Characterizing the Performance Bottlenecks of Irregular GPU Kernels

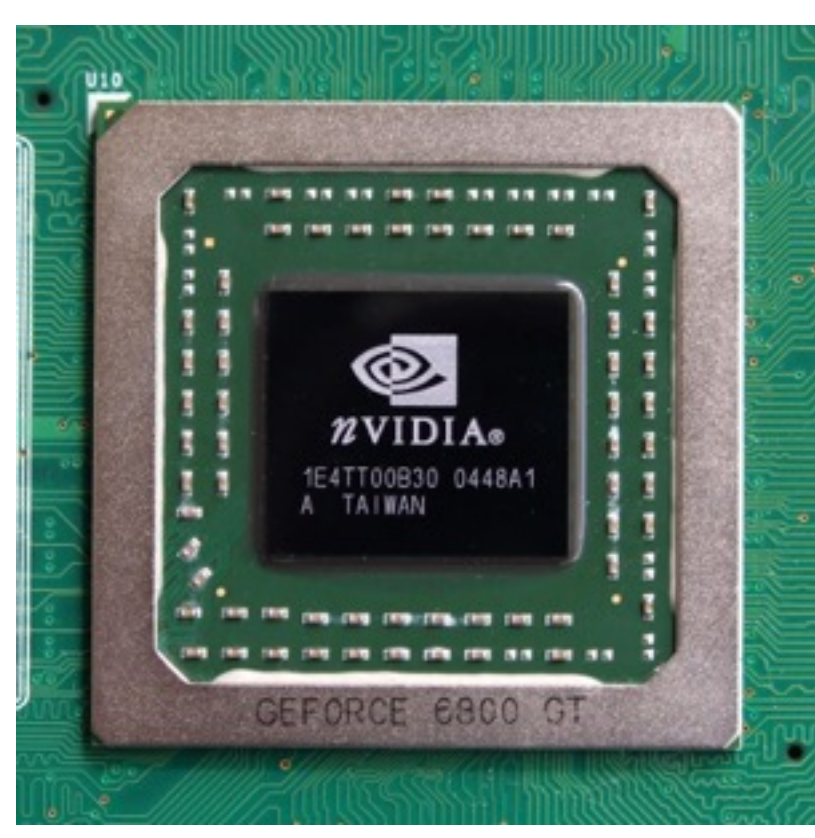
Molly A. O'Neil
M.S. Candidate, Computer Science

Thesis Defense

April 2, 2015

Committee Members:
Dr. Martin Burtscher
Dr. Apan Qasem
Dr. Dan Tamir

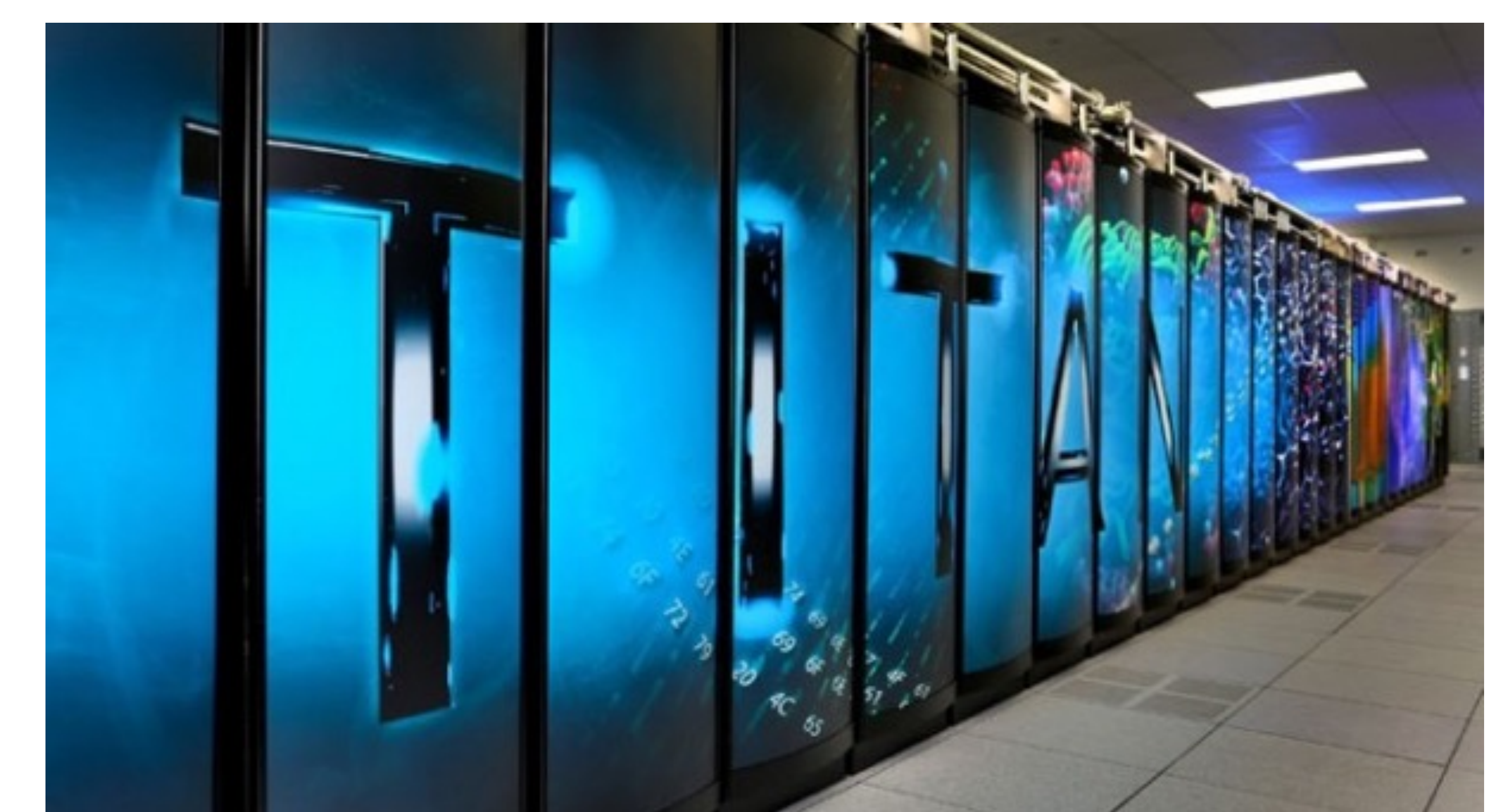
Highlights



- GPUs are everywhere — and really good at accelerating certain types of codes (regular, vector-based) in energy/cost-efficient manner
  - But lots of emerging, important codes are *irregular* in nature
  - Nobody knows yet how to efficiently run these codes on accelerators
- This thesis asks: *What are the biggest hurdles to enabling GPUs to efficiently accelerate these codes?*
  - Answer can help hardware designers broaden the acceleration capabilities of GPUs

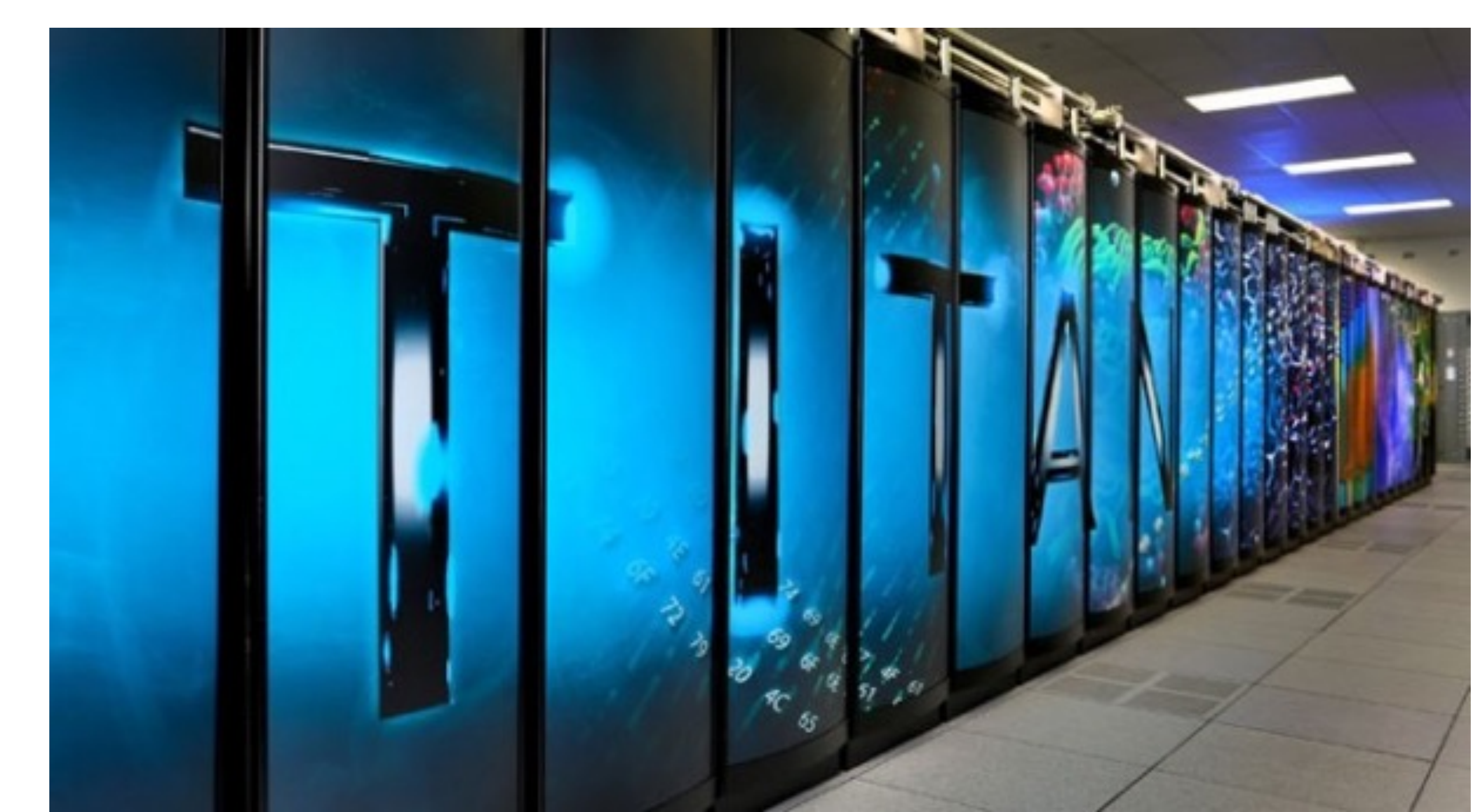
Background

- GPUs as general-purpose accelerators
  - *Ubiquitous* in HPC/supercomputers
  - Spreading in PCs and mobile devices
  - Performance and energy-efficiency benefits…



[ORNL Titan]

Background



[ORNL Titan]

- GPUs as general-purpose accelerators
  - *Ubiquitous* in HPC/supercomputers
  - Spreading in PCs and mobile devices
  - Performance and energy-efficiency benefits…
- …when code is well-suited!
  - *Regular* (input independent) vs. *irregular* (input determines control flow and memory accesses)
  - Lots of important irregular algorithms
    - Social networks, compilers, data mining, physics simulation, etc.
    - More difficult to parallelize, map less intuitively to GPUs

Motivation

[NVIDIA]

- GPUs likely to continue to grow in importance
- Need to better understand irregular applications' specific demands on GPU hardware
  - How they differ from those of regular codes
- Identify most significant architectural limitations for irregular GPU kernels
  - To help software developers *better optimize irregular codes*
  - As a baseline for *exploring hardware support* for broader classes of general-purpose codes

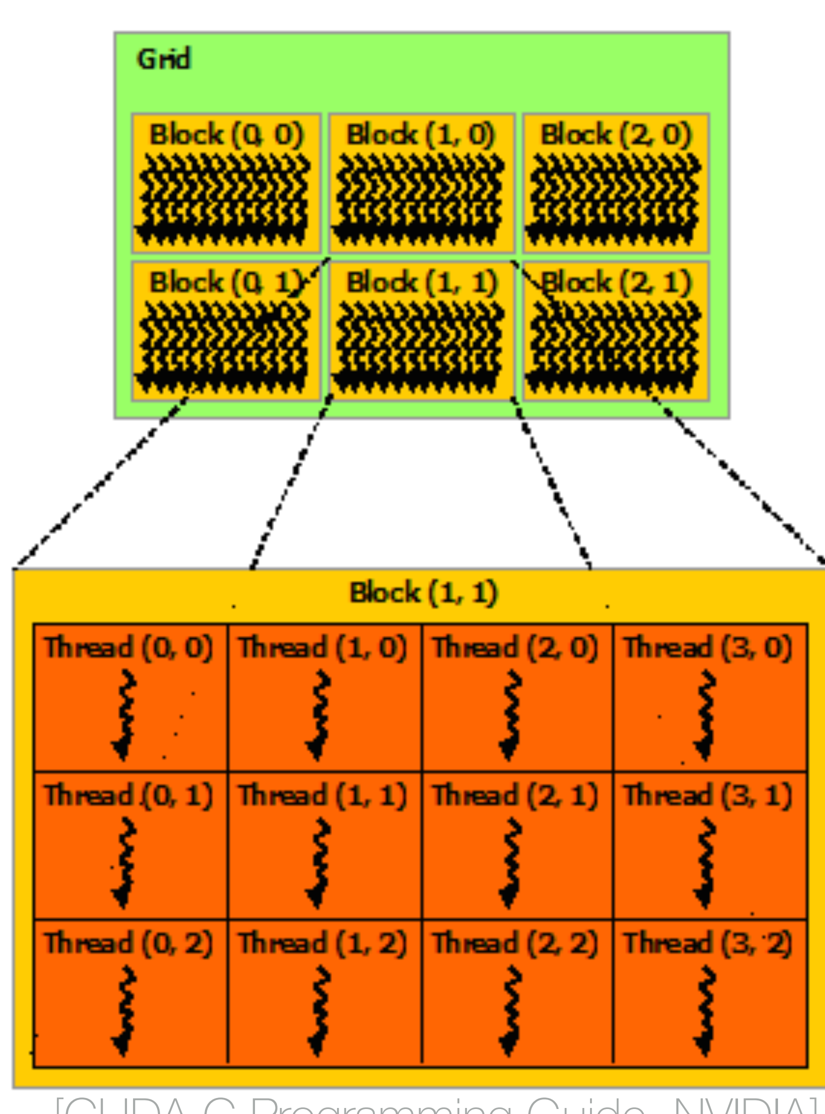
Related Literature

- Simulator-based characterization studies
  - Bakhoda et al. (ISPASS'09), Goswami et al. (IISWC'10), Che et al. (IISWC'10), Blem et al. (EAMA'11), Lee and Wu (ISPASS'14)
    - CUDA SDK, Rodinia, Parboil (no focus on irregularity)
  - IISWC'14: *O'Neil and Burtscher* [1] [LonestarGPU]; Xu et al. [graph codes]
- Emulator studies (also SDK, Rodinia, Parboil)
  - Kerr et al. (IISWC'09), Wu et al. (CACHES'11)
- Hardware performance counters
  - Burtscher et al. (IISWC'12) [LonestarGPU], Che et al. (IISWC'13)

[1] O'Neil and Burtscher, "Microarchitectural Performance Characterization of Irregular GPU Kernels," IISWC 2014.

GPU Review

CUDA GPUs

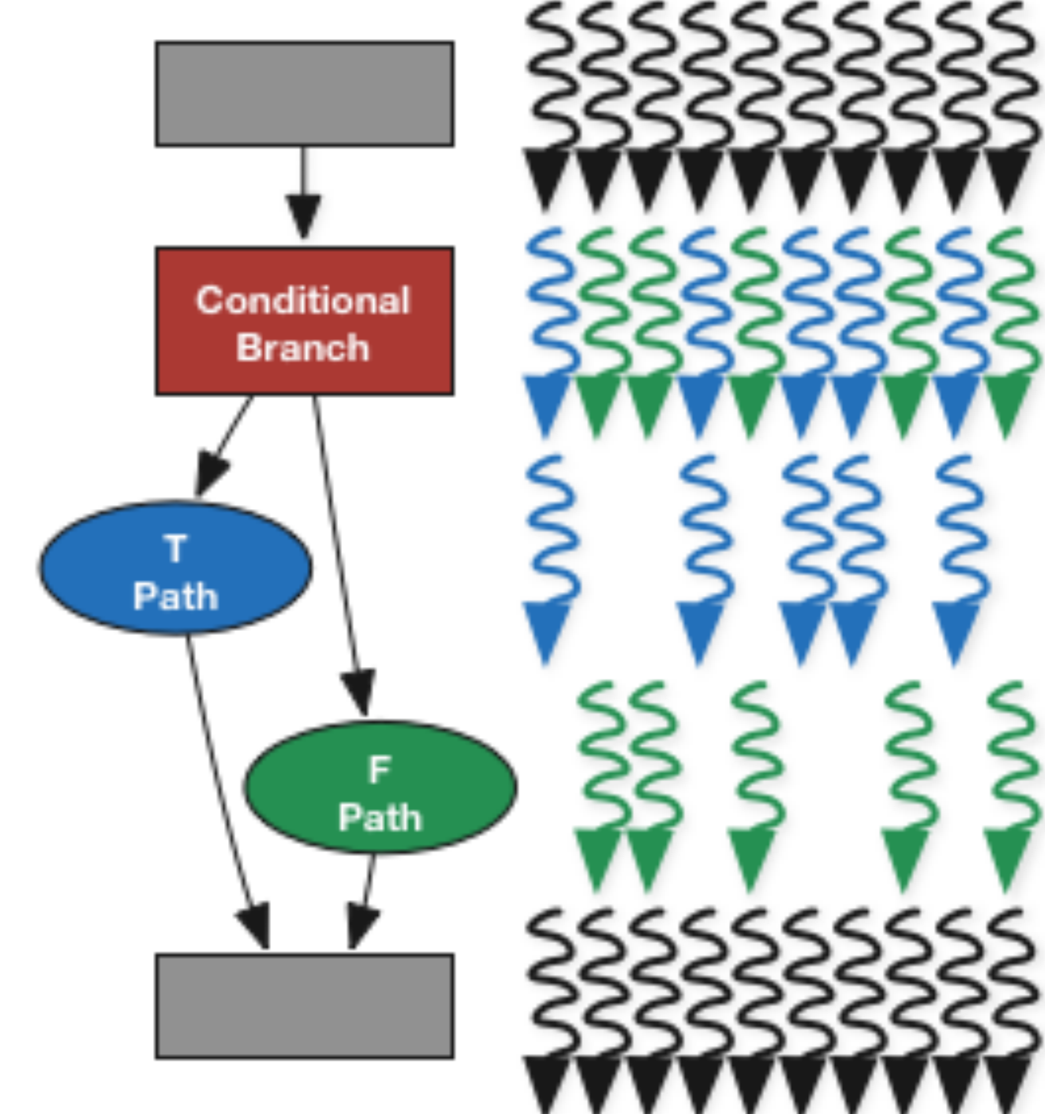


[CUDA C Programming Guide, NVIDIA]

- Two-level compute hierarchy
  - *Streaming multiprocessors (SMs)* each composed of tightly-coupled *processing elements (PEs)*
- CUDA program specifies the behavior of a *kernel grid*, the threads of which are grouped into *thread blocks* and dynamically assigned to SMs
  - Threads within a block share on-chip cache and fast synchronization
- PEs execute *warps* (sets of 32 adjacent threads that execute as a vector instruction operating conditionally on 32 elements)
  - PEs fed with warps in multithreading style, interleaving between blocks

Branch Divergence

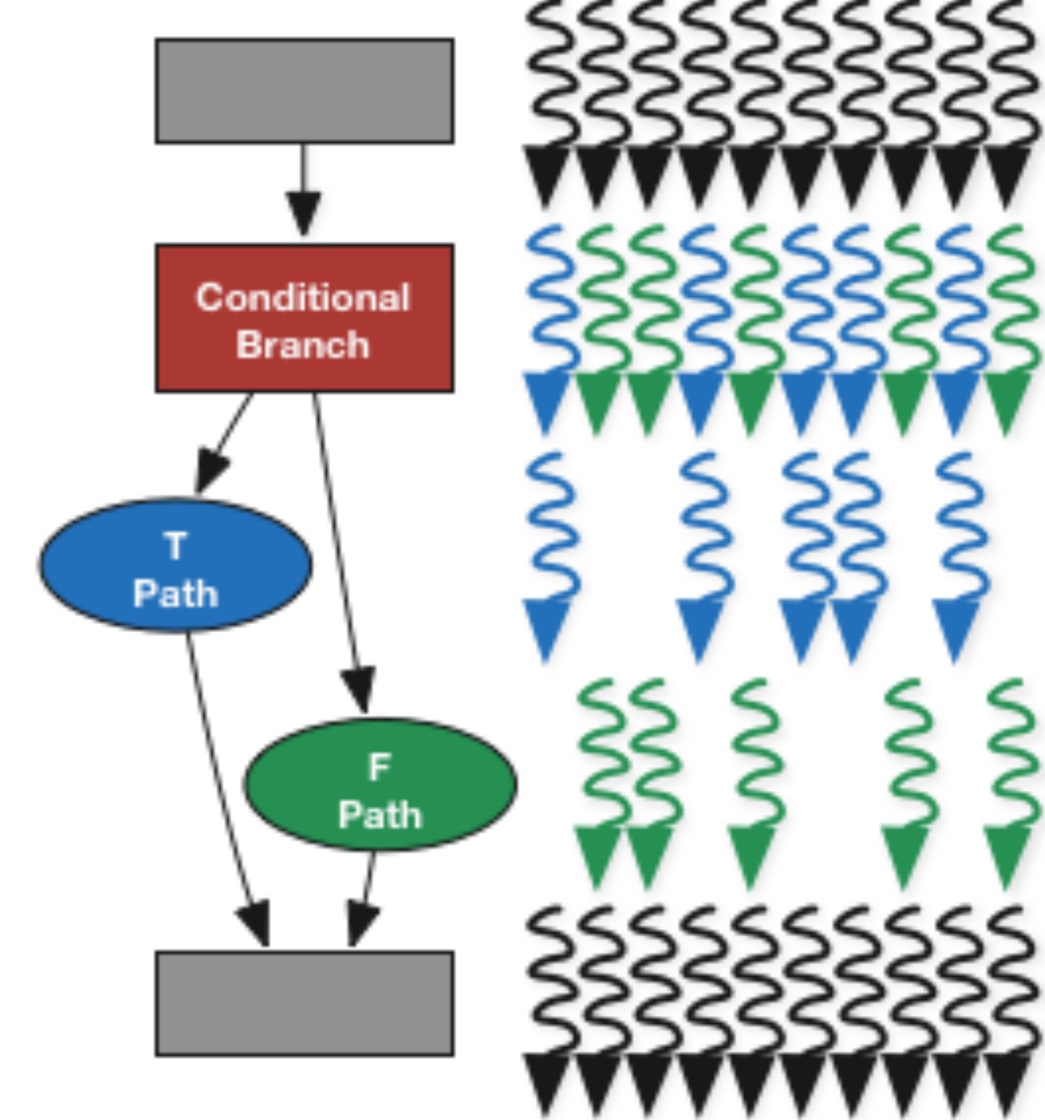
- To execute in parallel, threads in a warp must share *identical control flow*
  - If not, execution serialized by hardware into smaller groups of threads such that all threads in subset execute the same instruction



- Good performance requires minimal *branch divergence*

Branch Divergence

- To execute in parallel, threads in a warp must share *identical control flow*
    - If not, execution serialized by hardware into smaller groups of threads such that all threads in subset execute the same instruction

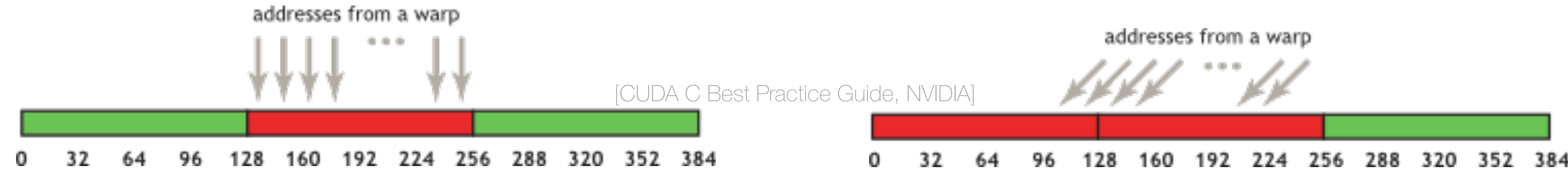


- Good performance requires minimal *branch divergence*

Irregular control flow makes divergence difficult to avoid!

Memory Coalescing

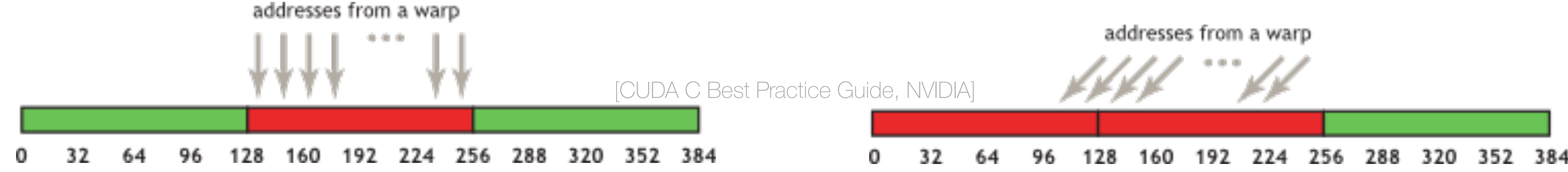
- For good performance, memory accesses within a warp must be *coalesced* (fall within the same cache line)
  - If a warp instruction touches multiple 128-byte segments, accesses to additional lines are serialized
  - Possible for single warp instruction to result in 32 separate transactions



[CUDA C Best Practice Guide, NVIDIA]

Memory Coalescing

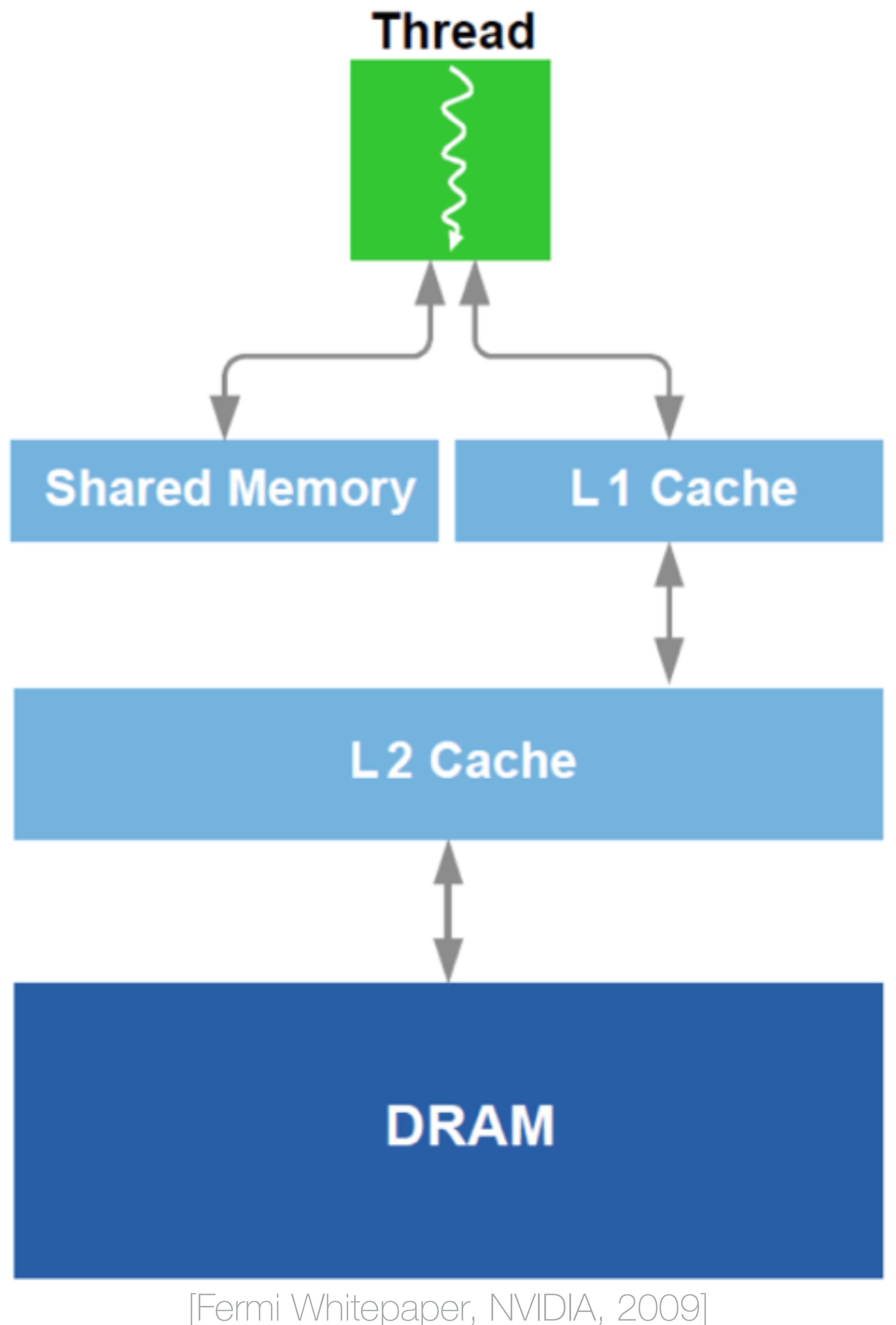
- For good performance, memory accesses within a warp must be *coalesced* (fall within the same cache line)
  - If a warp instruction touches multiple 128-byte segments, accesses to additional lines are serialized
  - Possible for single warp instruction to result in 32 separate transactions



Irregular access patterns make coalescing difficult to achieve!

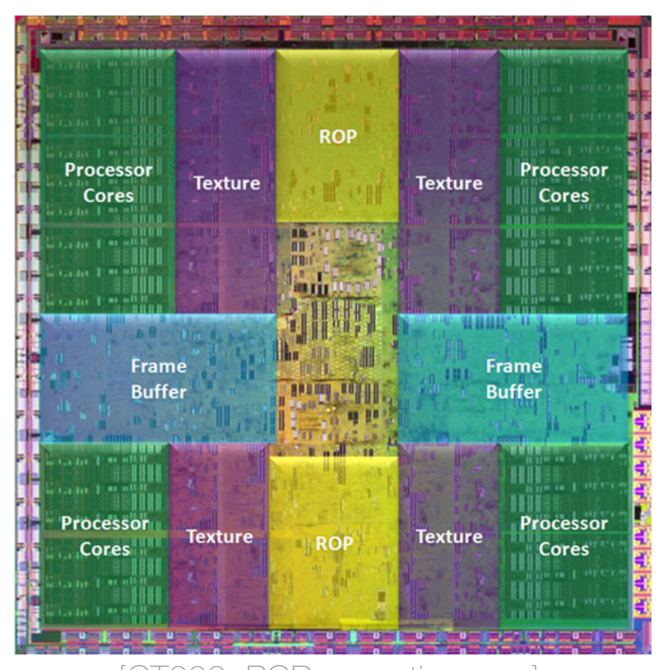
Cache & Memory Hierarchy

- All SMs share global memory (DRAM) as well as a unified L2 cache (GTX 480: 768 kB)
- Each SM has a programmer-controlled *shared memory* (16 kB - 48 kB)
    - Shared between blocks resident on SM
- Each Fermi SM has incoherent L1 data cache (16 kB - 48 kB)



[Fermi Whitepaper, NVIDIA, 2009]

Methodology



[GT200, PCPerspective.com]

Problem Statement

- For a set of irregular and regular applications, understand the impact of control-flow and memory-access irregularity on…
  - Branch divergence
  - Memory coalescing
  - Cache effectiveness

Problem Statement

- For a set of irregular and regular applications, understand the impact of control-flow and memory-access irregularity on…
  - Branch divergence
  - Memory coalescing
  - Cache effectiveness

Problem Statement

- For a set of irregular and regular applications, understand the impact of control-flow and memory-access irregularity on…
  - Branch divergence
  - Memory coalescing
  - Cache effectiveness

Code behavior

Hardware performance

Problem Statement

- For a set of irregular and regular applications, understand the impact of control-flow and memory-access irregularity on…

  - Branch divergence
  - Memory coalescing
  - Cache effectiveness

Hardware performance

- Assess the sensitivity of these applications to hardware design parameters such as…
  - Cache and memory latency
  - Cache and memory bandwidth
  - Cache size
  - Coalescing behavior
  - Warp scheduling policy

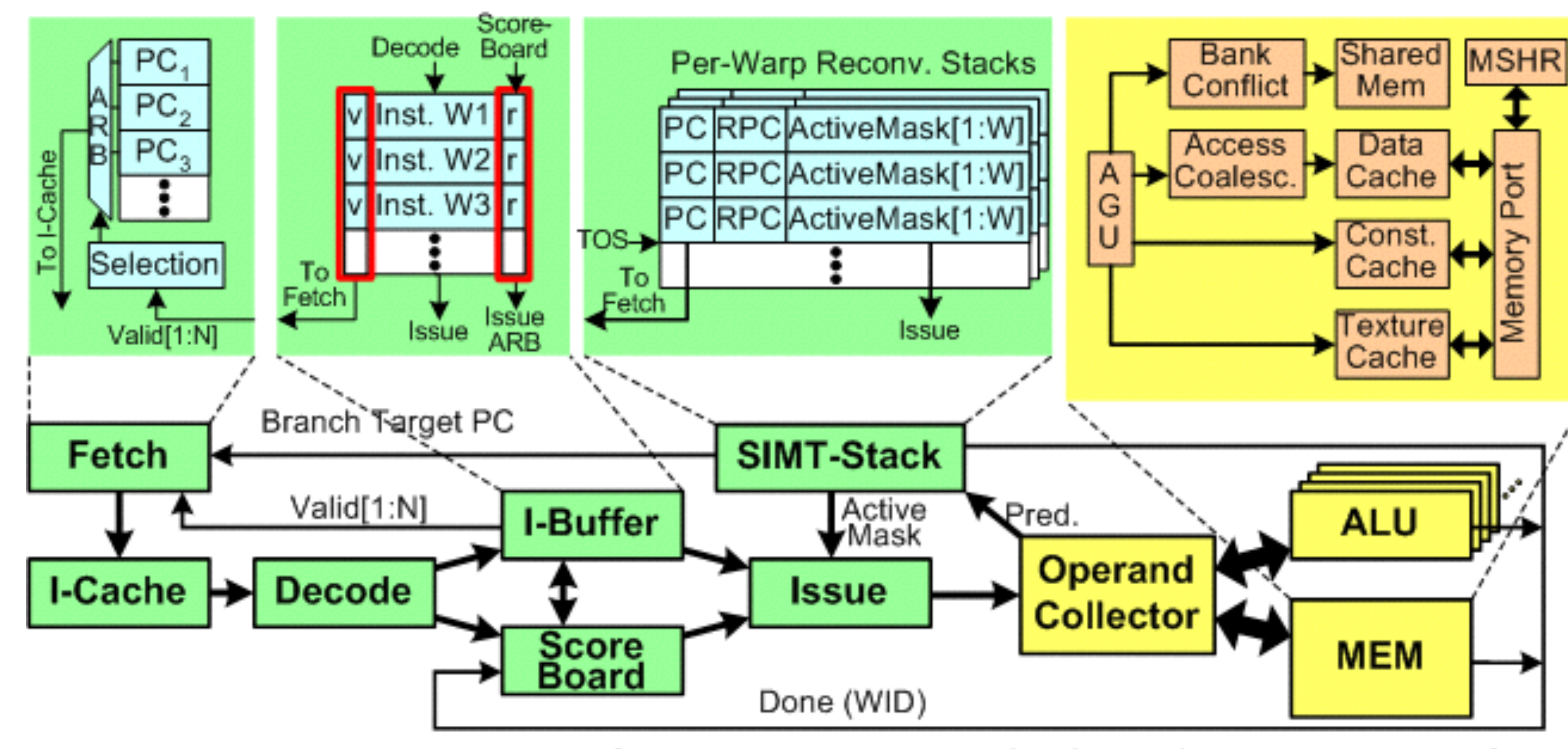
Hardware parameters critical to the performance factors above

Objective

- **My goal**: From patterns of behavior in studied benchmarks, abstract an *understanding of the impact of irregular code* (data structures, algorithms, implementation choices) on hardware performance
- **NOT my goal**: Determine the best particular configuration of hardware parameters for this particular set of codes and GPU device
  - Why not?
    - No claim of completeness in benchmark suite
    - GPU microarchitecture (and macro-architecture!) still in flux
- I want to identify the *major bottlenecks in hardware* where architects should be focusing their attention
  - Versus the sources of performance loss that programmers can address on their own

GPGPU-Sim

- Cycle-level microarchitectural simulator of a CUDA GPU
  - Functional PTX simulator (NVIDIA’s virtual ISA)
  - Timing model for the SMs, caches, shared memory, interconnect network, memory partitions (including L2 cache), and off-chip DRAM

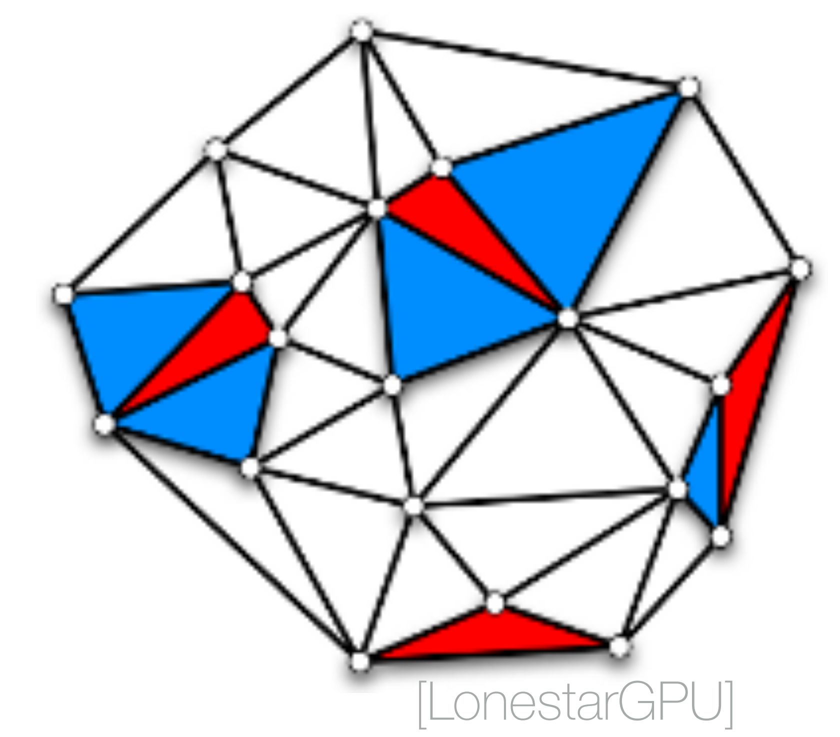


[Aamodt and Fung, GPGPU-Sim v3.x Manual]

- GPGPU-Sim v. 3.2.1
  - GTX 480 (Fermi) configuration
  - Plus bug fixes, additional operations, extra performance counters, and new hardware configuration options

Applications

Irregular Applications (LonestarGPU)


[LonestarGPU]

- Labels each node in graph with minimum level from start node
- *BFS*: Topology-driven
- *BFS-unroll*: Multiple frontiers per iteration w/ local worklist
- *BFS-wlw*: Data-driven, node per thread
- *BFS-wlc*: Data-driven, edge per thread (Merrill et al., PPoPP'12)

- **Barnes-Hut (BH)**
  - Approximate N-body algorithm using octree to decompose space around bodies
- **Mesh Refinement (DMR)**
  - Iteratively transforms 'bad' triangles by retriangulating surrounding cavity

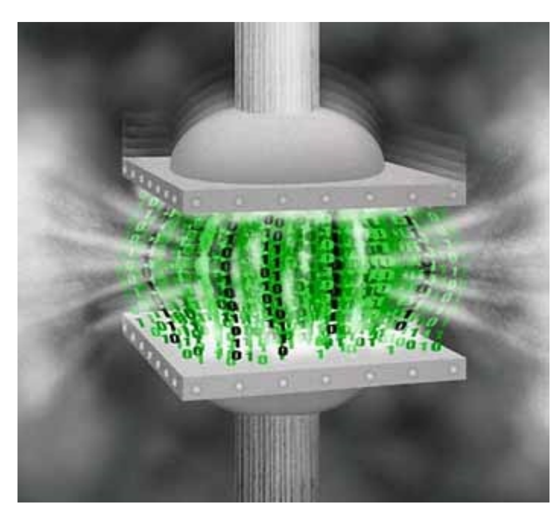
Irregular Applications (LonestarGPU)

[StackExchange.com]

- **Minimum Spanning Tree (MST)**
  - Applying Boruvka's algorithm, successively contracts minimum weight edge until single node
- **Survey Propagation (SP)**
  - Heuristic SAT-solver based on Bayesian inference, represents Boolean formula as bipartite graph of variables and clauses
- **Single-Source Shortest Paths (SSSP)**
  - Labels each node in graph with minimum level from start node
  - *SSSP*: Topology-driven
  - *SSSP-wln*: Data-driven, node per thread
  - *SSSP-wlc*: Data-driven, edge per thread

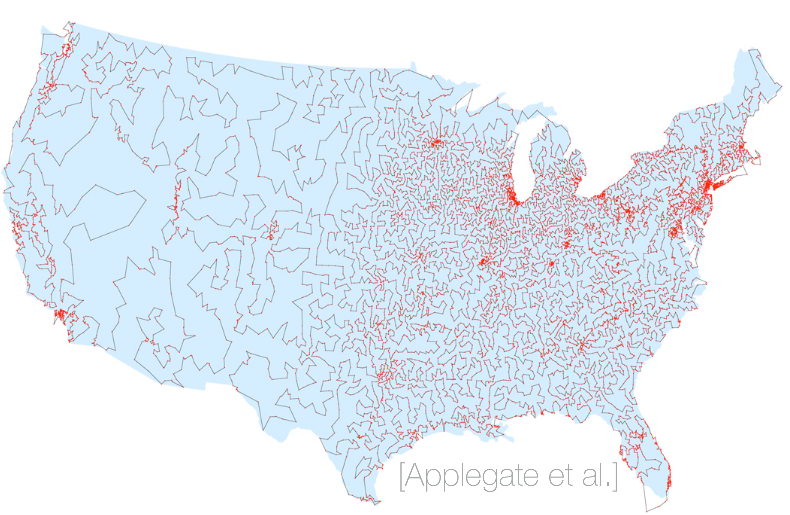
Semi-Regular Applications

- **FP Compression (FPC)** [1]
  - Lossless data compression for DP floating-point values
  - Irregular control flow



[AnalyticBridge.com]

- **Traveling Salesman (TSP)** [2,3]
  - Find minimal tour in graph using iterative hill climbing
  - Irregular memory accesses



[Applegate et al.]

[1] O'Neil and Burtscher, "Floating-Point Data Compression at 75 Gb/s on a GPU," GPGPU 2011.

[2] O'Neil, Tamir, and Burtscher, "A Parallel GPU Version of the Traveling Salesman Problem," PDPTA 2011.

[3] O'Neil and Burtscher, "Rethinking the Parallelization of Random-Restart Hill Climbing," GPGPU 2015.

Regular Applications

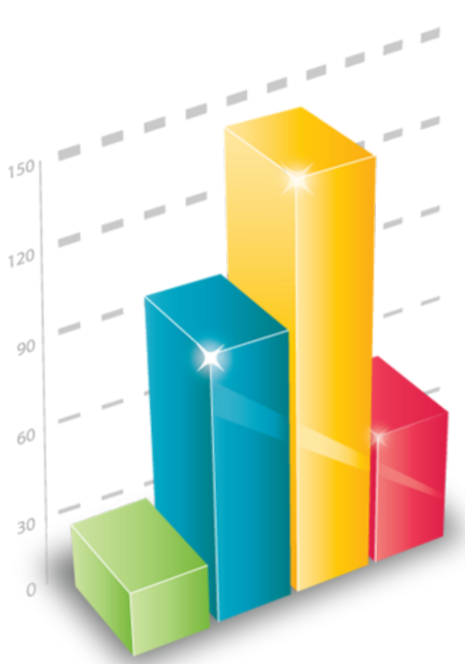
- **Monte Carlo (MC)**
  - Evaluates fair call price for set of options using Monte Carlo method
  - CUDA SDK version
- **N-Body (NB)**
  - N-body algorithm using all-to-all force calculation
  - Texas State ECL version (outperforms SDK version)

Regular Applications

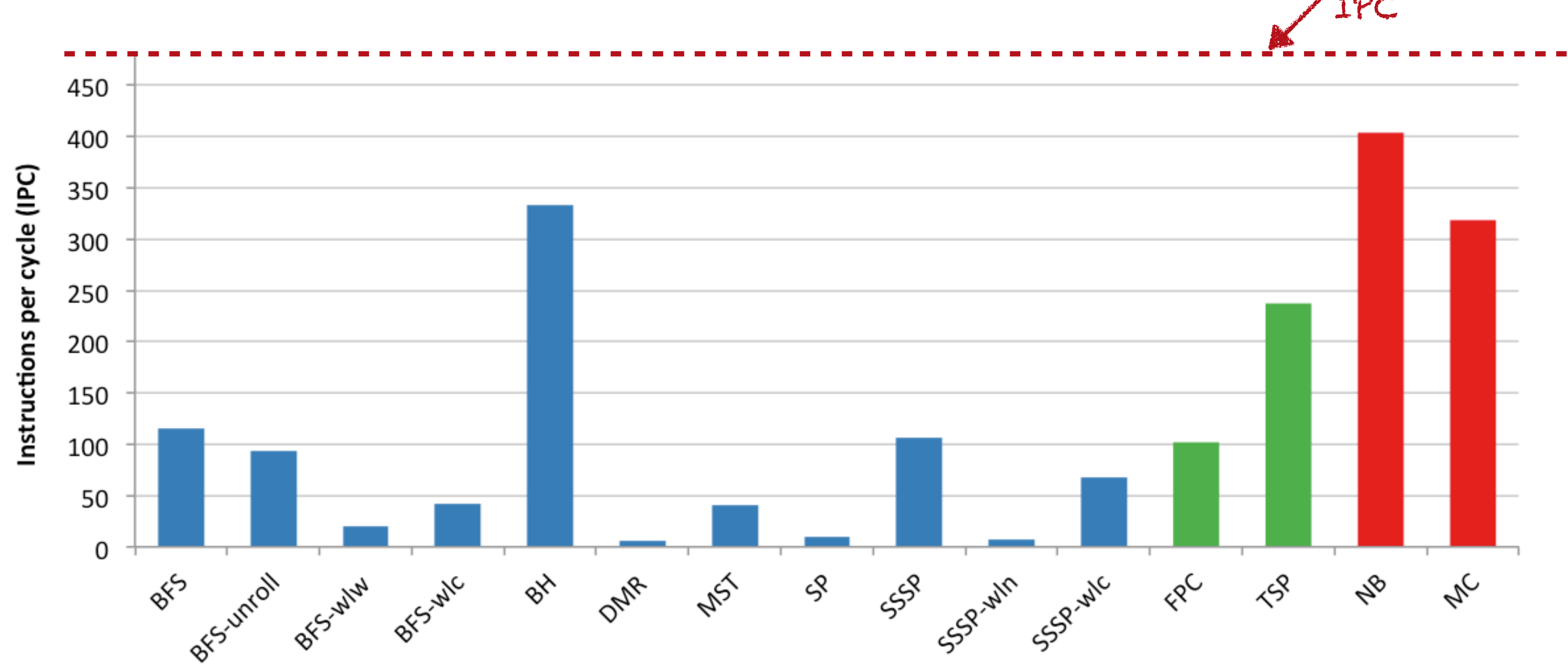
- **Monte Carlo (MC)**
  - Evaluates fair call price for set of options using Monte Carlo method
  - CUDA SDK version
- **N-Body (NB)**
  - N-body algorithm using all-to-all force calculation
  - Texas State ECL version (outperforms SDK version)

- Input for each benchmark:
  - Small enough to result in reasonable simulation runtimes (<2 weeks) but large enough to keep simulated hardware busy
  - Where possible, working set ≥ 5 times the default L2 cache size

Results & Analysis

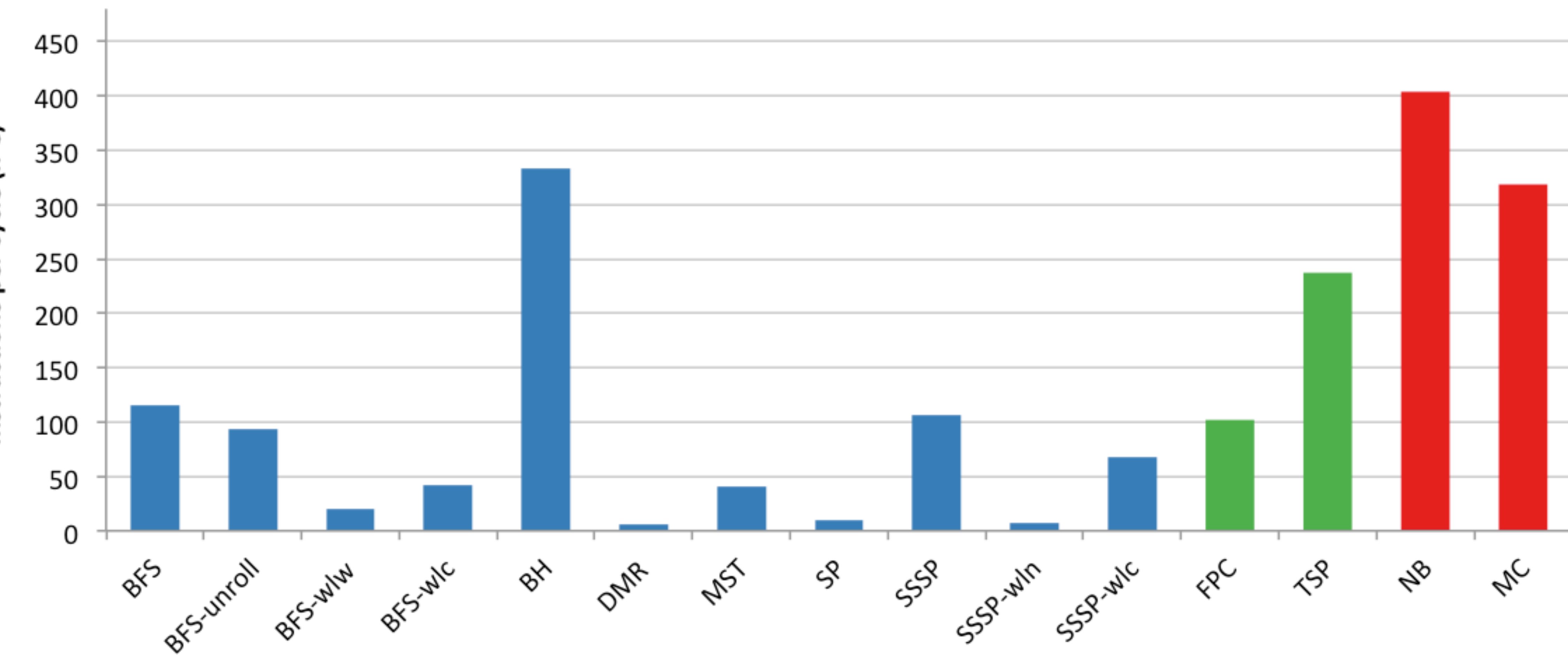


Application Performance



Peak: 480 IPC

Application Performance



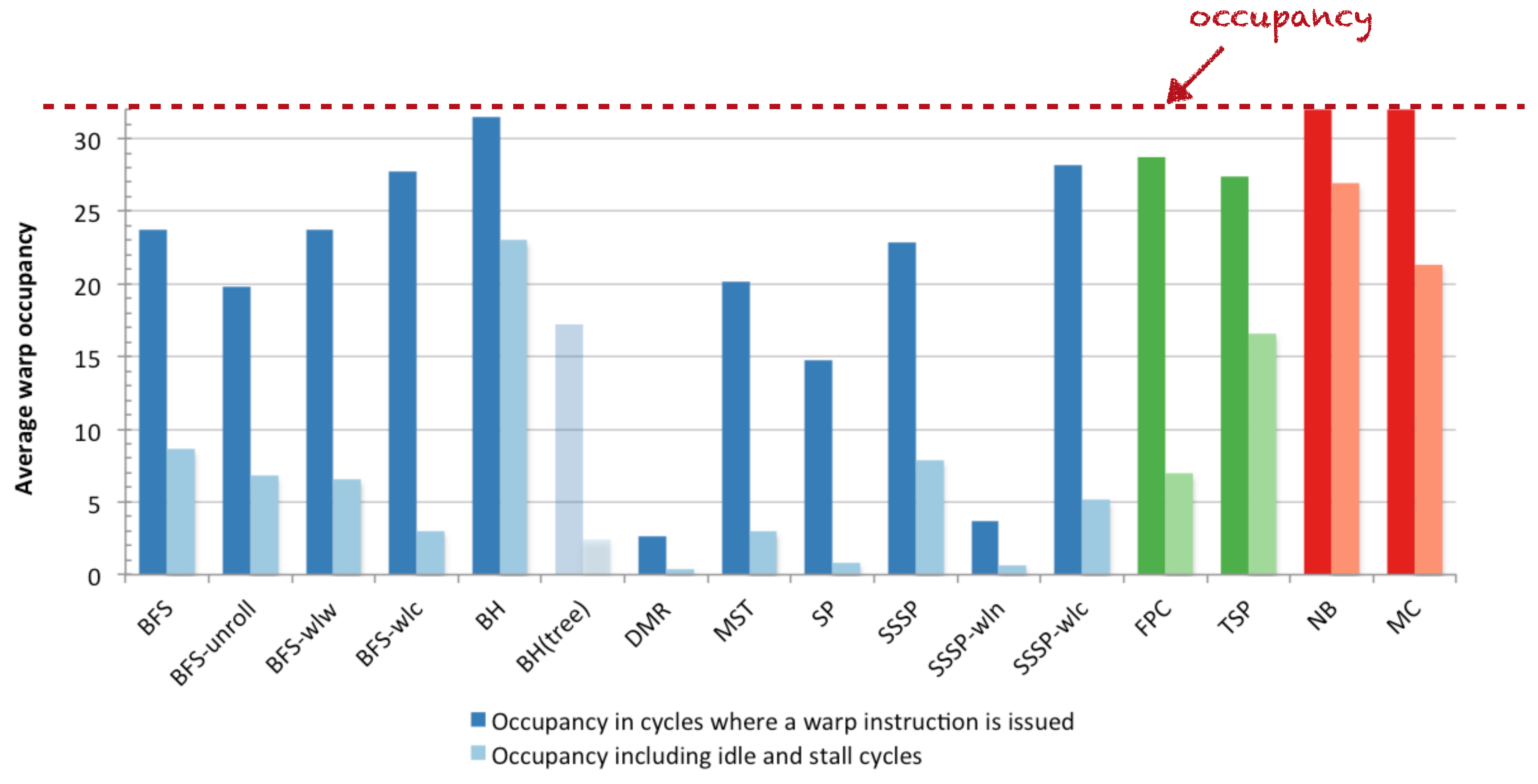
- *Strong correlation* between regularity of code and IPC
- BH is an exception (runtime-dominating kernel has been regularized)
- *No simple or fixed rule* to delineate the performance of irregular and regular codes

Sources of Performance Limitation

Sources of Performance Limitation

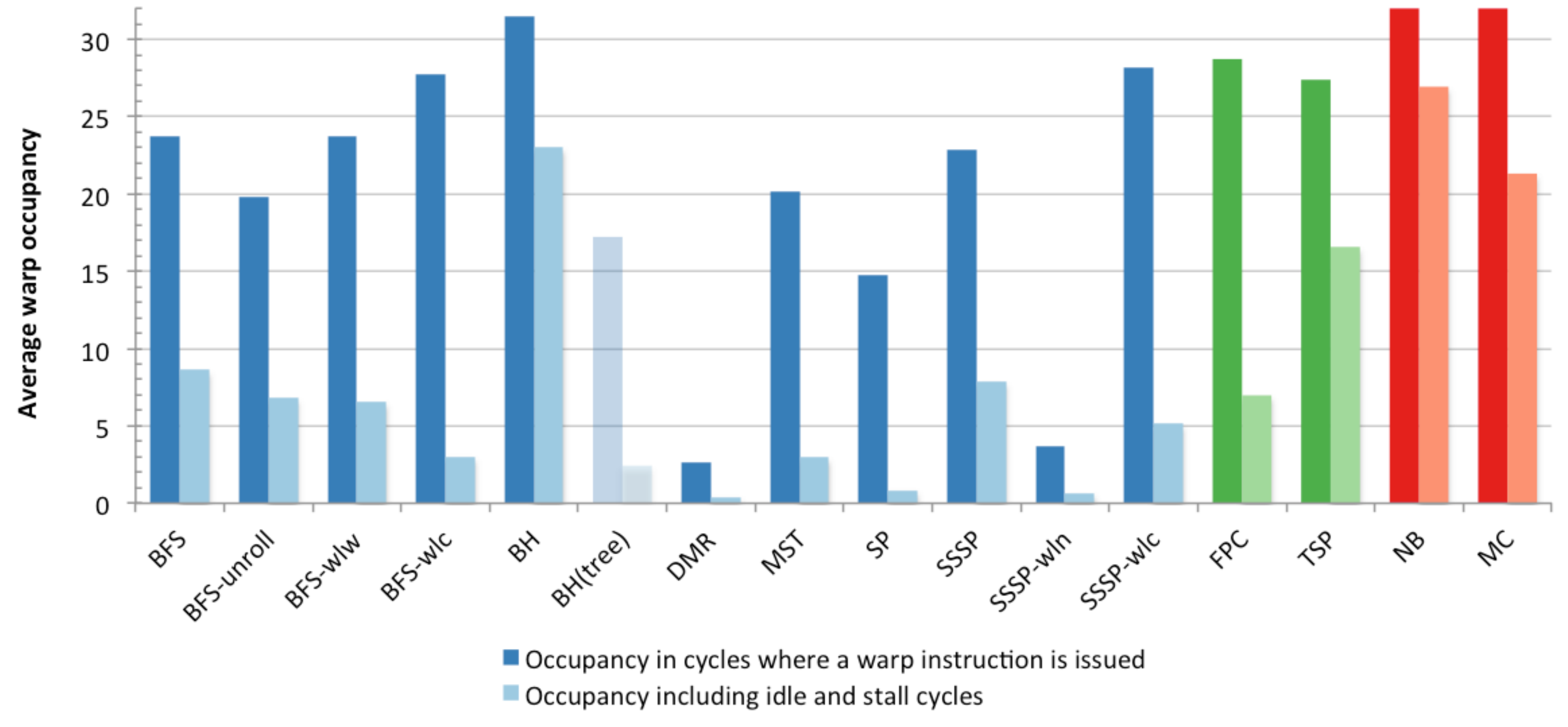
Divergence + Un-Coalesing are not factors we have to consider when writing parallel CPU code!

Branch Divergence

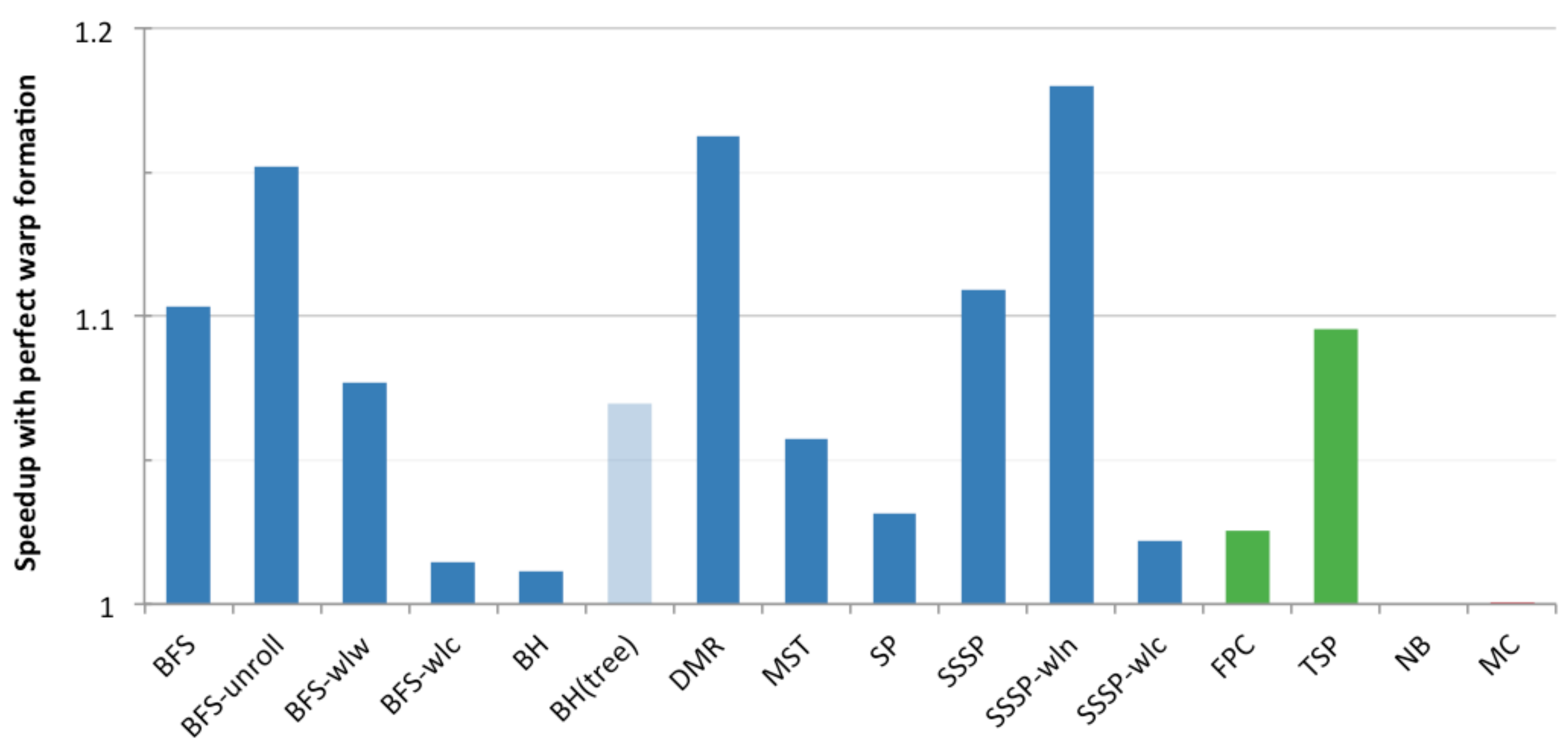


Branch Divergence

- Irregular codes more diverged
  - But only two <~50% occupied
  - BH an outlier again

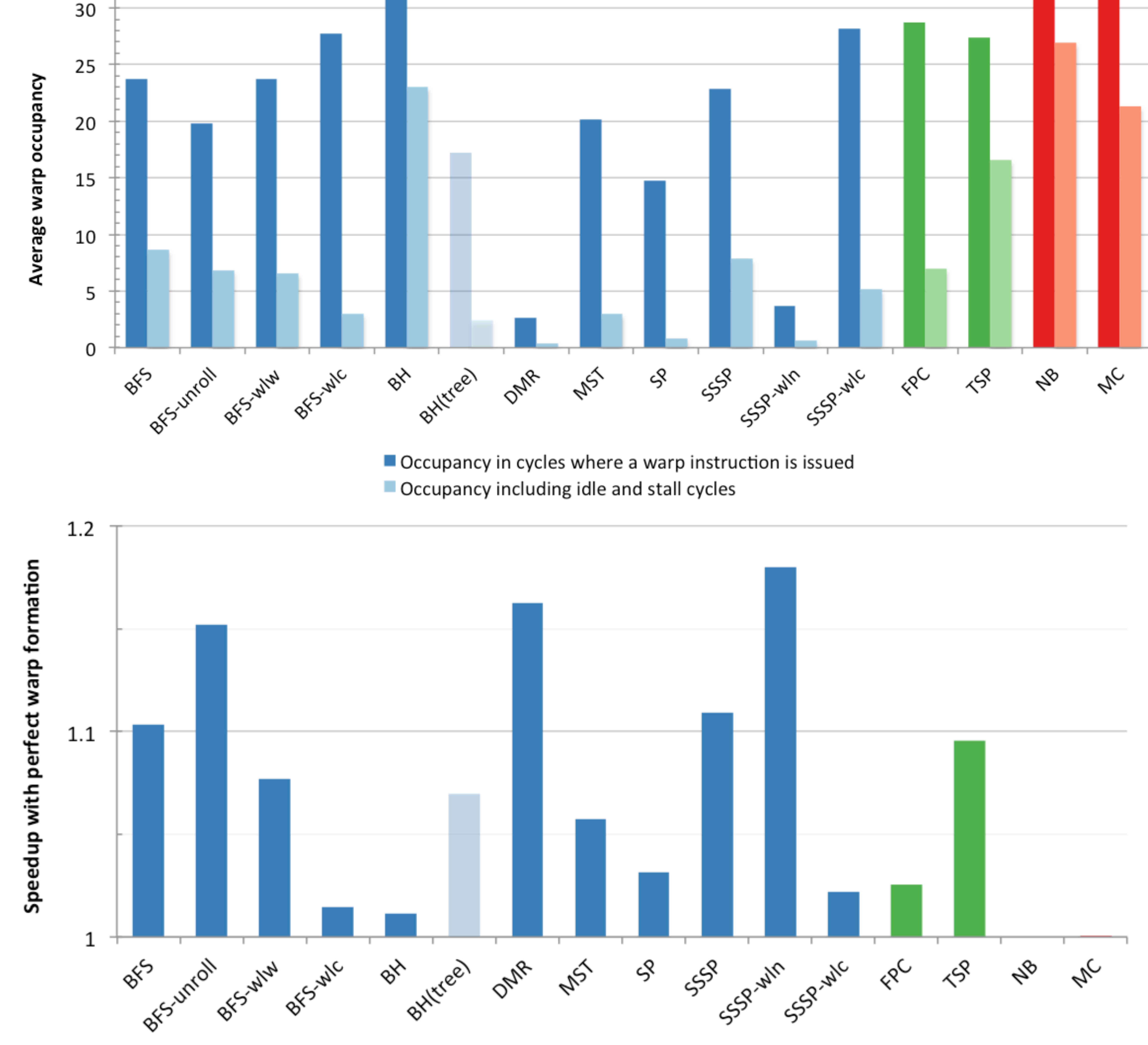


Branch Divergence



Branch Divergence

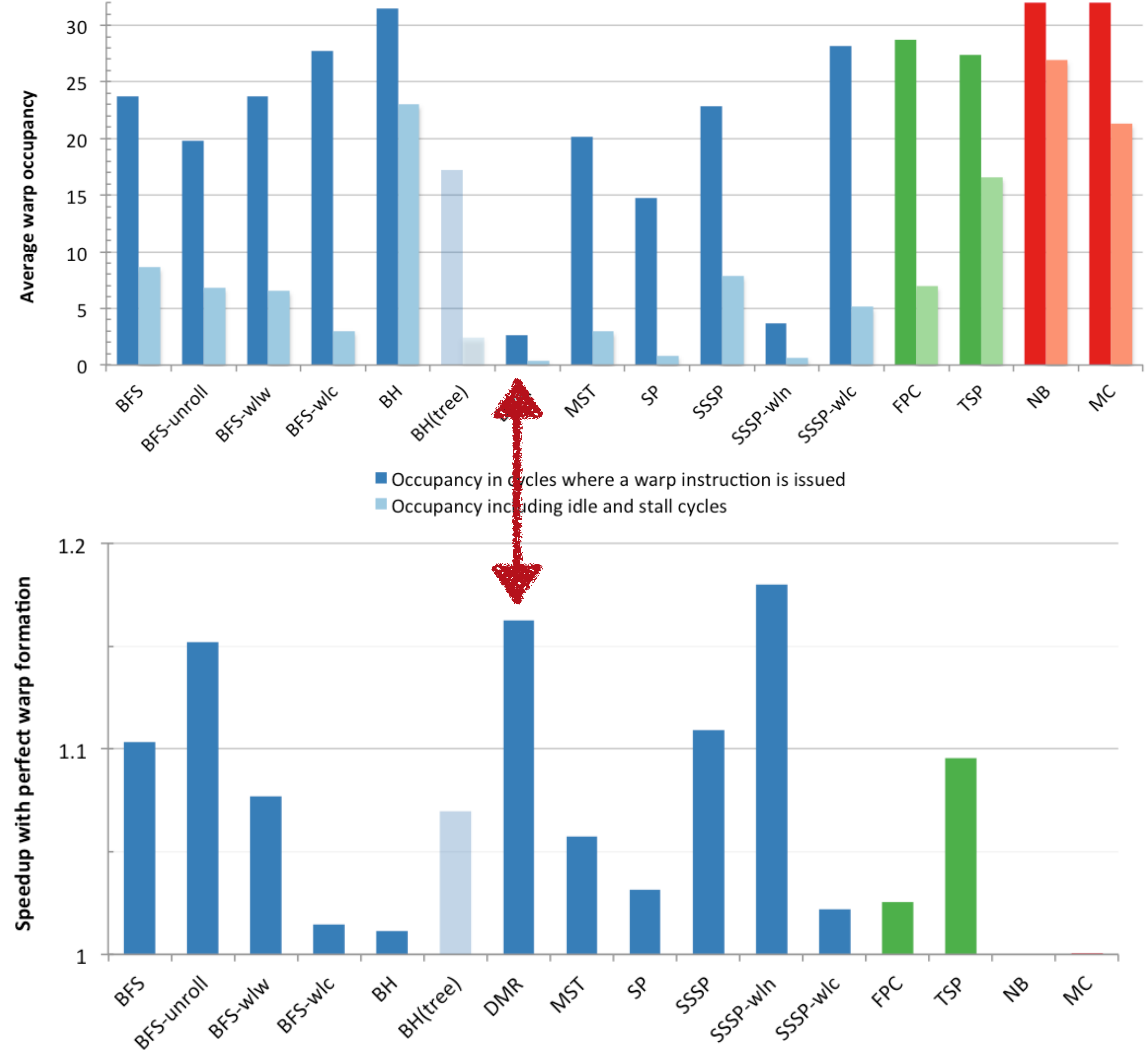
- Irregular codes more diverged
  - But only two <~50% occupied
  - BH an outlier again
- Only a *few codes >~10% speedup* even with perfect warp formation



Branch Divergence

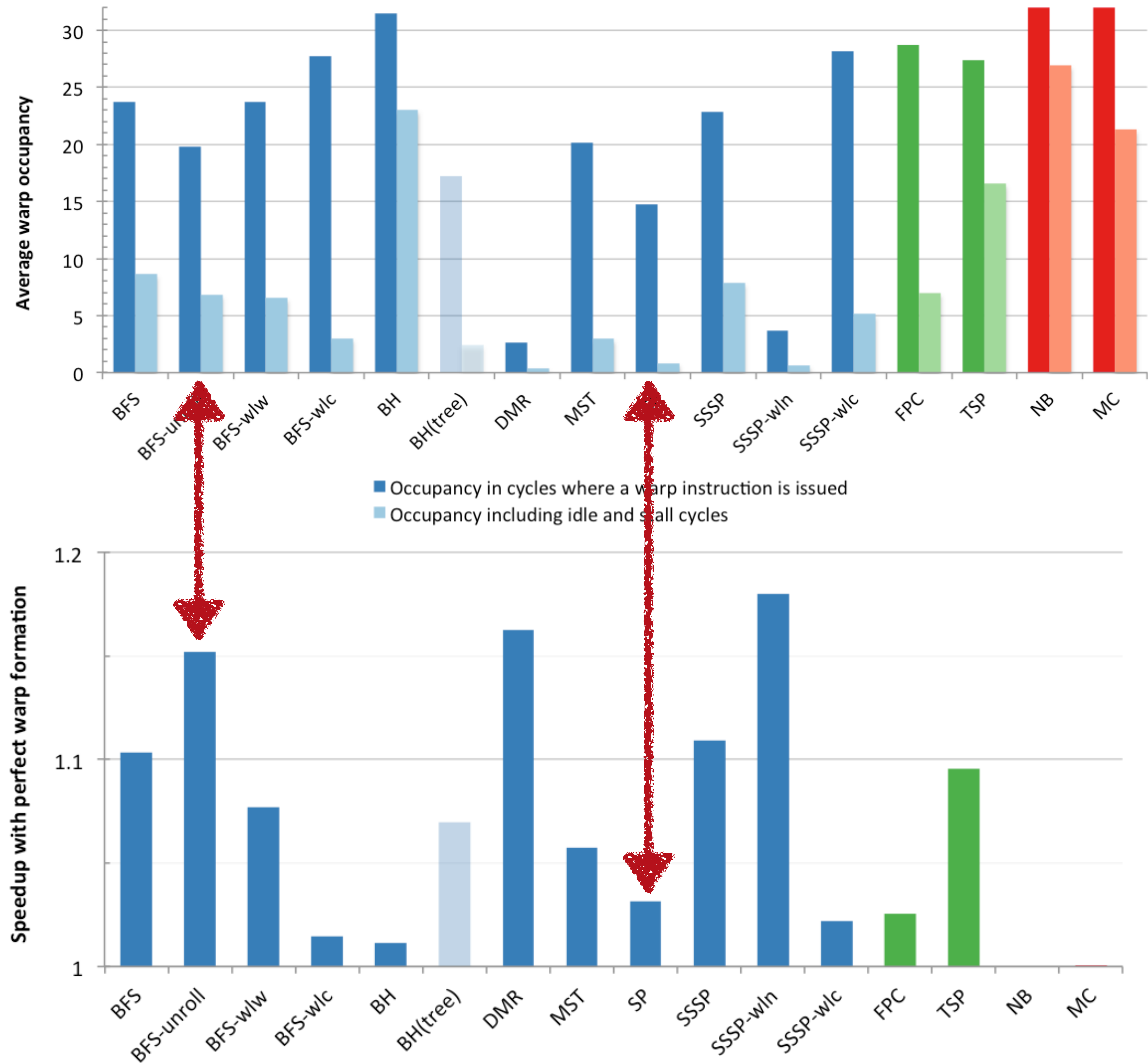
- Irregular codes more diverged
  - But only two <~50% occupied
  - BH an outlier again

- Only a *few codes >~10% speedup* even with perfect warp formation

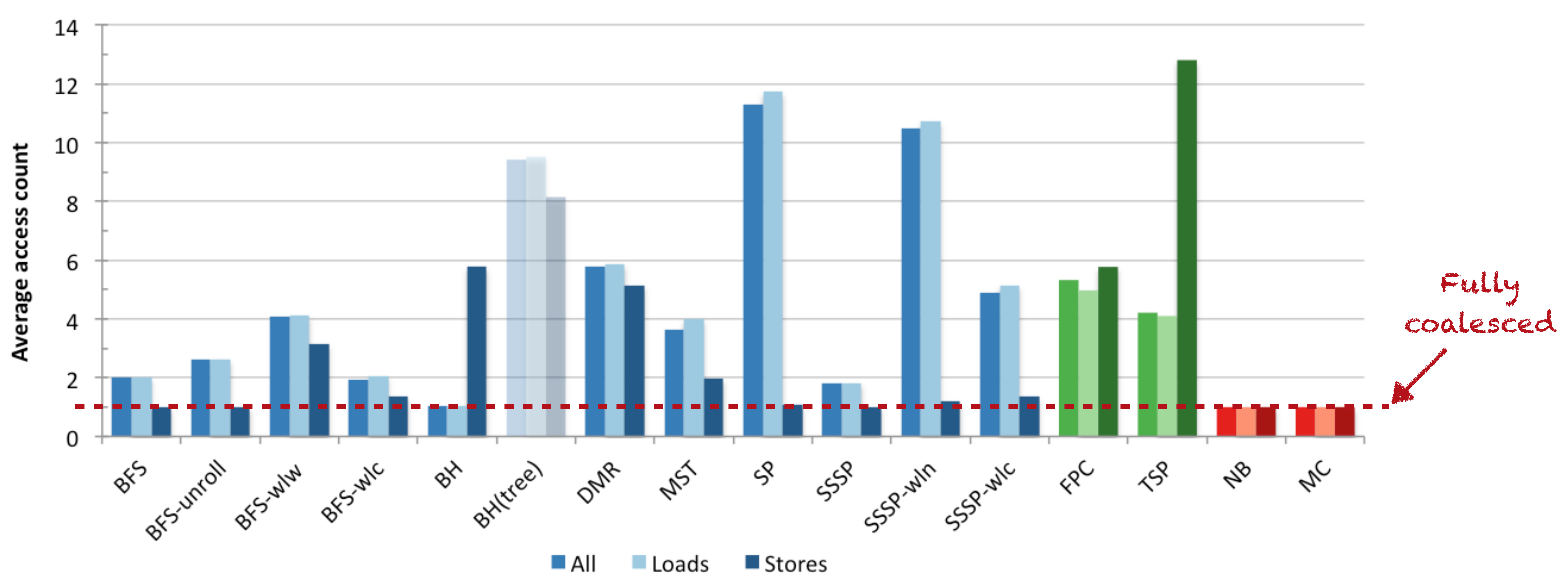


Branch Divergence

- Irregular codes more diverged
  - But only two <~50% occupied
  - BH an outlier again
- Only a *few codes >~10% speedup* even with perfect warp formation

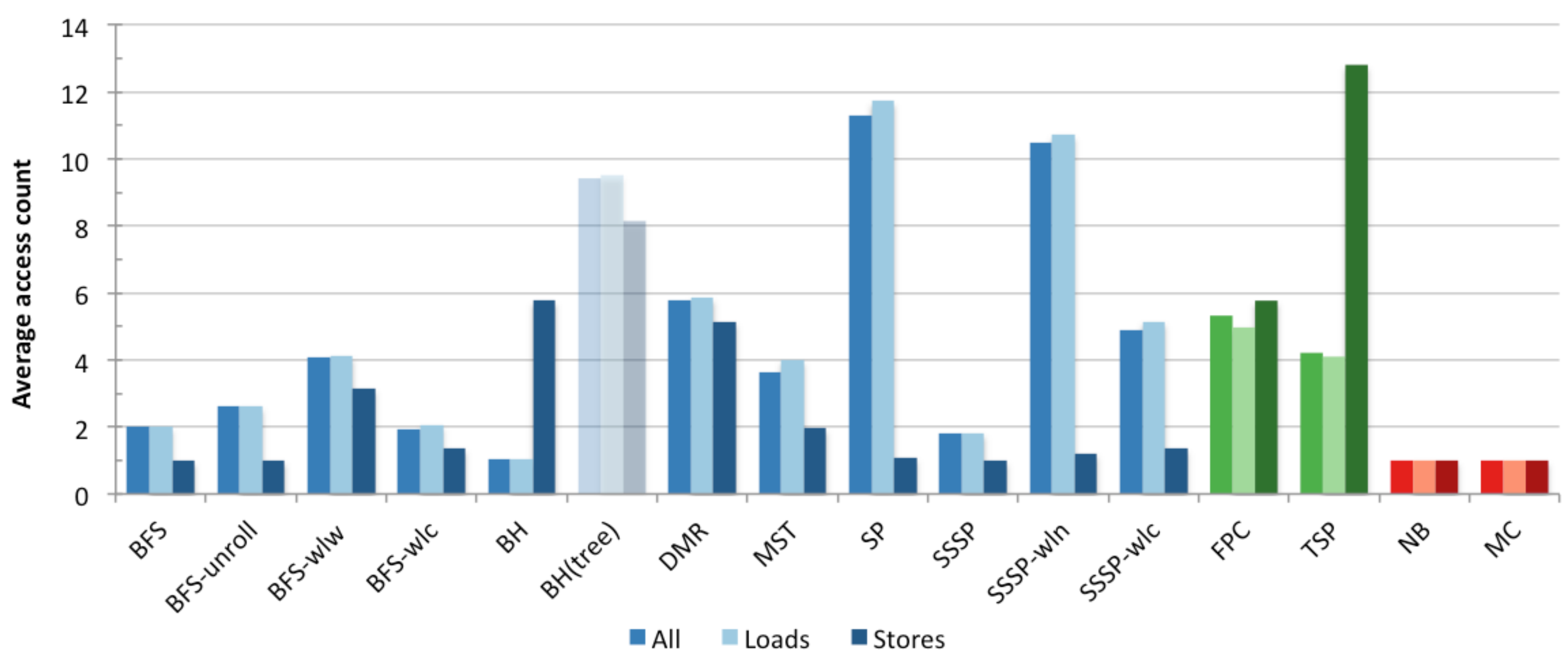


Memory Coalescing

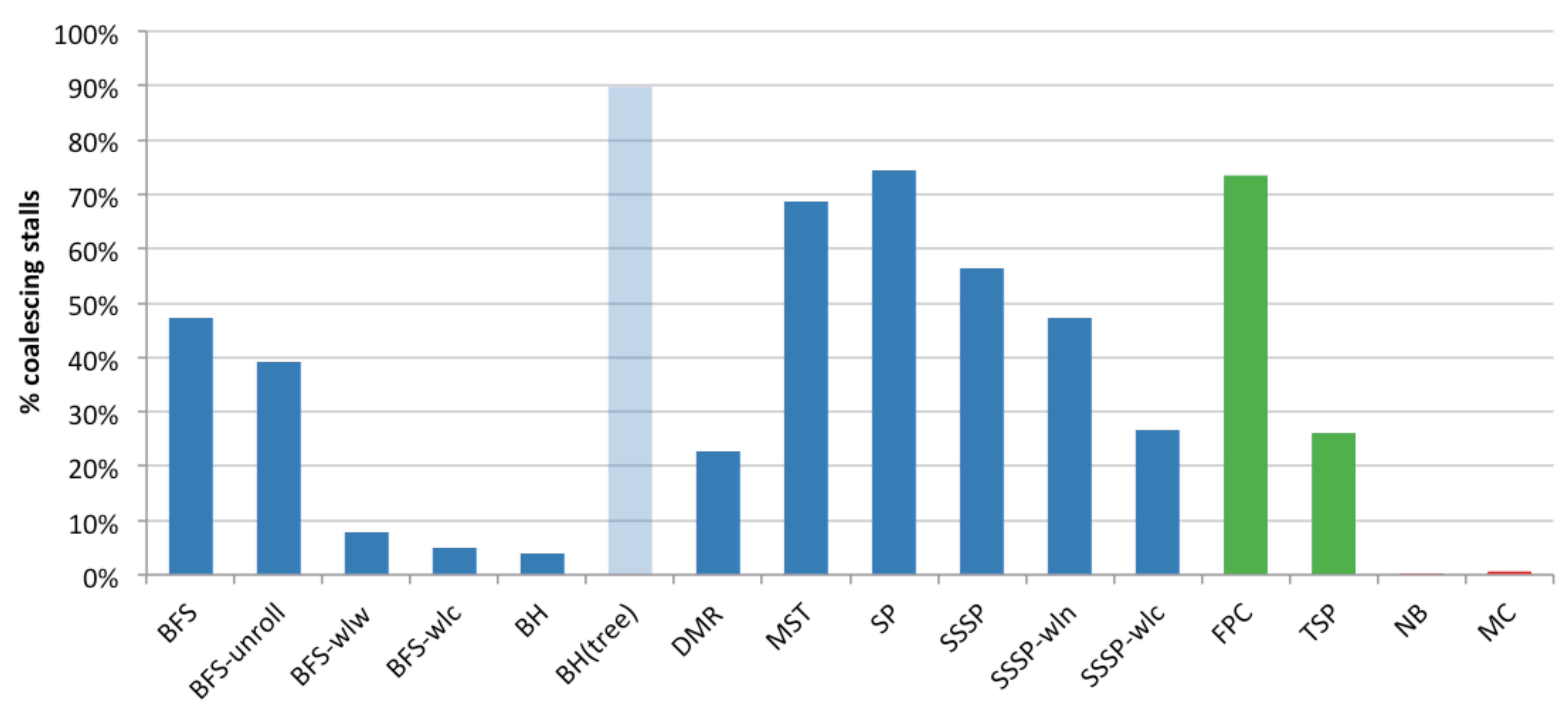


Memory Coalescing

- Regular applications are fully coalesced
- TSP: byte-granular stores to same word serialized by hardware
- BH tree construction, SP, and SSSP-wln all very un-coalesced
  - Very scattered access patterns
  - Topological BFS + SSSP quite coalesced, but…

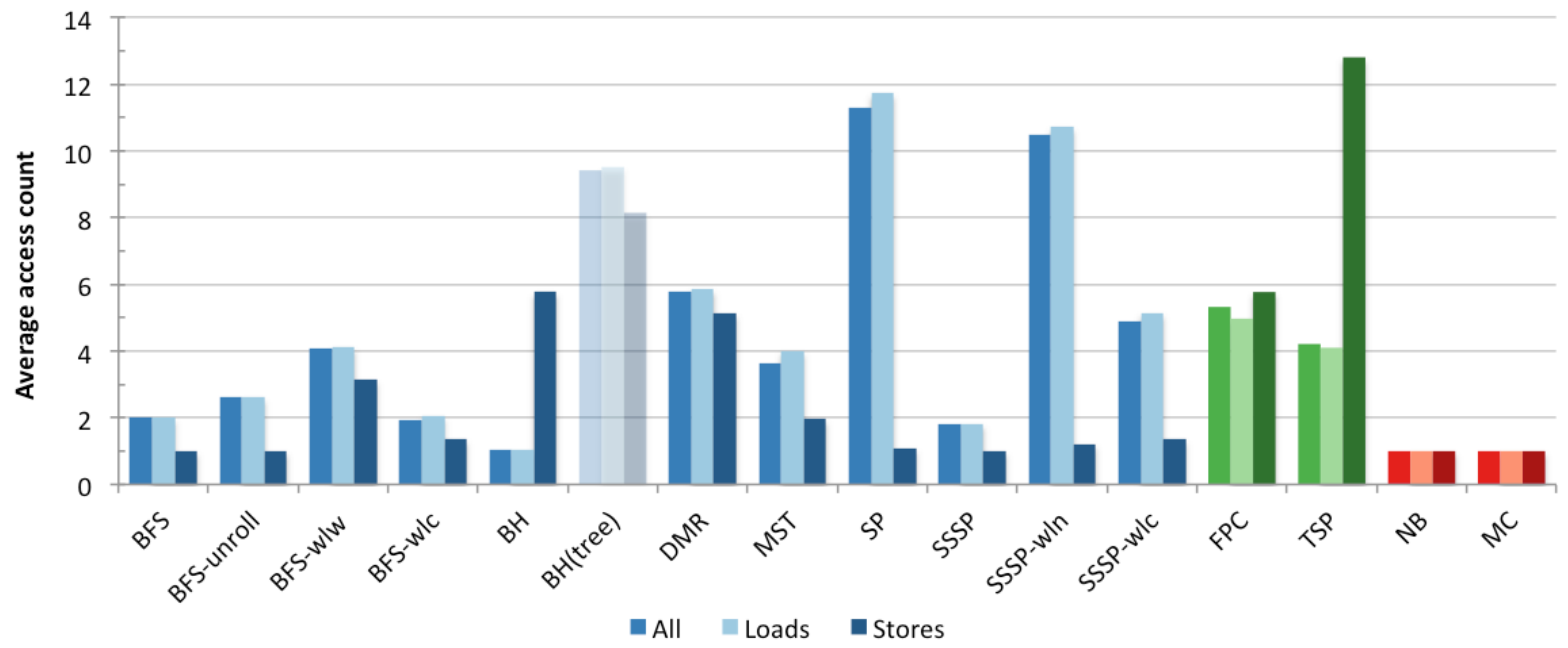


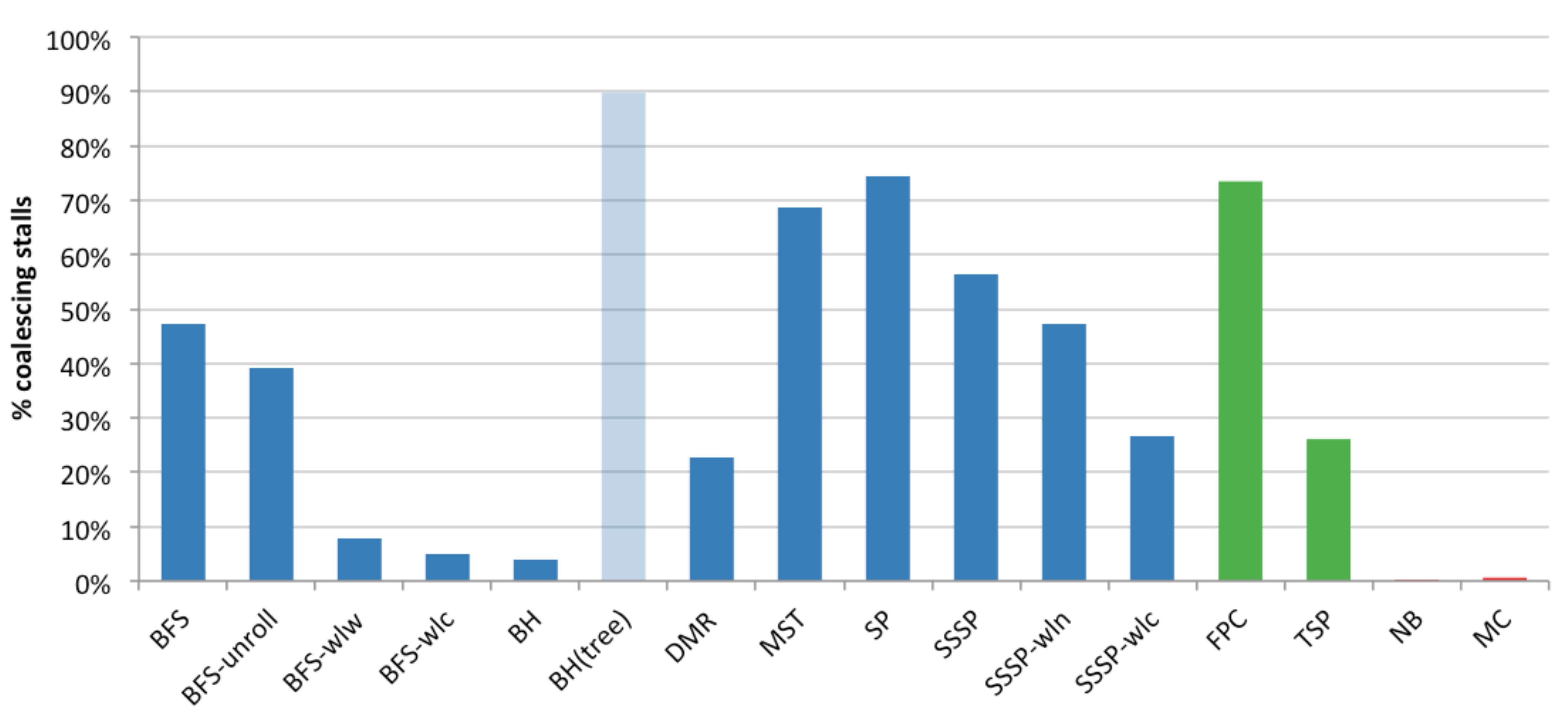
Memory Coalescing



Memory Coalescing

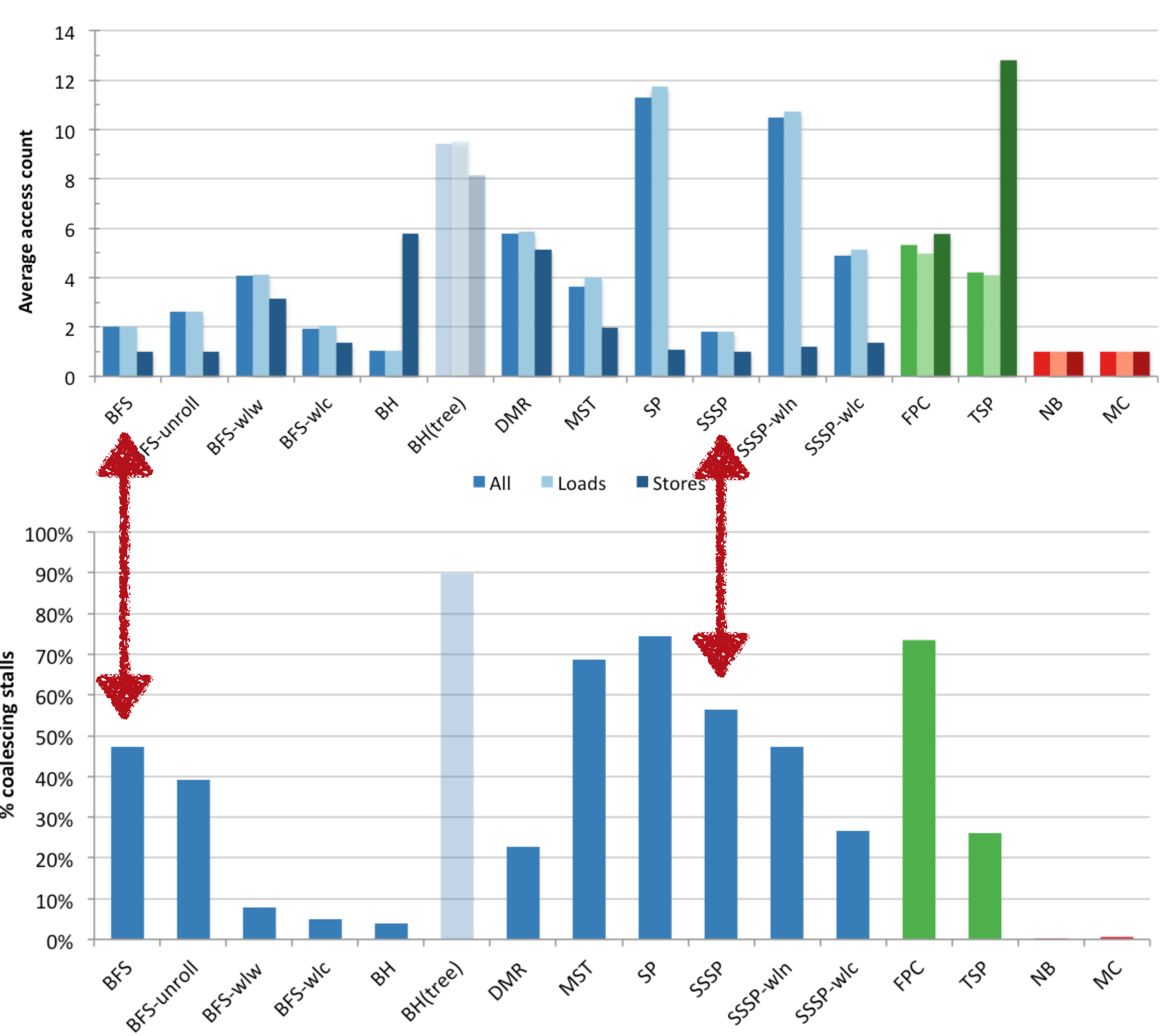
- Regular applications are fully coalesced
- TSP: byte-granular stores to same word serialized by hardware
- BH tree construction, SP, and SSSP-wln all very un-coalesced
  - Very scattered access patterns
  - Topological BFS + SSSP quite coalesced, but…





Memory Coalescing

- Regular applications are fully coalesced
- TSP: byte-granular stores to same word serialized by hardware
- BH tree construction, SP, and SSSP-wln all very un-coalesced
  - Very scattered access patterns
  - Topological BFS + SSSP quite coalesced, but…
- High load instruction count → *even a small amount of un-coalescing hurts*

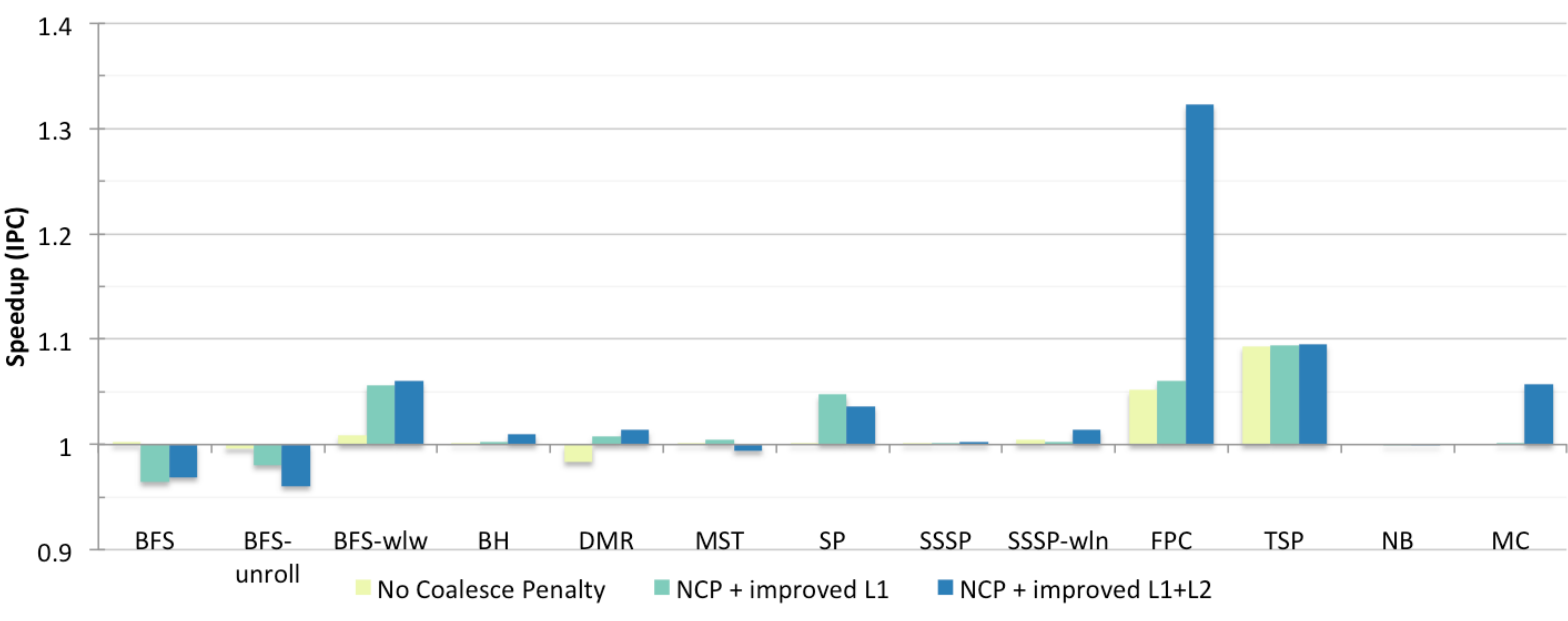


Memory Coalescing

[CUDA C Best Practice Guide, NVIDIA]

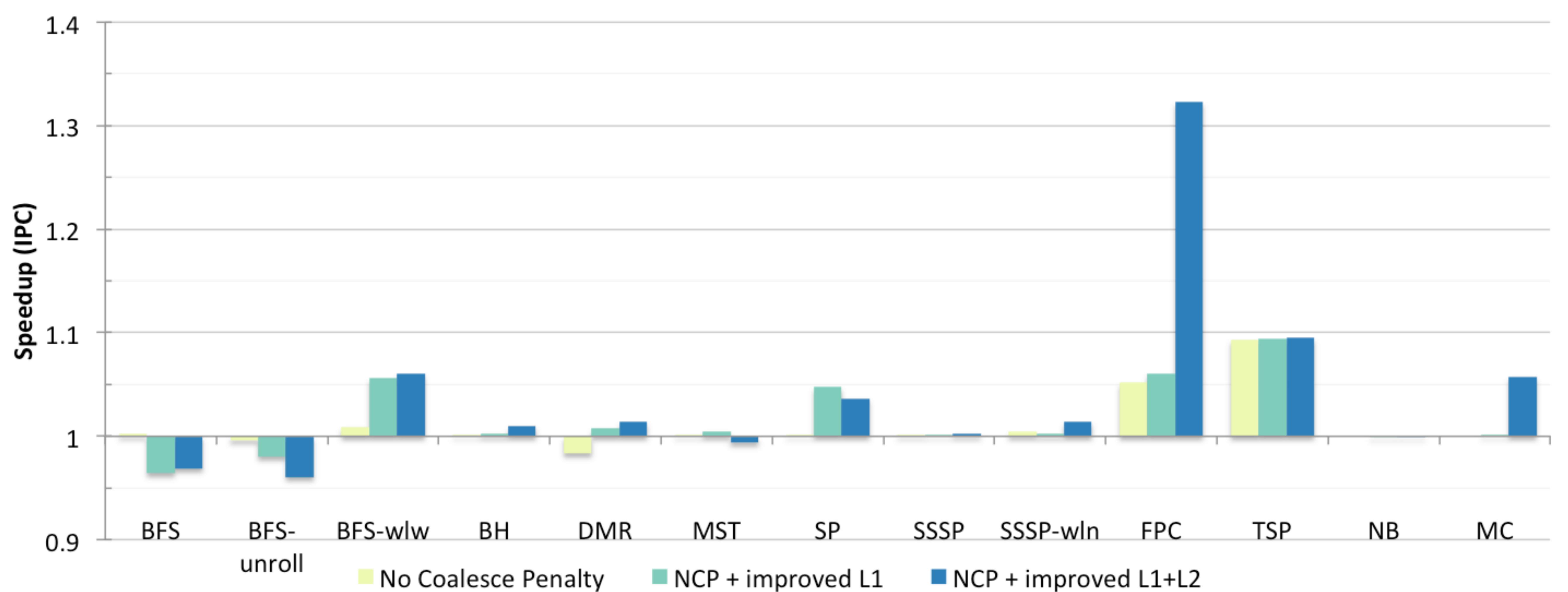
- Two components to coalescing
  1. Pipe stall or replay necessary to perform cache lookup, set up memory request, etc.
  2. Extra memory traffic
- New GPGPU-Sim configuration: No Coalesce Penalty (NCP)
  - Artificially removes the pipeline stall for non-coalesced accesses
  - No other improvement to memory pipeline to handle additional traffic
  - Not intended to be a realistic hardware improvement

Memory Coalescing



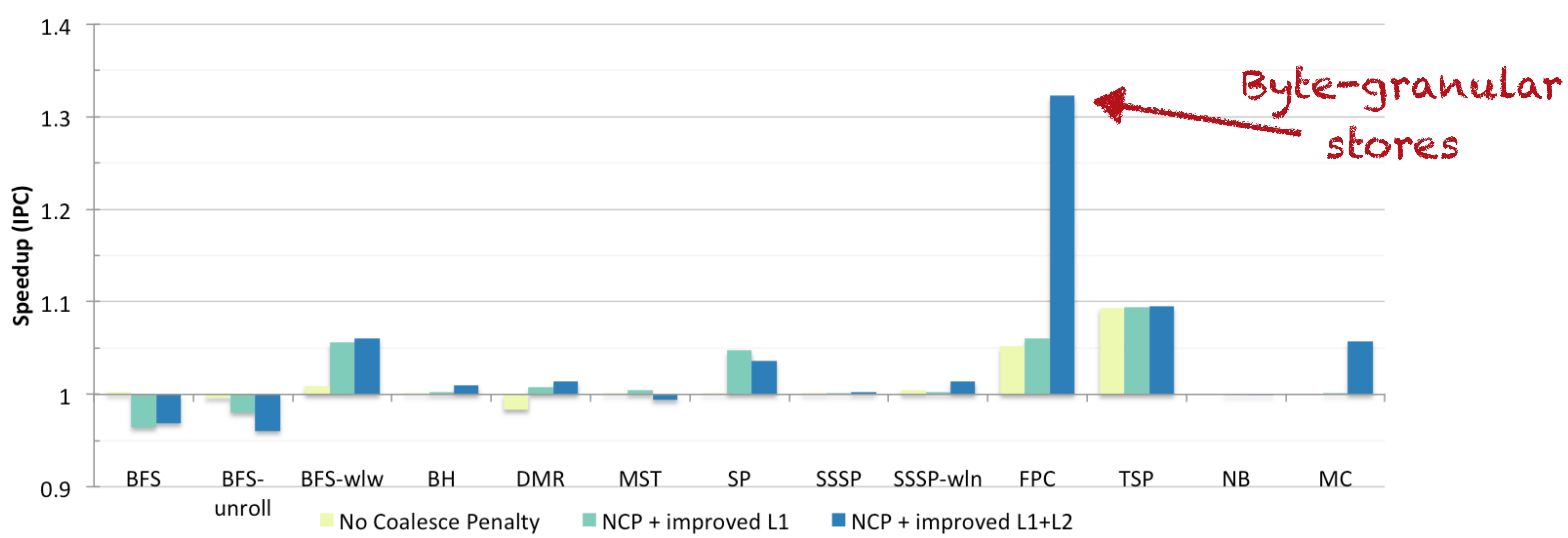
Memory Coalescing

- Applied NCP config by itself as well as in combination with increased cache buffers
- Removing pipeline penalty alone does little good (and sometimes hurts)
  - Improving miss-handling capacity in the cache doesn't help much, either!
- *H/W improvements aimed at reducing coalescing pipeline penalty unlikely to help* irregular codes unless combined with improved memory bandwidth or cache usage

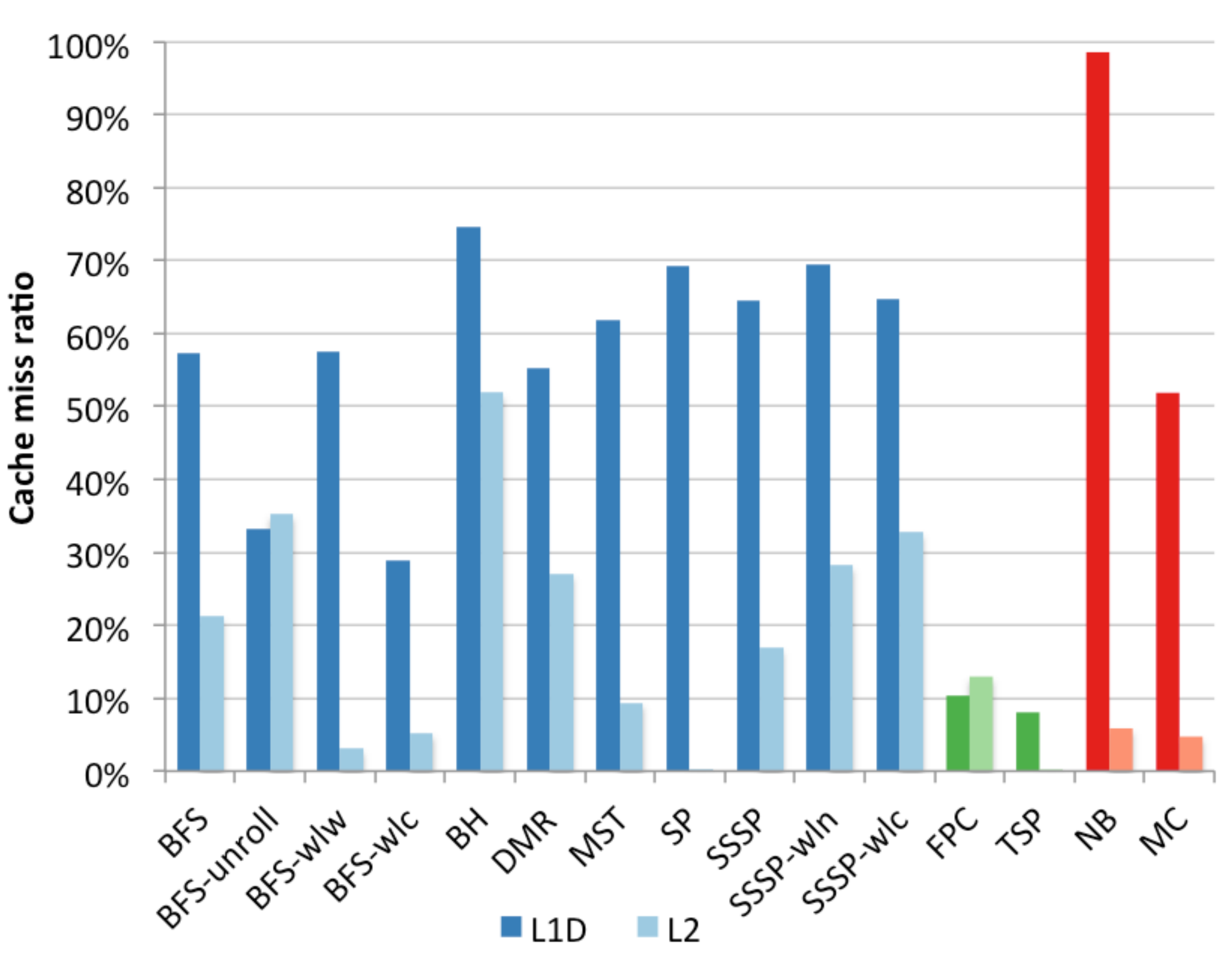


Memory Coalescing

- Applied NCP config by itself as well as in combination with increased cache buffers
- Removing pipeline penalty alone does little good (and sometimes hurts)
  - Improving miss-handling capacity in the cache doesn't help much, either!
- *H/W improvements aimed at reducing coalescing pipeline penalty unlikely to help* irregular codes unless combined with improved memory bandwidth or cache usage

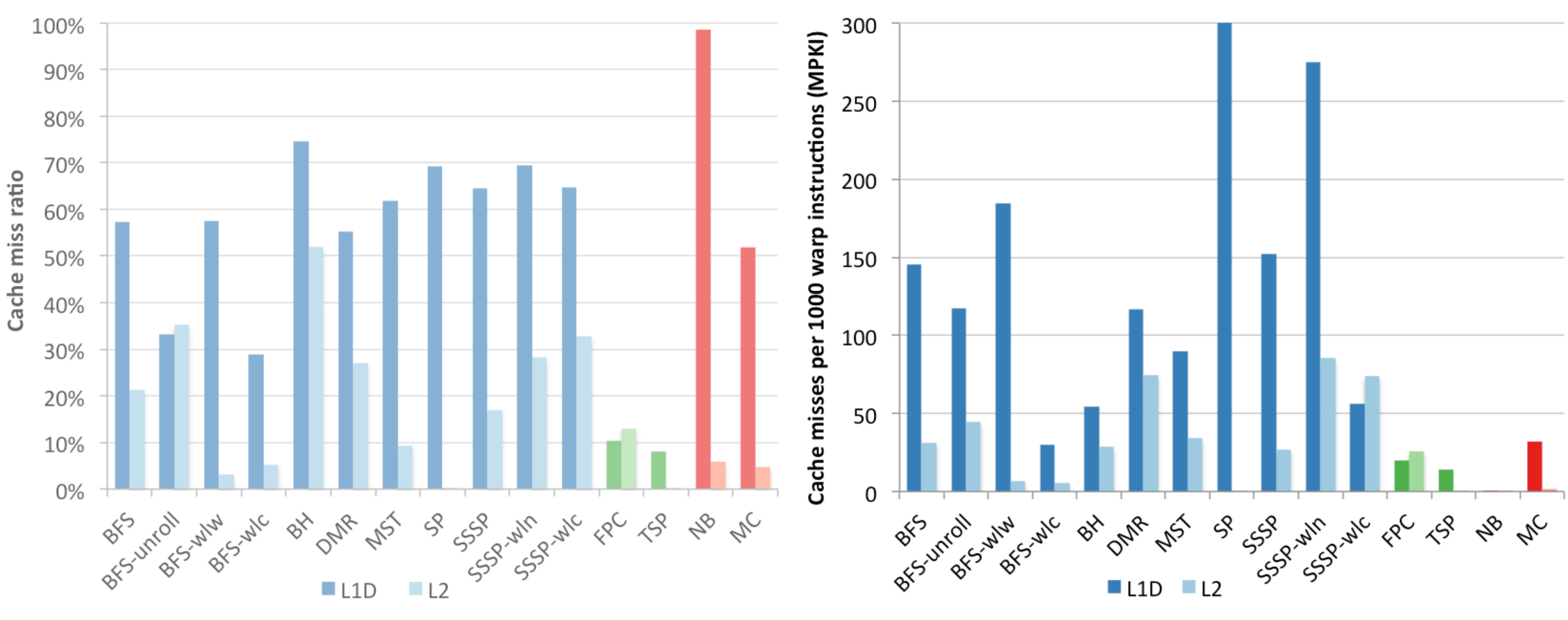


Cache Effectiveness



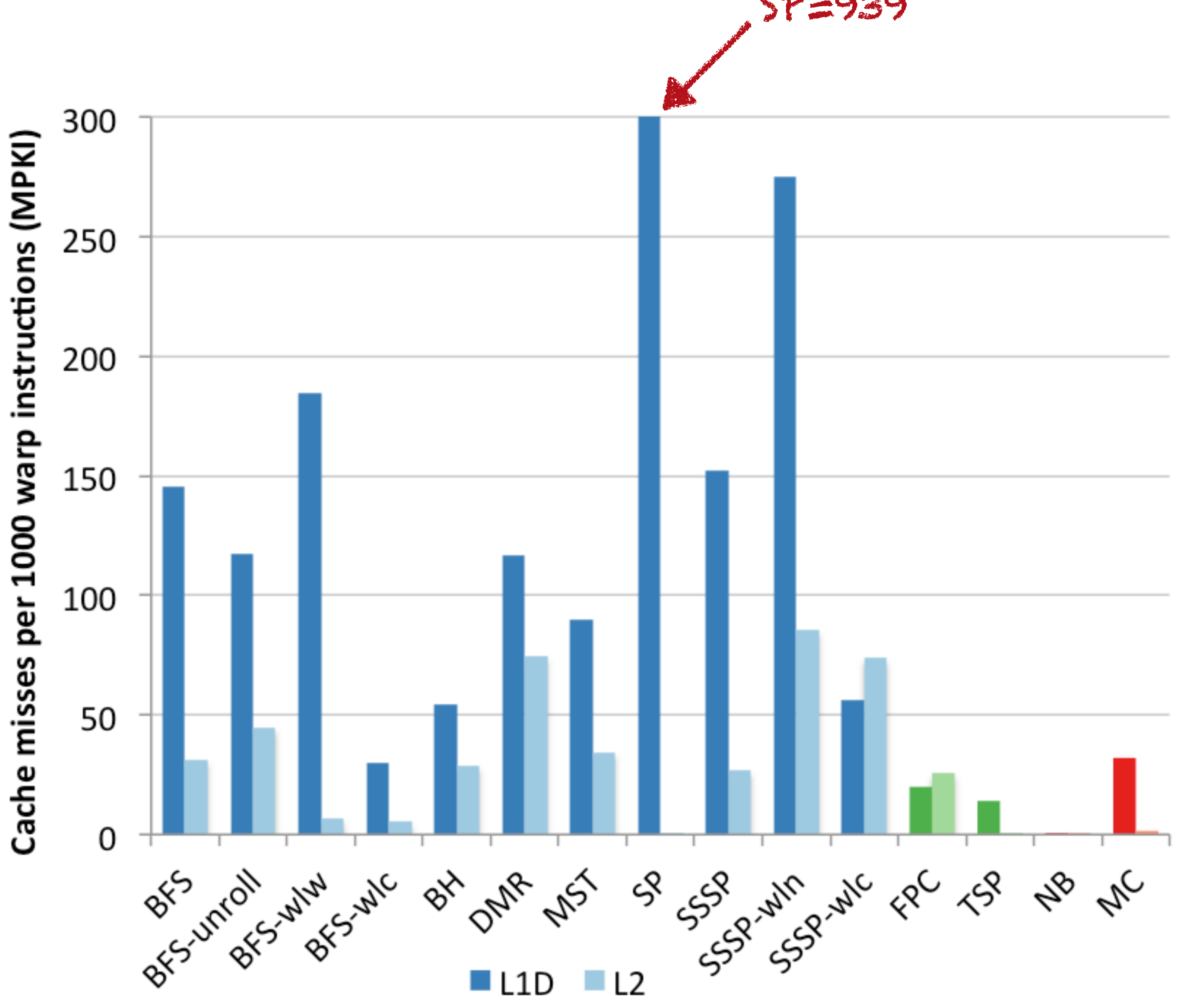
- Very high miss ratios (most >50% in the L1)
- GPUs and CPUs have caches for different reasons

Cache Effectiveness



Cache Effectiveness

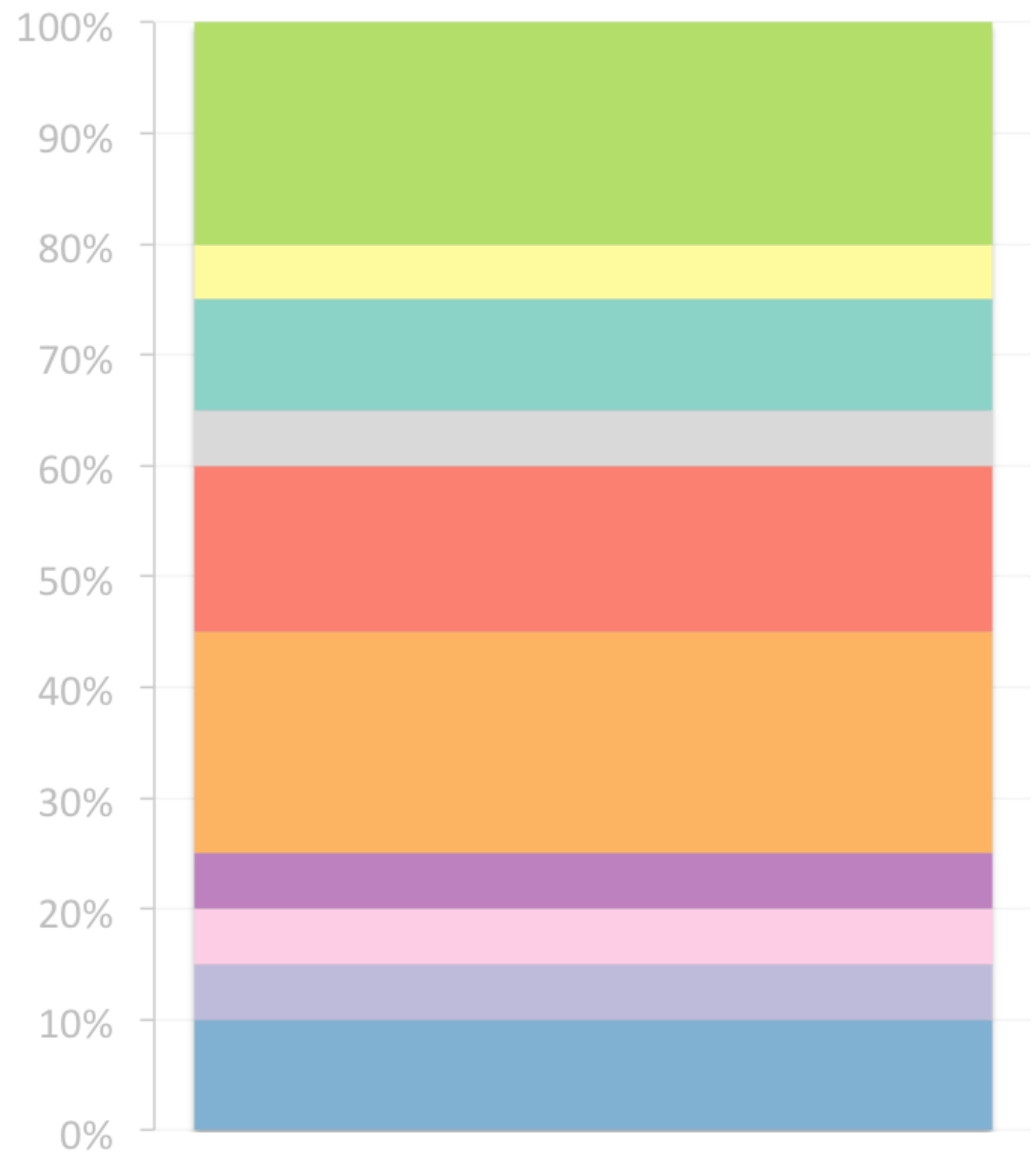
- Irregular codes look *very different* than regular codes
  - Lots of pointer chasing
  - Not much spatial locality
- SP has highest average access count of these codes → absurdly high miss rate



Individual Applications

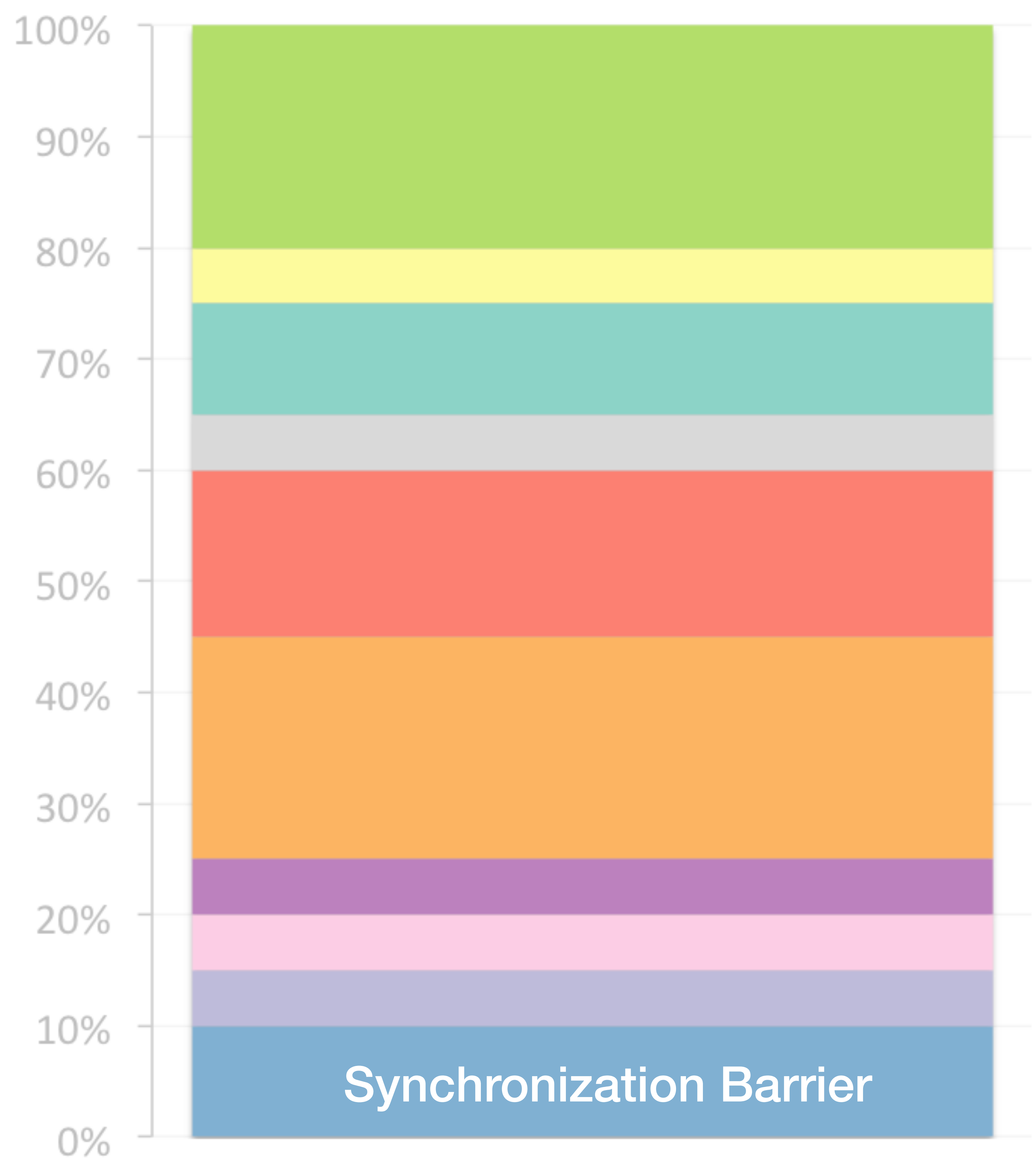
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



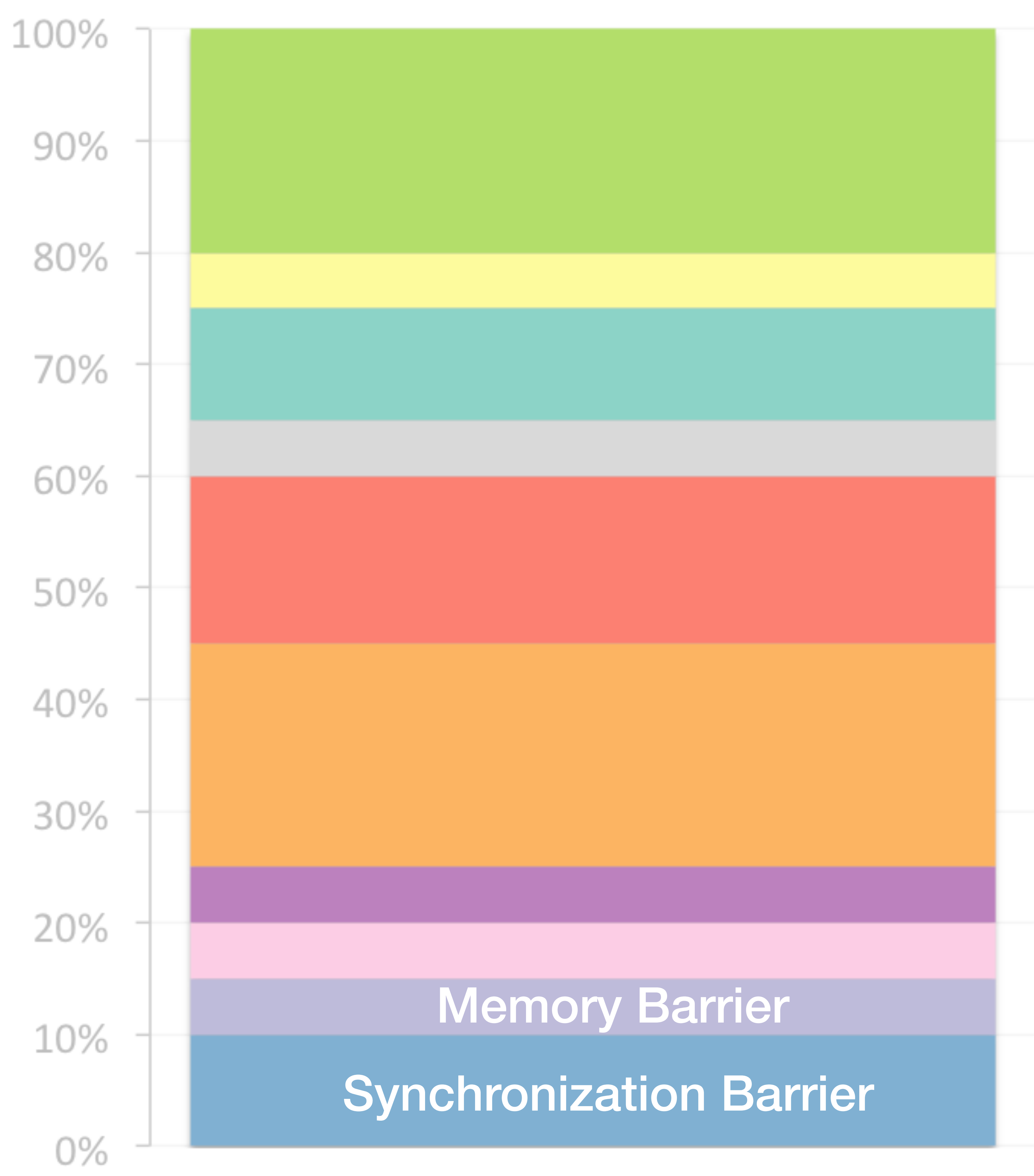
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



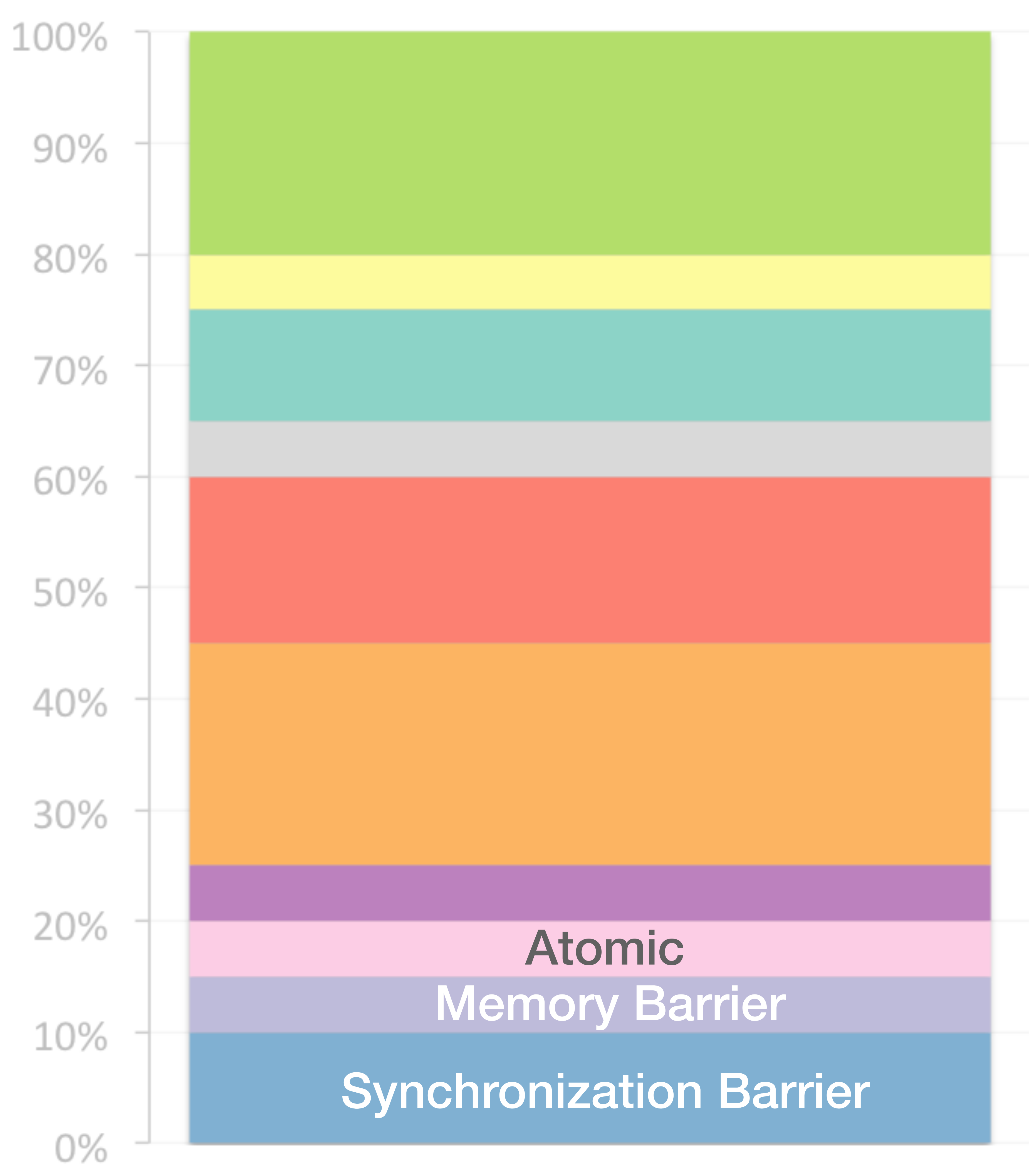
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



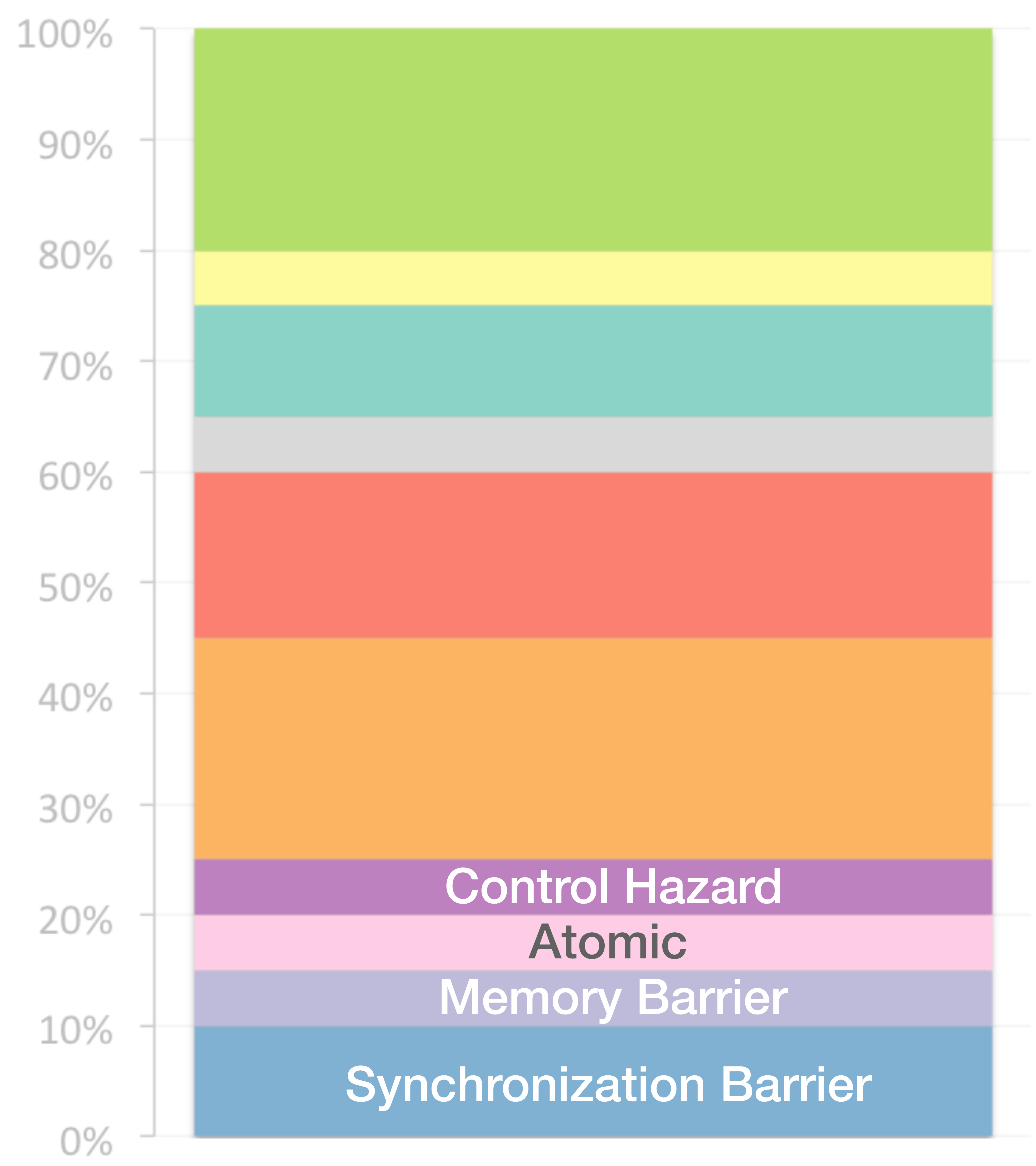
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



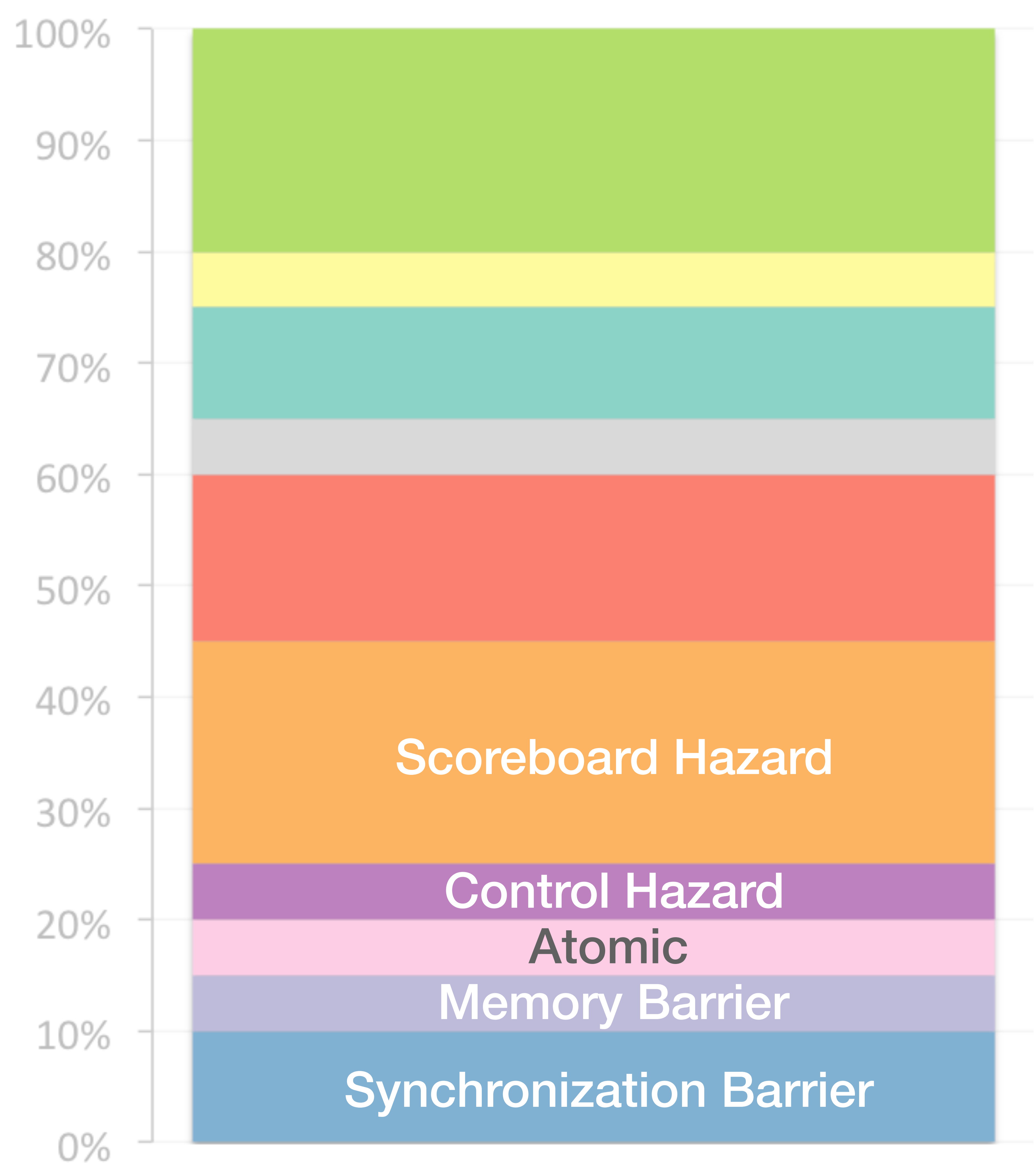
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



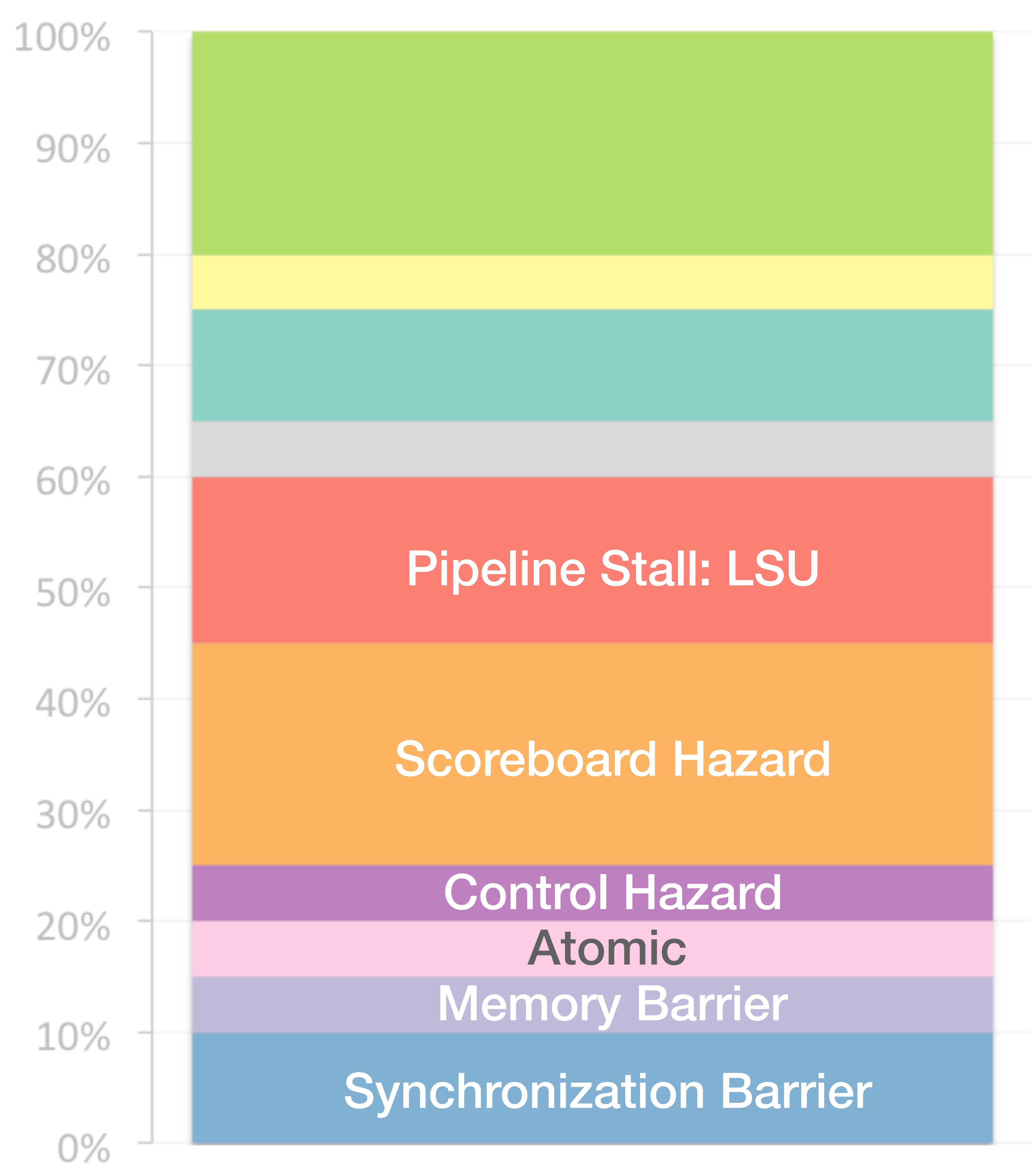
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



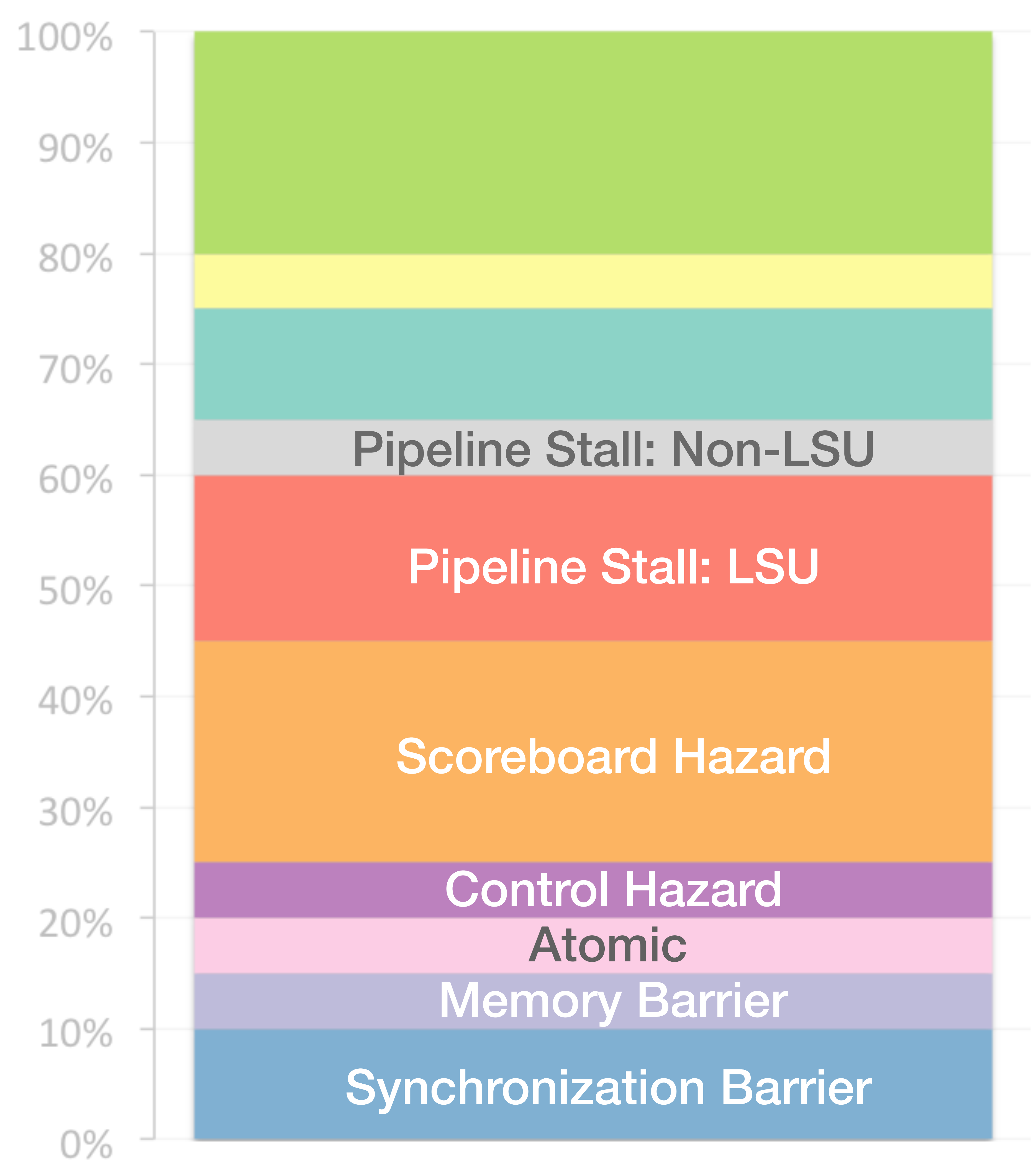
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



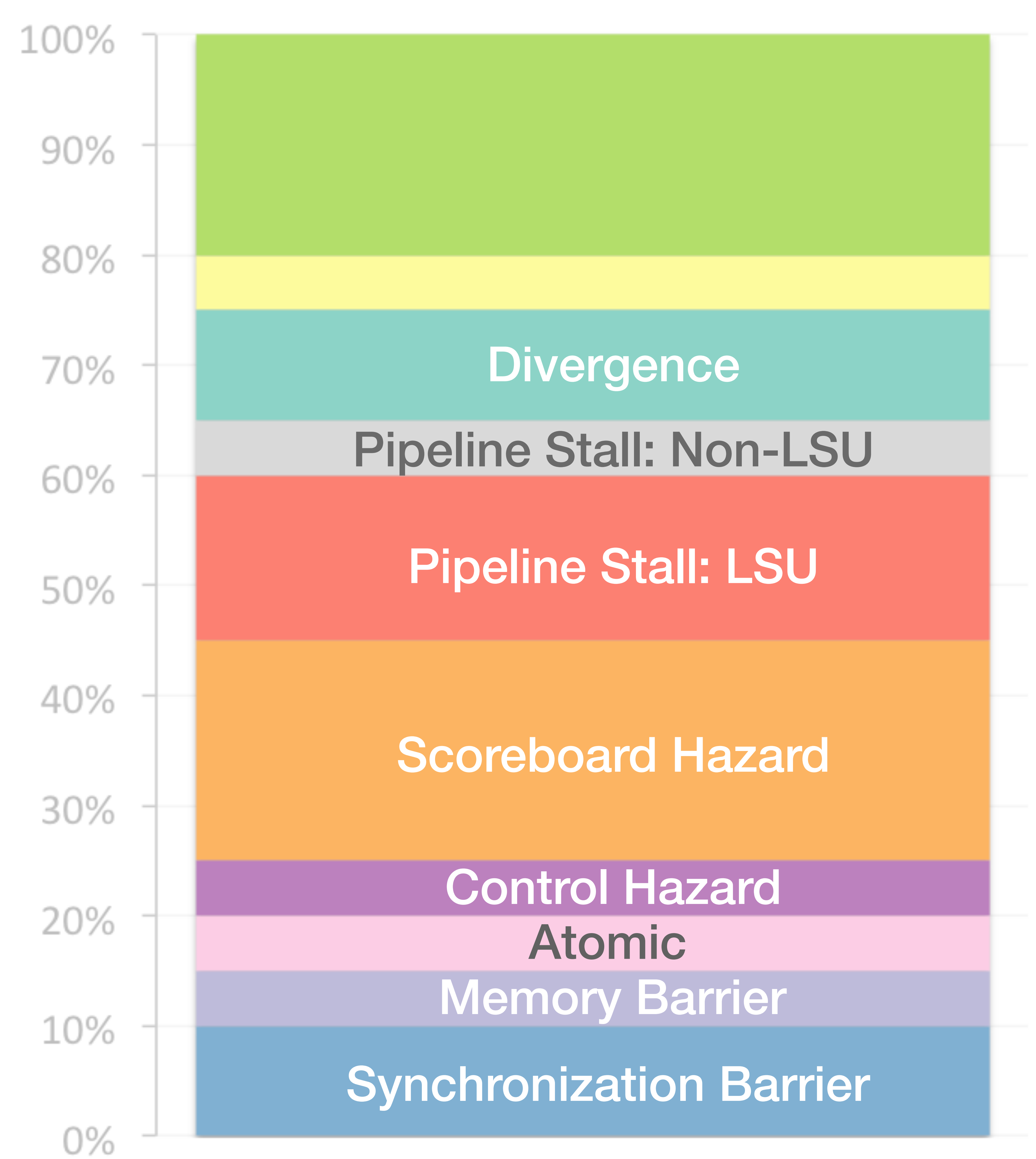
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



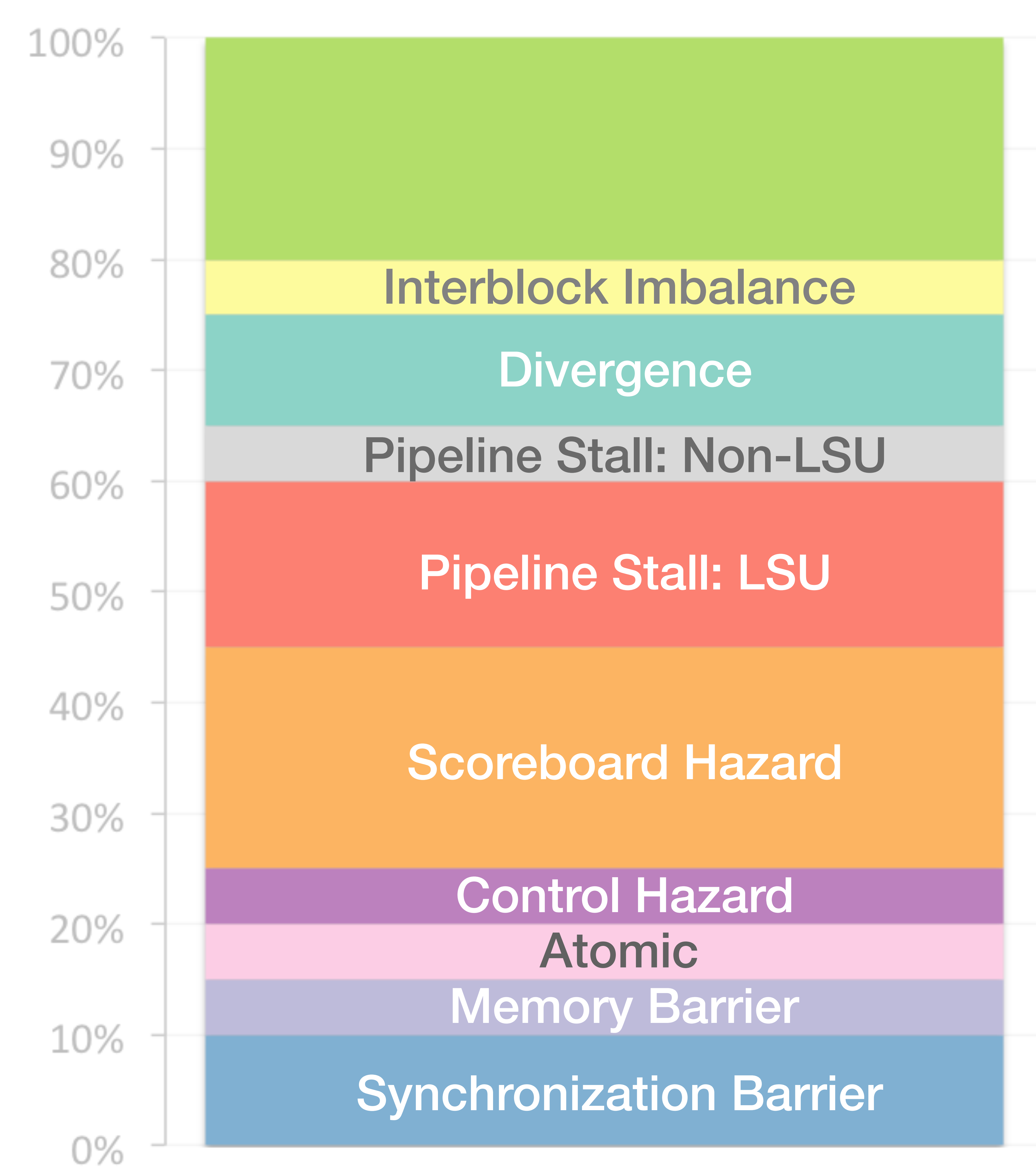
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



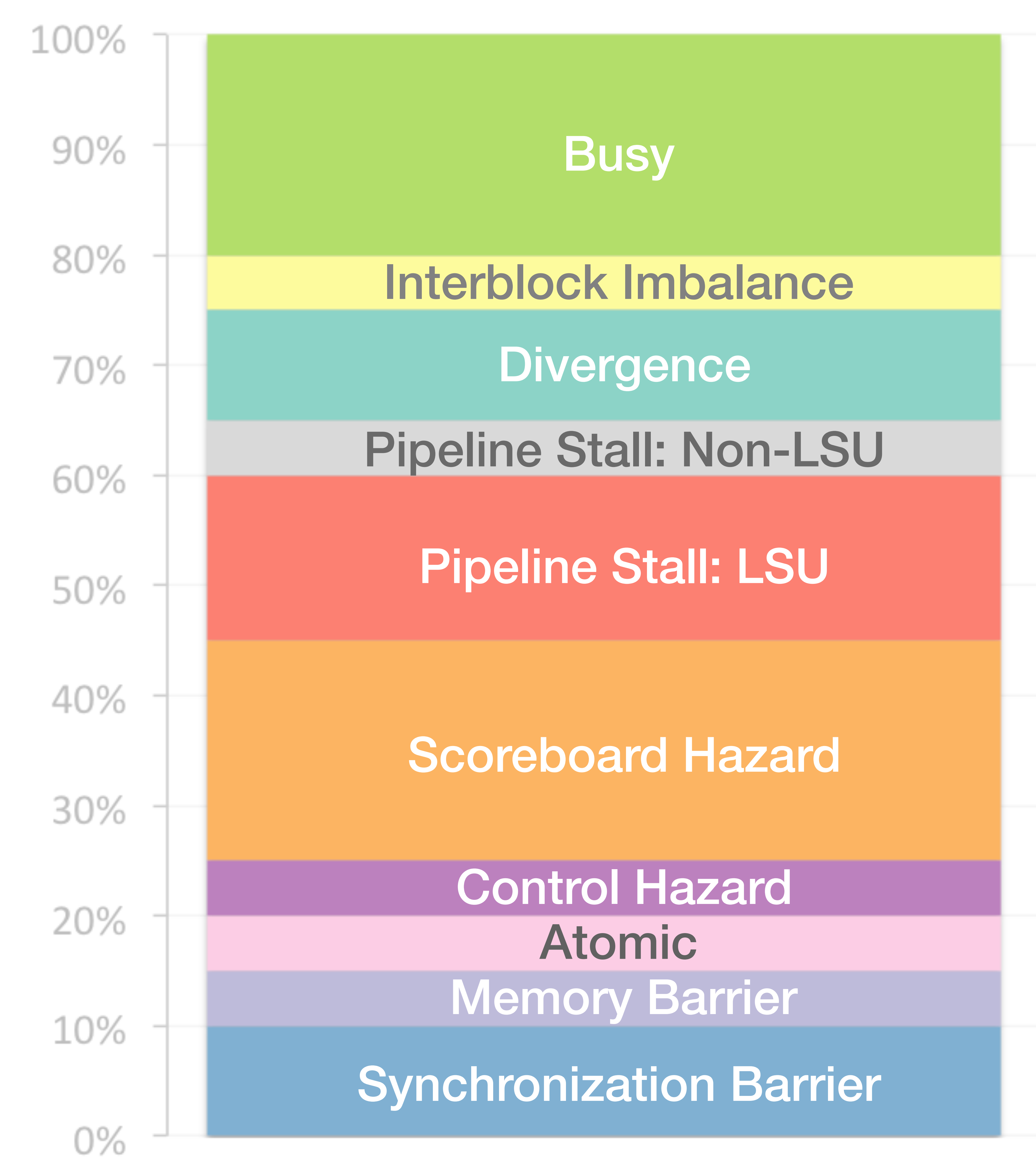
Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue

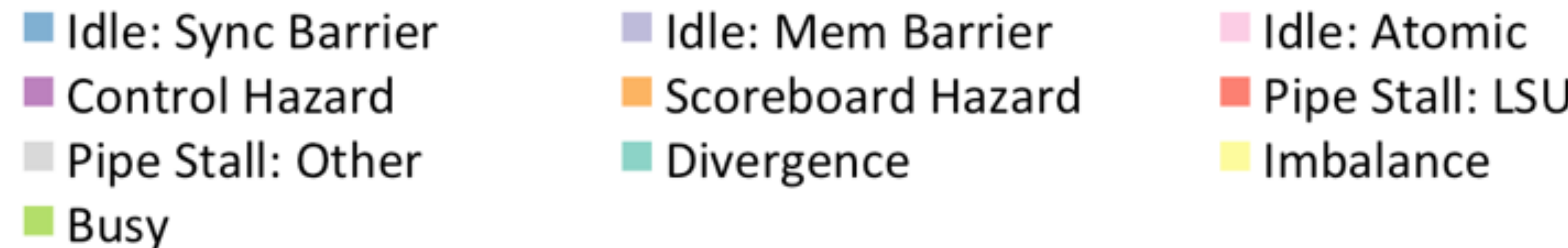


Measuring Application Slowdown

- Histogram of *underutilized vs. fully-occupied cycles* in each benchmark
  - In issue stage of each SM
  - Based on active threads in warp
  - If no issue: track deepest pipeline stage responsible for no-issue



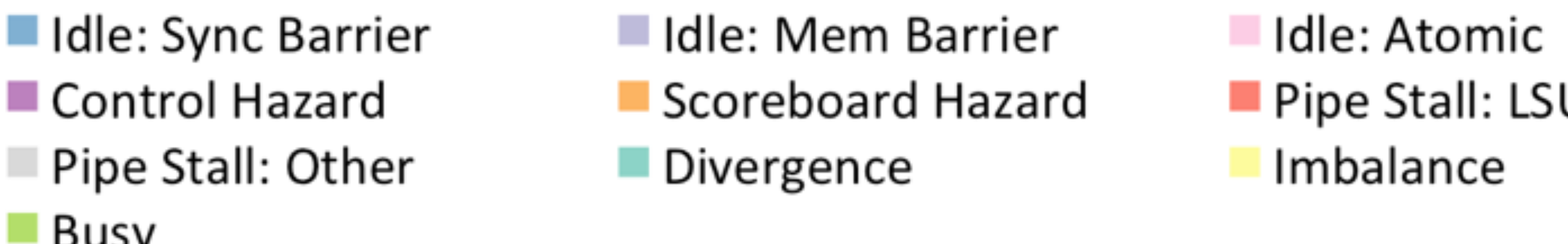
Applications



100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS | BFS-unroll | BFS-wlw | BFS-wlc | BH | BH(tree) | DMR | MST | SP | SSSP | SSSP-wln | SSSP-wlc | FPC | TSP | NB | MC

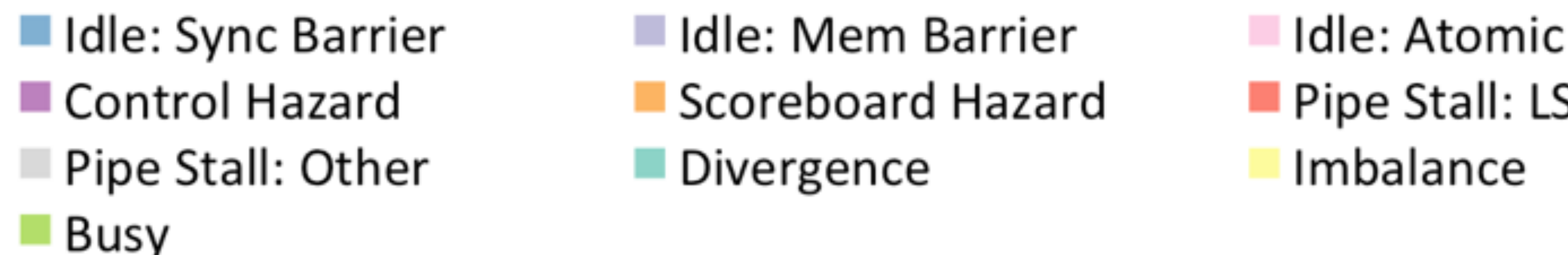
Applications: BFS & SSSP



100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS
BFS-unroll
BFS-wlw
BFS-wlc
BH
BH(tree)
DMR
MST
SP
SSSP
SSSP-wln
SSSP-wlc
FPC
TSP
NB
MC

Applications: BFS & SSSP

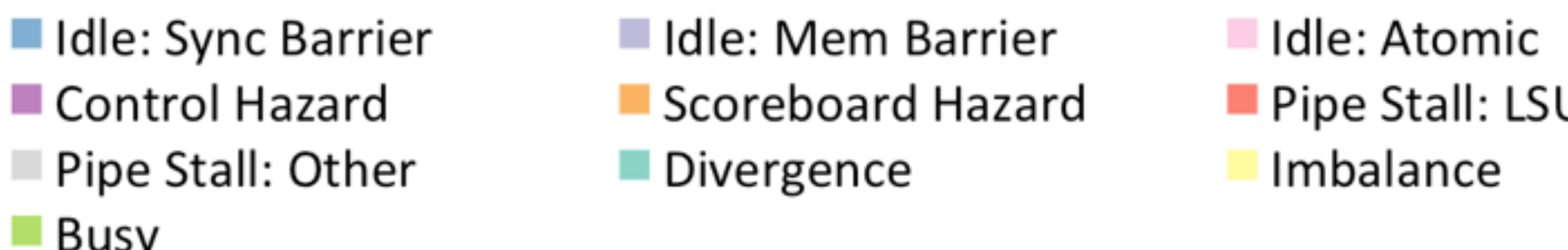


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS | BFS-unroll | BFS-wlw | BFS-wlc | BH | BH(tree) | DMR | MST | SP | SSSP | SSSP-wln | SSSP-wlc | FPC | TSP | NB | MC

Large memory access penalty in nearly all of the irregular apps

Applications: BFS & SSSP



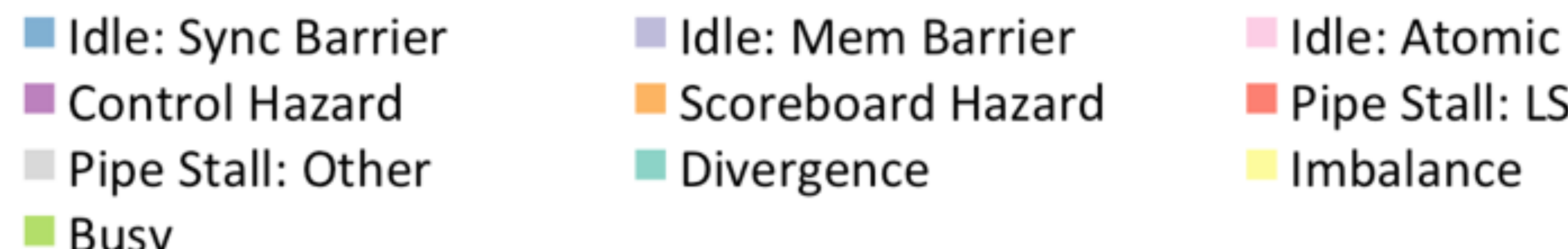
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS | BFS-unroll | BFS-wlw | BFS-wlc | BH | BH(tree) | DMR | MST | SP | SSSP | SSSP-wln | SSSP-wlc | FPC | TSP | NB | MC

Large memory access penalty in nearly all of the irregular apps

Data-driven codes have high synchronization penalty

Applications: BFS & SSSP



100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

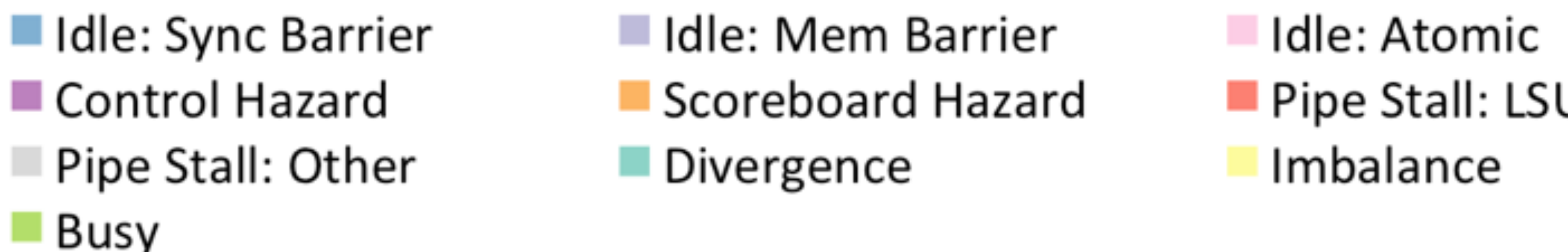
BFS
BFS-unroll
BFS-wlw
BFS-wlc
BH
BH(tree)
DMR
MST
SP
SSSP
SSSP-wln
SSSP-wlc
FPC
TSP
NB
MC

Large memory access penalty in nearly all of the irregular apps

Data-driven codes have high synchronization penalty

Additional atomic operations add mostly RAW hazard stalls

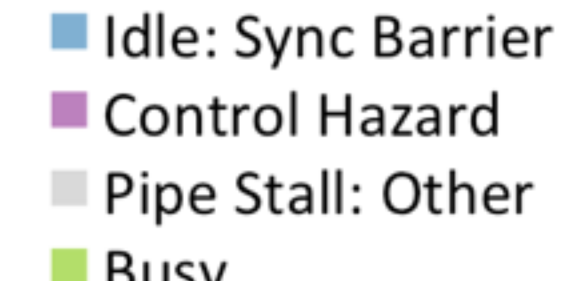
Applications

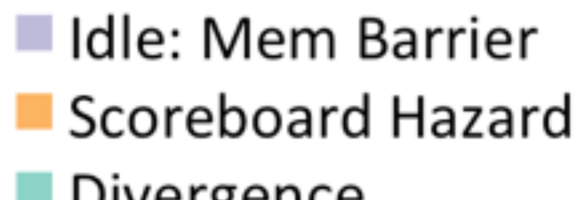


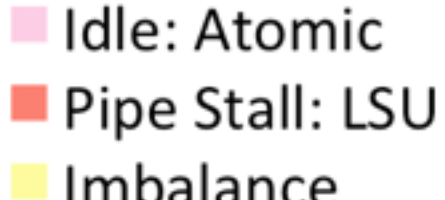
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS
BFS-unroll
BFS-wlw
BFS-wlc
BH
BH(tree)
DMR
MST
SP
SSSP
SSSP-wln
SSSP-wlc
FPC
TSP
NB
MC

Applications: Other Irregular



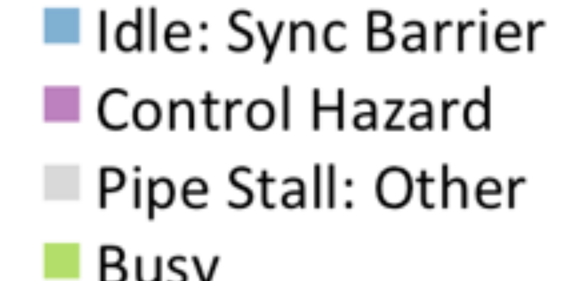


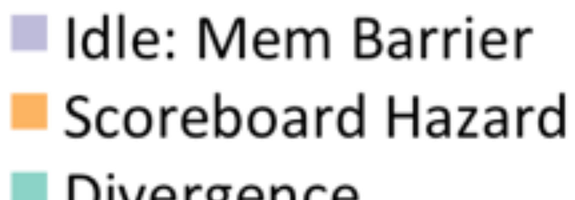


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

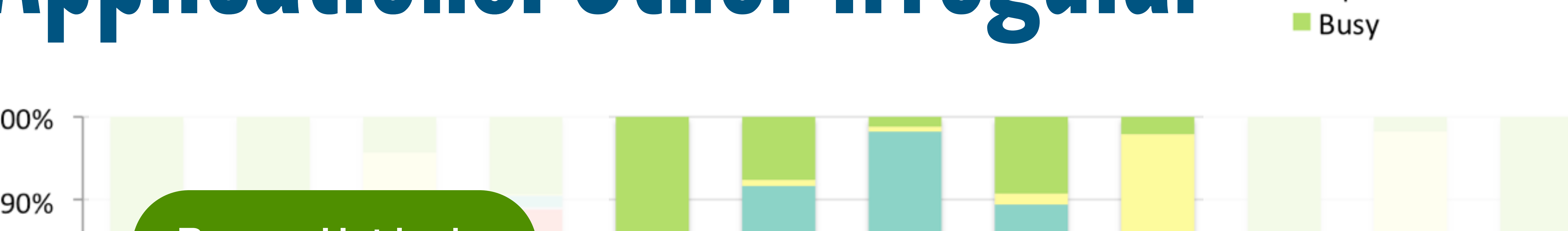
BFS
BFS-unroll
BFS-wlw
BFS-wlc
BH
BH(tree)
DMR
MST
SP
SSSP
SSSP-wln
SSSP-wlc
FPC
TSP
NB
MC

Applications: Other Irregular

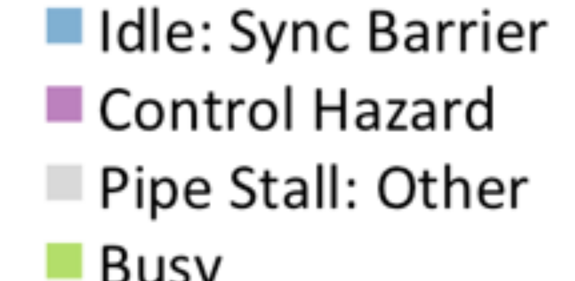


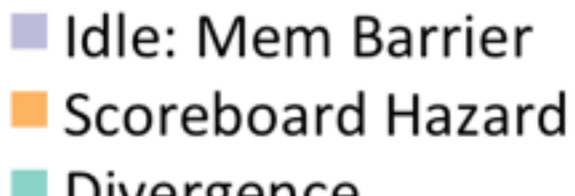


Idle: Atomic
Pipe Stall: LSU
Imbalance



Applications: Other Irregular





Idle: Atomic
Pipe Stall: LSU
Imbalance

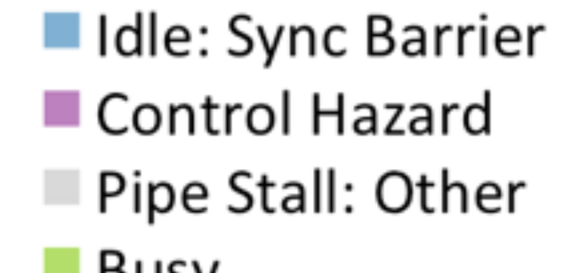
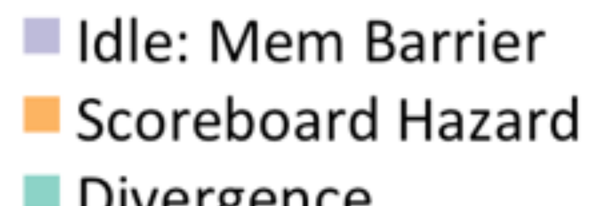
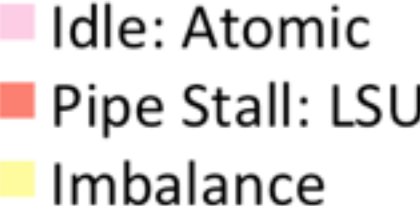
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS
BFS-unroll
BFS-wlw
BFS-wlc
BH
BH(tree)
DMR
MST
SP
SSSP
SSSP-wln
SSSP-wlc
FPC
TSP
NB
MC

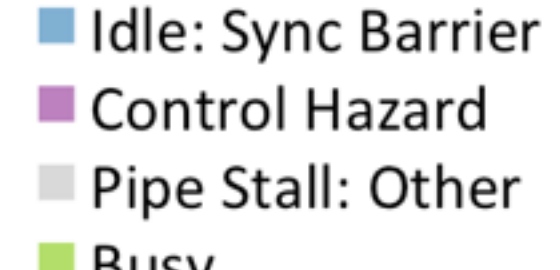
Barnes-Hut looks quite regular…

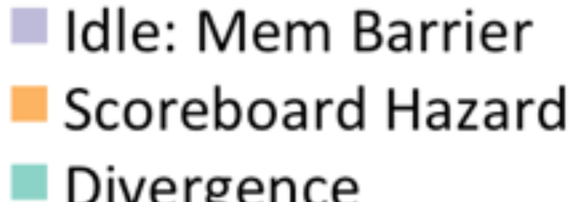
…but not all of its kernels do

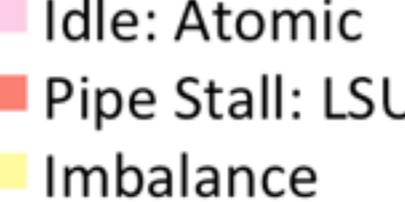
Applications: Other Irregular

Applications: Other Irregular







100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS, BFS-unroll, BFS-wlw, BFS-wlc, BH, BH(tree), DMR, MST, SP, SSSP, SSSP-wln, SSSP-wlc, FPC, TSP, NB, MC

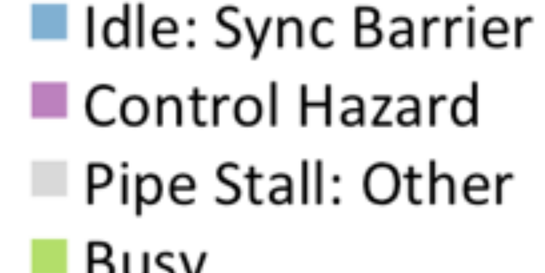
Barnes-Hut looks quite regular…

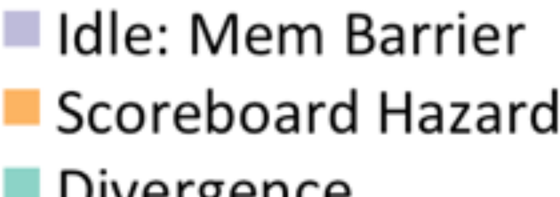
…but not all of its kernels do

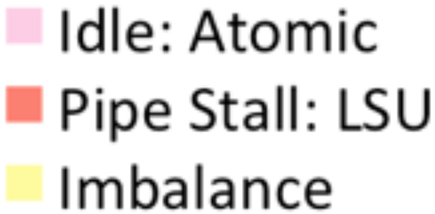
In general, less divergence penalty than expected

Synchronization/atomics penalty also somewhat below expectation

Applications: Other Irregular







100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

BFS | BFS-unroll | BFS-wlw | BFS-wlc | BH | BH(tree) | DMR | MST | SP | SSSP | SSSP-wln | SSSP-wlc | FPC | TSP | NB | MC

Barnes-Hut looks quite regular…

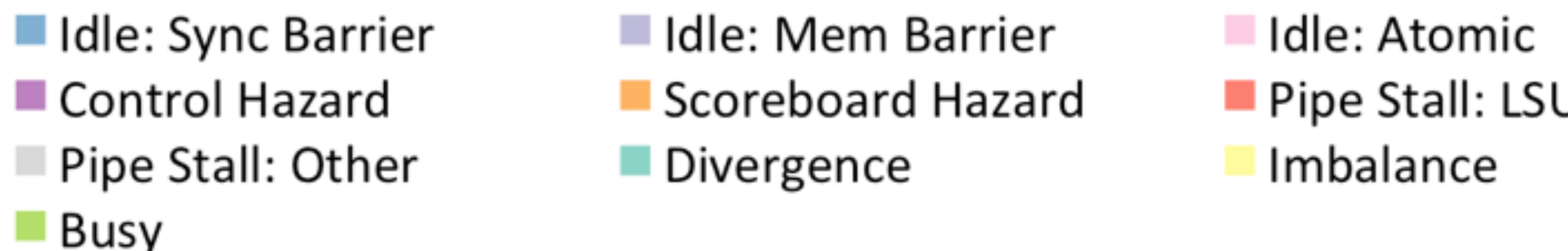
…but not all of its kernels do

In general, less divergence penalty than expected

Synchronization/atomics penalty also somewhat below expectation

Source code optimizations already addressing these bottlenecks more effectively than memory-related slowdown

Applications

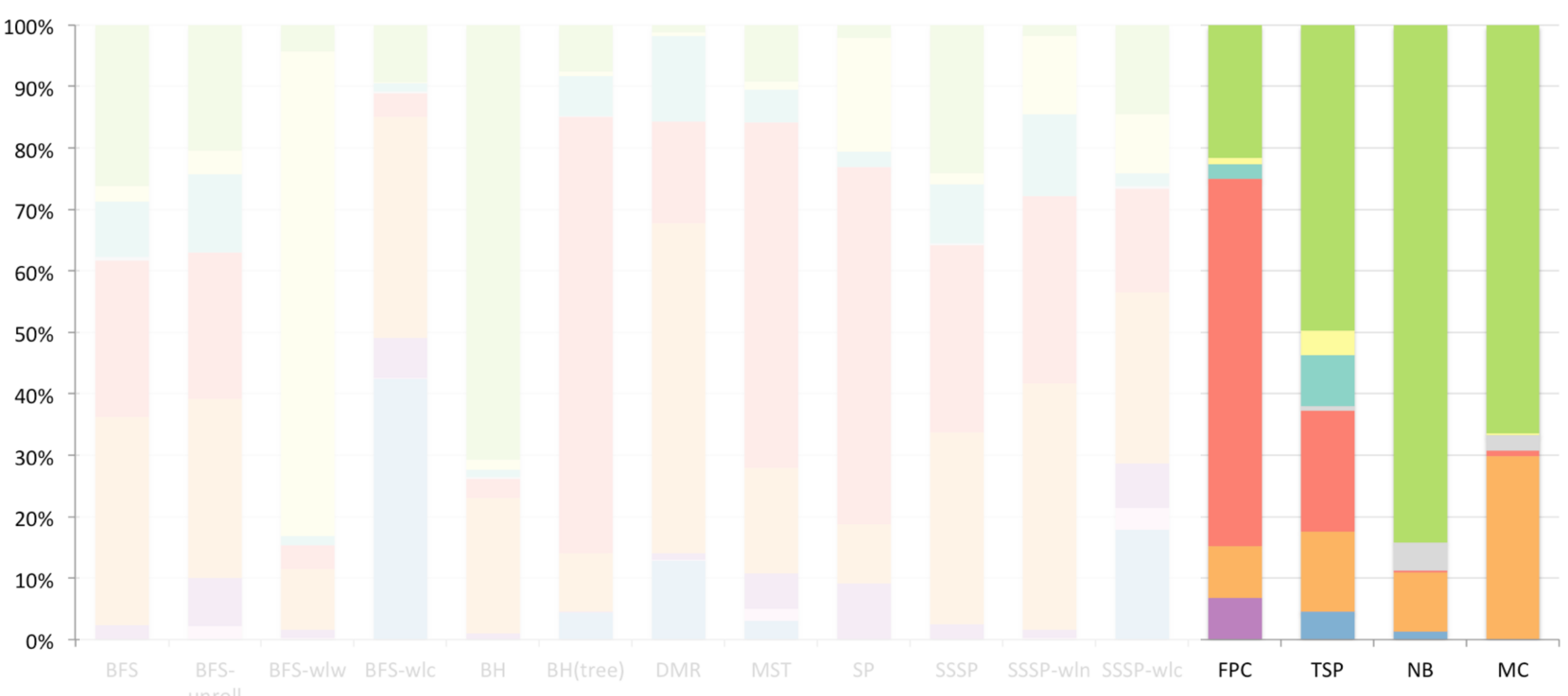


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%

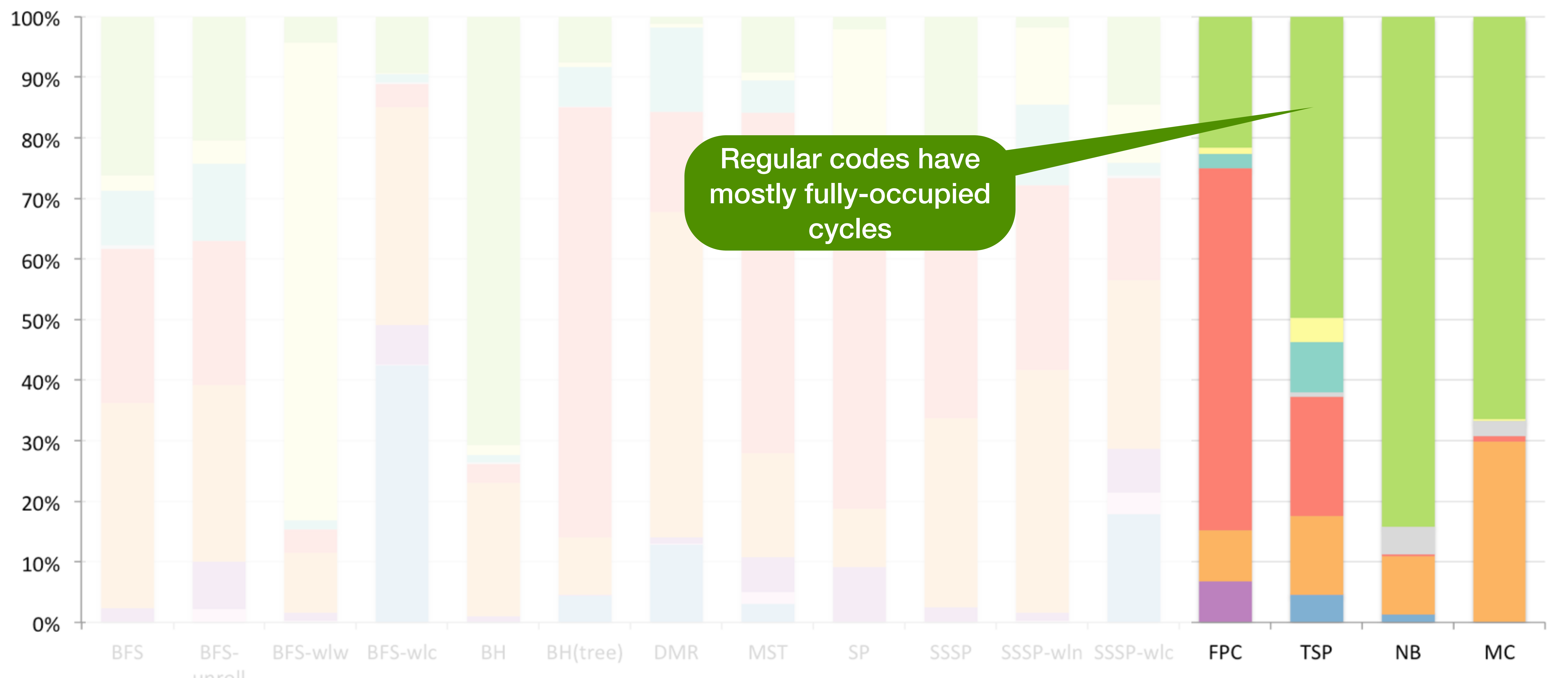
BFS
BFS-unroll
BFS-wlw
BFS-wlc
BH
BH(tree)
DMR
MST
SP
SSSP
SSSP-wln
SSSP-wlc
FPC
TSP
NB
MC

Applications: (Semi-)Regular

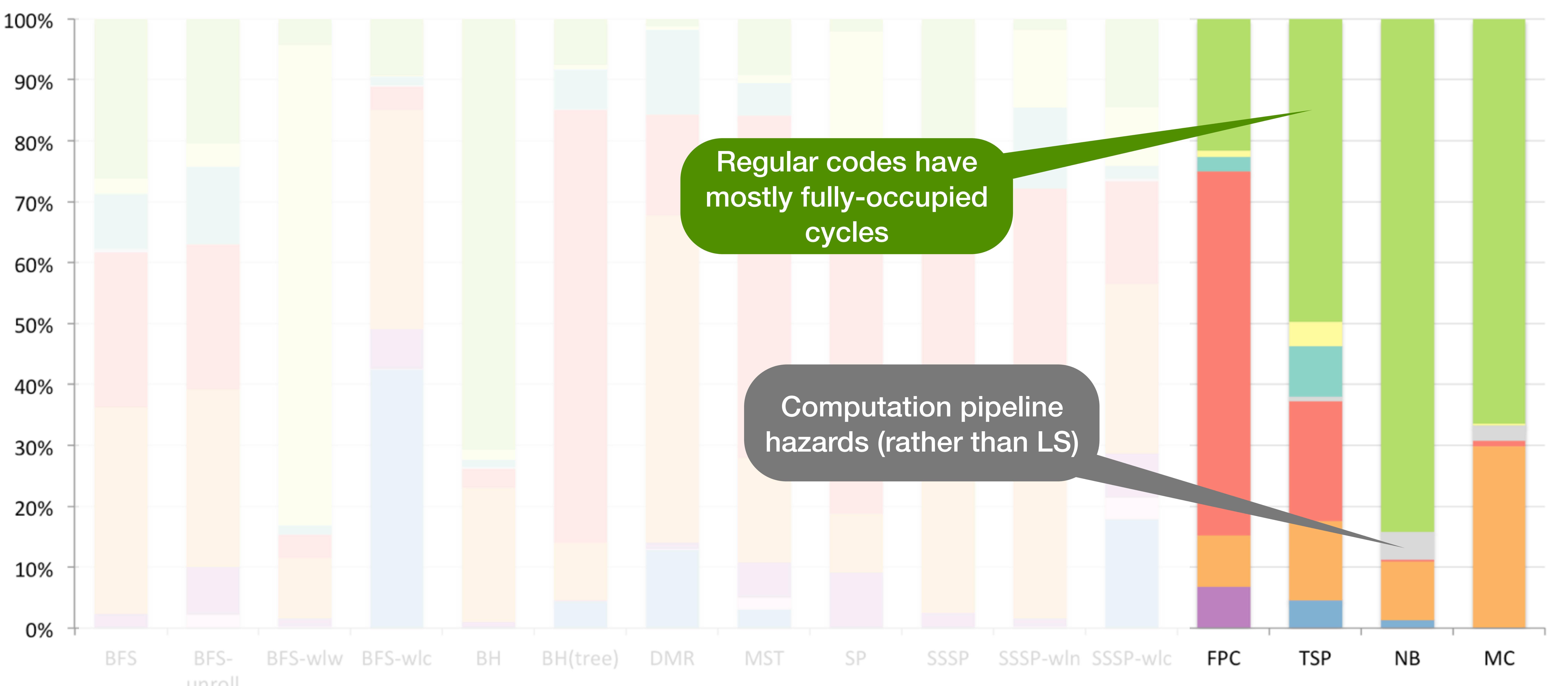
- Idle: Sync Barrier
- Idle: Mem Barrier
- Idle: Atomic
- Control Hazard
- Scoreboard Hazard
- Pipe Stall: LSU
- Pipe Stall: Other
- Divergence
- Imbalance
- Busy



Applications: (Semi-)Regular

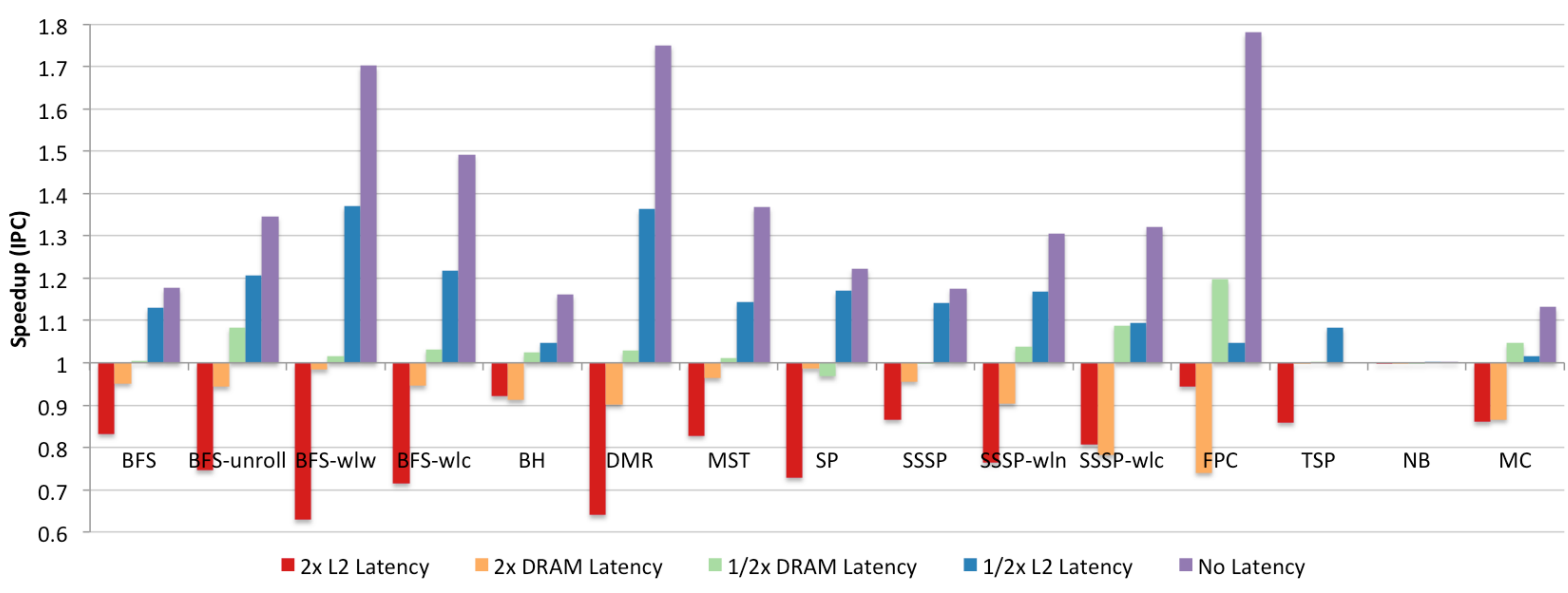


Applications: (Semi-)Regular



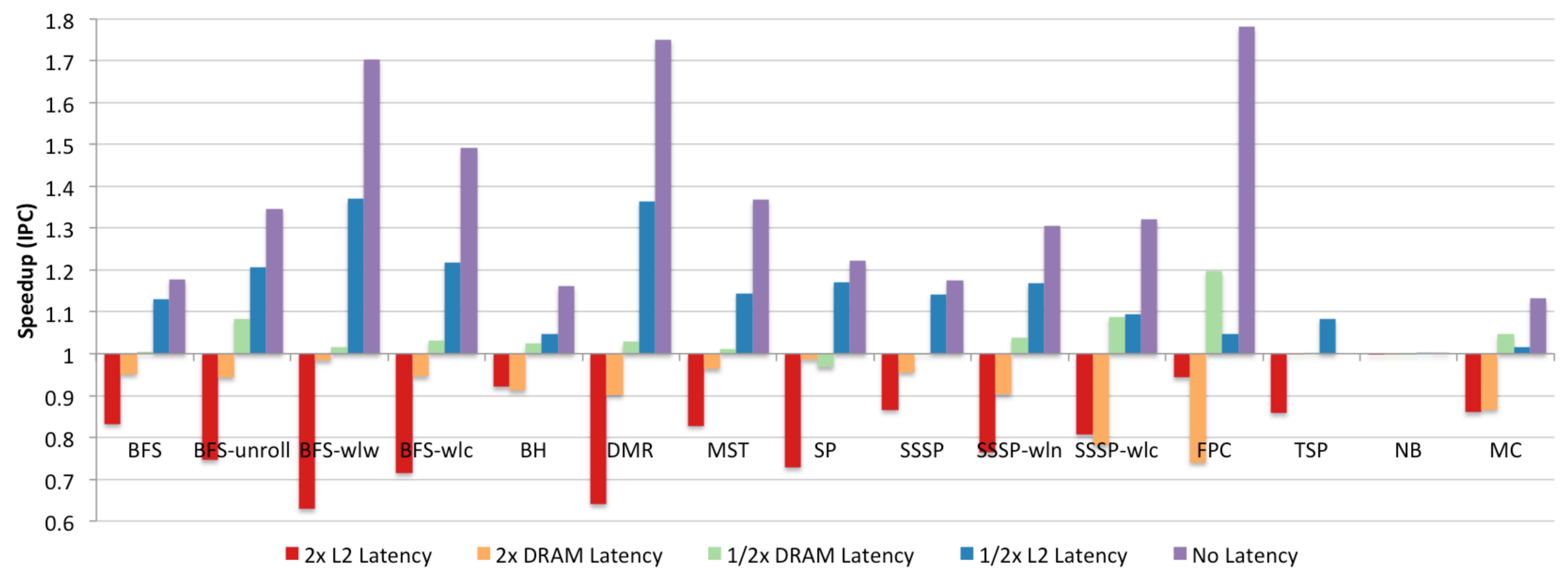
Hardware Modifications

Latency Scaling

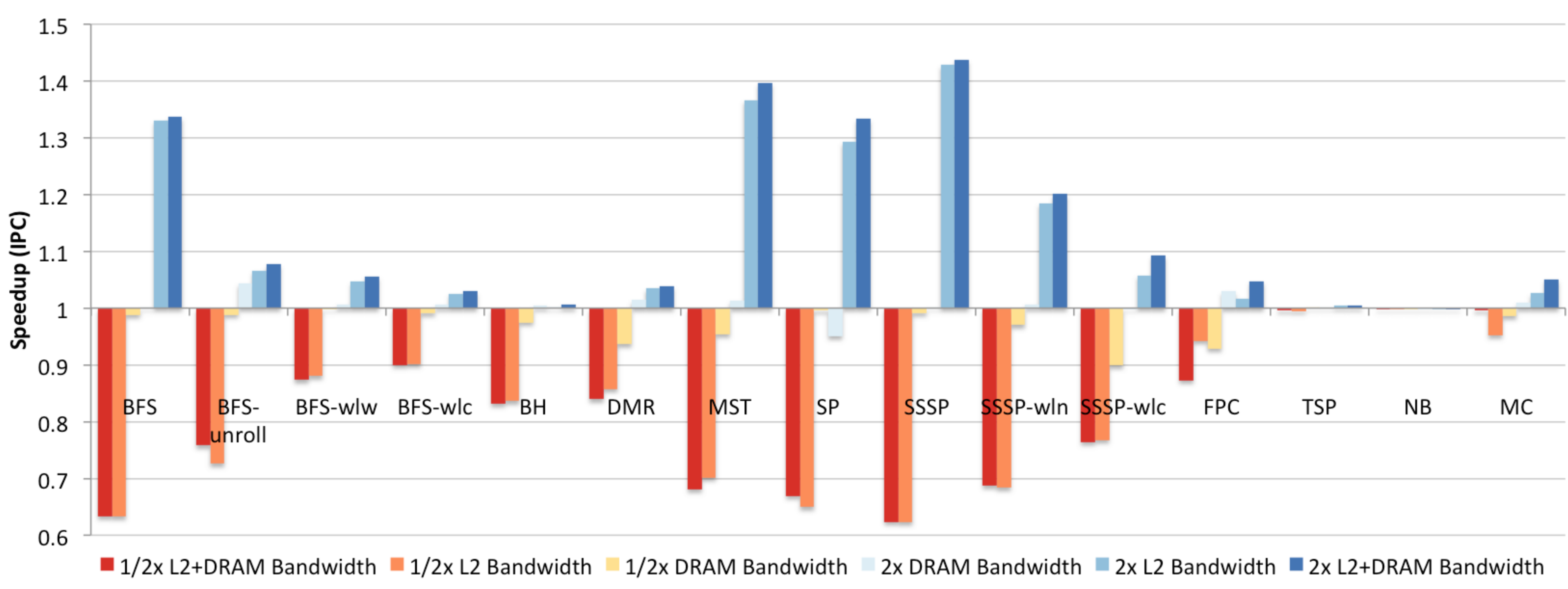


Latency Scaling

- Regular codes largely insensitive to latency
- FPC: quite sensitive to L2 latency (streaming code, high spatial locality)
- Overall, *L2 latency appears more important than DRAM latency*
  - Even for inputs with working set sizes several times larger than the L2

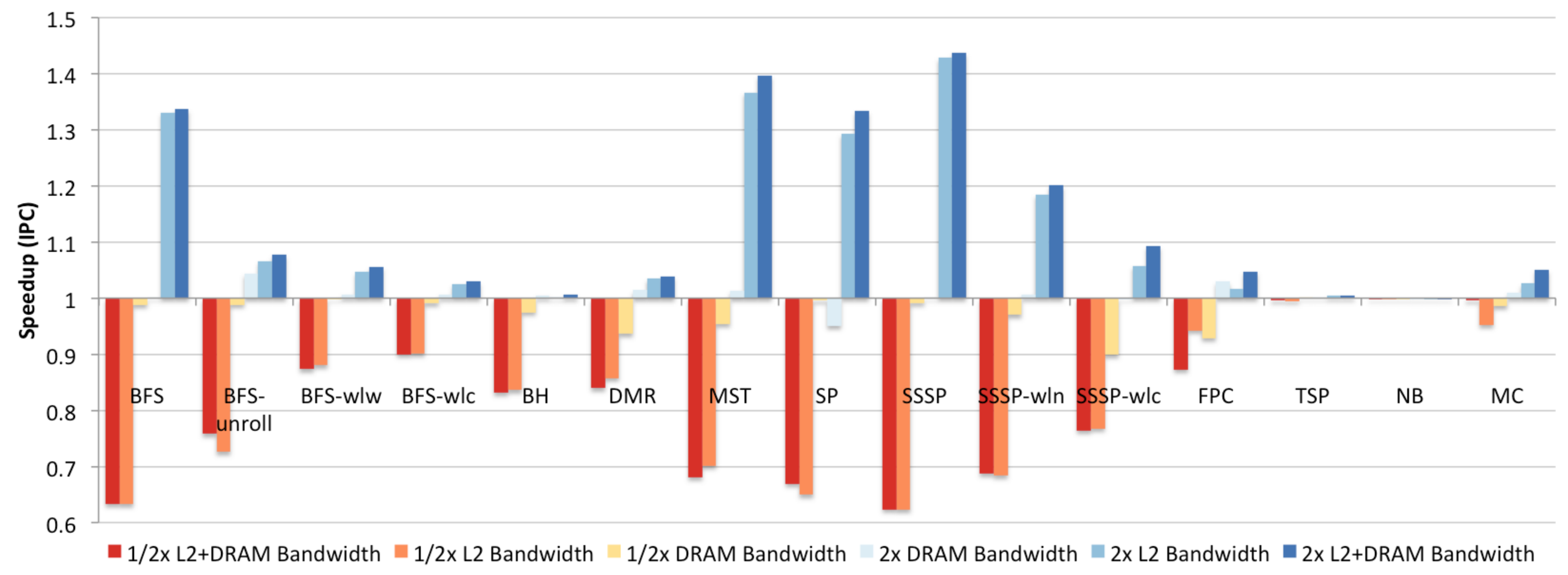


Bandwidth Scaling

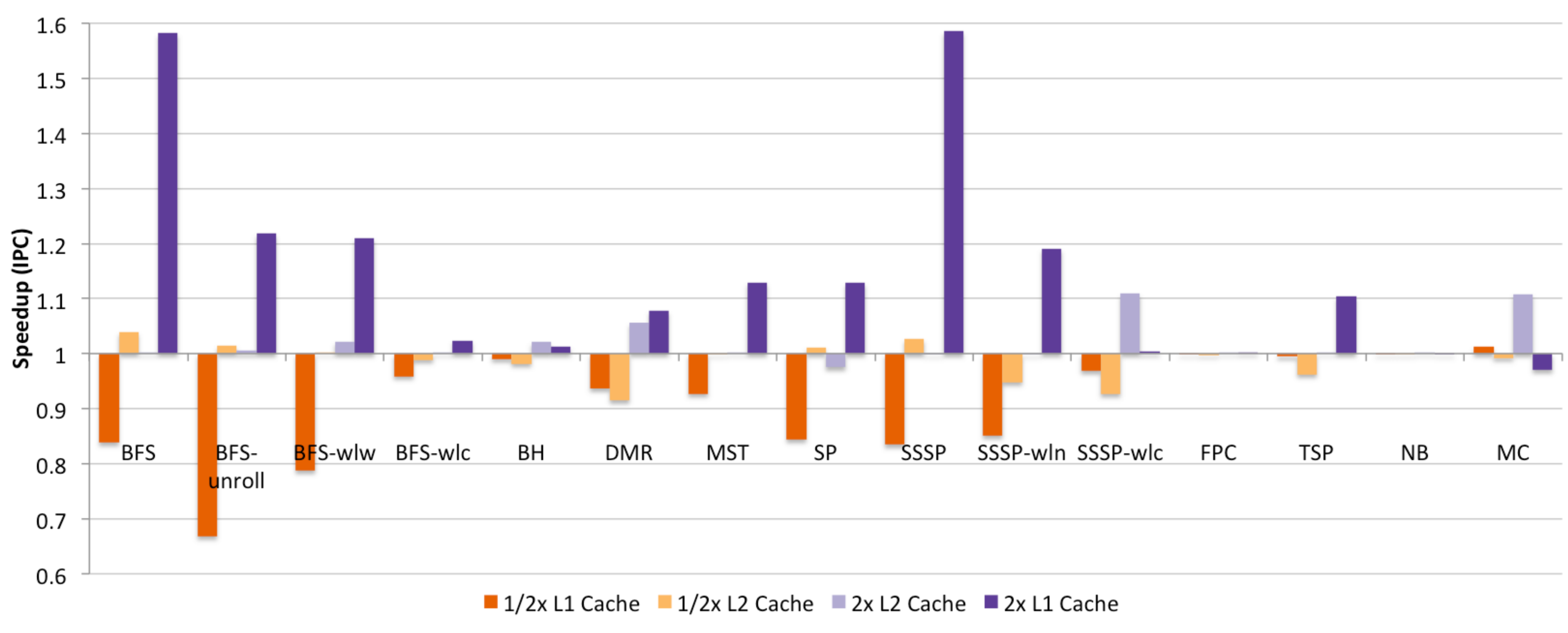


Bandwidth Scaling

- Similarly to latency results, most applications are much *more sensitive to L2 bandwidth than to DRAM bandwidth*
  - Regular/vector codes largely unaffected by bandwidth scaling
- For tested inputs, the L2 is large enough that sufficient L2 bandwidth keeps enough warps able to execute

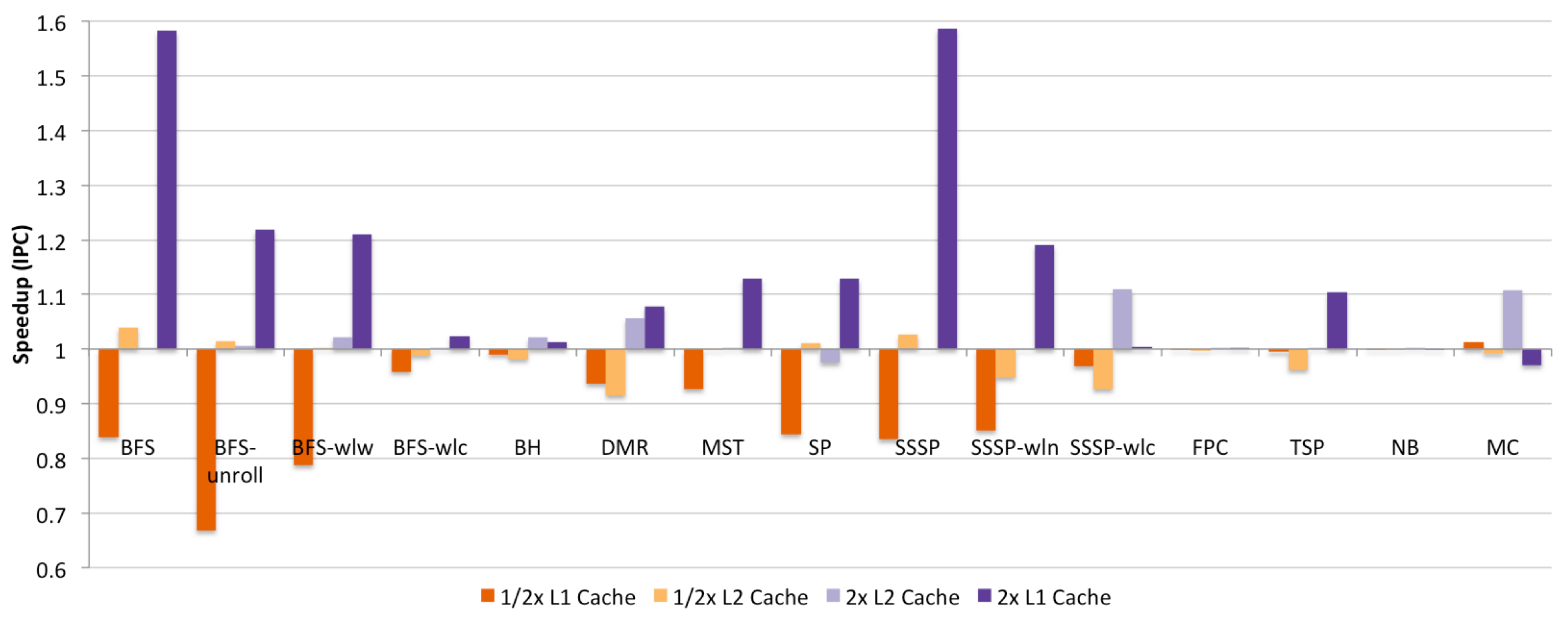


Cache Size Scaling



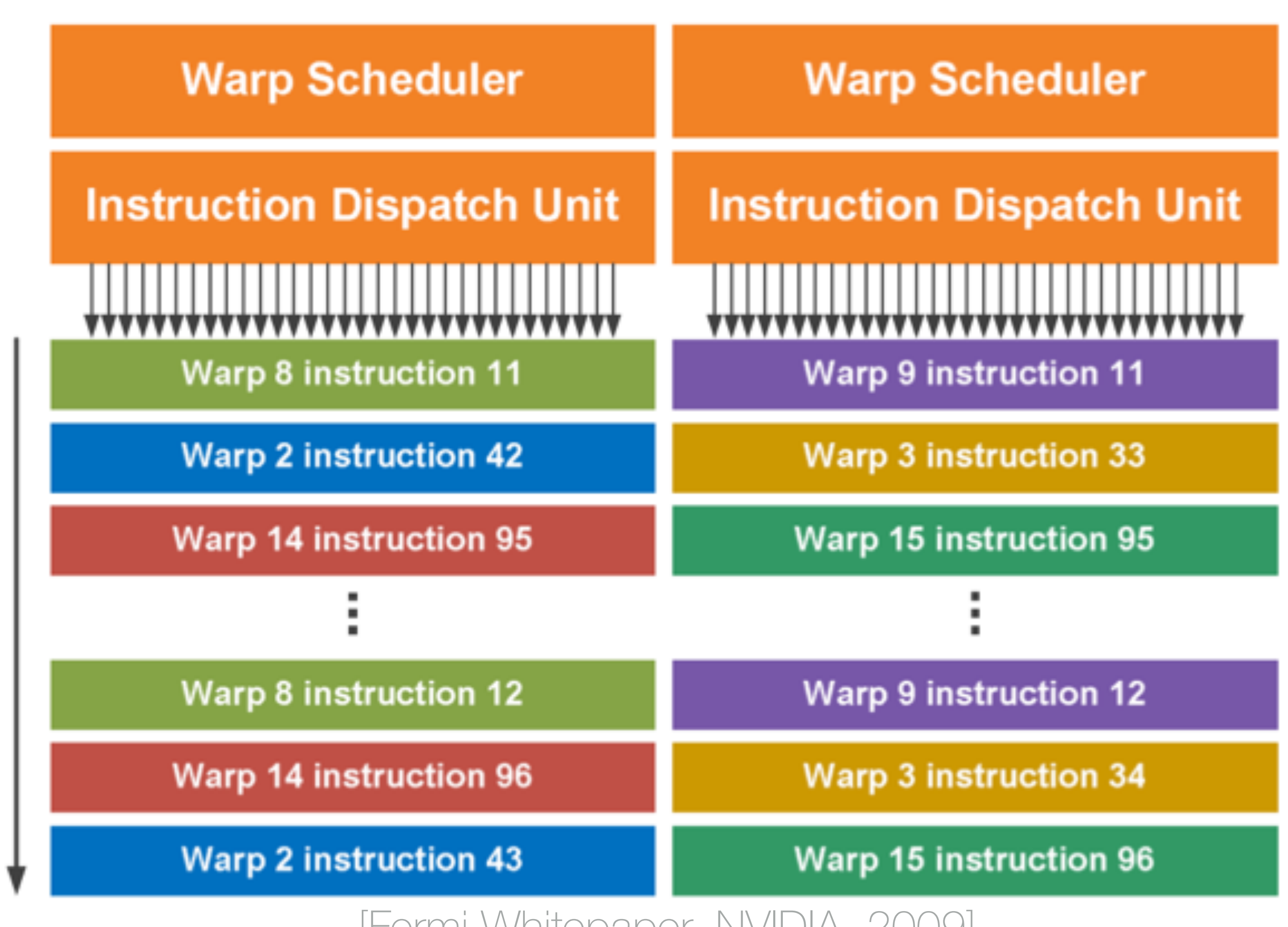
Cache Size Scaling

- Codes sensitive to L2 bandwidth are also sensitive to L1D size
- Most irregular codes *hurt more by a smaller L1 than a smaller L2*
  - Regular codes are the opposite, but the effect is much less pronounced



Warp Scheduling

- GPUs cannot hide latency without multiple warps from which to issue on every SM
  - Multithread instructions from inflight warps
  - If a warp encounters a long operation (e.g., RAW hazard on load data or stall), SM can issue from another warp instead
  - If no other warp can issue its next instruction → underutilization



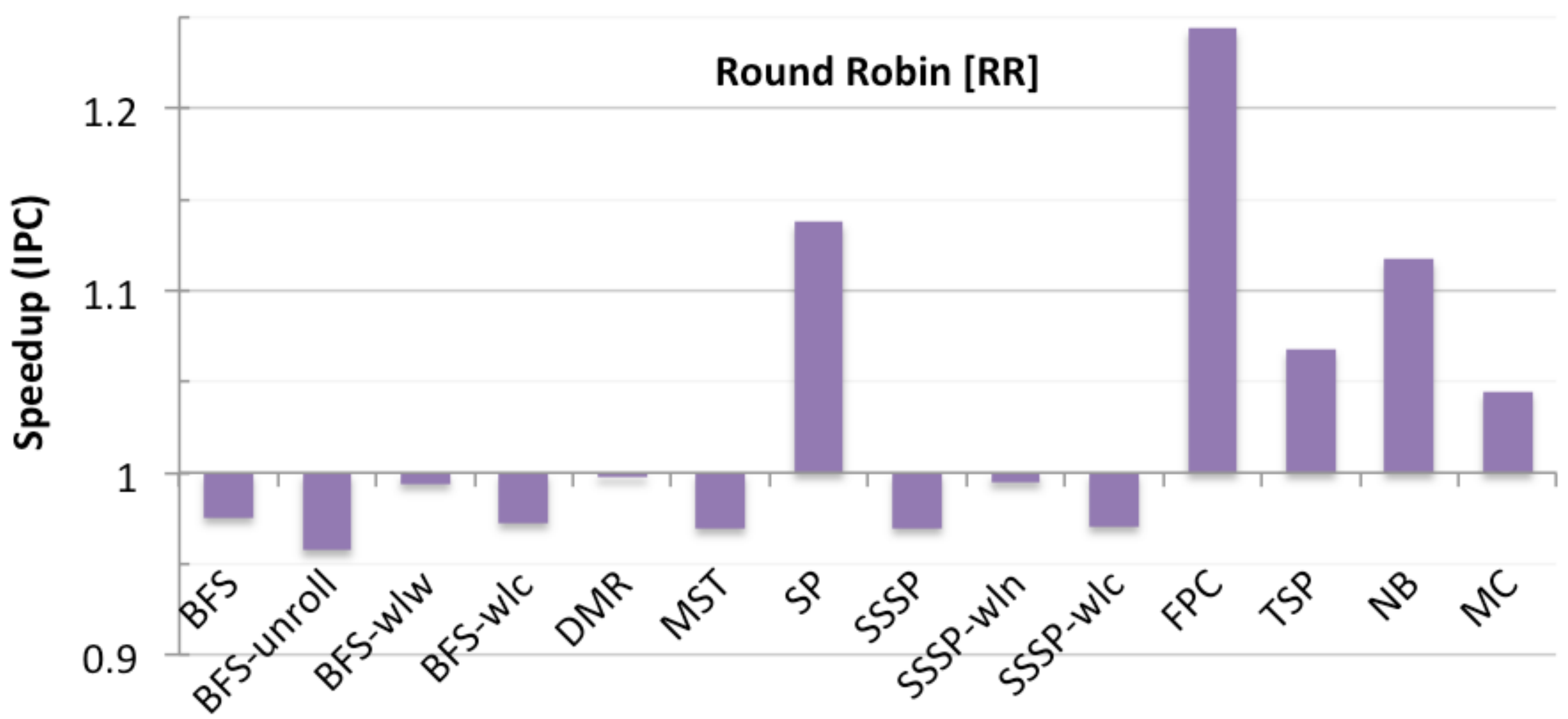
[Fermi Whitepaper, NVIDIA, 2009]

Warp Scheduling

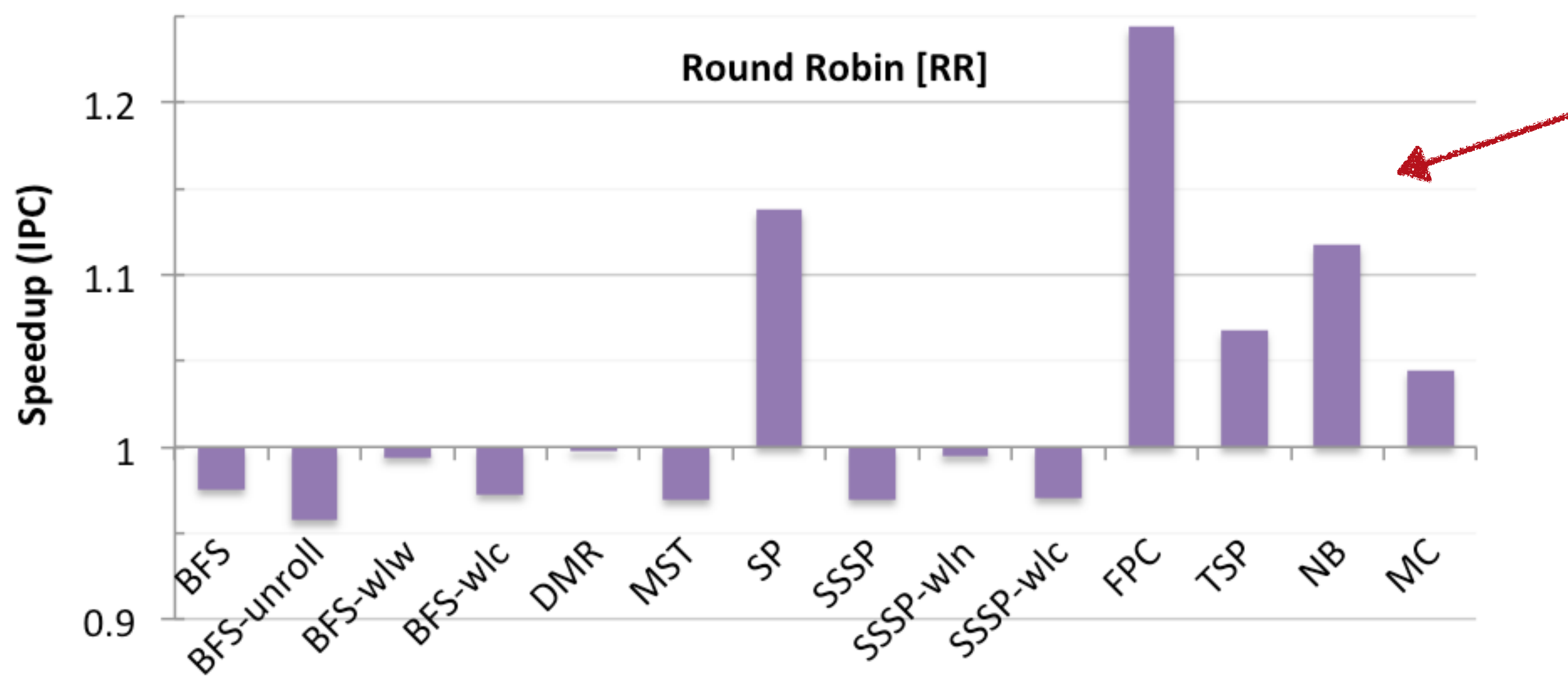
- Selection policy to choose next warp to issue can significantly impact GPU's ability to hide latencies
  - *Round Robin (RR)*
  - *Greedy-Then-Oldest (GTO)* ← GPGPU-Sim default
  - *Two-Level Active Scheduler*

- Round-robin schedulers: Good for preserving inter-warp locality, but warps tend to arrive at long-latency operations in close time proximity
- Greedy schedulers: Lose memory access locality as warps run progressively out-of-sync, but mitigate the all-stall-together issue

Warp Scheduling

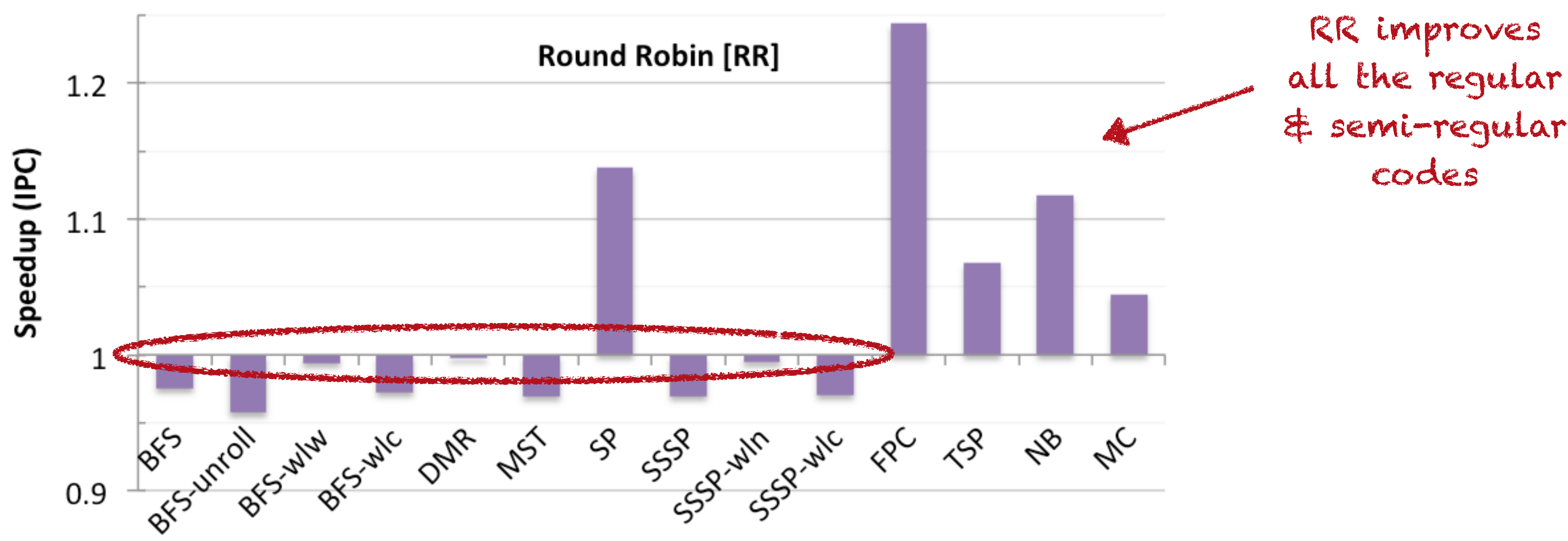


Warp Scheduling



RR improves all the regular & semi-regular codes

Warp Scheduling



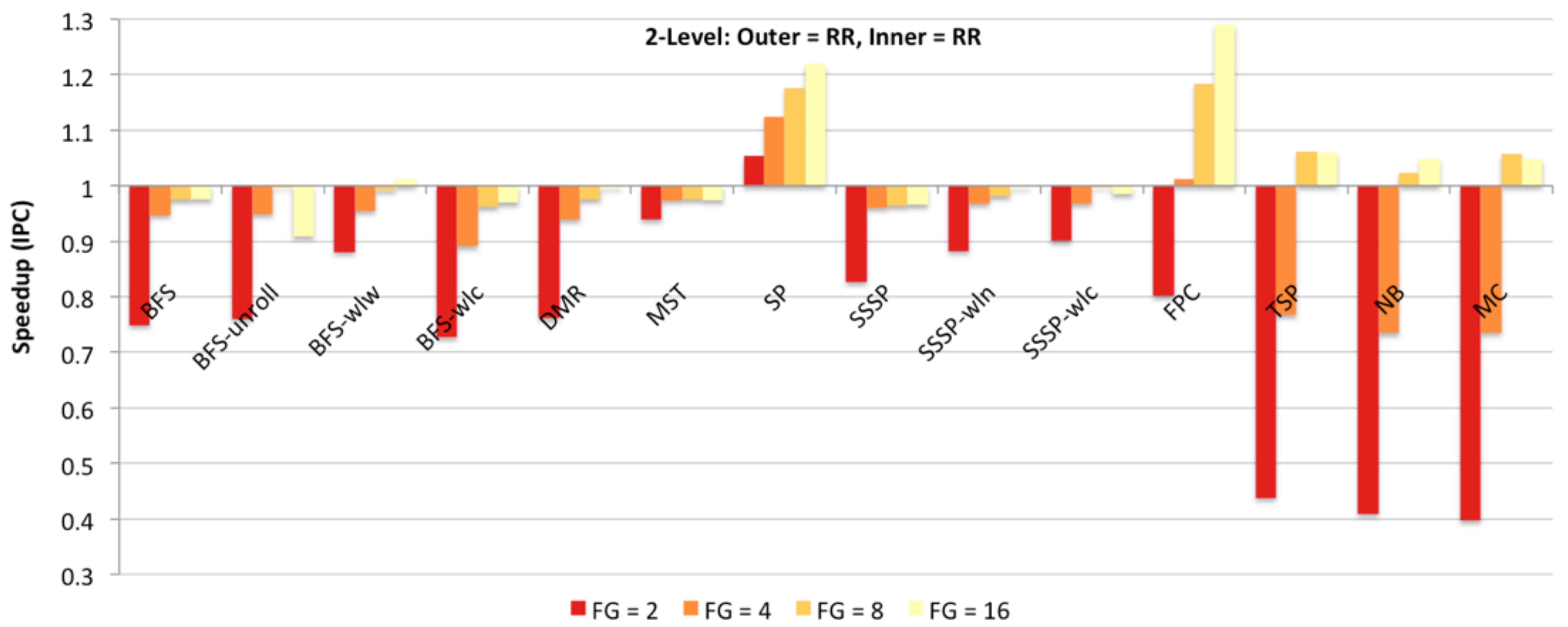
- *GTO scheduling superior for irregular codes*, which often possess little inter-warp locality

Warp Scheduling

- **Two-level scheduling** (Narasiman et al., MICRO'11) splits active warps on each SM into fetch groups (FGs)
  - Prioritize issue from single FG until stall
  - Designed to balance pros and cons of greedy vs. RR scheduling

Warp Scheduling

- **Two-level scheduling** (Narasiman et al., MICRO'11) splits active warps on each SM into fetch groups (FGs)
  - Prioritize issue from single FG until stall
  - Designed to balance pros and cons of greedy vs. RR scheduling

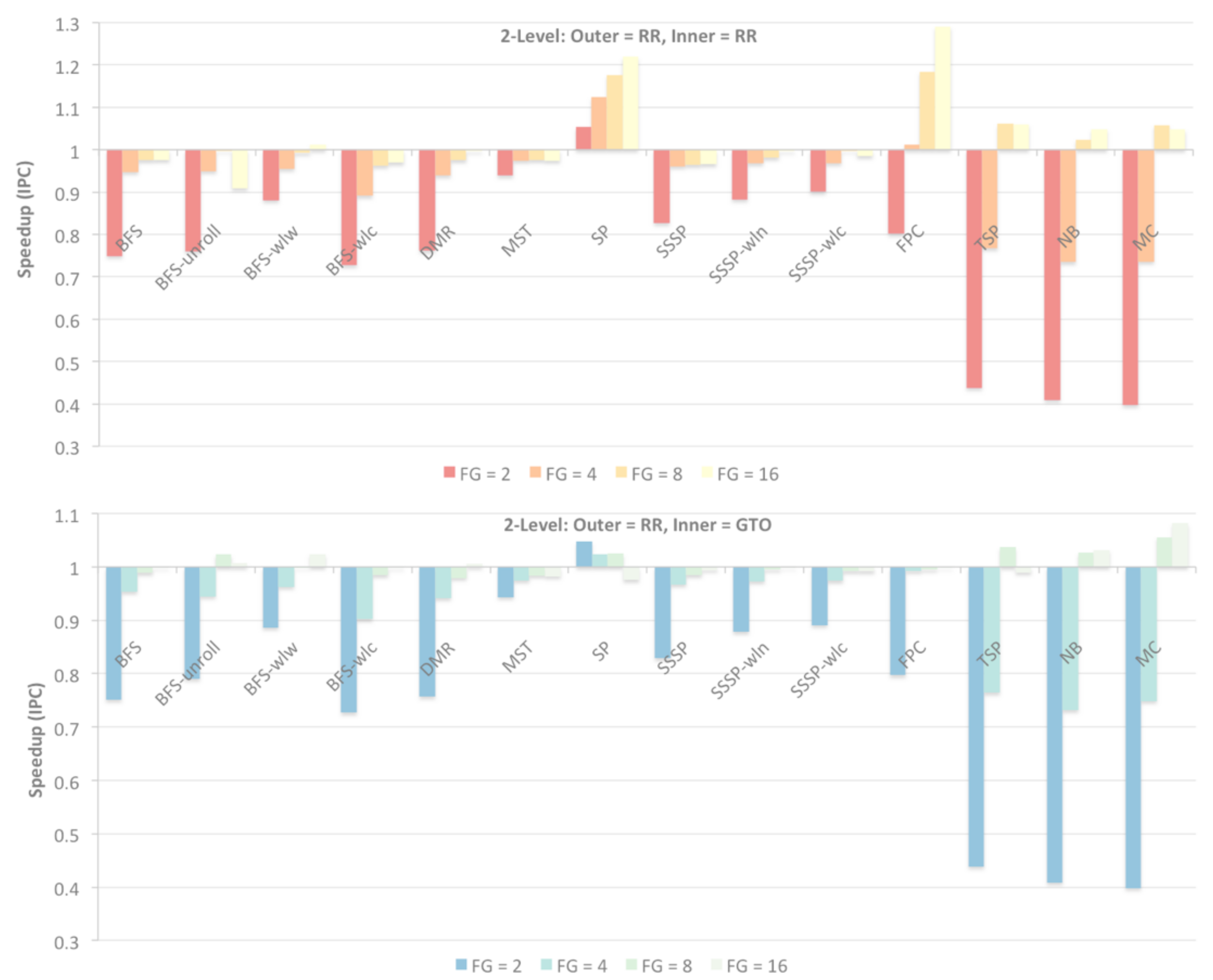


Warp Scheduling

- **Two-level scheduling** (Narasiman et al., MICRO'11) splits active warps on each SM into fetch groups (FGs)
  - Prioritize issue from single FG until stall
  - Designed to balance pros and cons of greedy vs. RR scheduling

Warp Scheduling

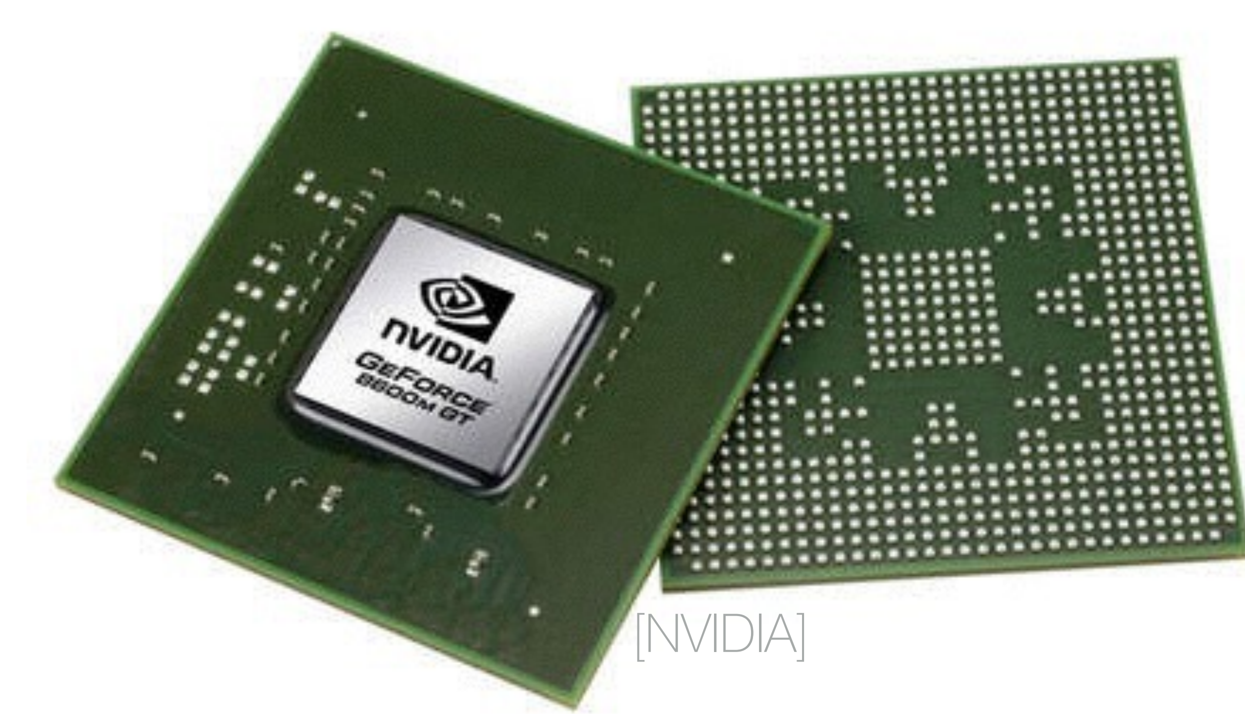
- **Two-level scheduling** (Narasiman et al., MICRO'11) splits active warps on each SM into fetch groups (FGs)
  - Prioritize issue from single FG until stall
  - Designed to balance pros and cons of greedy vs. RR scheduling
- *Appears ineffective for irregular codes*



Conclusions

Recap

- First microarchitectural, cycle-level-simulation-based workload characterization focusing on irregular GPU kernels



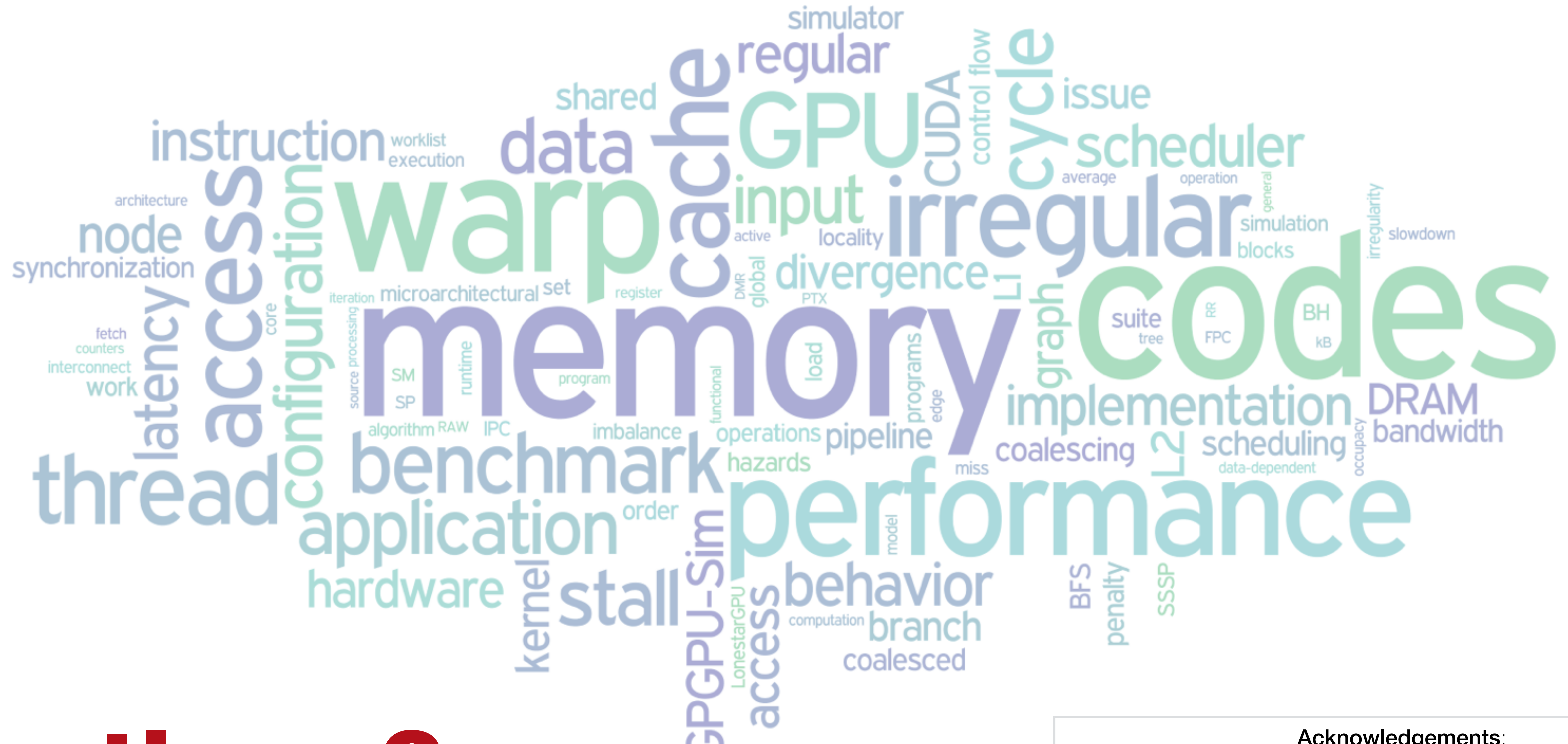
[NVIDIA]

- Findings
  - Irregular codes have more load imbalance, branch divergence, and uncoalesced memory accesses than regular codes
  - For most applications, less branch divergence, atomics, and synchronization penalty than expected
    - Software designers successfully addressing these performance issues

Key Takeaways for GPU Architects

- *Improved memory and last-level cache latency and bandwidth, enhanced cache effectiveness* are the most important factors for supporting irregular codes on GPUs
    - Improving *L2 latency/bandwidth appears more important* than improving DRAM latency/bandwidth
    - Strategies to reduce coalescing pipeline penalty unlikely to help without corresponding increases in memory bandwidth
- Simple warp schedulers (e.g., two-level active scheduling) that improve regular codes are ineffective for irregular codes
    - *Greedy scheduling* is the best simple strategy for these codes
    - Addressing slowdown via warp scheduling likely to require more complex schedulers

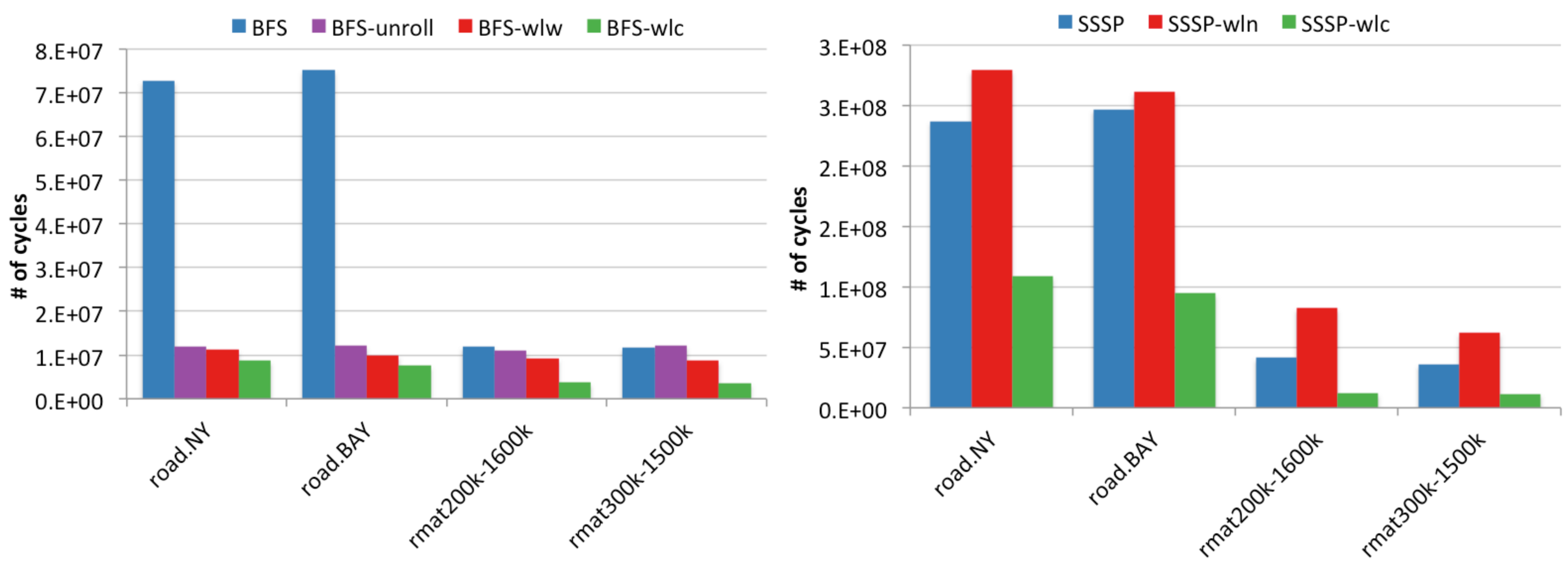
THANK YOU!



Questions?

Acknowledgements:
NSF Graduate Research Fellowship Program (grant 1144466)
NSF grants 1141022, 1217231, 1406304, and 1438963
Grants and gifts from NVIDIA Corporation

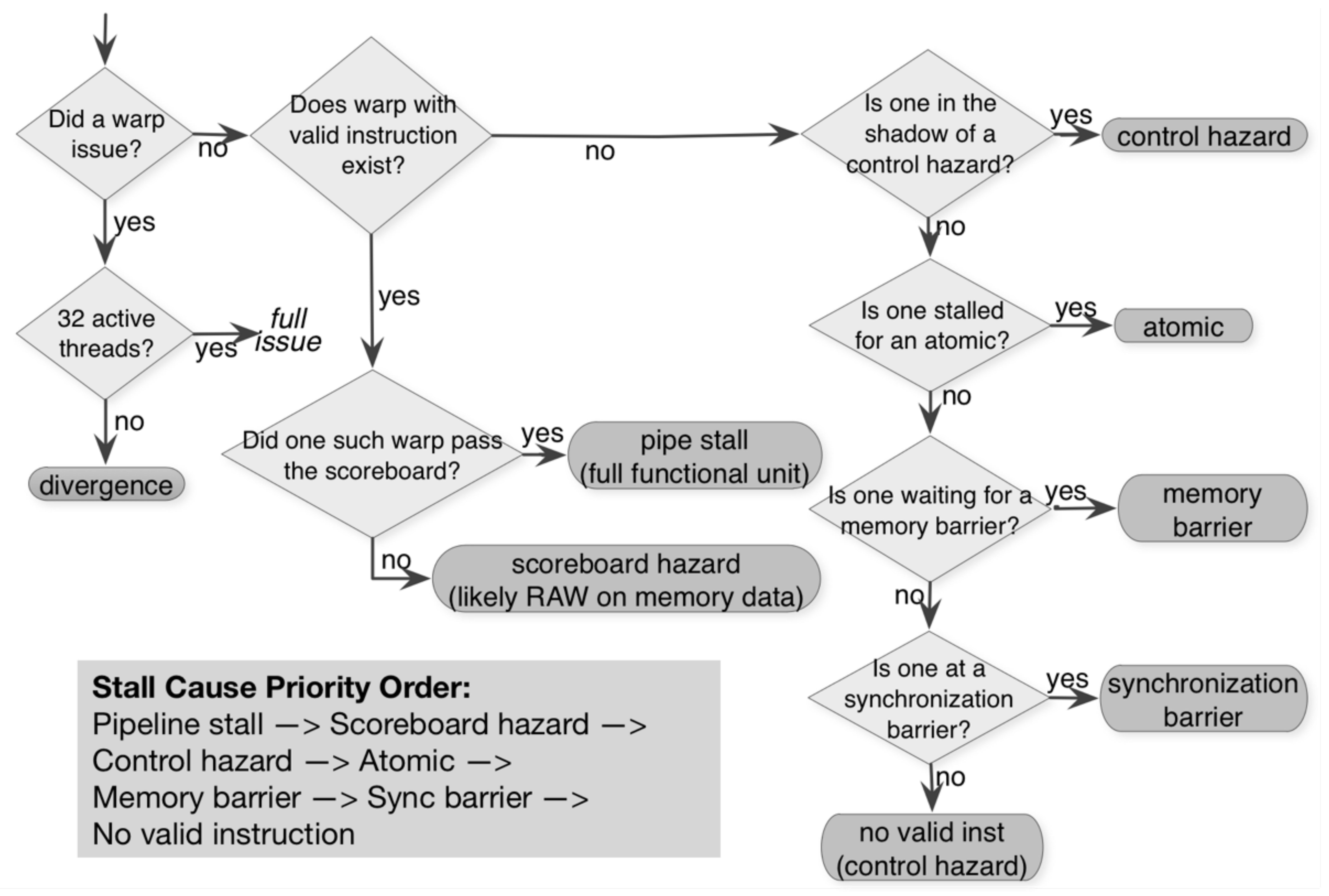
IPC vs. Runtime



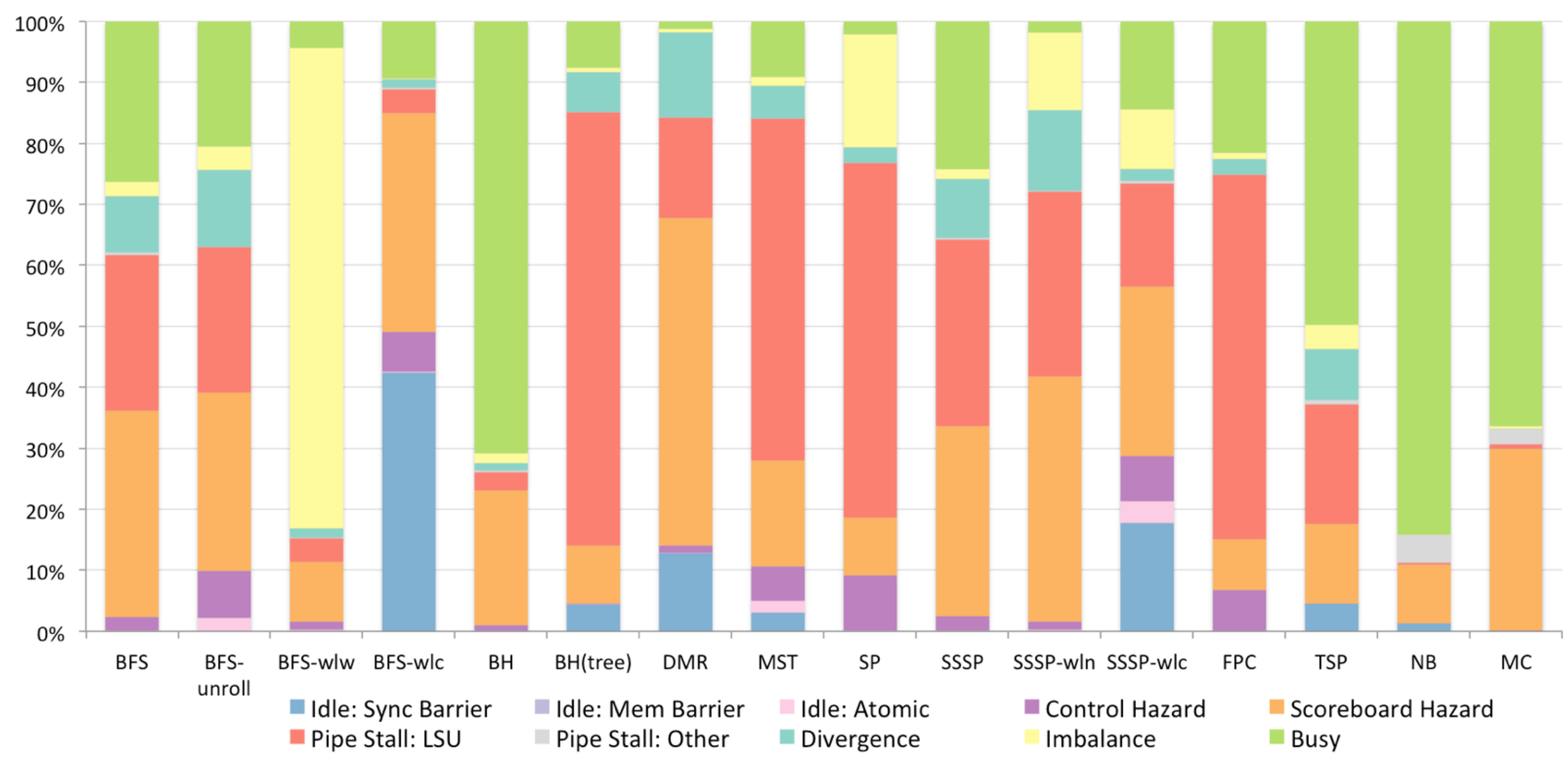
Input Sizes

Application	Name	Description	Working Set	L2 Size Multiplier
BFS, SSSP	USA_road_d.NY	NY roads (264K nodes, 734K edges)	3899 kB	5.08
	USA_road_d.BAY	SF Bay Area roads (321K nodes, 800K edges)	4380 kB	5.70
	rmat200k-1600k	R-MAT (200K nodes, 1600K edges)	7031 kB	9.16
	rmat264k-734k	R-MAT (264K nodes, 734K edges)	3898 kB	5.08
BH	494,000 1 (seed=7)	494K bodies, 1 timestep	7718 kB	10.05
	494,000 1 (seed=1)	494K bodies, 1 timestep	7718 kB	10.05
DMR	massive.2	100.3K triangles, maxfactor=10	7840 kB	10.21
	30k	60K triangles, maxfactor=10	4688 kB	6.10
	25k	50K triangles, maxfactor=10	3906 kB	5.09
MST	USA_road_d.NY	NY roads (264K nodes, 734K edges)	3898 kB	5.08
	USA_road_d.BAY	SF Bay Area roads (321K nodes, 800K edges)	4380 kB	5.70
	rmat30k-250k	R-MAT (30K nodes, 250K edges)	1093 kB	1.42
SP	random-4200-1000-3-seed23.cnf	4.2K clauses, 1K literals, 3 literals/clause	414 kB	0.54
	random-4200-1000-3-seed27.cnf	4.2K clauses, 1K literals, 3 literals/clause	414 kB	0.54
	random-4200-1000-3-seed71.cnf	4.2K clauses, 1K literals, 3 literals/clause	414 kB	0.54
FPC	obs_error	60 MB dataset, 30 blocks, 24 warps/block, dimensionality=24	60 MB	78.12
	num_plasma	34 MB dataset, 30 blocks, 24 warps/block, dimensionality=2	34 MB	44.27
	msg_lu	X MB dataset, 30 blocks, 24 warps/block, dimensionality=5	186 MB	242.19
TSP	att48.tsp 15,000	48 cities, 15K restarts	9 kB	0.01
	eil51.tsp 15,000	51 cities, 15K restarts	10 kB	0.01
	pr76.tsp 20,000	76 cities, 20K restarts	23 kB	0.03
NB	23,040 1 (seed=7)	23,040 bodies, 1 timestep	360 kB	0.47
	23,040 1 (seed=19)	23,040 bodies, 1 timestep	360 kB	0.47
	23,040 1 (seed=43)	23,040 bodies, 1 timestep	360 kB	0.47
MC	(default)	SDK input w/ 262,144 paths	1024 kB	1.33

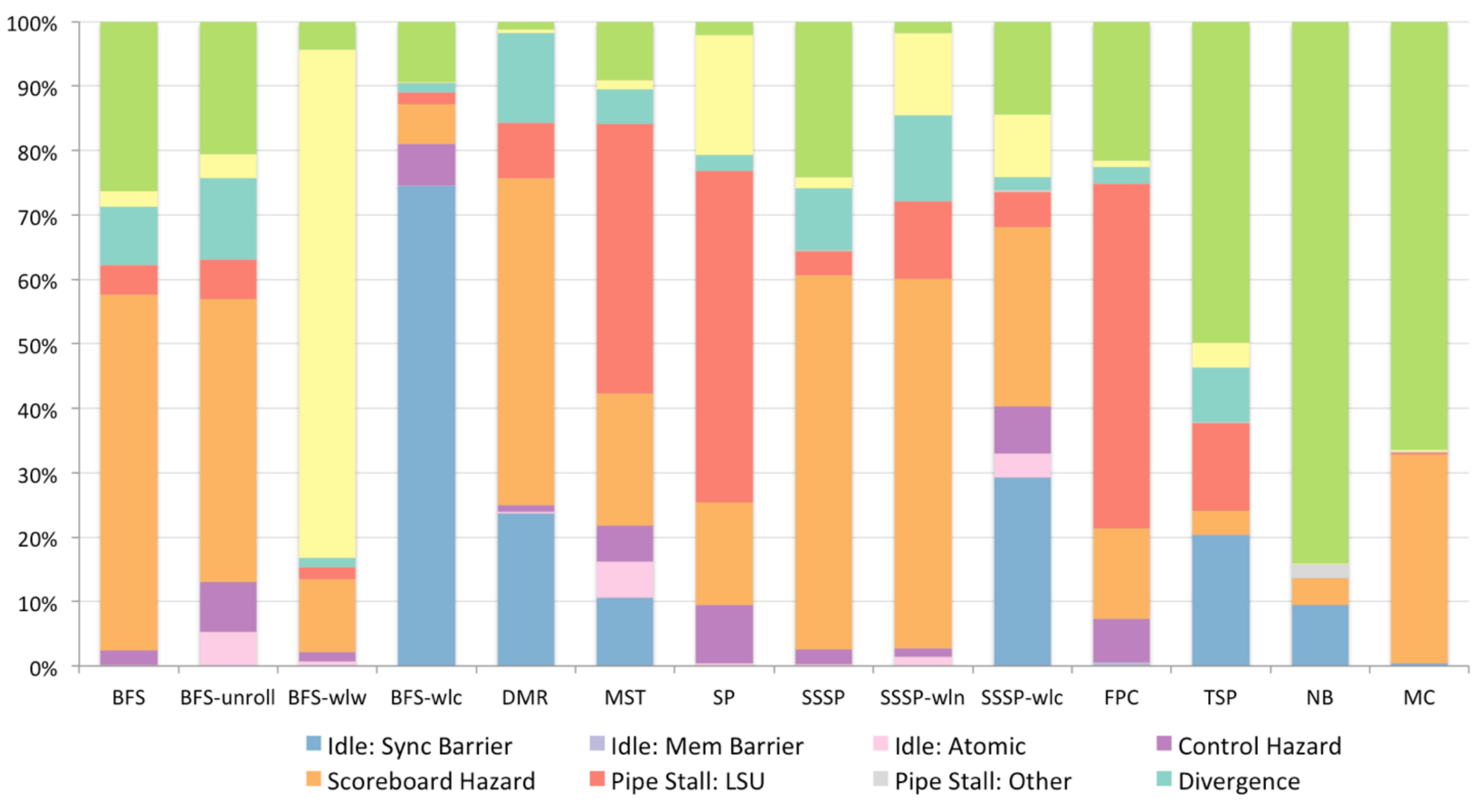
Stall Cause Prioritization



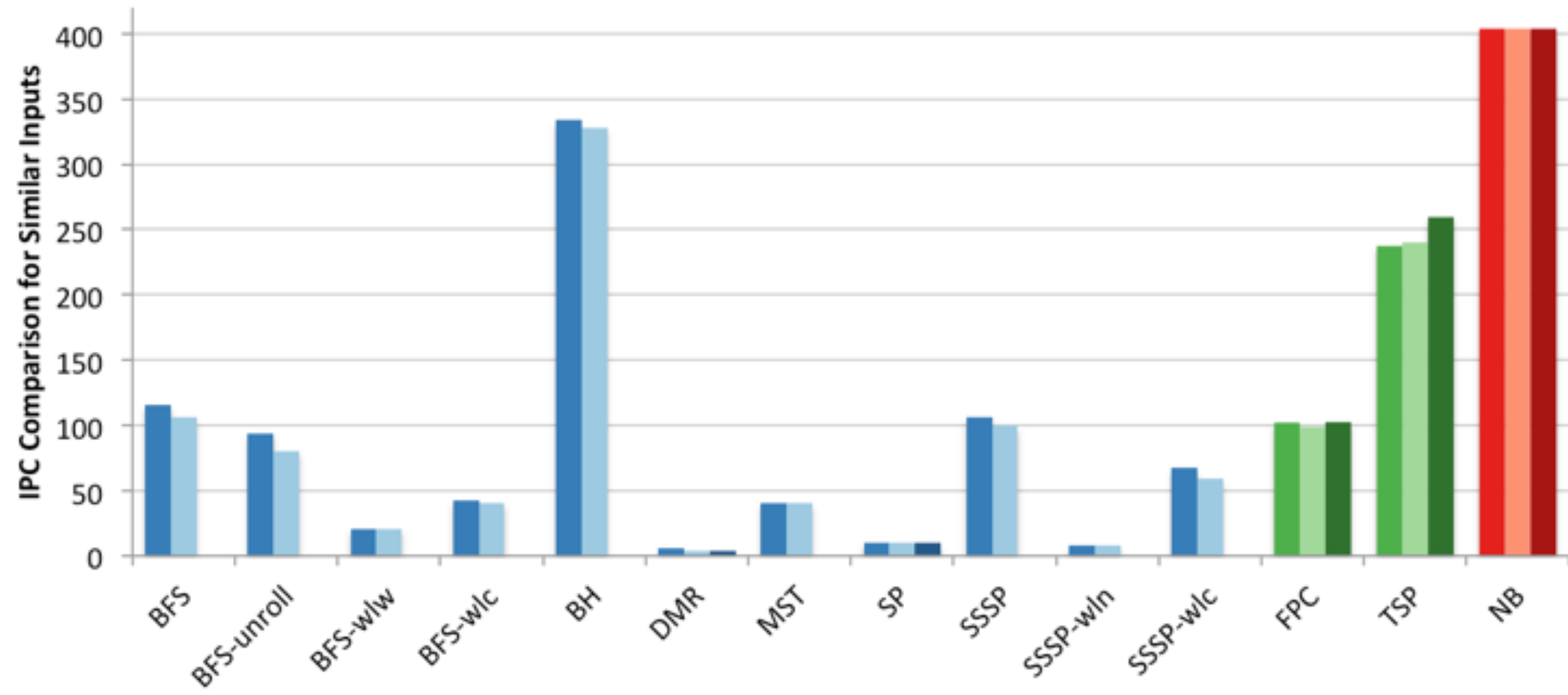
Histogram: Deepest Pipeline Stage

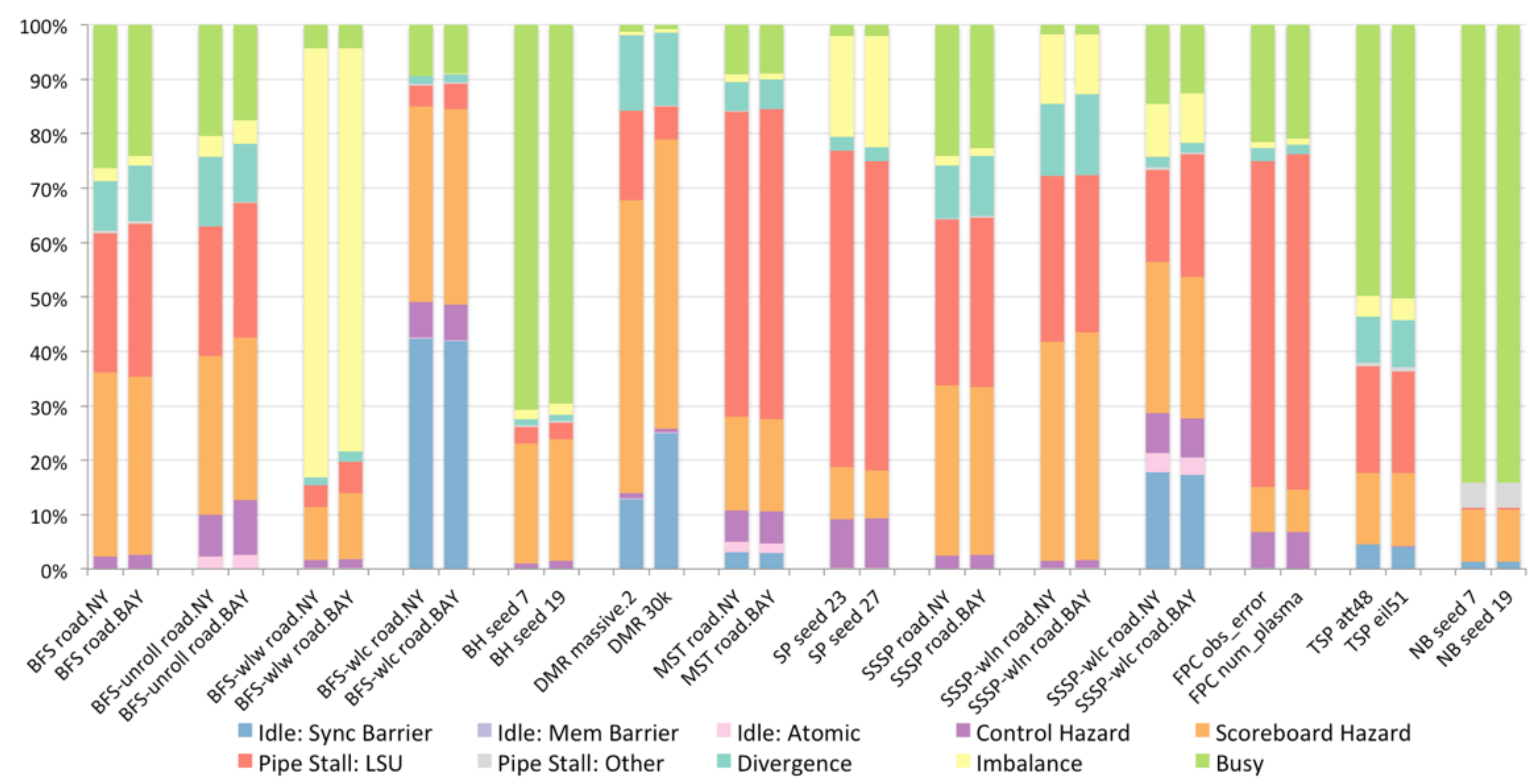


Histogram: Most Impacted Warps

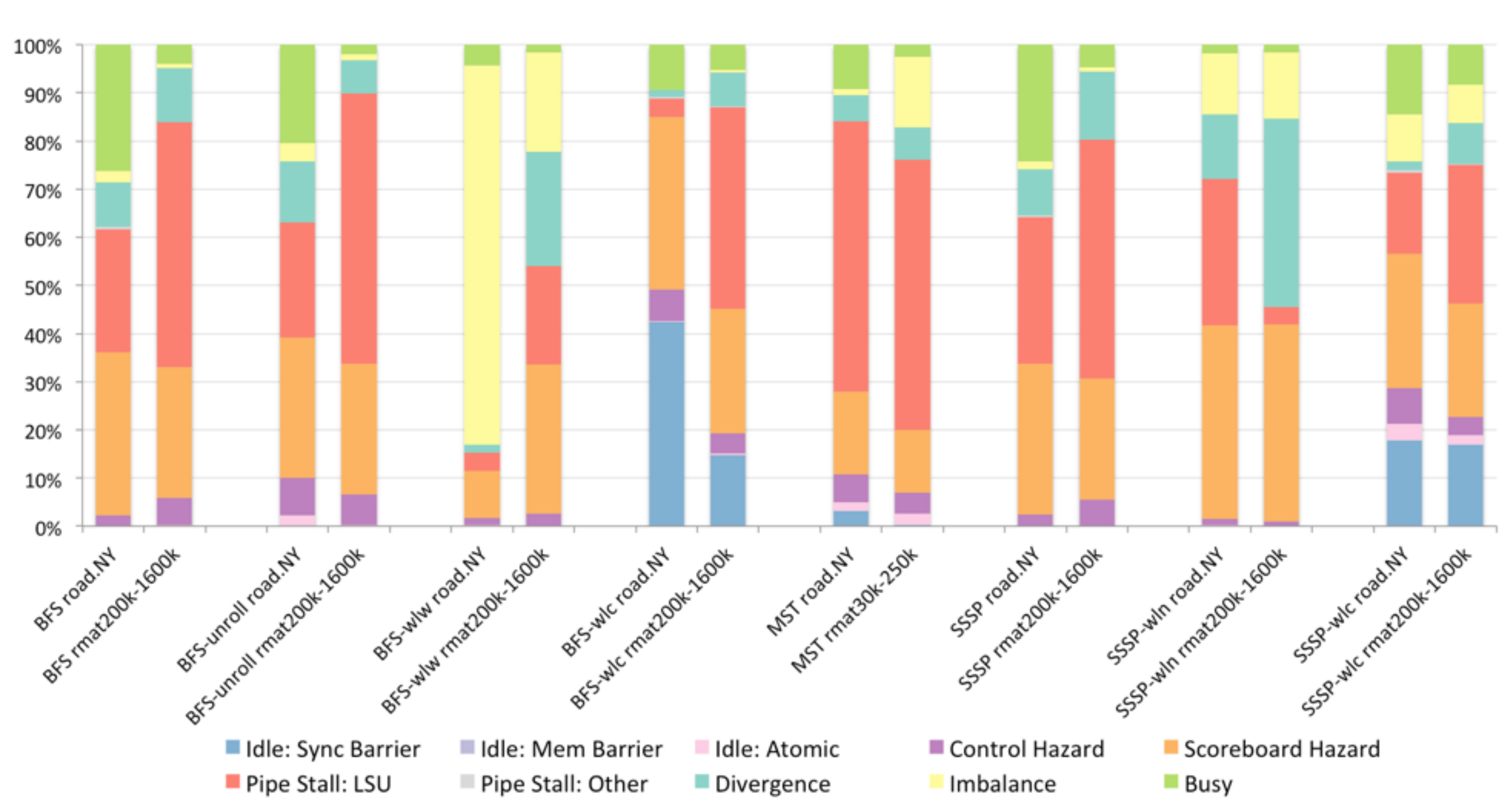


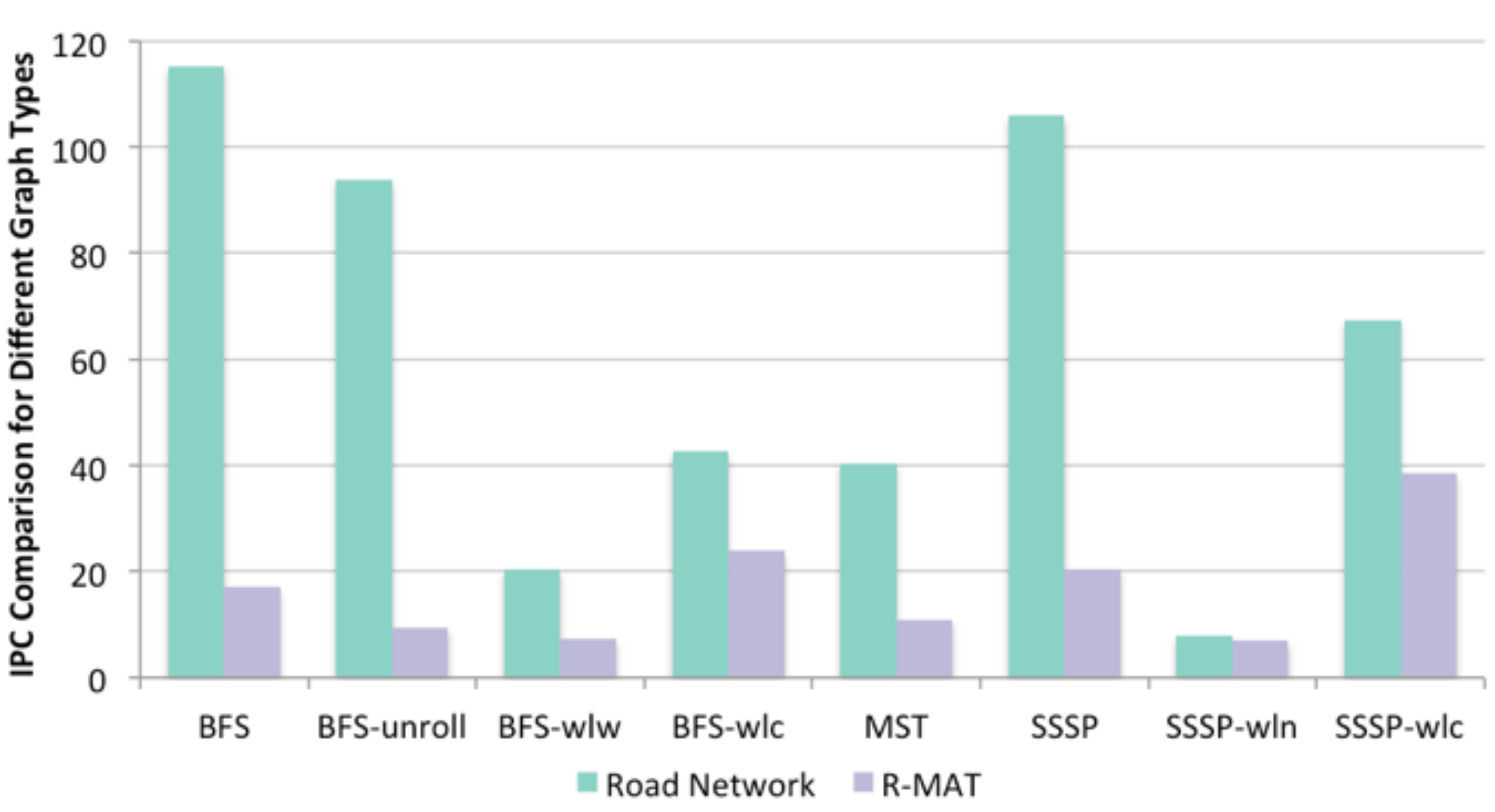
Input Variation: Similar Inputs





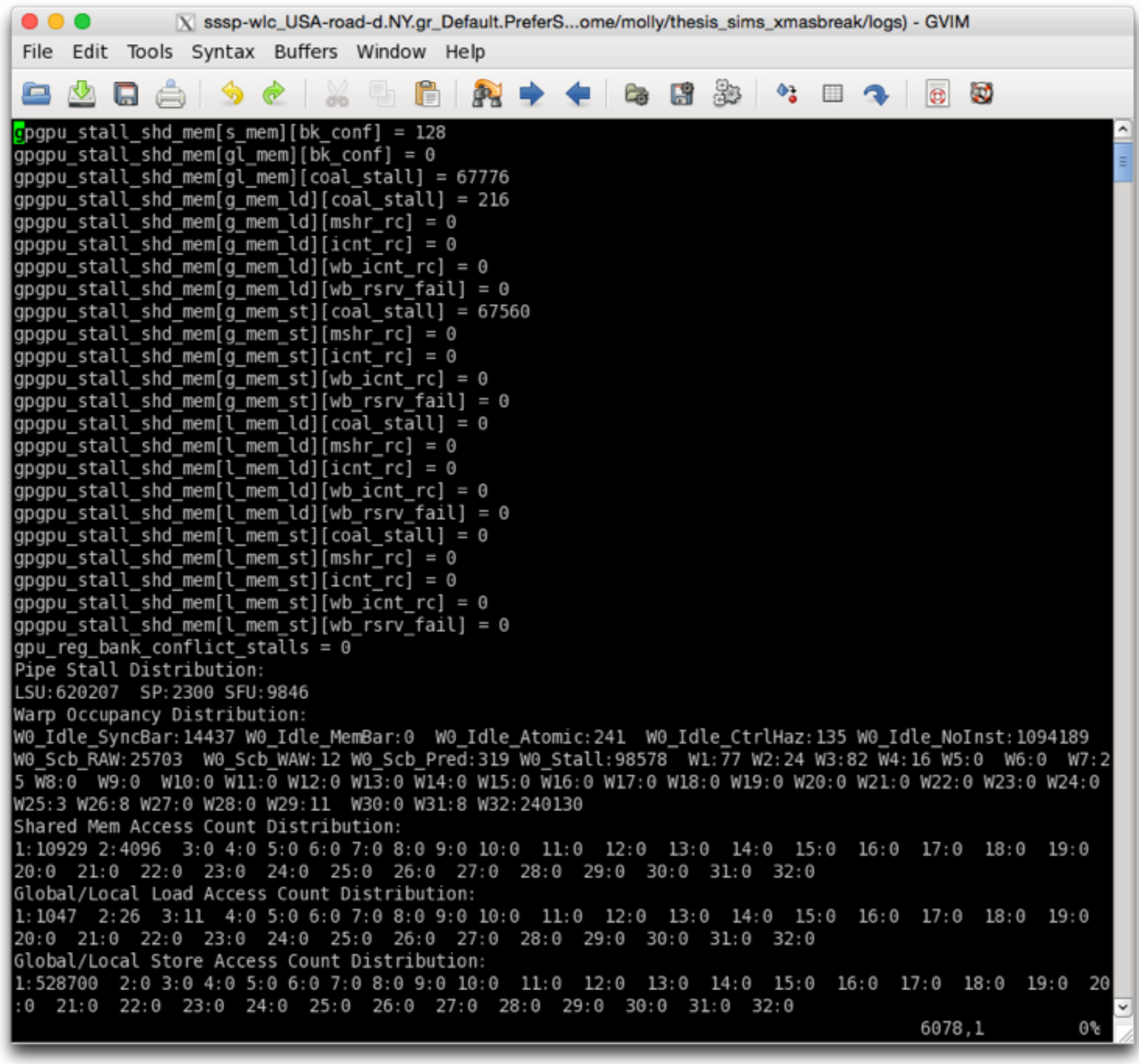
Input Variation: Road Networks vs. R-MAT





My GPGPU-Sim Toolkit

- By default, GPGPU-Sim gives you ~3 GB of this (per run) dumped to stdout:



- Management of multiple runs with different benchmarks, inputs, and configs and feasible data analysis required…
  1. Regression infrastructure supporting multiple cores, smart directory and log file handling, per-sim config specification
  2. Log file parser to auto-create .xlsx spreadsheets with high-value data, pre-drawn charts

Some Fun Numbers

- The results presented in the graphs and charts in this thesis represent simulation times that on a single CPU would have consumed approximately…

22,000+ hours

-or- **922+ days**

-or- **30+ months!!!**