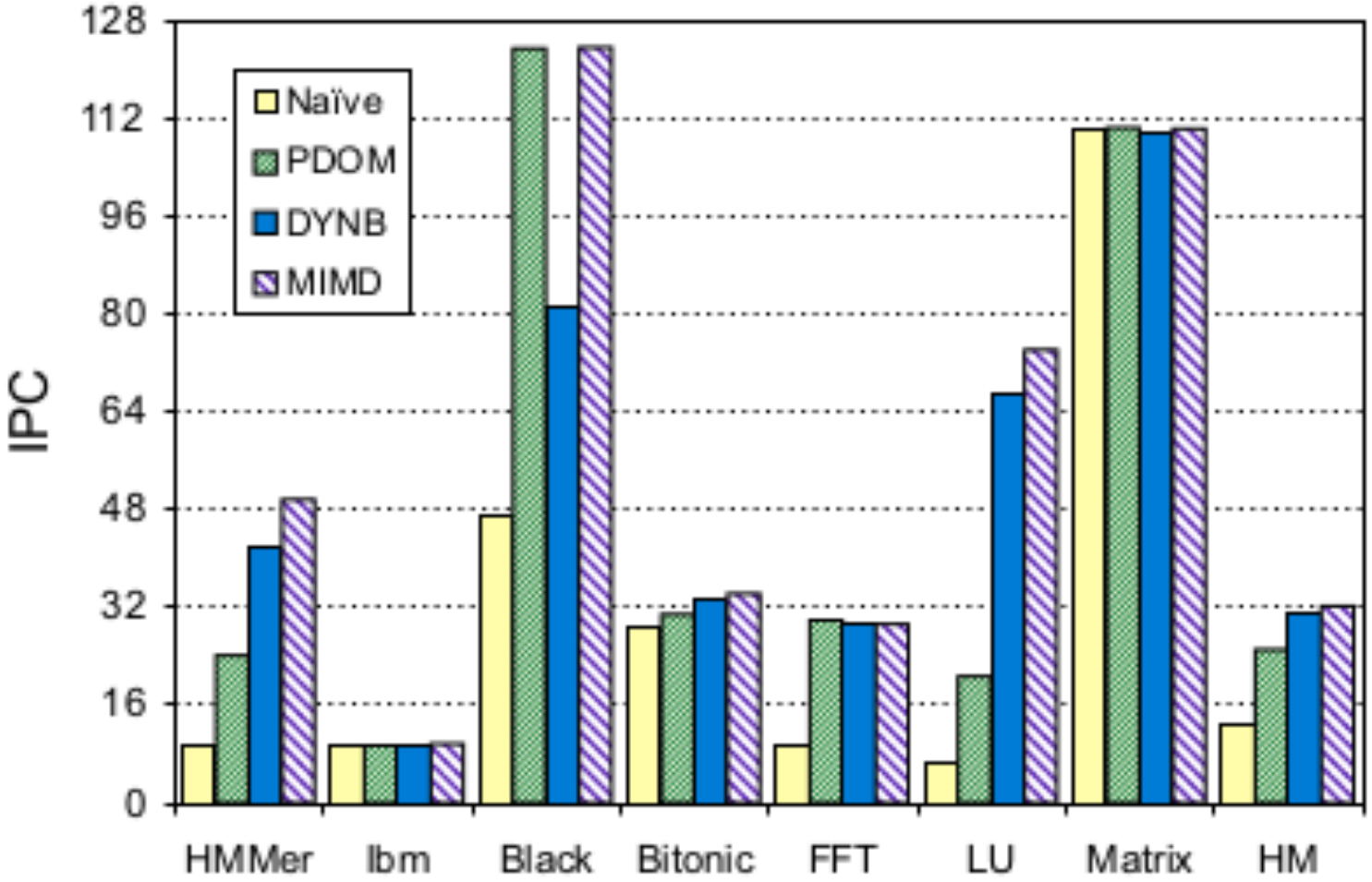


Figure 12. Performance comparison of Naïve, PDOM, and DYNB versus MIMD.

Conclusion

- Branch divergence can significantly degrade a GPU's performance.

  50.5% performance loss with SIMD width =16

- Dynamic Warp Formation & Scheduling
  - 20.7% on average better than reconvergence
  - 4.7% area cost

Pro’s and Con’s

Pro’s :

Dynamic approach unlike PDOM

Con’s:

Register file micro-architecture becomes complicated.

Context info about the threads need to be maintained to allow for lane-aware scheduling.