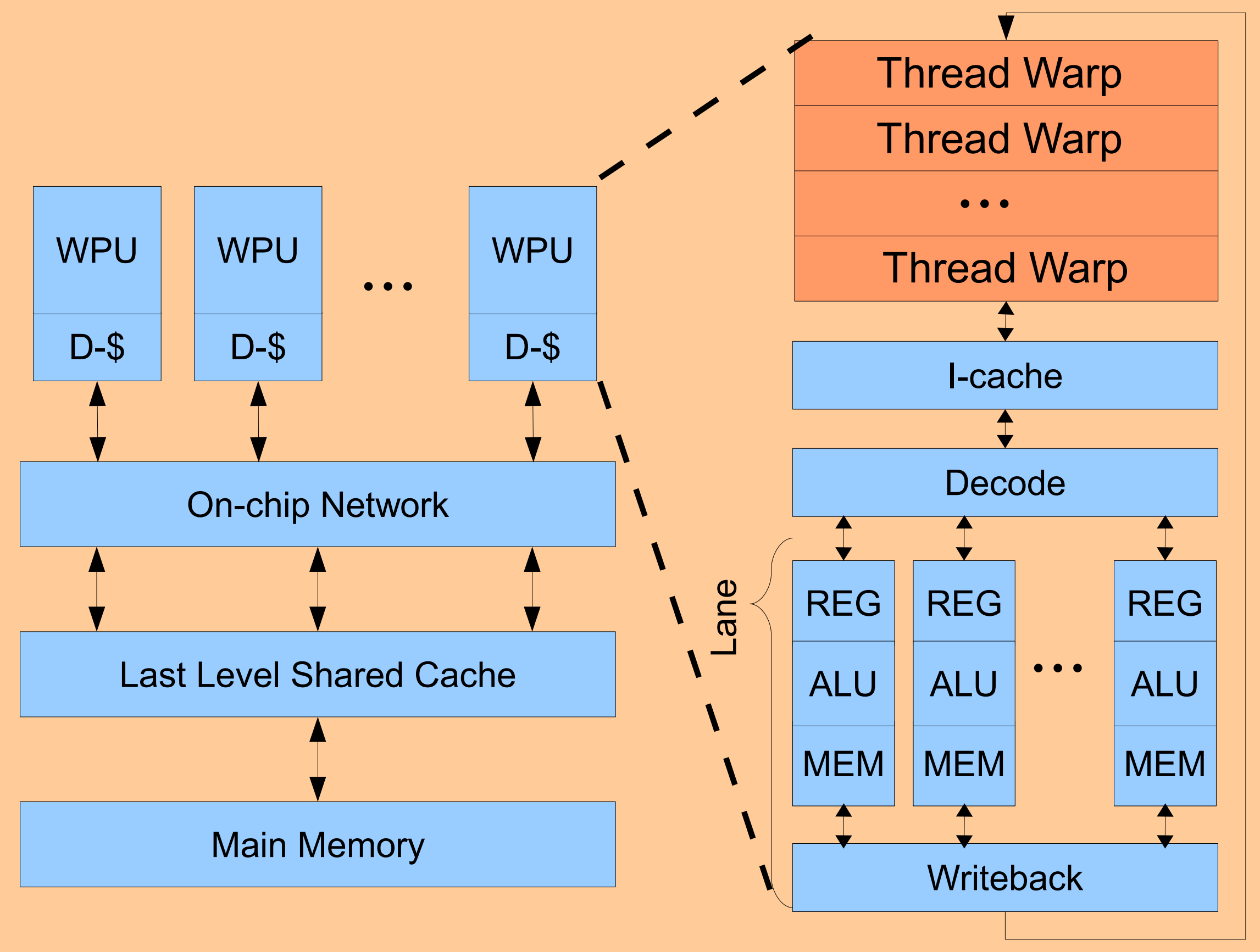
Dynamic Warp Subdivision for Non-Speculative Runahead SIMT Gather

Jiayuan Meng, Kevin Skadron
Department of Computer Science, University of Virginia

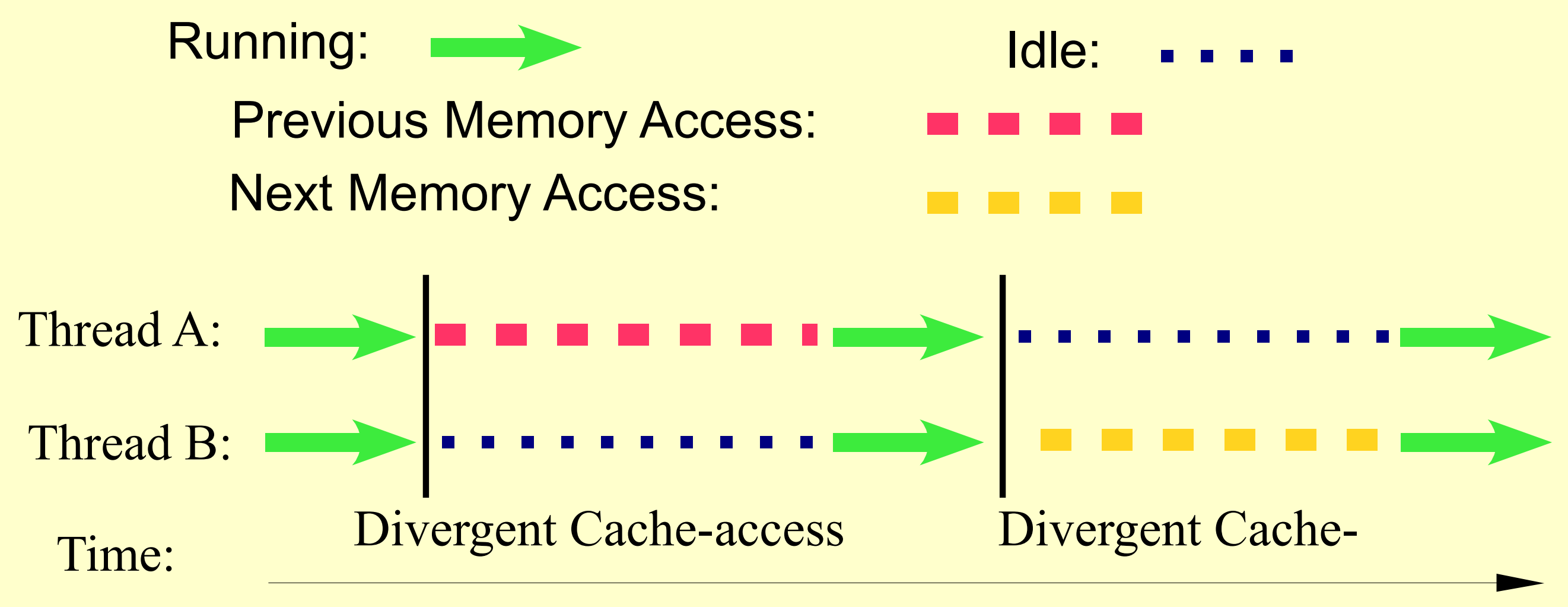
1. Background: SIMT Architecture

Scalar threads are grouped into warps that operate with a common instruction sequence in lockstep. (e.g. NVIDIA's Tesla architecture [1])

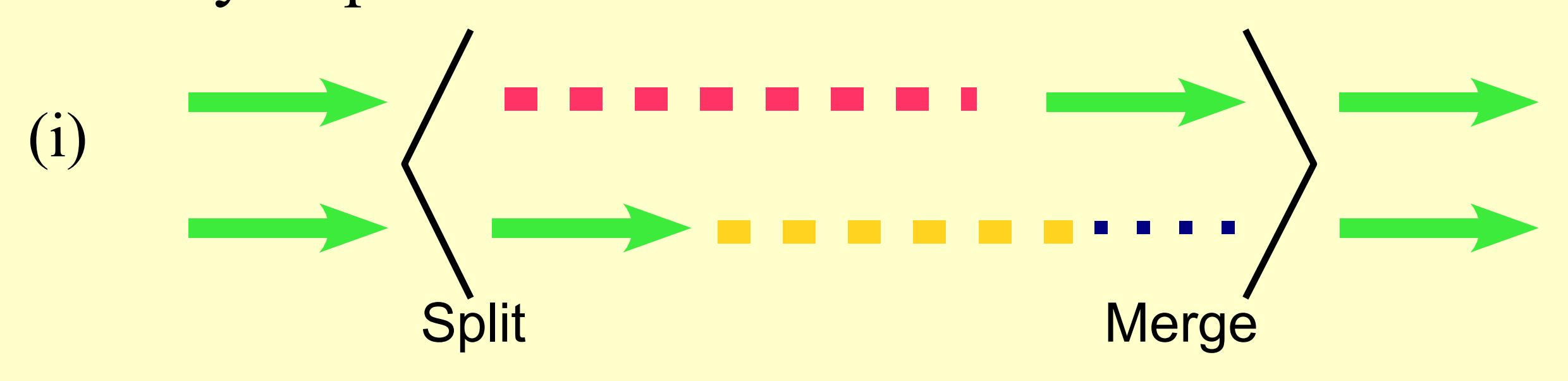


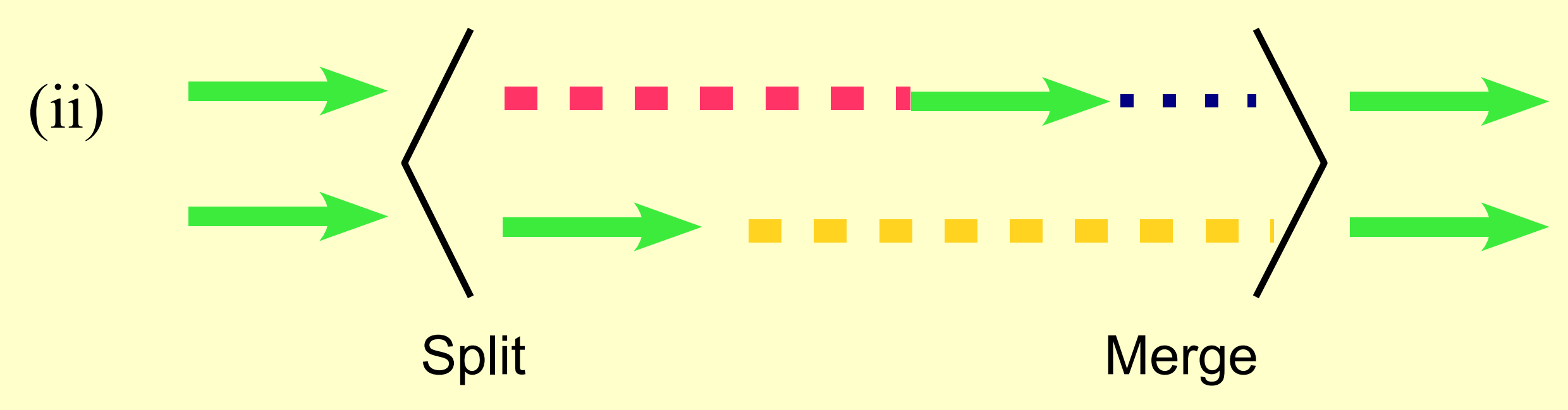
2. Motivation: Divergent Cache-accesses

- Cache miss caused by an individual thread suspends the entire warp.



- We subdivide a warp and allow threads that hit to proceed and issue more memory requests in parallel. As a result, threads that missed the cache previously may
  (i) not have to stall , or
  (ii) only have to stall for a much shorter period upon the next memory request.



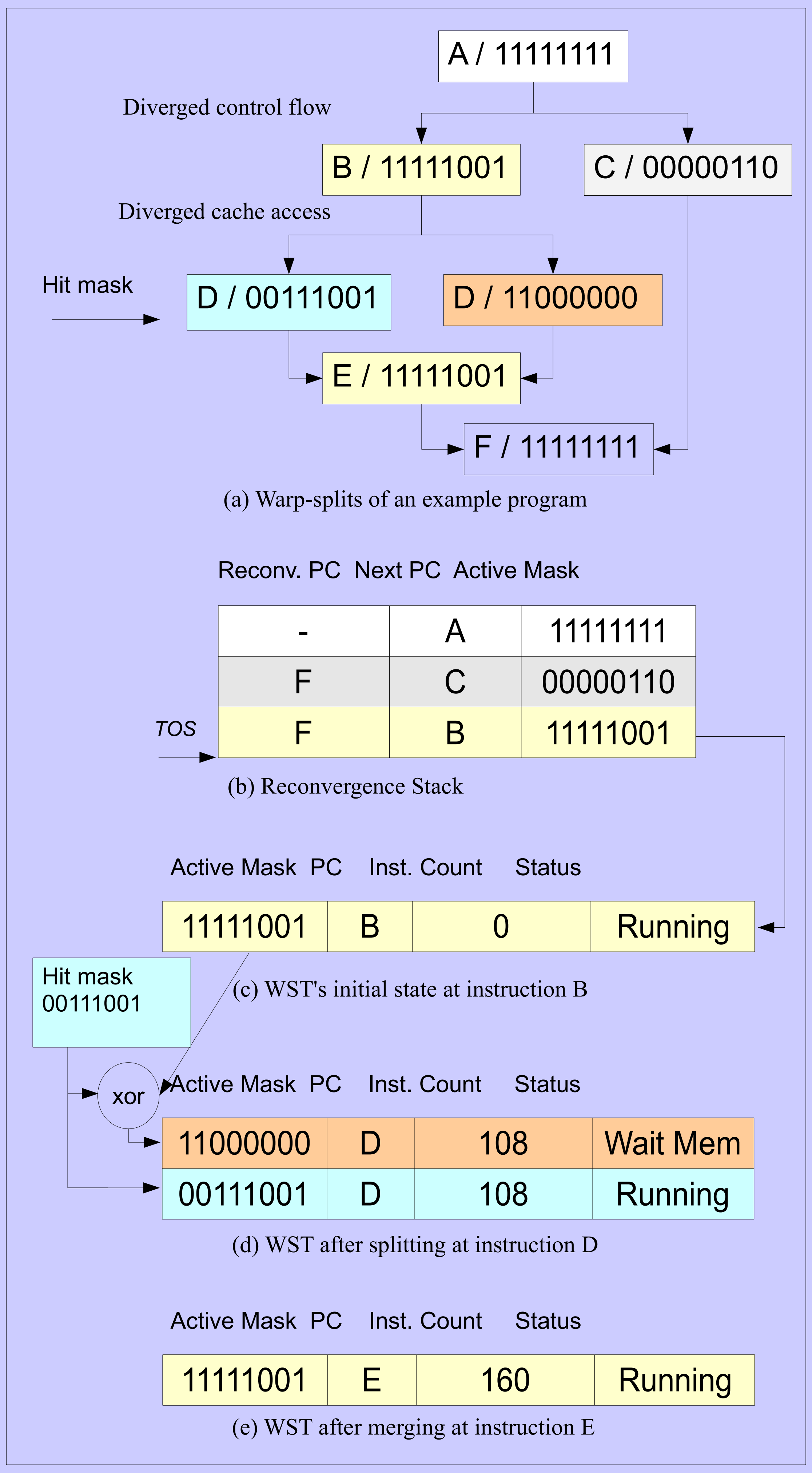


3. Challenges

- Compatibility with Branch Divergence Handling:
  - Upon conditional branches, the *reconvergence stack* [1, 3] subdivides a warp as well.
  - Predication is limited to non-nested branches and small branch sections. *Adaptive slip* proposed by Tarjan et al. [2] relies on aggressive predication.
- Pipeline Utilization: Aggressive subdivision leads to a large number of narrow warp-splits that only exploit a fraction of the SIMT pipelines.
- Latency Hiding: Warp-subdivision may not be necessary if other warps can hide memory latency sufficiently.

4 Implementation and Optimizations

- Combining Branch- and Memory- Divergence
  - Only manipulate active threads marked by the top of the reconvergence stack
  - Subdivided warp-splits are maintained in a warp-split table (WST)
  - Fall-behind warp-splits are resumed when their requests are fulfilled and merged into the run-ahead warp-splits when they catch up
- Latency Hiding and Pipeline Utilization:
  - *LazySplit* subdivides warps only when all the other warps are waiting for memory. Pipeline under-utilization may still occur.
  - *LatSpec (latency speculation)* dynamically speculates the remaining miss-free cycles (MFCs) of a warp to make a better decision upon divergent cache-accesses.
- Loop Bypassing
  - Allowing a run-ahead warp-split to continue across iteration boundaries to exploit more MLP.
  - Detecting loops using the reconvergence stack.
  - A generalization of loop slip [3]
- Warp Scheduling
  - SWF (shallowest warp first) policy first executes warp-splits that are likely to miss the cache in the near future.



(a) Warp-splits of an example program

Reconv. PC	Next PC	Active Mask
-	A	11111111
F	C	00000110
F	B	11111001

(b) Reconvergence Stack

Active Mask	PC	Inst. Count	Status
11111001	B	0	Running

(c) WST's initial state at instruction B

Active Mask	PC	Inst. Count	Status
11000000	D	108	Wait Mem
00111001	D	108	Running

(d) WST after splitting at instruction D

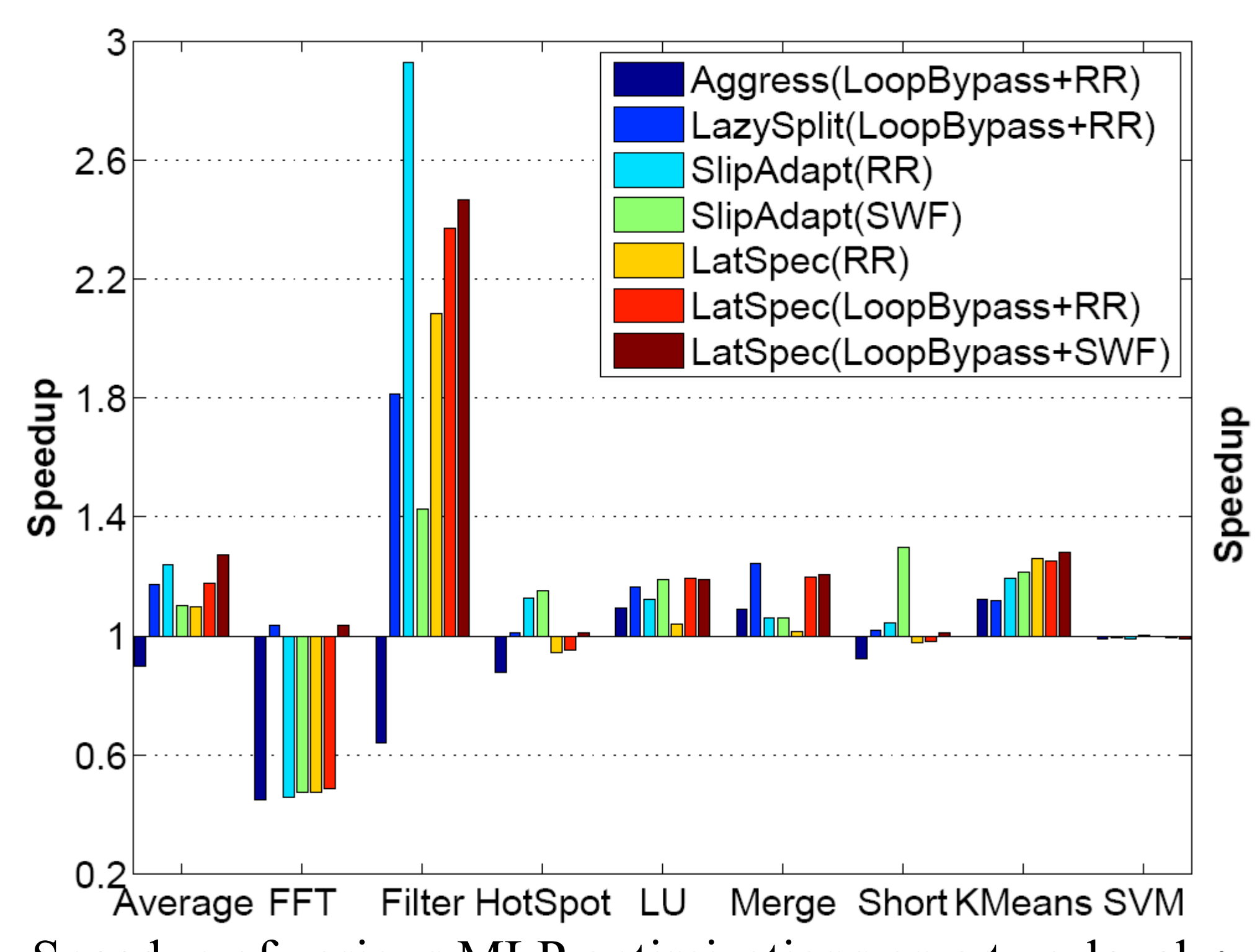
Active Mask	PC	Inst. Count	Status
11111001	E	160	Running

(e) WST after merging at instruction E

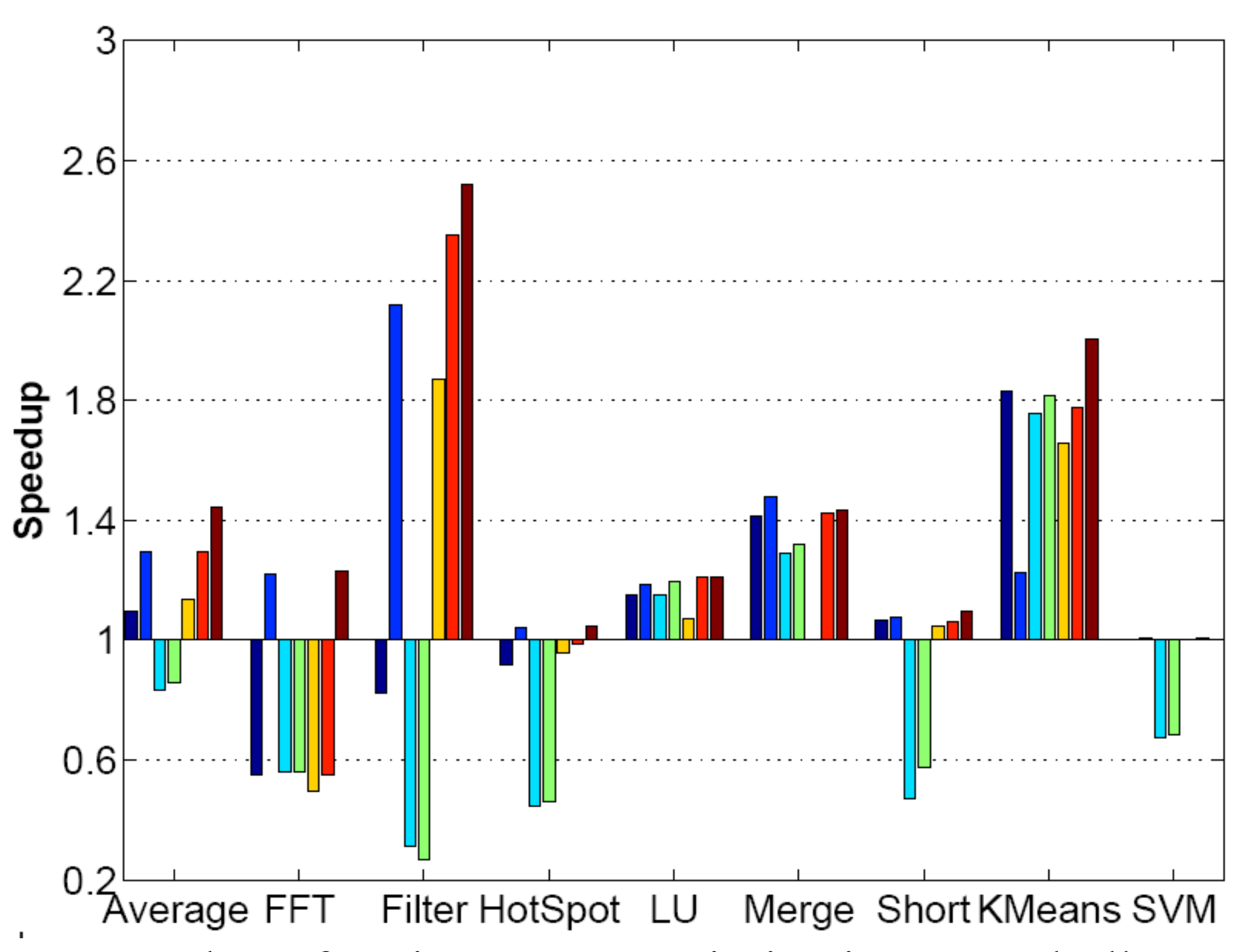
5 Results

Average speedup:

- 1.44X on the bulk-synchronous cache organization with a maximum speedup of 2.47X
- 1.28X on a coherent cache hierarchy with a maximum speedup of 2.53X
- Area overhead: < 2%.



Speedup of various MLP optimizations on a two-level coherent cache hierarchy.



Speedup of various MLP optimizations on a bulk-synchronous organization

References

[1] NVIDIA Corporation. Geforce GTX 280 specifications. 2008.

[2] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt. Dynamic warp formation and scheduling for efficient GPU control flow. In *MICRO '07*, pages 407-420, Washington, DC, USA.

[3] David Tarjan, Jiayuan Meng, and Kevin Skadron. Increasing memory miss tolerance for SIMD cores. To appear in SC '09

Acknowledgements

This work was supported in part by SRC grant No. 1607, NSF grant nos. IIS-0612049 and CNS-0615277, a grant from Intel Research, and a professor partnership award from NVIDIA Research.

LAVA Lab