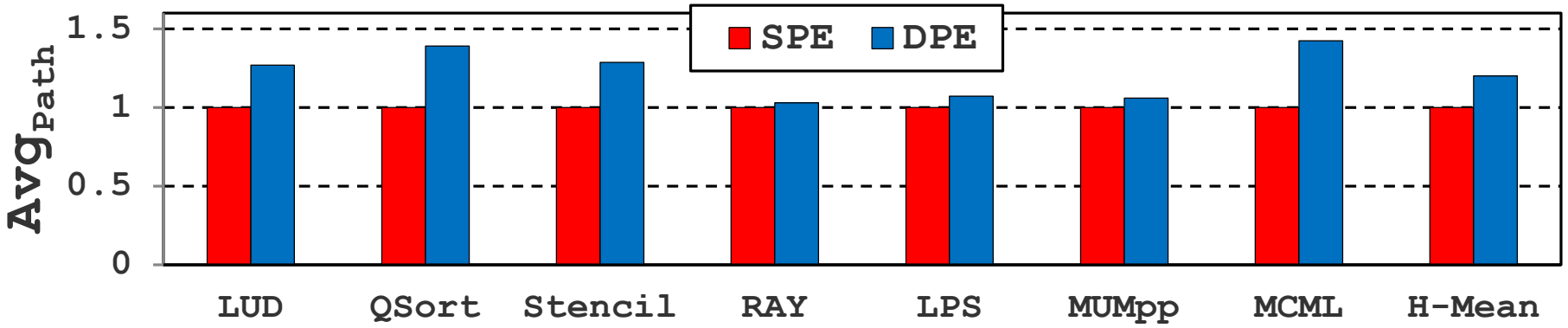
The Dual-Path Execution Model for Efficient GPU Control Flow

Minsoo Rhu and Mattan Erez

The University of Texas at Austin
Electrical and Computer Engineering Department

DPE: Dual-Path Execution Model

- Single-instruction multiple-thread (SIMT) model
  - Dominant model in current NVIDIA/AMD GPUs
  - ***Stack***-based reconvergence: optimal structured control flow
- Problem
  - Only a ***single*** path is considered for scheduling at a time
  - Restricts thread-level parallelism (aka *path* parallelism)
- DPE solution
  - Expose ***both*** taken&not-taken paths to the thread-scheduler
  - Reconvergence *identical* to baseline single-path exec (**SPE**)

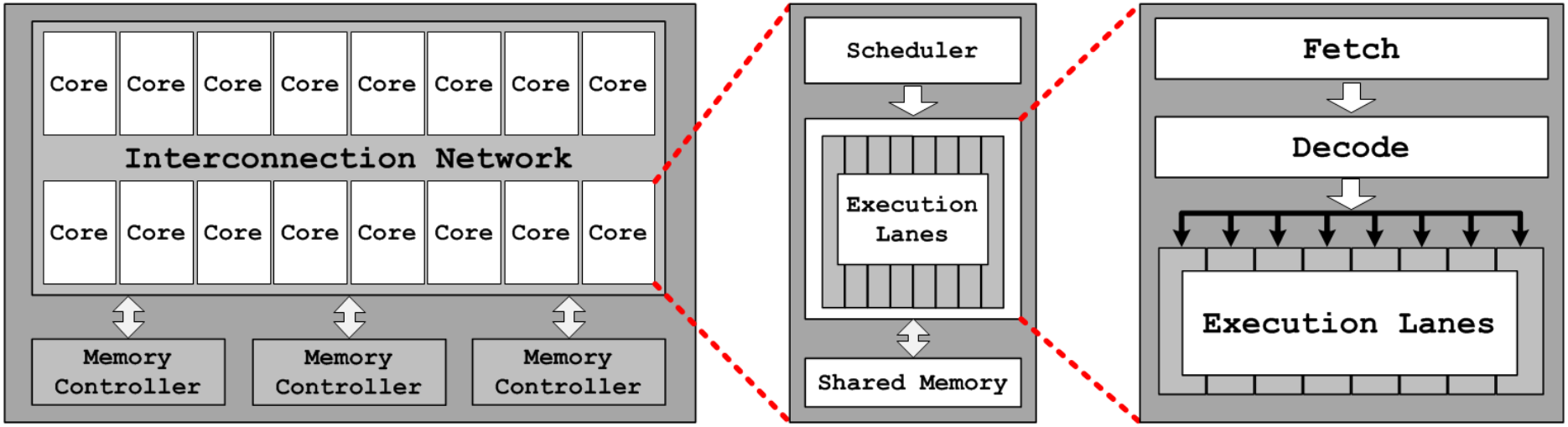


Outline

- **GPU and the SIMT stack-based reconvergence**
- Limitation of the single-path stack model
- DPE– **d**ual-**p**ath **e**xecution model
- Related works
- Evaluation

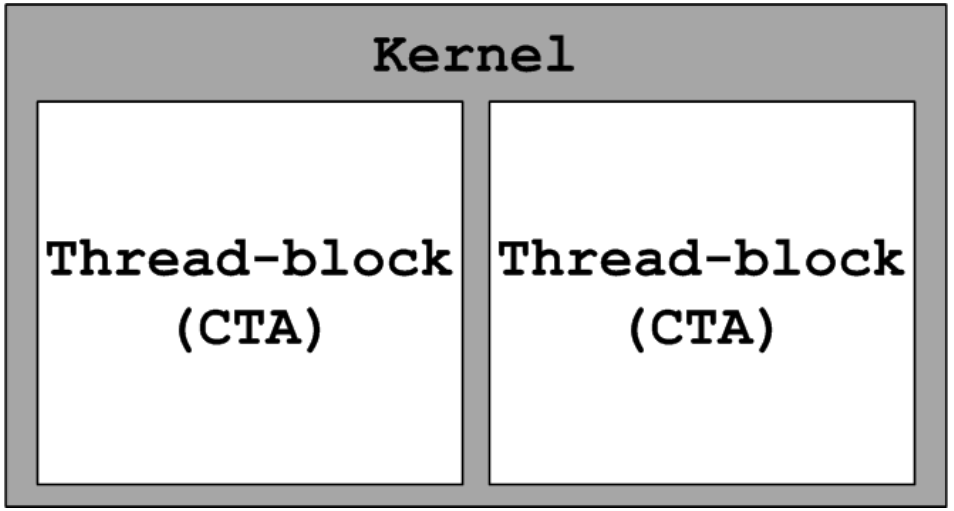
Graphic Processing Units (GPUs)

- General-purpose many-core accelerators
  - Supports non-graphics APIs (e.g. CUDA, OpenCL)
- Scalar frontend (fetch & decode) + parallel backend
  - Amortizes the cost of frontend and control



CUDA exposes hierarchy of data-parallel threads

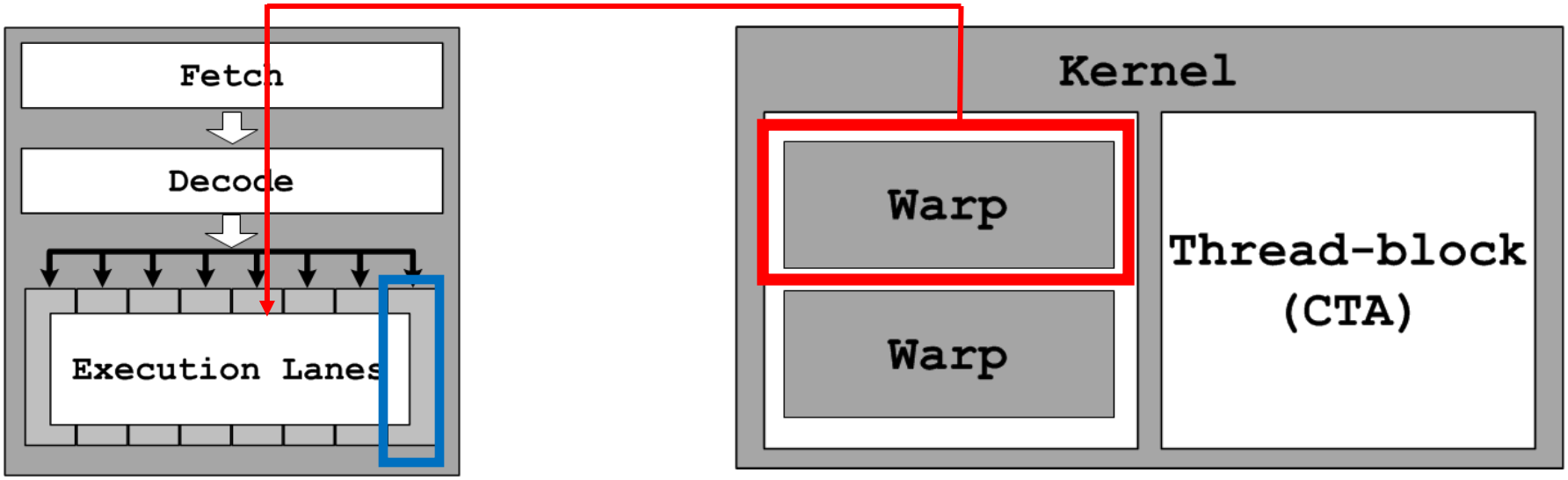
- **SPMD model**: single ***kernel*** executed by all threads
- **Kernel / Thread-block**
  - Multiple **thread-blocks** (concurrent-thread-arrays(**CTA**s)) compose a **kernel**



CUDA exposes hierarchy of data-parallel threads

- **SPMD model**: single ***kernel*** executed by all threads
- **Kernel / Thread-block / *Warp* / *Thread***
  - Multiple **warps** compose a **thread-block**
  - Multiple **threads (32)** compose a warp

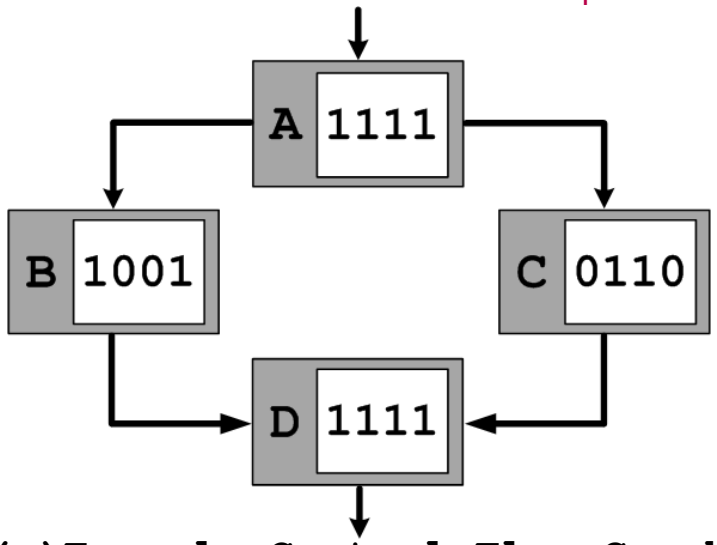
A warp is scheduled as a *batch* of threads



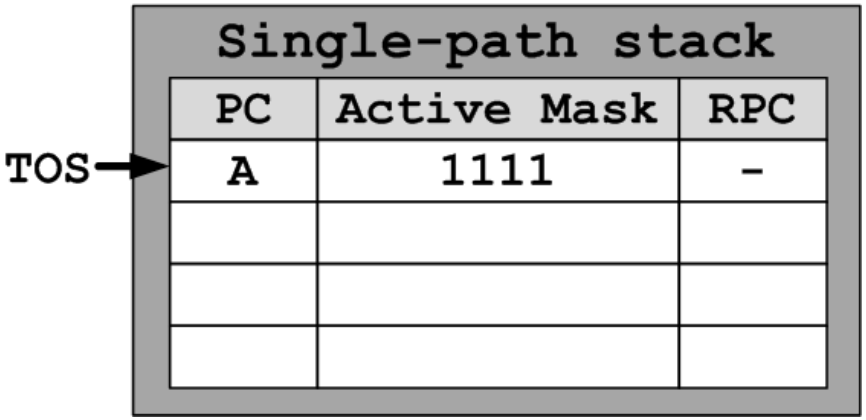
: Thread executes in its dedicated lane

GPUs have HW support for conditional branches

- Control-divergence: threads in warp branch differently
  - **Predicate-Bitmasks**: allows subset of a warp to commit results
  - ***Stack***-based re-convergence model
  - Always execute ***active***-threads at the ***top-of-stack* (TOS)**.
  - Reconvergence PC (RPC) derived at compile-time
    - RPC: ***immediate post-dominator (PDOM)*** of the branching-point
    - Guarantees optimal reconvergence of threads for structured control flow



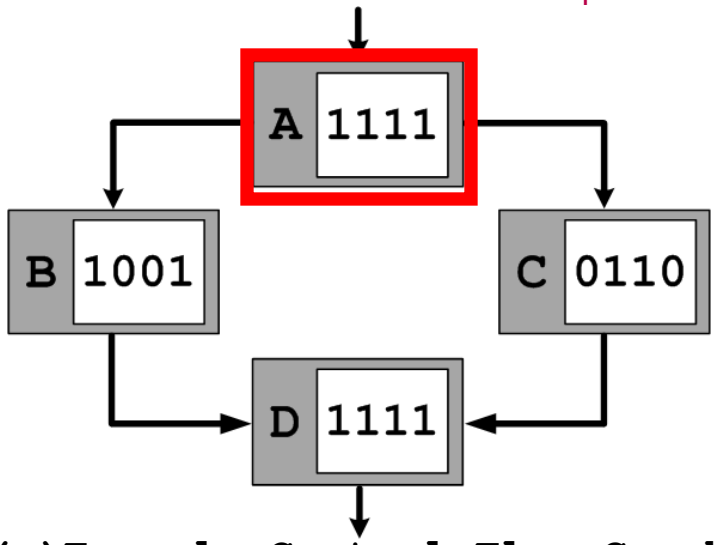
(a) Example Control Flow Graph
1: Active, 0: Inactive



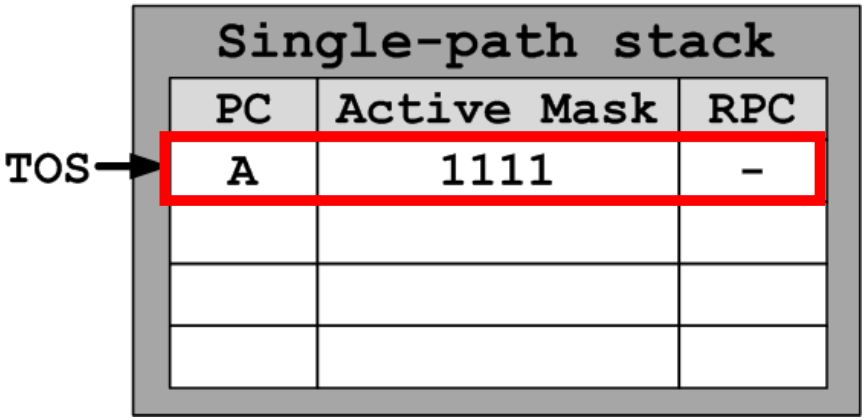
(b) Using the stack for control flow management

GPUs have HW support for conditional branches

- Control-divergence: threads in warp branch differently
  - **Predicate-Bitmasks**: allows subset of a warp to commit results
  - ***Stack***-based re-convergence model
  - Always execute ***active***-threads at the ***top-of-stack* (TOS)**.
  - Reconvergence PC (RPC) derived at compile-time
    - RPC: ***immediate post-dominator (PDOM)*** of the branching-point
    - Guarantees optimal reconvergence of threads for structured control flow



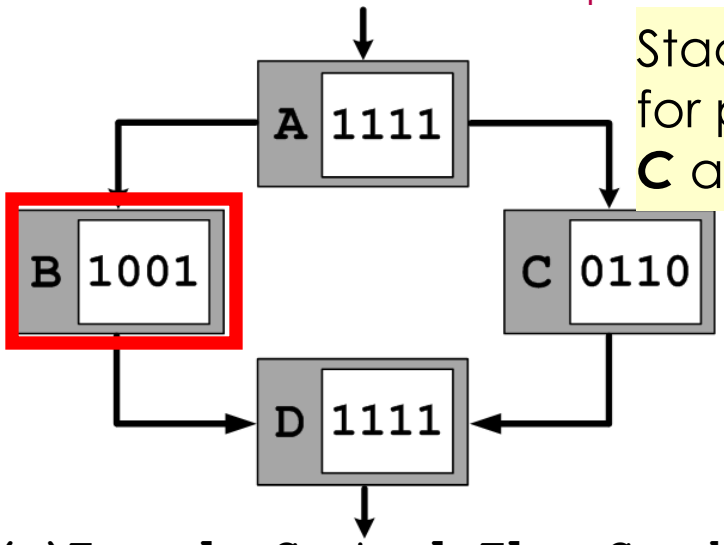
(a) Example Control Flow Graph
1: Active, 0: Inactive



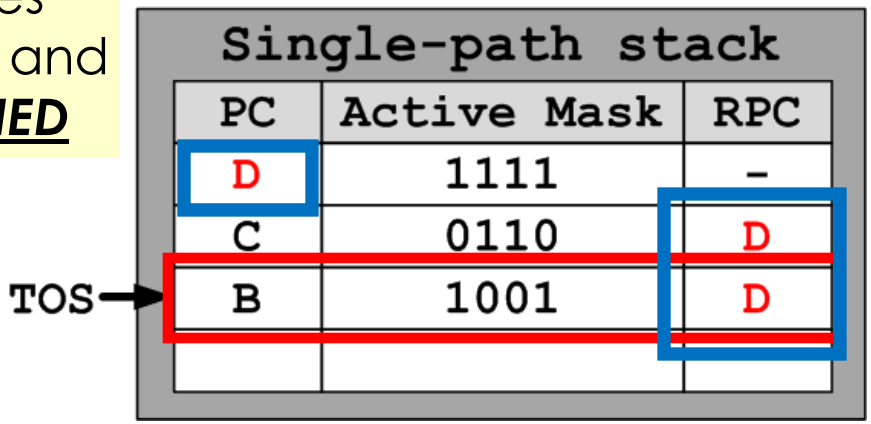
(b) Using the stack for control flow management

GPUs have HW support for conditional branches

- Control-divergence: threads in warp branch differently
  - **Predicate-Bitmasks**: allows subset of a warp to commit results
  - ***Stack***-based re-convergence model
  - Always execute ***active***-threads at the ***top-of-stack* (TOS)**.
  - Reconvergence PC (RPC) derived at compile-time
    - RPC: ***immediate post-dominator (PDOM)*** of the branching-point
    - Guarantees optimal reconvergence of threads for structured control flow



Stack entries for paths ***B*** and ***C*** are ***PUSHED***

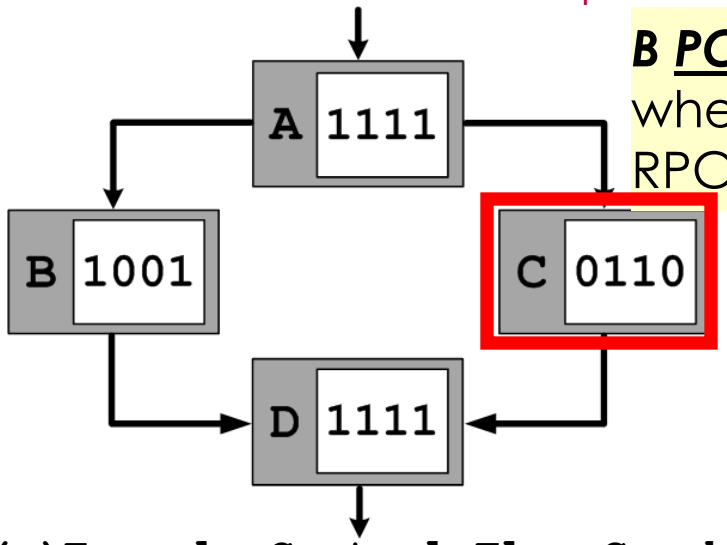


(a) Example Control Flow Graph
1: Active, 0: Inactive

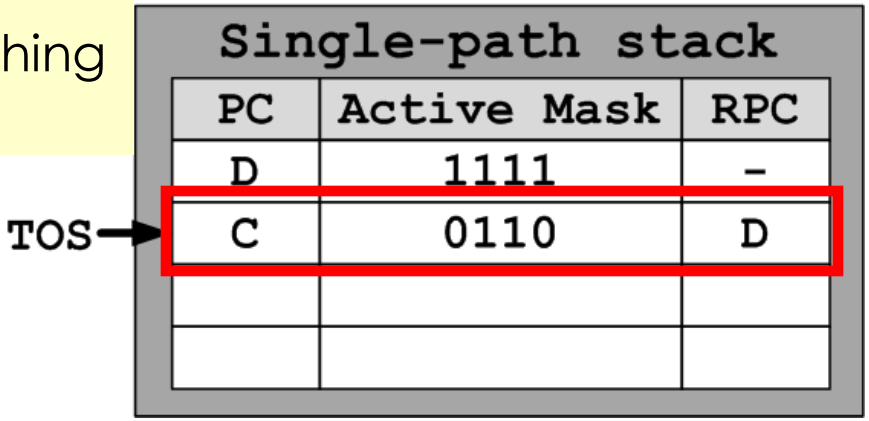
(b) Using the stack for control flow management

GPUs have HW support for conditional branches

- Control-divergence: threads in warp branch differently
  - **Predicate-Bitmasks**: allows subset of a warp to commit results
  - ***Stack***-based re-convergence model
  - Always execute ***active***-threads at the ***top-of-stack* (TOS)**.
  - Reconvergence PC (RPC) derived at compile-time
    - RPC: ***immediate post-dominator (PDOM)*** of the branching-point
    - Guarantees optimal reconvergence of threads for structured control flow



B POPPED when reaching RPC (**D**)

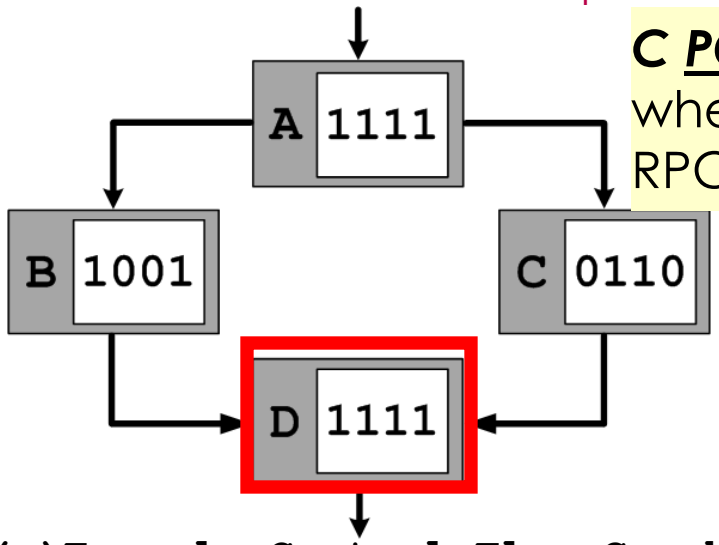


(a) Example Control Flow Graph
1: Active, 0: Inactive

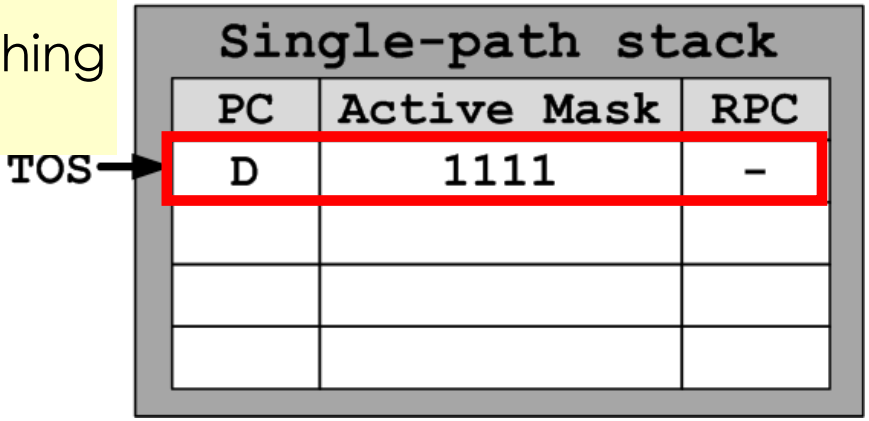
(b) Using the stack for control flow management

GPUs have HW support for conditional branches

- Control-divergence: threads in warp branch differently
  - **Predicate-Bitmasks**: allows subset of a warp to commit results
  - ***Stack***-based re-convergence model
  - Always execute ***active***-threads at the ***top-of-stack* (TOS)**.
  - Reconvergence PC (RPC) derived at compile-time
    - RPC: ***immediate post-dominator (PDOM)*** of the branching-point
    - Guarantees optimal reconvergence of threads for structured control flow



C POPPED when reaching RPC (**D**)



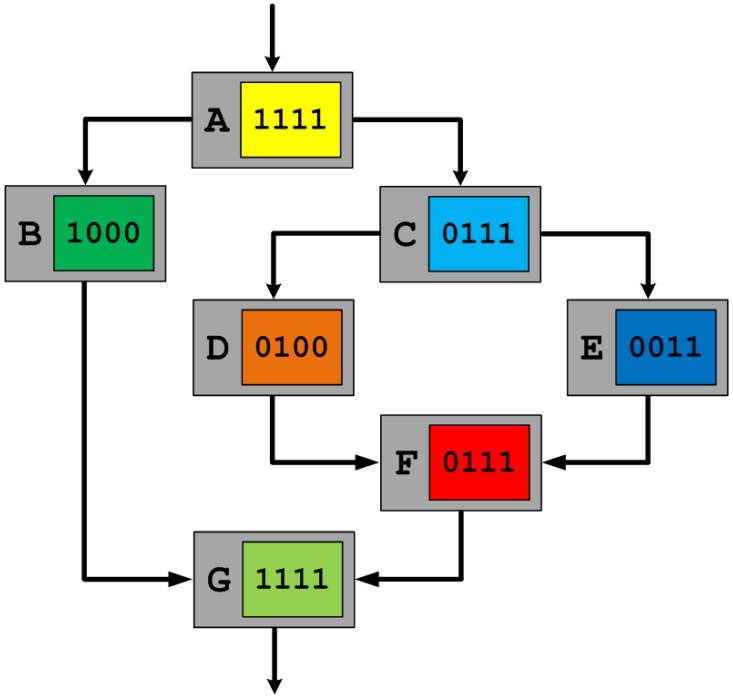
(a) Example Control Flow Graph
1: Active, 0: Inactive

(b) Using the stack for control flow management

Outline

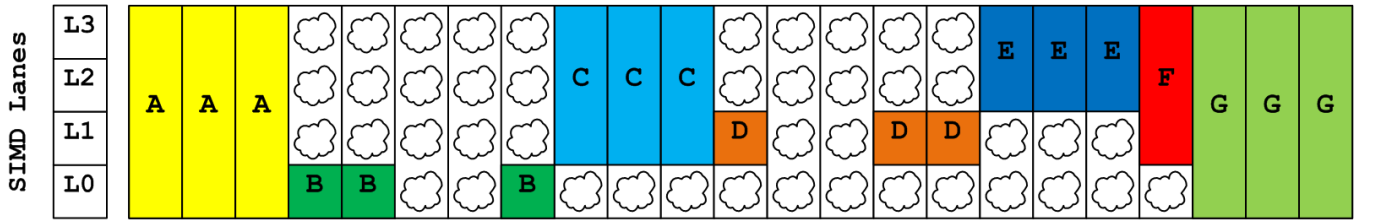
- GPU and the SIMT stack-based reconvergence
- **Limitation of the single-path stack model**
- DPE– **d**ual-**p**ath **e**xecution model
- Related works
- Evaluation

1. Underutilization of SIMD lanes



(a) Example Control Flow Graph

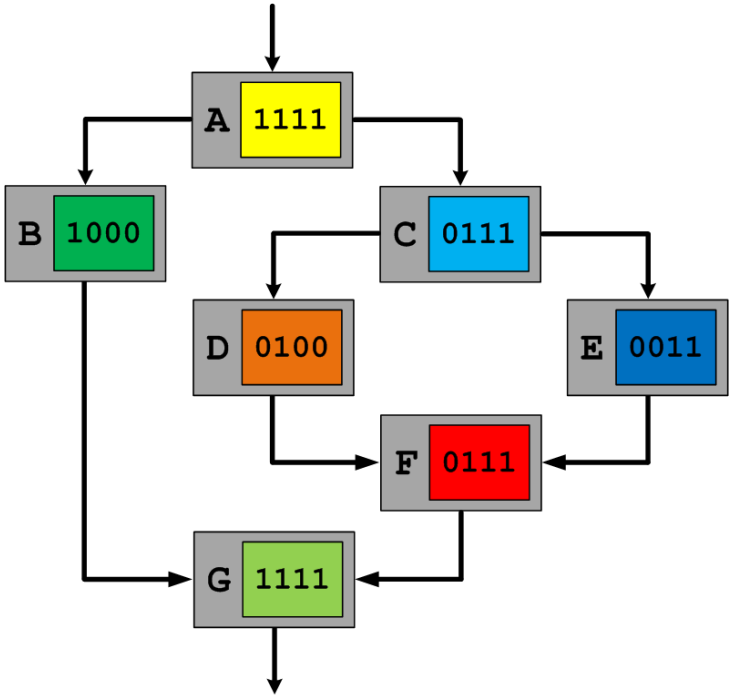
- Number of active threads in a warp decreases every time control diverges
- Active area of research
  - Thread block compaction [Fung'11]
  - Larger warps [Narasiman'11]
- **NOT what this work solves!**



(b) Execution flow using SPE.

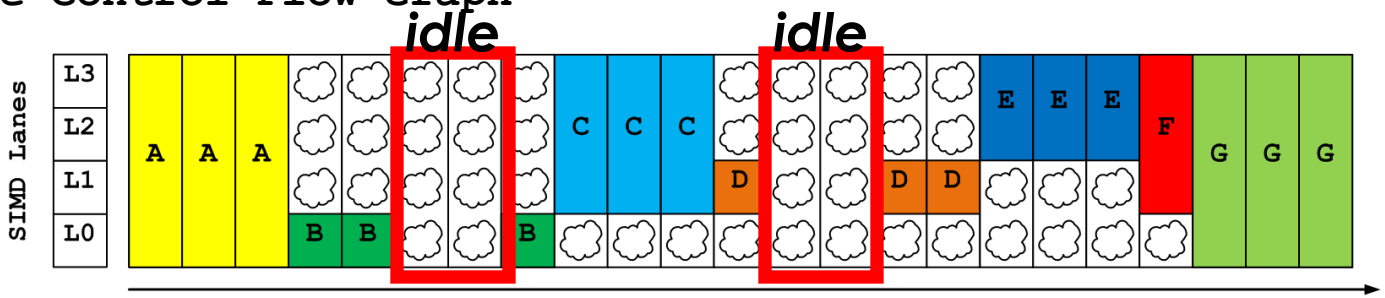
: Threads (lanes) masked out from execution, remaining idle.

2. Serialization of execution paths



(a) Example Control Flow Graph

- Execution transitions to another path ***only*** after ***all*** the instructions in the ***current*** path (basic-block) are executed.
- Misses opportunities to schedule from *other* ***concurrent*** paths
  - E.g. paths **B/C**



(b) Execution flow using SPE.

idle cycles: phases where the scheduler is short of warps to schedule

Outline

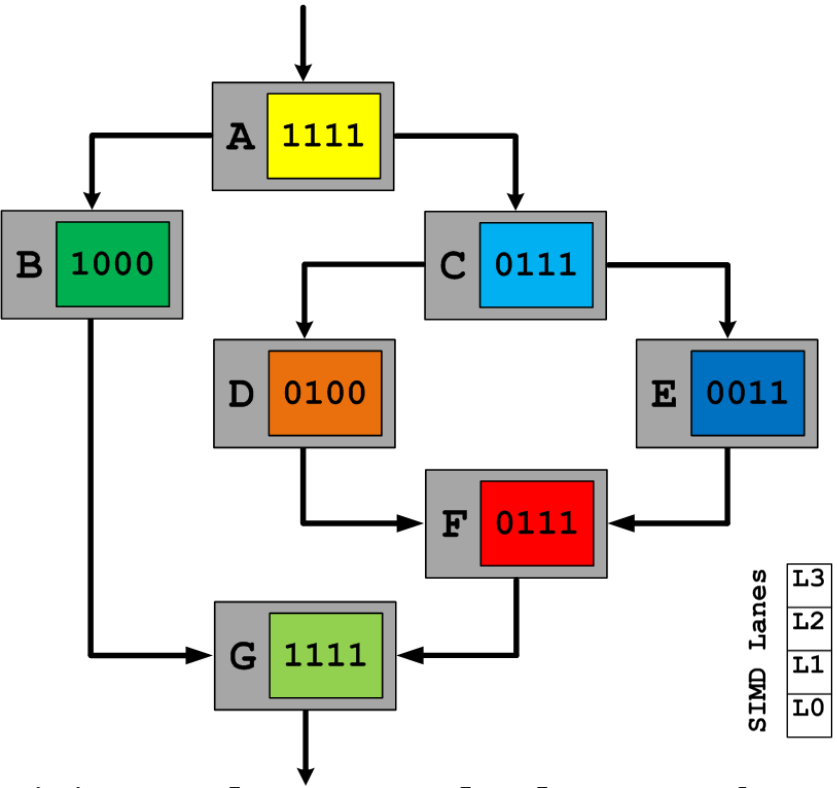
- GPU and the SIMT stack-based reconvergence
- Limitation of the single-path stack model
- **DPE– dual-path execution model**
- Related works
- Evaluation

Dual-Path Execution Model (DPE)

- Maintain simplicity of reconvergence stack
- Minimal extensions to the SPE model
- Enhance path-parallelism
  - Without sacrificing SIMD lane utilization
- No additional compiler-support/heuristics required
  - Purely a microarchitectural approach

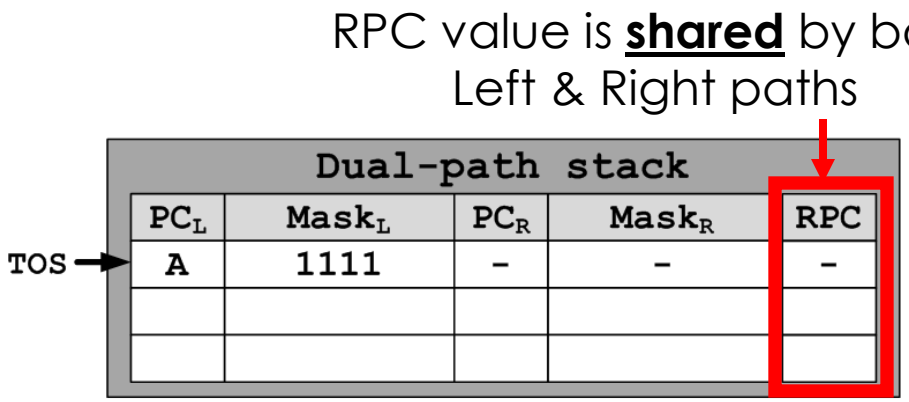
DPE Components: Dual-Path Stack

- DPE stack microarchitecture
  - Each entry accommodates ***both*** paths of a branching point
  - Single dual-path entry instead of two separate entries in SPE



(a) Example Control Flow Graph

RPC value is **shared** by both Left & Right paths

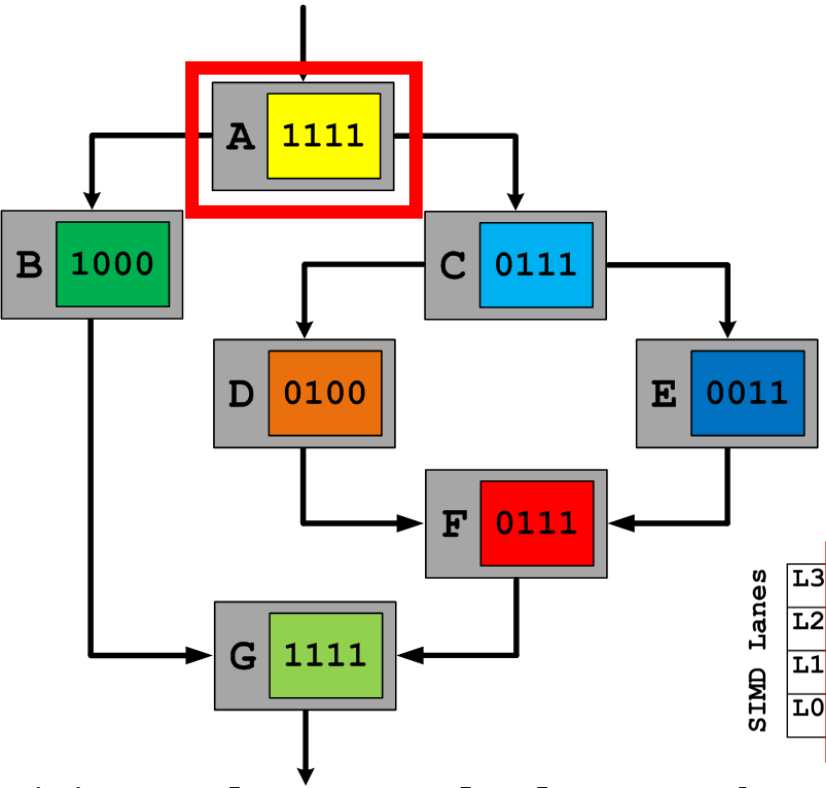


(b) Using the DPE stack for control flow management

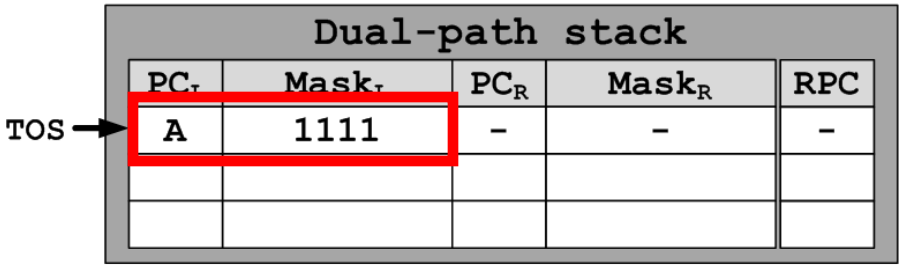
(c) Execution flow using DPE.

DPE Components: Dual-Path Stack

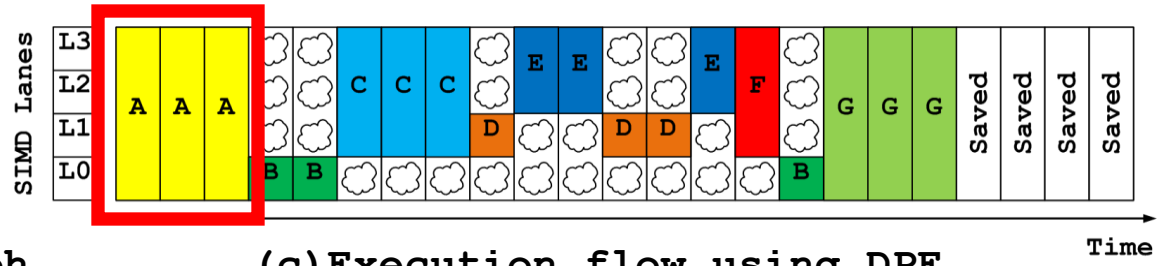
- DPE stack microarchitecture
  - Each entry accommodates ***both*** paths of a branching point
  - Single dual-path entry instead of two separate entries in SPE



(a) Example Control Flow Graph



(b) Using the DPE stack for control flow management

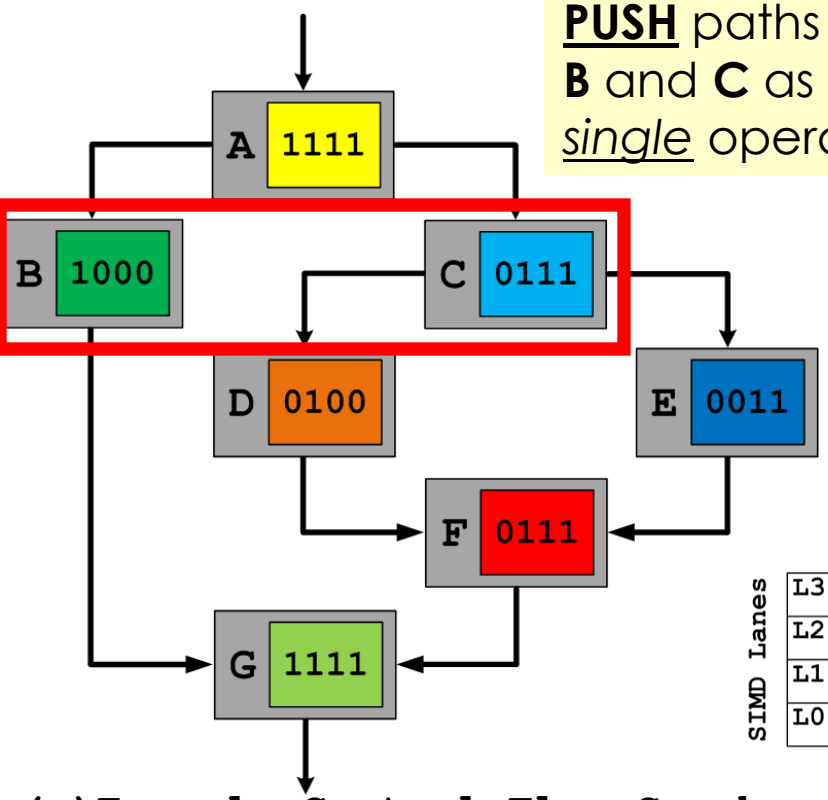


(c) Execution flow using DPE.

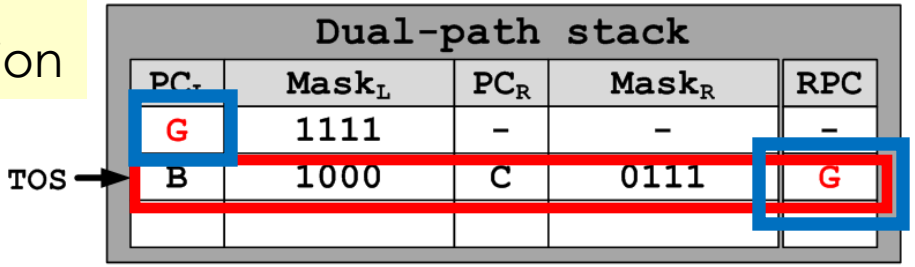
DPE Components: Dual-Path Stack

- DPE stack microarchitecture
  - Each entry accommodates ***both*** paths of a branching point
  - Single dual-path entry instead of two separate entries in SPE

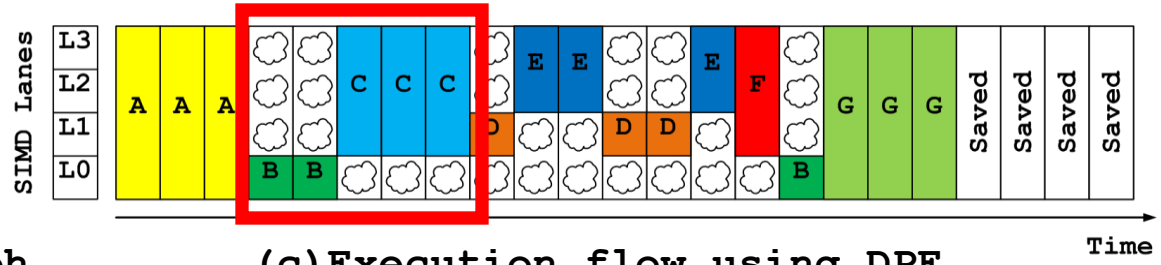
PUSH paths **B** and **C** as a *single* operation



(a) Example Control Flow Graph



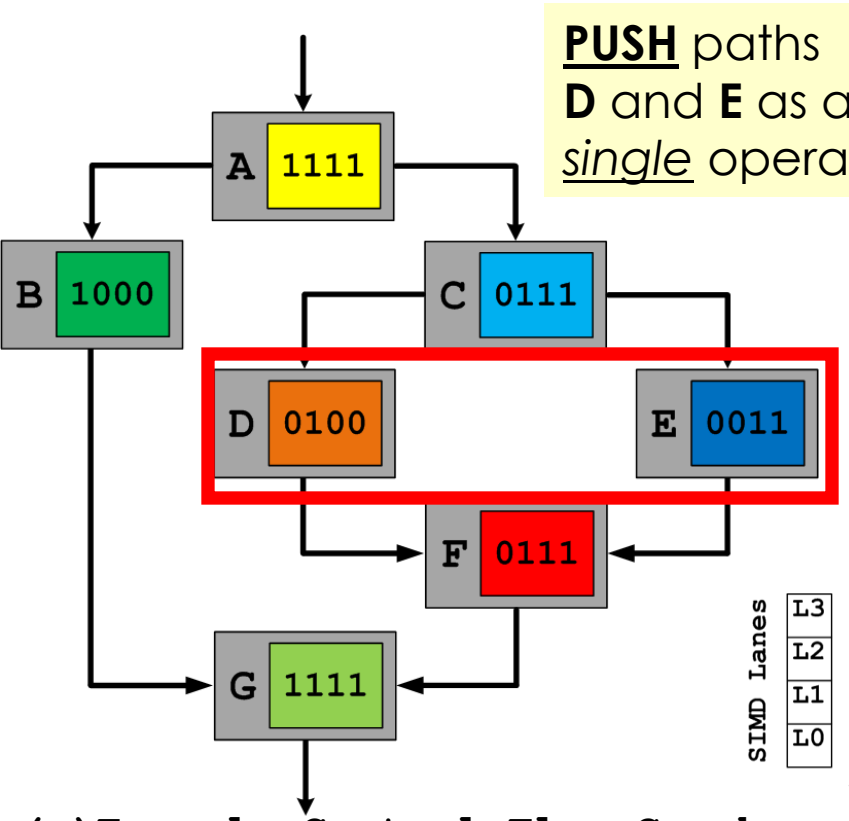
(b) Using the DPE stack for control flow management



(c) Execution flow using DPE.

DPE Components: Dual-Path Stack

- DPE stack microarchitecture
  - Each entry accommodates ***both*** paths of a branching point
  - Single dual-path entry instead of two separate entries in SPE



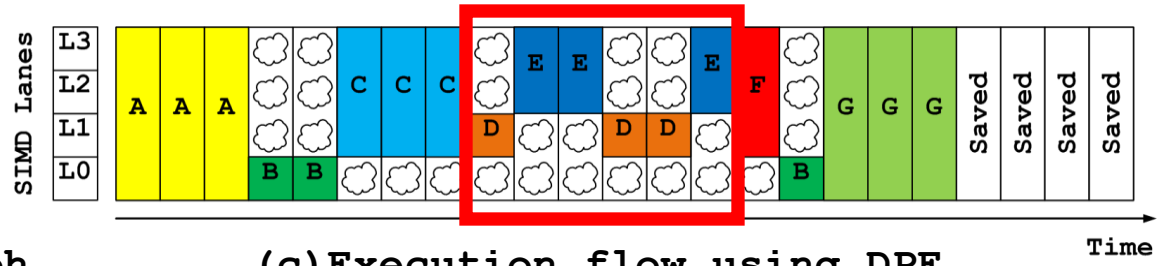
(a) Example Control Flow Graph

PUSH paths **D** and **E** as a *single* operation

Dual-path stack

	PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
	G	1111	-	-	-
	B	1000	F	0111	G
TOS →	D	0100	E	0011	F

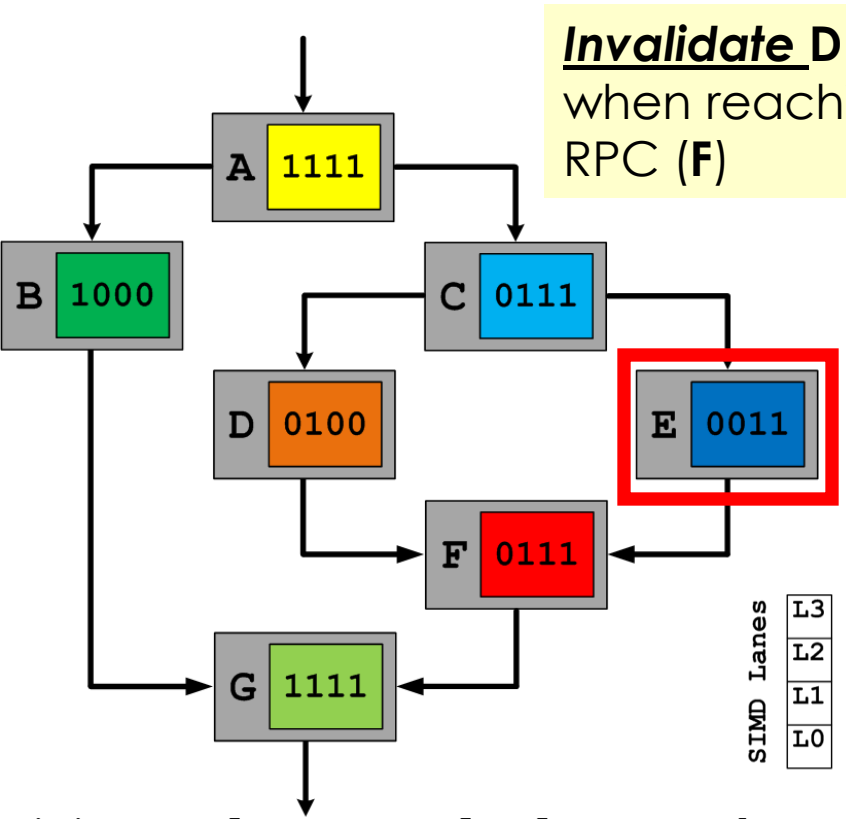
(b) Using the DPE stack for control flow management



(c) Execution flow using DPE.

DPE Components: Dual-Path Stack

- DPE stack microarchitecture
  - Each entry accommodates ***both*** paths of a branching point
  - Single dual-path entry instead of two separate entries in SPE



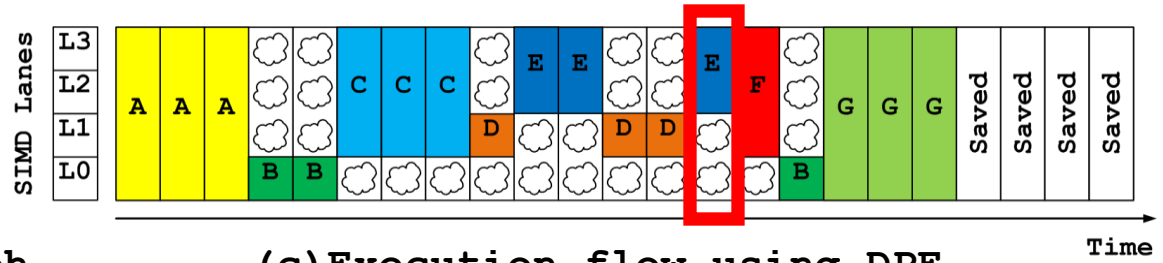
(a) Example Control Flow Graph

Invalidate **D** when reaching RPC (**F**)

Dual-path stack

	PC_L	$Mask_L$	PC_R	$Mask_R$	RPC
	G	1111	-	-	-
	B	1000	F	0111	G
TOS →	-	-	E	0011	F

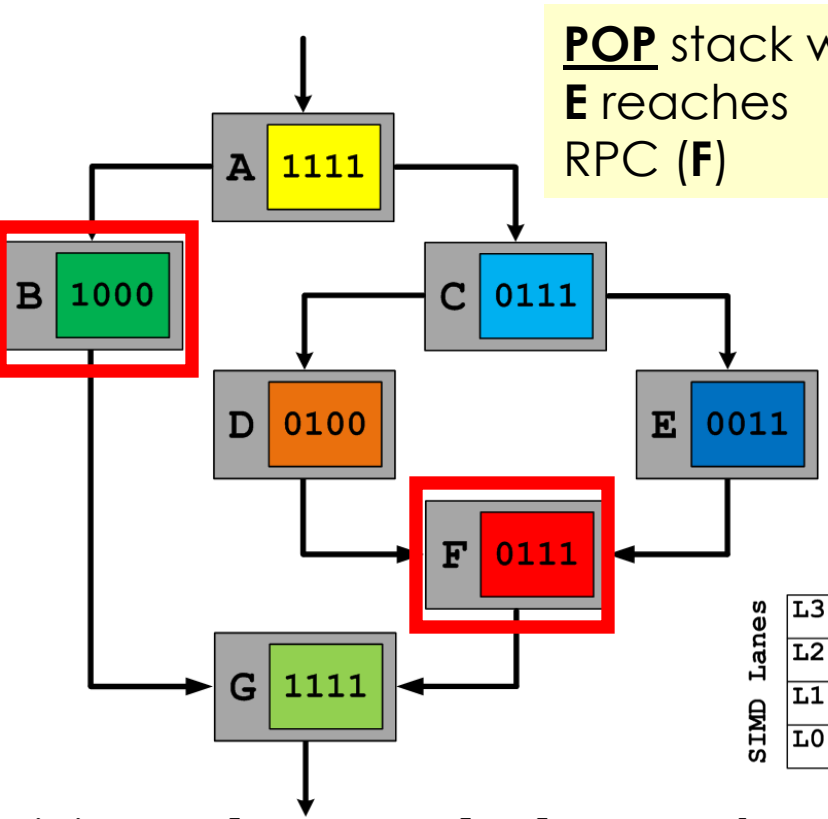
(b) Using the DPE stack for control flow management



(c) Execution flow using DPE.

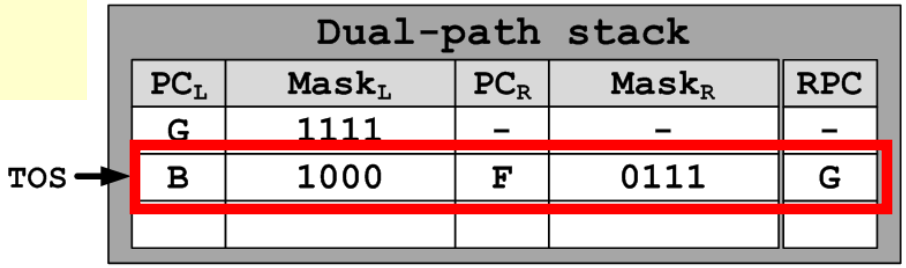
DPE Components: Dual-Path Stack

- DPE stack microarchitecture
  - Each entry accommodates ***both*** paths of a branching point
  - Single dual-path entry instead of two separate entries in SPE

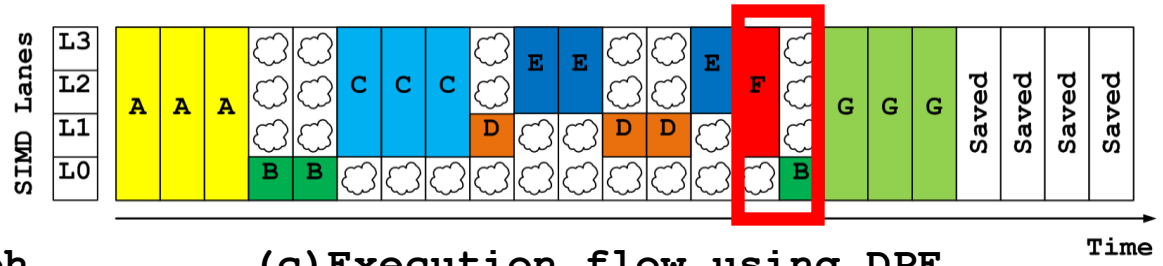


(a) Example Control Flow Graph

POP stack when **E** reaches RPC (**F**)



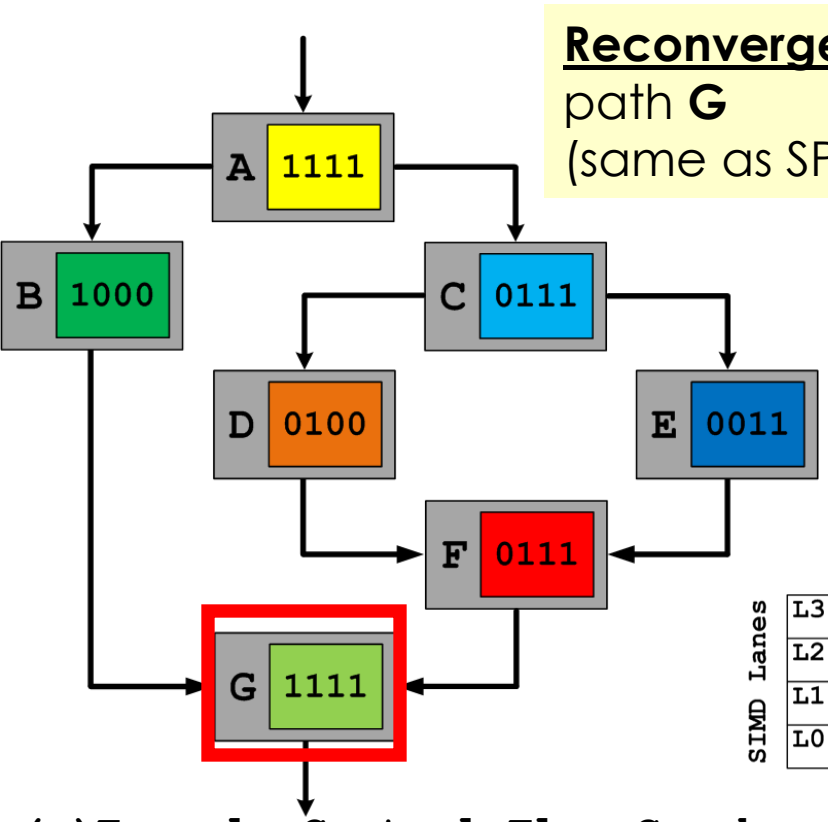
(b) Using the DPE stack for control flow management



(c) Execution flow using DPE.

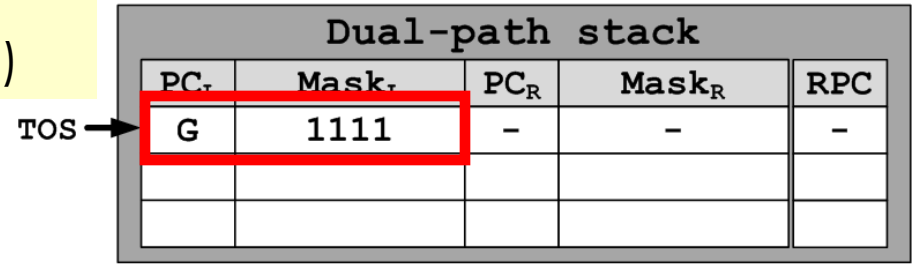
DPE Components: Dual-Path Stack

- DPE stack microarchitecture
  - Each entry accommodates ***both*** paths of a branching point
  - Single dual-path entry instead of two separate entries in SPE

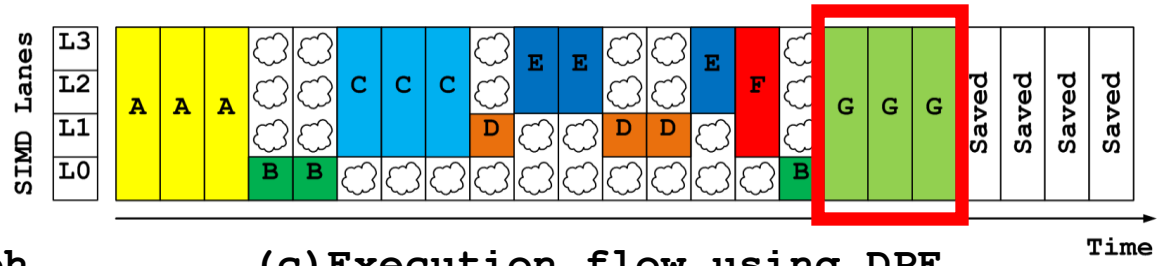


(a) Example Control Flow Graph

Reconverge at path **G** (same as SPE)



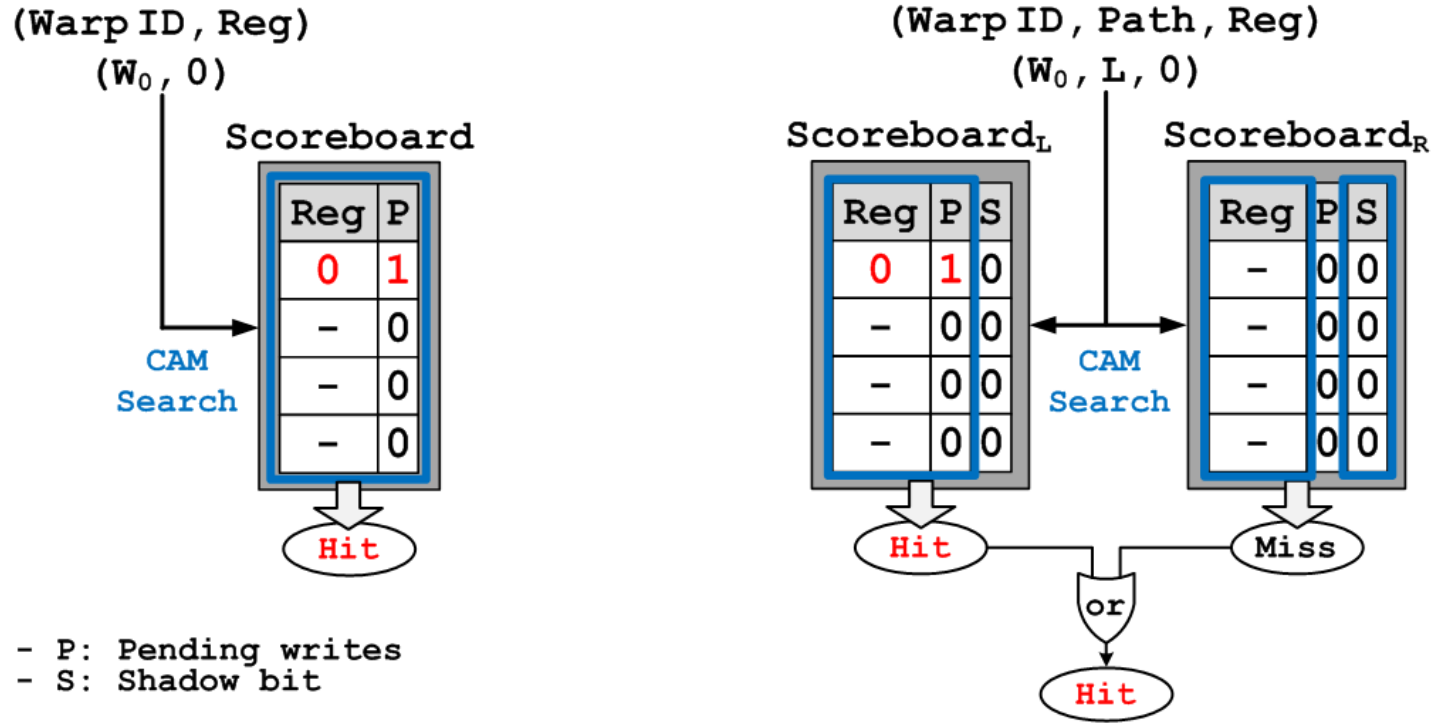
(b) Using the DPE stack for control flow management



(c) Execution flow using DPE.

DPE Components: Scoreboard

- Current GPUs execute ***intra***-warp threads back-to-back
- DPE complications because L/R paths share registers
  - Separate Left/Right scoreboards
  - Shadow-bits for pre-/post-divergence dependencies



(a) SPE Scoreboard (b) DPE Scoreboard

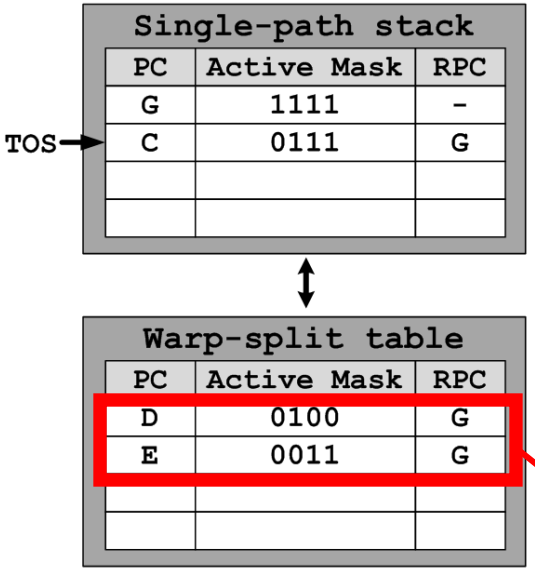
DPE Components: Warp Scheduler

- Scheduler enhanced to cover ***both*** L/R paths
  - For maximal benefits of DPE, scheduler overhead is doubled
  - *'Constrained'* scheduler
    - Warp-scheduler is fed with only a single path at the TOS
    - Path to be sent to the scheduler is rotated when a *long-latency operation* is executed
    - Benefits decrease from **14.9%** to **11.7%**

Outline

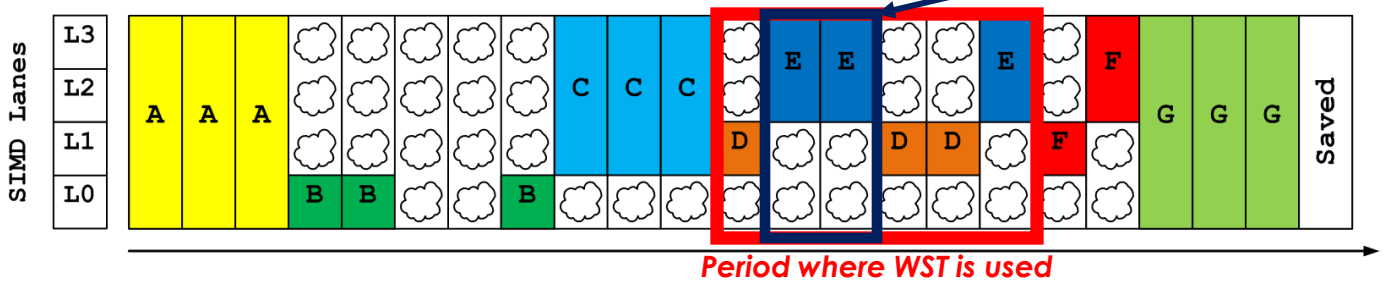
- GPU and the SIMT stack-based reconvergence
- Limitation of the single-path stack model
- DPE– dual-path execution model
- **Related works**
- Evaluation

Dynamic warp subdivision (DWS) [Meng'10]



(a) Adoption of a separate Warp-split-table for path interleaving

- Transition between …
  - **Serial** mode (single-path stack)
  - **Interleaving** (suddivision) mode (Warp-split table)
- When using the …
  - Stack: execution is ***same*** as in baseline SPE.
  - WST: concurrent paths in the WST can be interleaved, **filling in** the idle cycles.



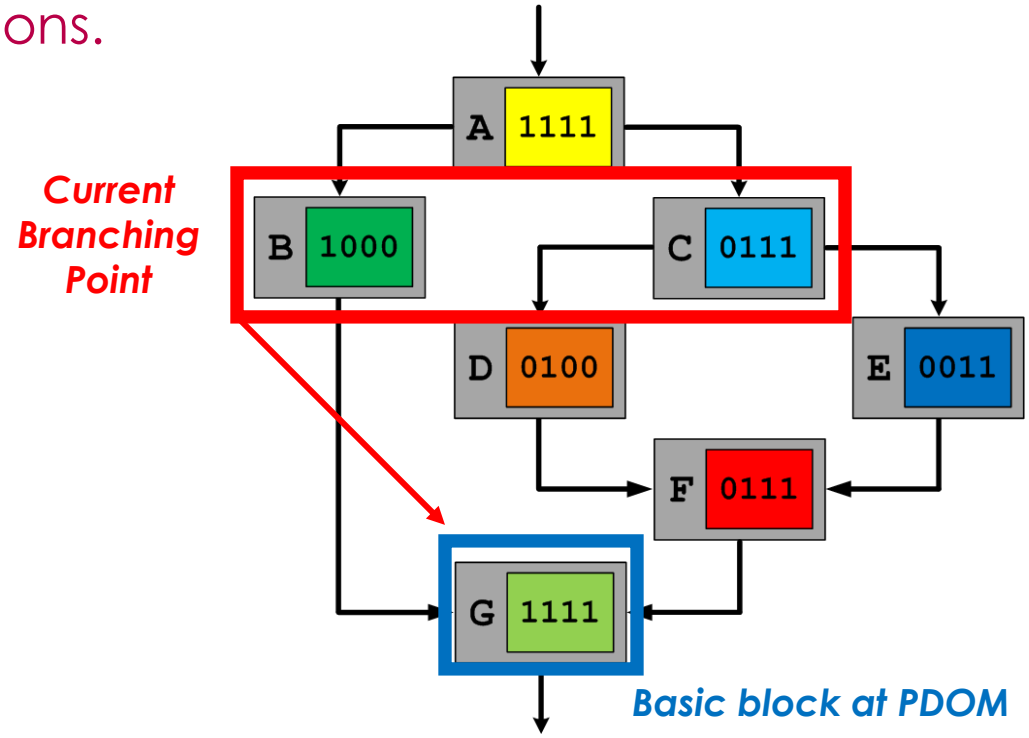
(b) Execution flow using DWS (subdivision is applied for path D/E)

Dynamic warp subdivision (DWS) [Meng'10]

- DWS has significant complexity and overheads
  - **Compiler modifications and sub-optimal heuristics**

    <Condition to activate "*interleaving*" mode>

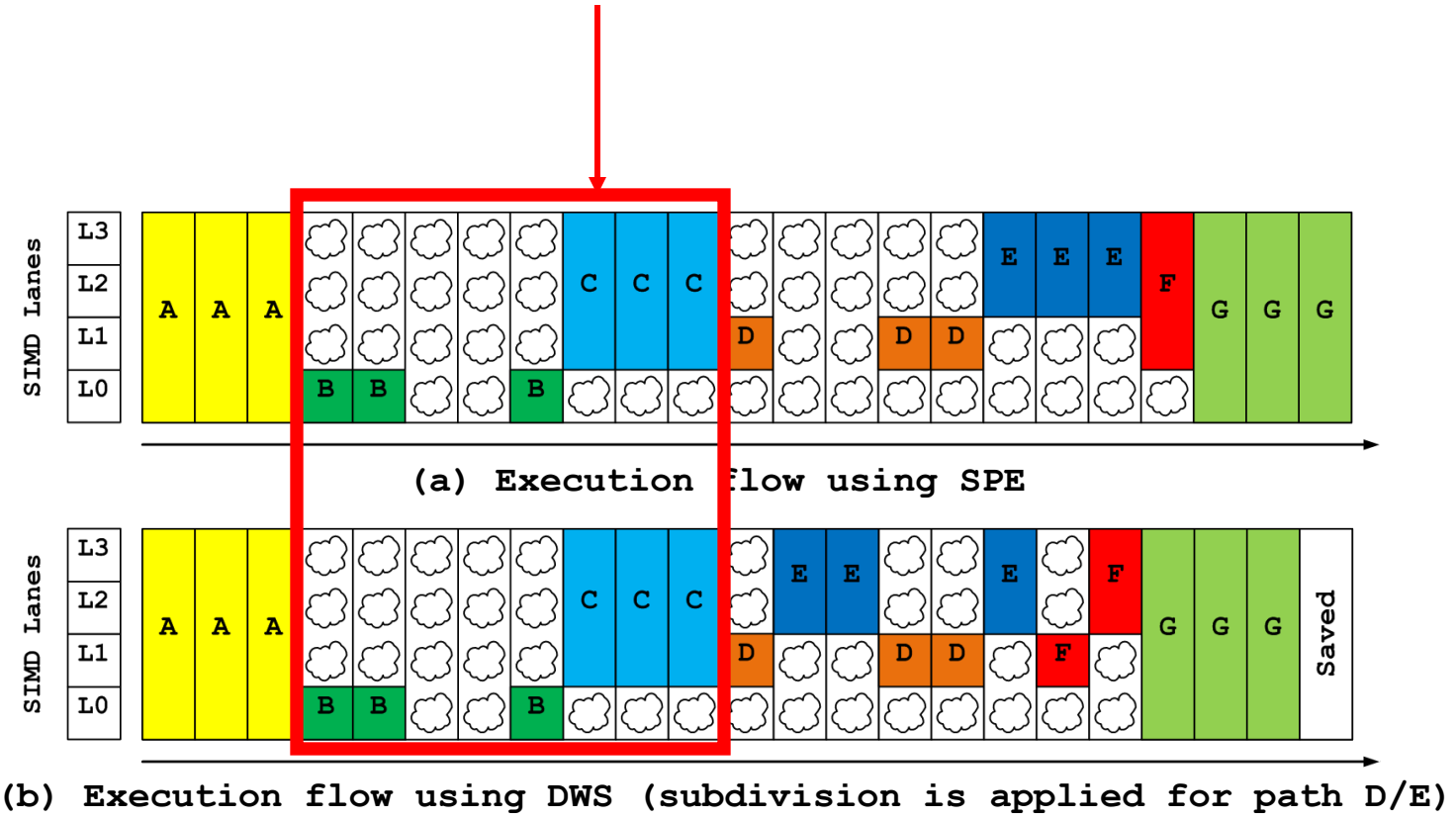
    Warp is subdivided ***only if*** the PDOM is followed by a **short** basic block of no more than ***'subdivision-threshold'*** number of instructions.



(a) Example Control Flow Graph

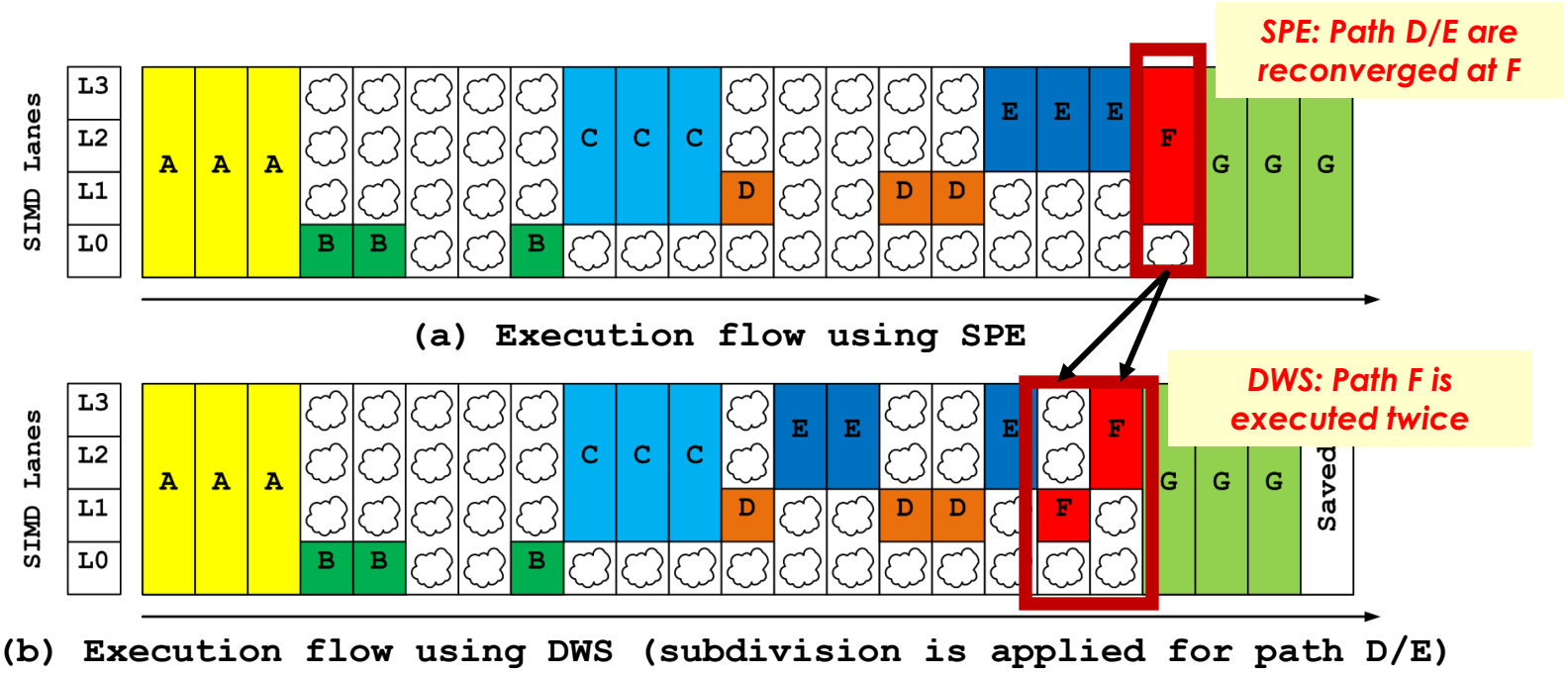
Dynamic warp subdivision (DWS) [Meng'10]

- DWS has significant complexity and overheads
  - **Compiler modifications and sub-optimal heuristics**
    - ***Heuristics***-based subdivision: cannot always exploit *ALL* interleaving opportunities.
      - i.e. path **B/C** are **NOT** interleaved below.



Dynamic warp subdivision (DWS) [Meng'10]

- DWS has significant complexity and overheads
  - Compiler modifications and sub-optimal heuristics
  - **Sub-optimal reconvergence**
    - SPE stack no longer used for *optimal* reconvergence
    - Reconvergence is '*opportunistically*' initiated using WST entry's PC values
    - A warp-split that diverges is *further* split into narrower warp-splits



Dual-Instruction Multiple-Thread (DIMT) [Brunie'12]

- Issue '***two***' different instructions to the SIMD pipeline at the same time
  - Maximum of '***two***' paths are chosen for scheduling (**like** DPE)
  - Uses the '***thread-frontiers***' model, rather than the stack-architecture (**unlike** DPE)
  - Thread-frontiers [Diamos'11]
    - Better than stack model in handling '***unstructured***' control flow
    - More complicated than the stack model

Outline

- GPU and the SIMT stack-based reconvergence
- Limitation of the single-path stack model
- DPE– dual-path execution model
- Related works
- **Evaluation**

Simulation Environment

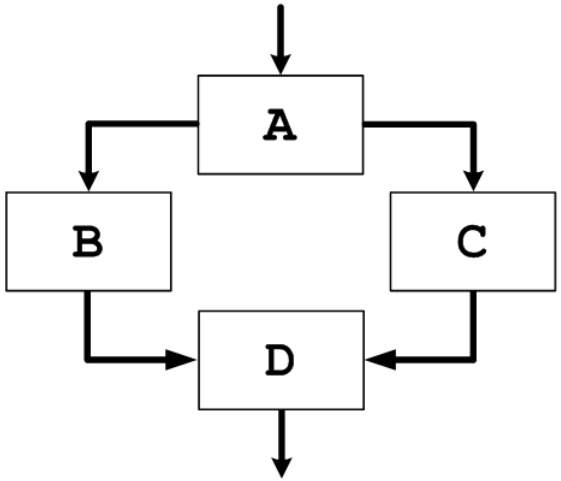
- GPGPU-Sim (v3.1.0)
  - 15 shader cores (Streaming Multiprocessors)
  - 1536 threads per core, 32K registers per core
  - Cache: 16kB L1, 768kB Unified L2
  - Warp scheduling policy: Round-Robin
  - Memory Controller : FR-FCFS
    - 29.6GB/s per channel, 6 channels overall
  - Configured similar to NVIDIA Fermi Architecture

- Workloads
  - Regular/Irregular apps with various input-sets
  - Chosen from CUDA-SDK(v3.0), Rodinia, Parboil, etc

Benchmark Characterization

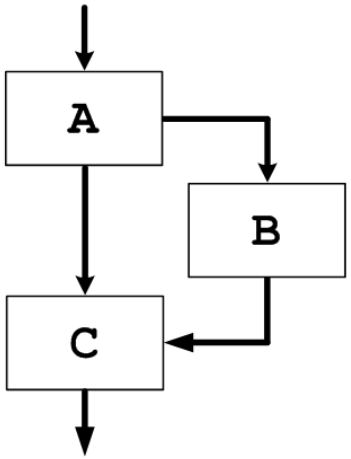
- Interleavable branch
  - Both "**if**" and "**else**" part active
  - DPE can *always* interleave
  - DWS can interleave paths *when* subdivision is activated

< Corresponding control flow graph >



- Non-interleavable branch
  - **No** '**else**' part
  - No added benefits with DPE
    - Operates the same as SPE
  - DWS can still interleave paths *when* subdivision is activated.

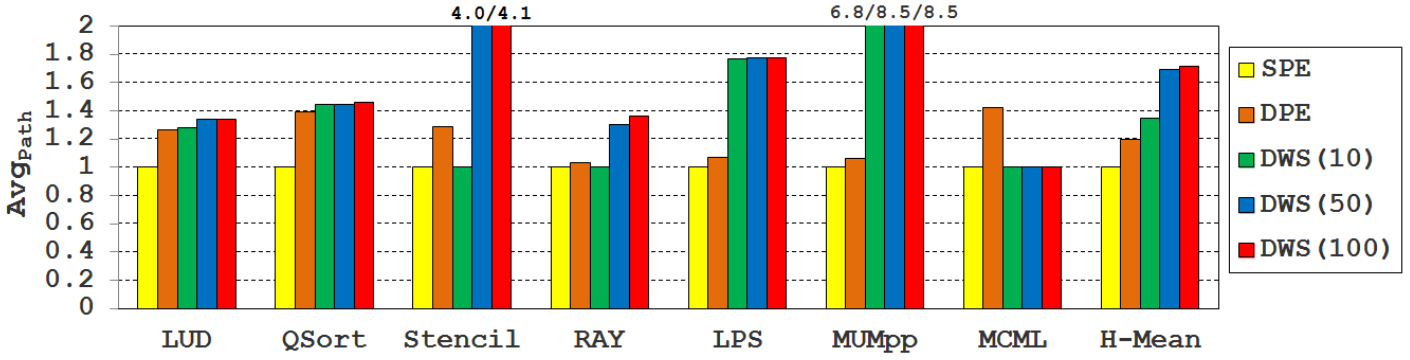
< Corresponding control flow graph >



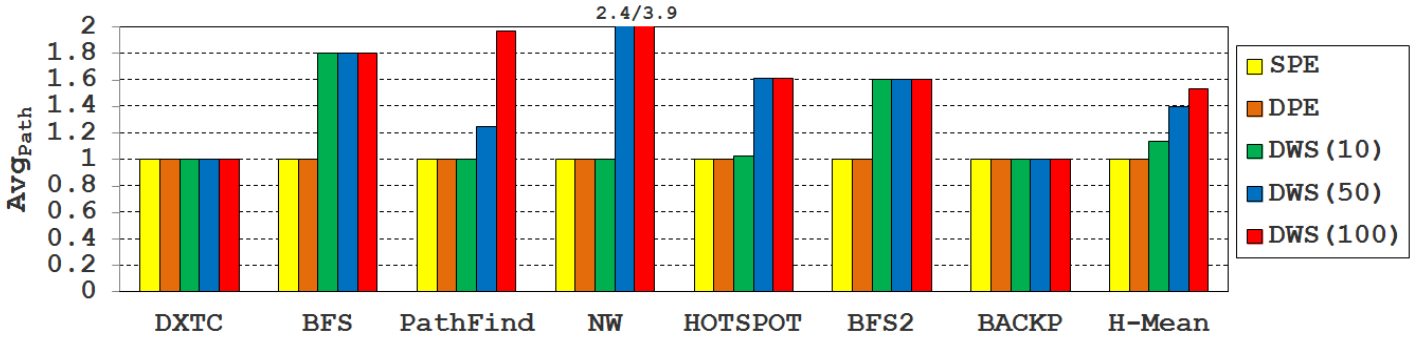
Average Path Parallelism

- SPE: always '1'
- DPE: '2' for interleavable branch and '1' otherwise
- DWS: '1' with stack and '**N**' with WST
  - **N** is the number of warp-splits in WST
  - DWS(**X**) designates a configuration using subdivision-threshold of '**X**'
  - The larger the **X**, the more aggressive subdivision is activated.

(a) Interleavable benchmarks



(b) Non-interleavable benchmarks

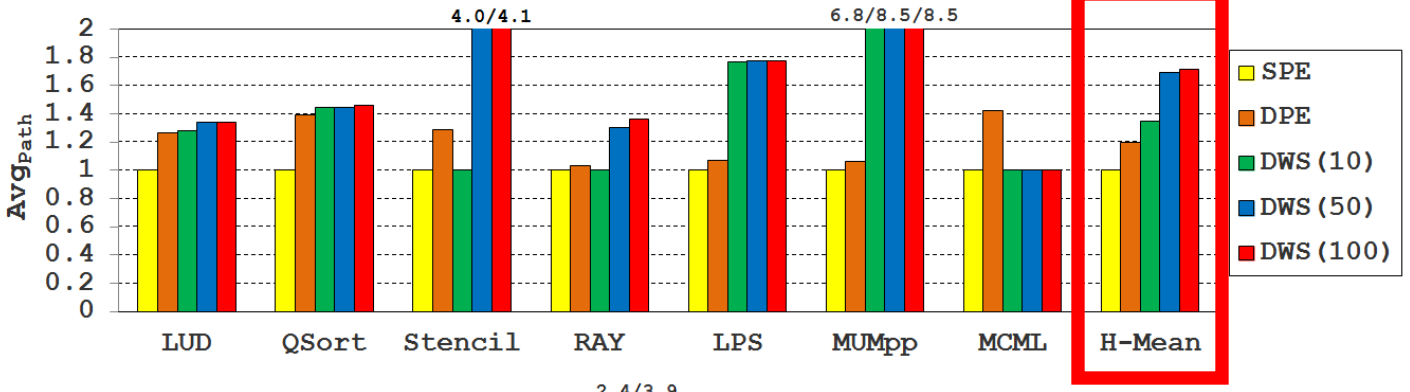


Average Path Parallelism

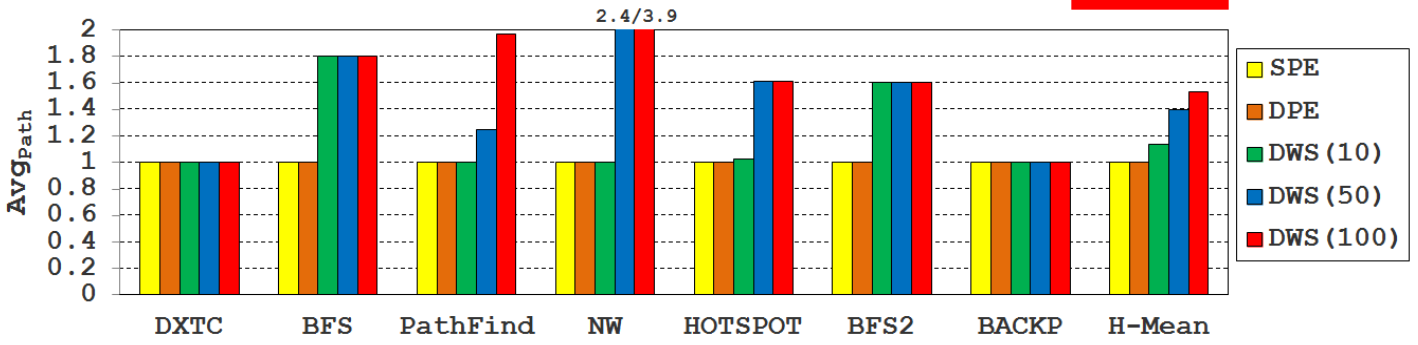
DPE: 20% increase
DWS(100) : 71% increase

- SPE: always '1'
- DPE: '2' for interleavable branch and '1' otherwise
- DWS: '1' with stack and '**N**' with WST
  - **N** is the number of warp-splits in WST
  - DWS(**X**) designates a configuration using subdivision-threshold of '**X**'
  - The larger the **X**, the more aggressive subdivision is activated.

(a) Interleavable benchmarks



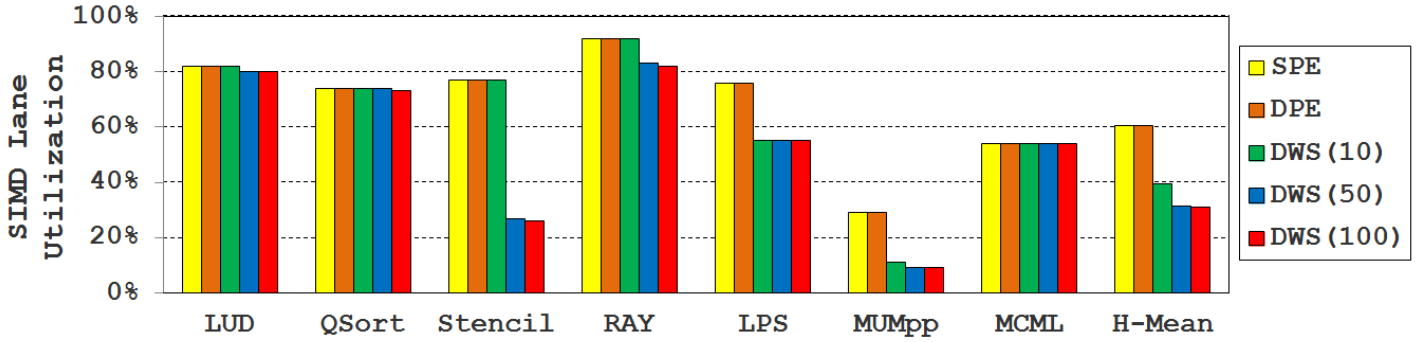
(b) Non-interleavable benchmarks



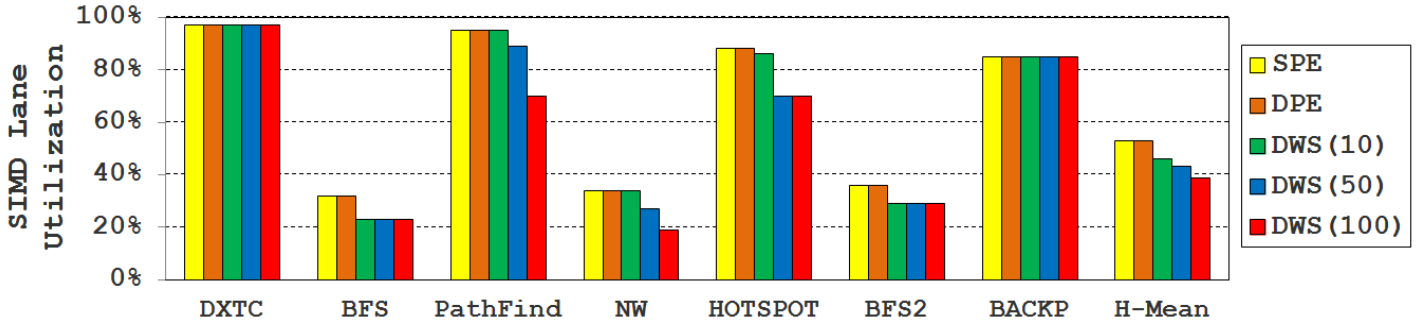
SIMD Lane Utilization ($SIMD_{util}$)

- SPE and DPE: ***always*** the same
- DWS
  - Identical when subdivision is '*never*' activated
  - Reduced when reconvergence is suboptimal

(a) Interleavable benchmarks



(b) Non-interleavable benchmarks



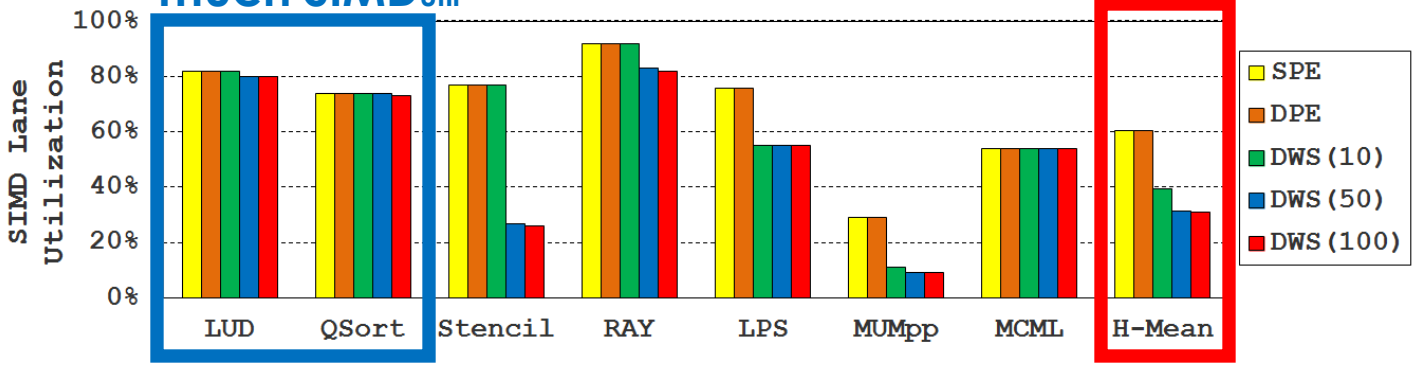
SIMD Lane Utilization ($SIMD_{util}$)

- SPE and DPE: ***always*** the same
- DWS
  - Identical when subdivision is '*never*' activated
  - Reduced when reconvergence is suboptimal

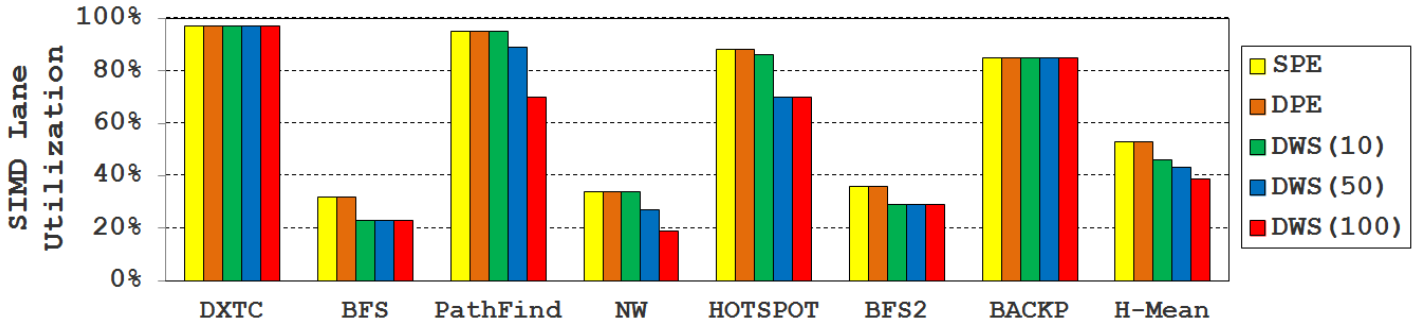
DPE: Same as SPE
DWS(100) : 51% reduction

DWS does not lose too much $SIMD_{util}$

(a) Interleavable benchmarks



(b) Non-interleavable benchmarks

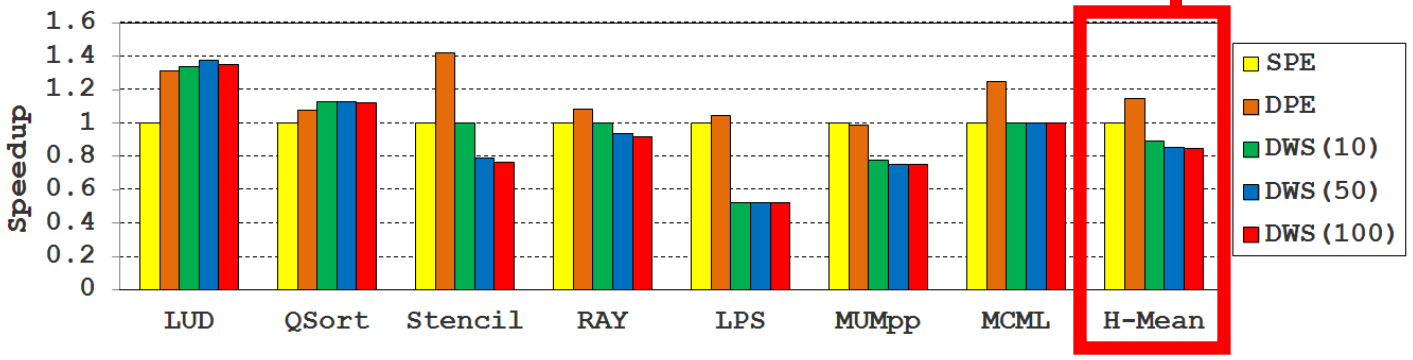


Speedup

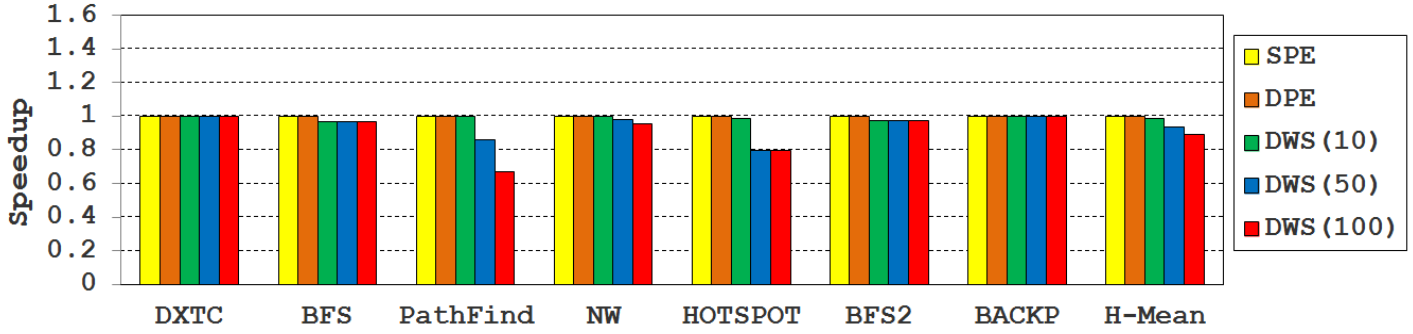
DPE: 14.9% increase (avg.)
DWS(100): 15.2% reduction (avg.)

- DPE == SPE for *non*-interleavable applications
- DPE is ***robust*** and outperforms other models when interleavable
  - An average 14.9% speedup over SPE (max 42%)

(a) Interleavable benchmarks



(b) Non-interleavable benchmarks



Conclusions

- DPE provides the best of both SPE and DWS
  - Always exploits path interleaving for taken/not-taken paths
  - Achieves the same optimal SIMT reconvergence for structured control flow
- Throughput improvements
  - **14.9**% (max 42%) improvements on top of SPE
- DPE model can be extended for further optimizations