A Variable Warp-Size Architecture

Timothy G. Rogers
Daniel R. Johnson
Mike O'Connor
Stephen W. Keckler

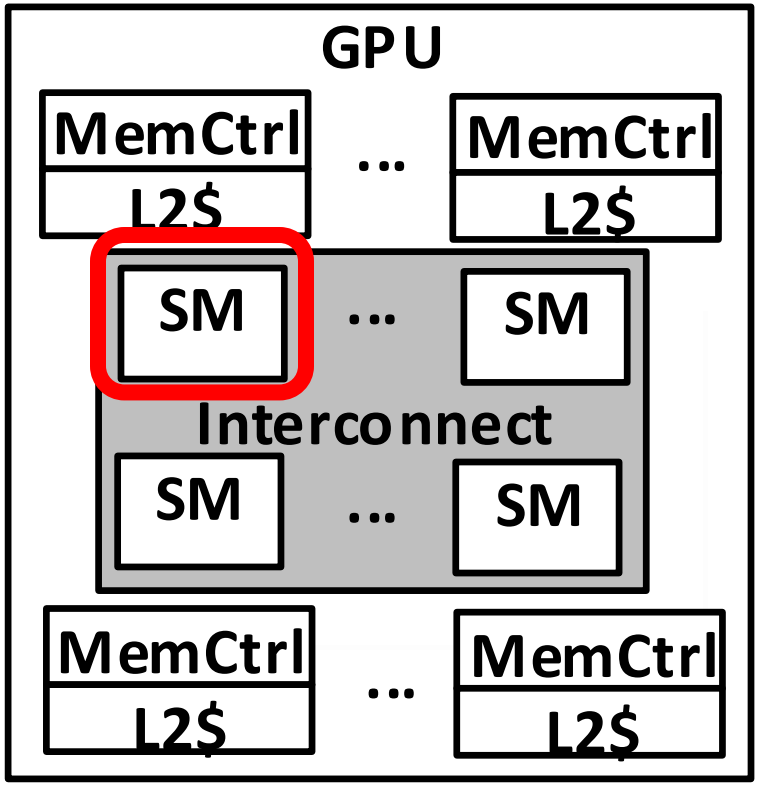
UBC

NVIDIA

THE UNIVERSITY OF TEXAS AT AUSTIN

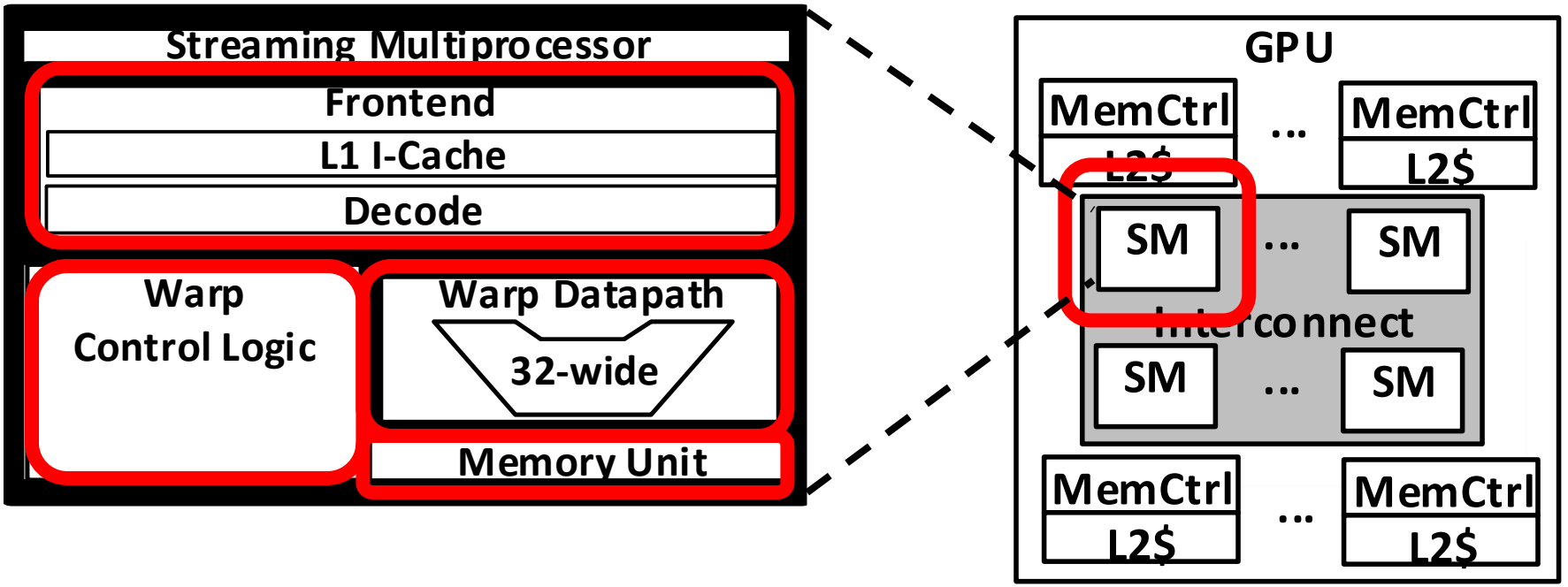
Contemporary GPU

- Massively Multithreaded
- 10,000's of threads concurrently executing on 10's of Streaming Multiprocessors (SM)



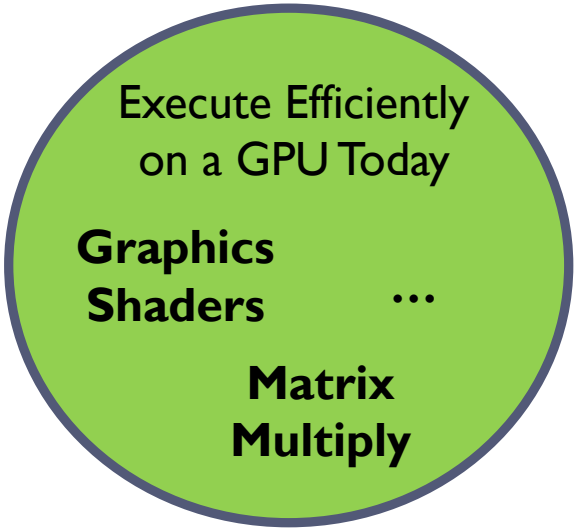
Contemporary Streaming Multiprocessor (SM)

- 1,000's of schedulable threads
- Amortize front end and memory overheads by grouping threads into warps.
  - Size of the warp is fixed based on the architecture



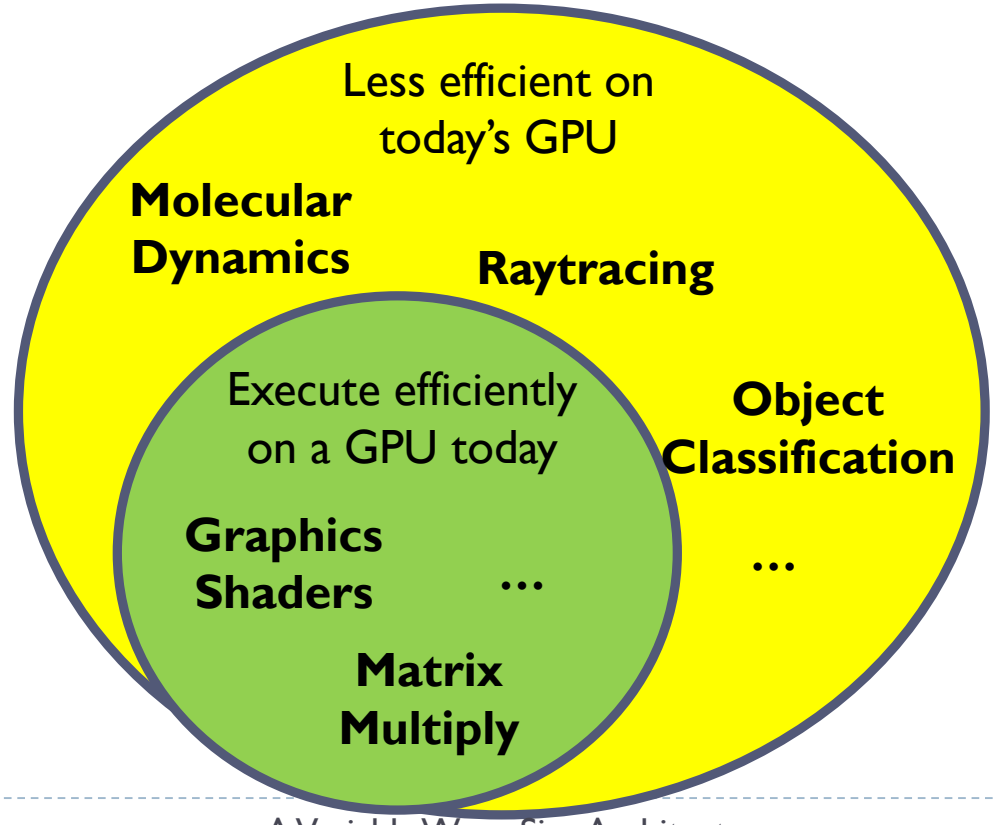
Contemporary GPU Software

- Regular structured computation
- Predictable access and control flow patterns
- Can take advantage of HW amortization for increased performance and energy efficiency



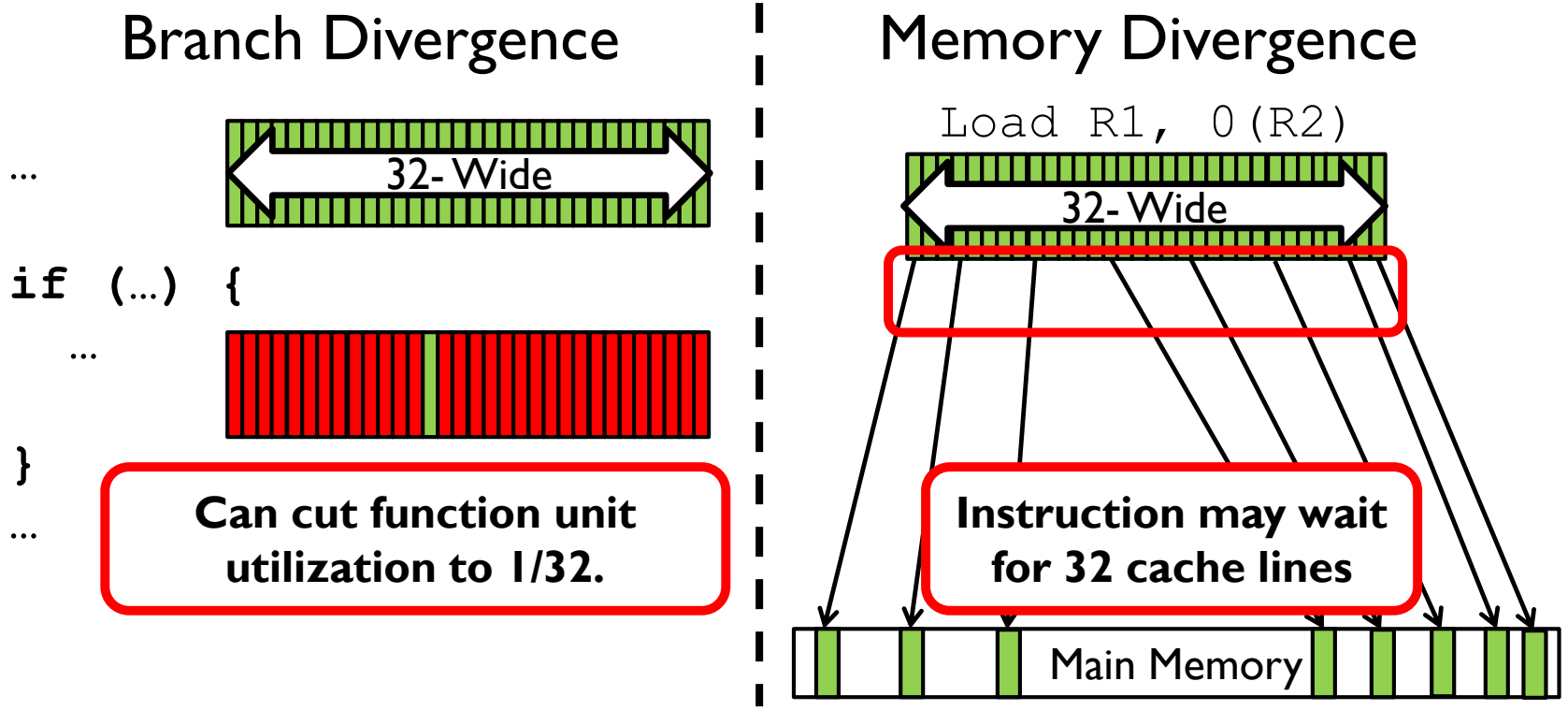
Forward-Looking GPU Software

- Still Massively Parallel
- Less Structured
  - Memory access and control flow patterns are less predictable

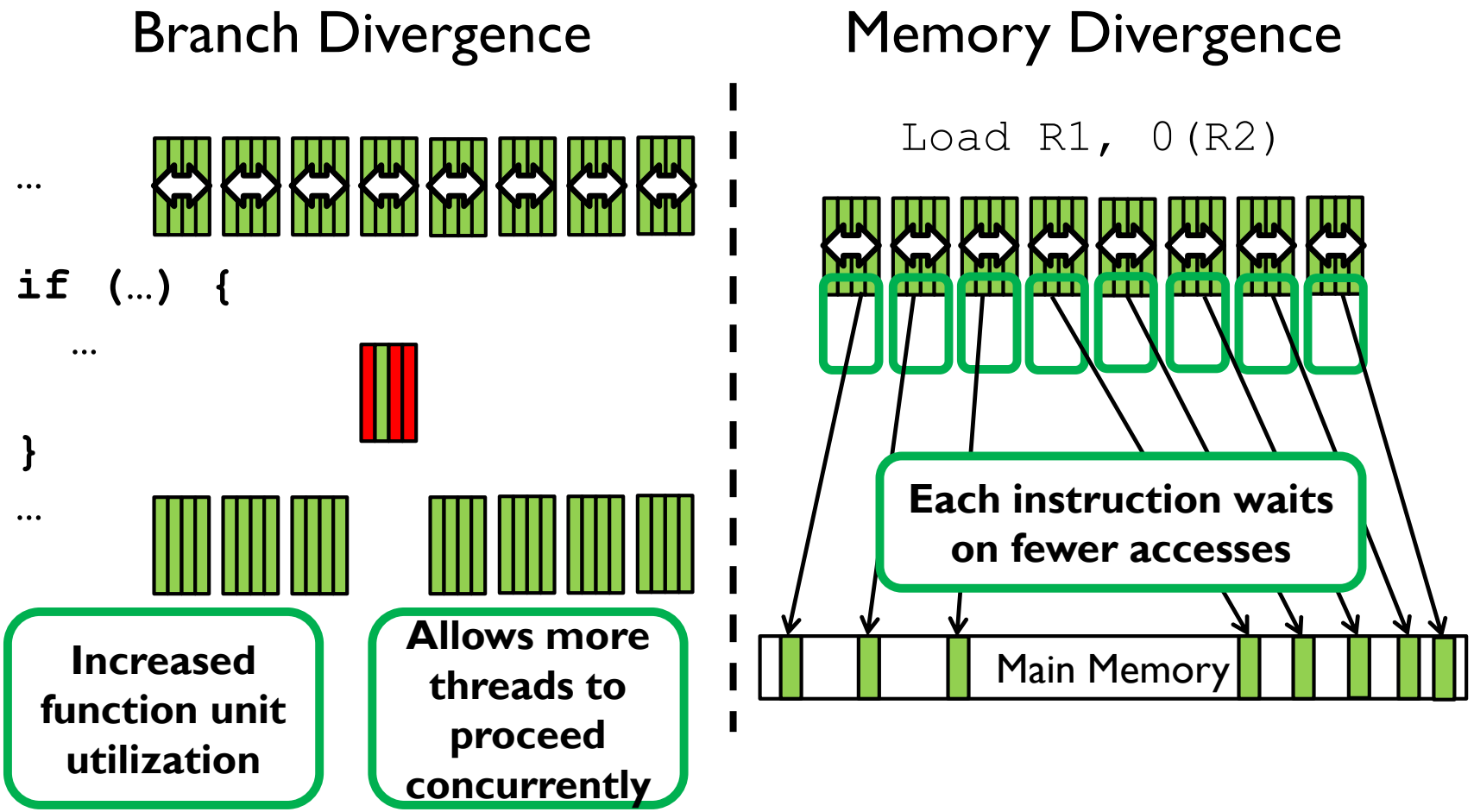


Divergence: Source of Inefficiency

- Regular hardware that amortizes front end and overhead
- Irregular software with many different control flow paths and less predicable memory accesses.



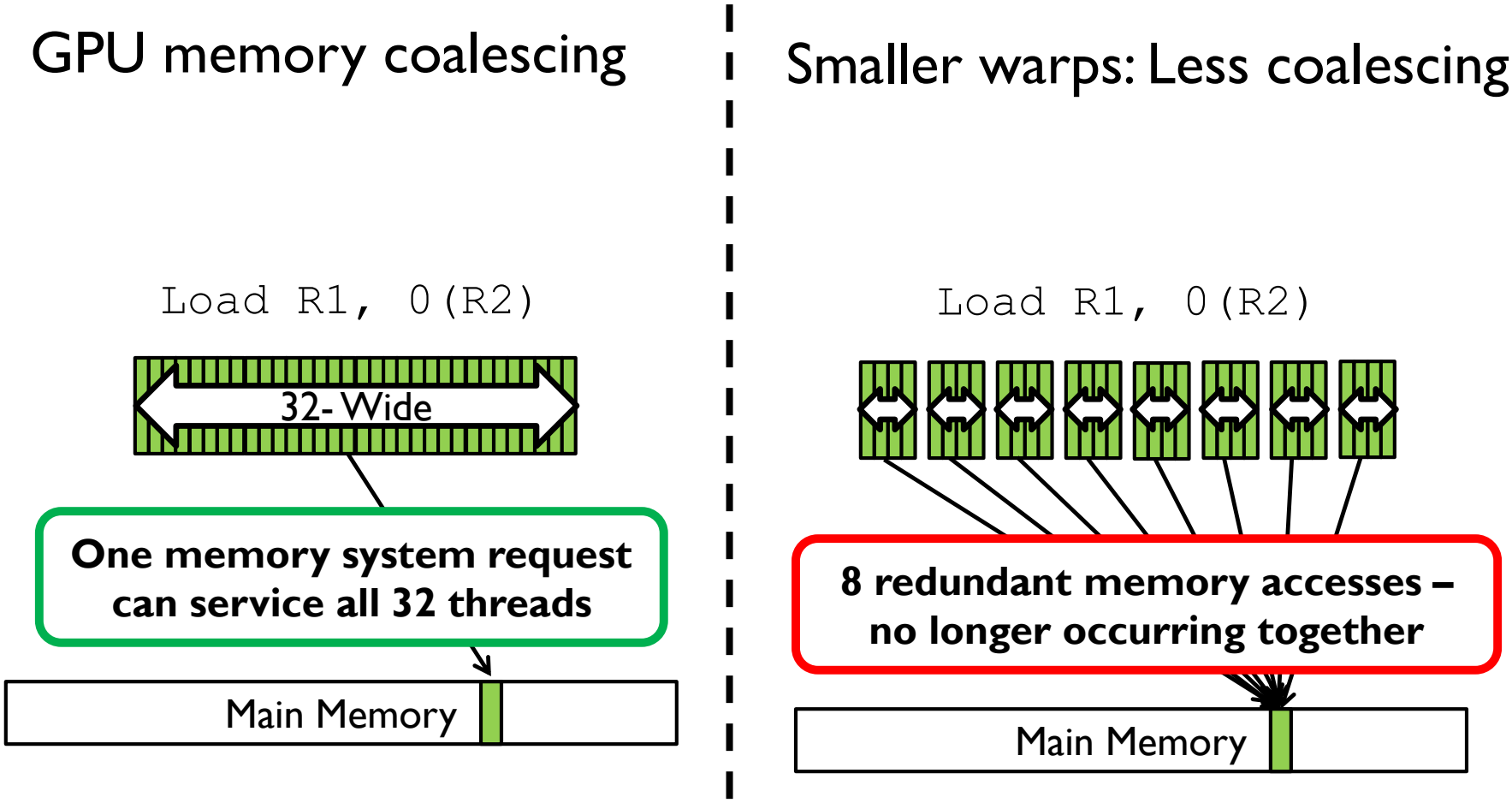
Irregular "Divergent" Applications: Perform better with a smaller warp size



Negative effects of smaller warp size

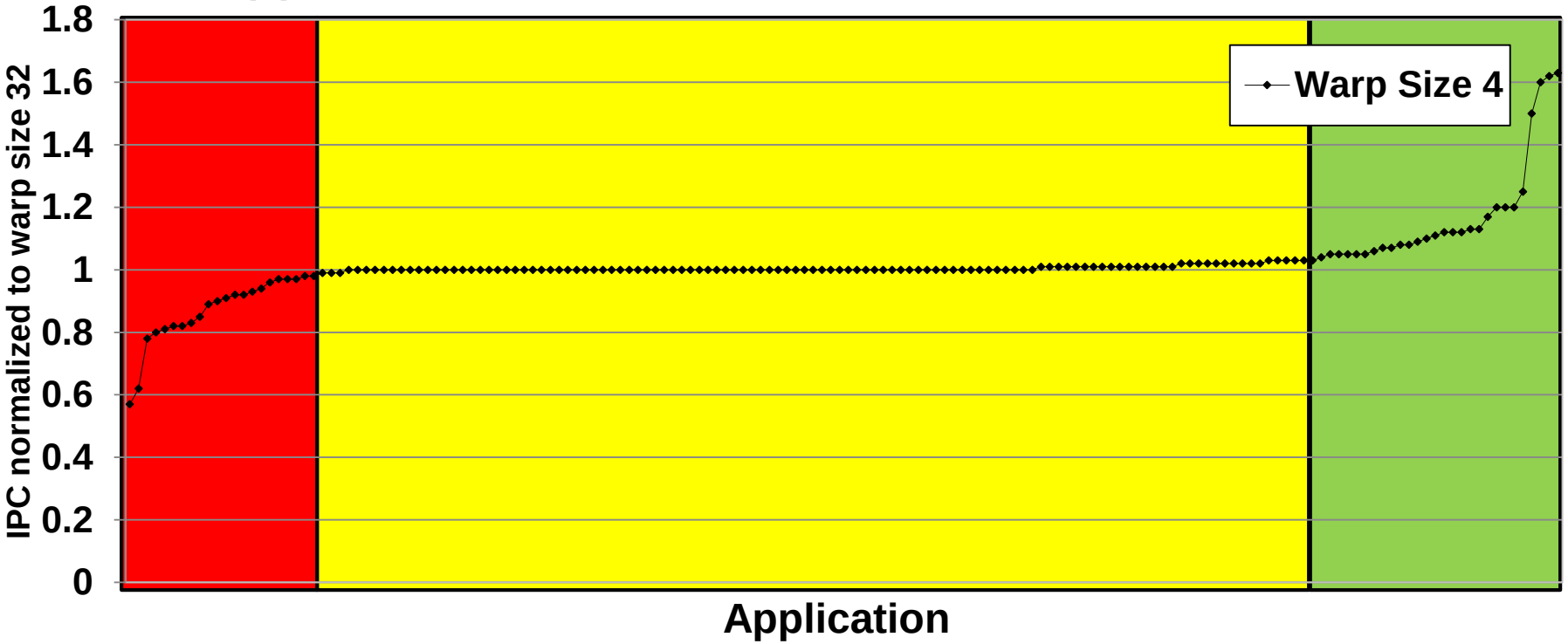
- Less front end amortization
  - Increase in fetch/decode energy
- Negative performance effects
  - Scheduling skew increases pressure on the memory system

Regular "Convergent" Applications: Perform better with a wider warp



Performance vs. Warp Size

- 165 Applications



Convergent Applications

Warp-Size Insensitive Applications

Divergent Applications

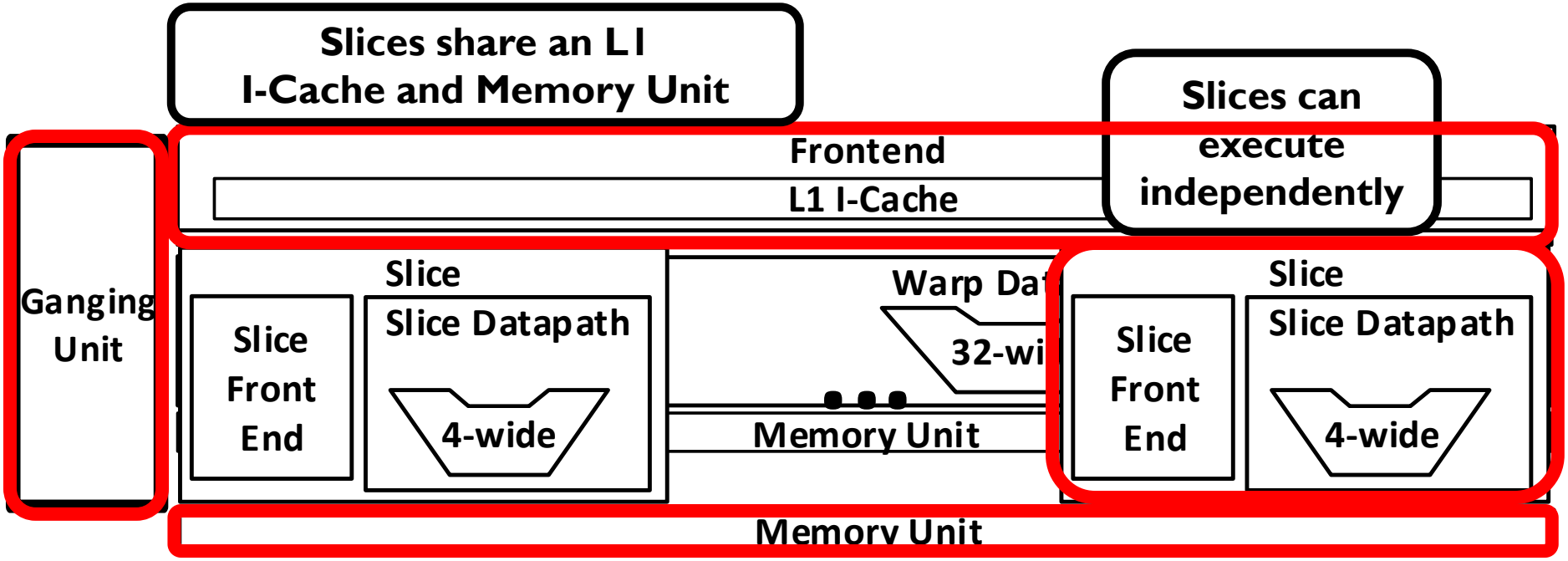
Goals

- Convergent Applications
  - Maintain wide-warp performance
  - Maintain front end efficiency
- Warp Size Insensitive Applications
  - Maintain front end efficiency
- Divergent Applications
  - Gain small warp performance

Set the warp size based on the executing application

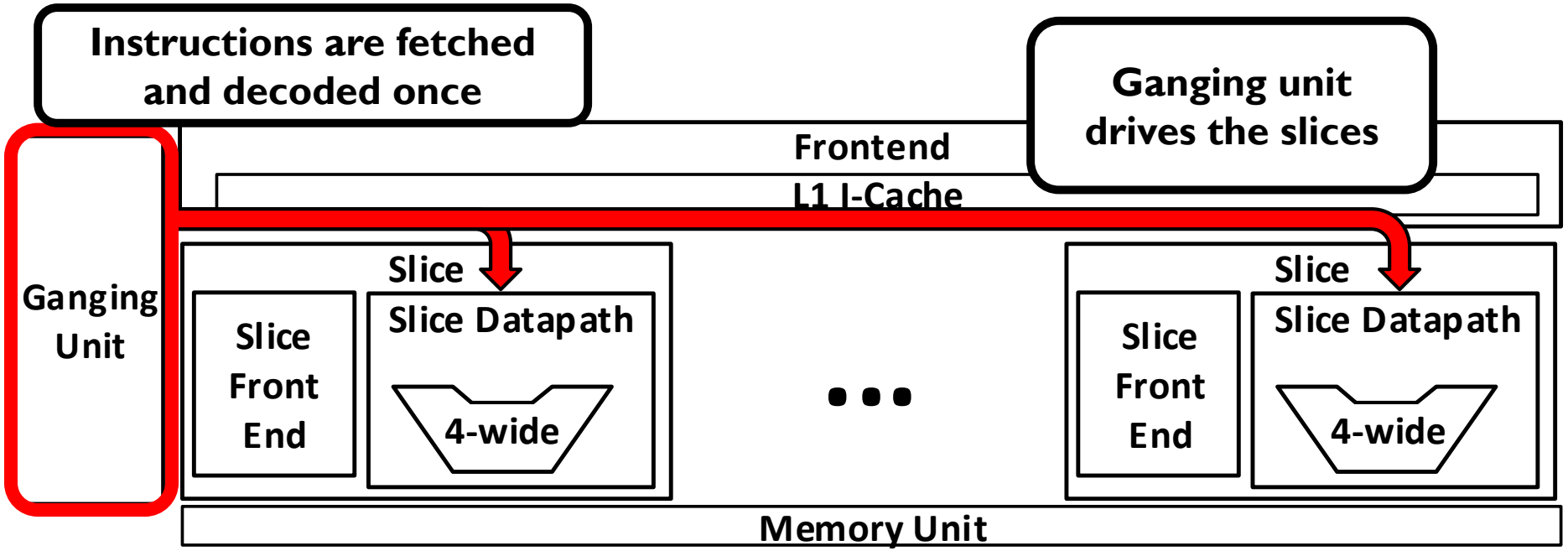
Sliced Datapath + Ganged Scheduling

- Split the SM datapath into narrow **slices**.
  - Extensively studied 4-thread slices
- Gang slice execution to gain efficiencies of wider warp.



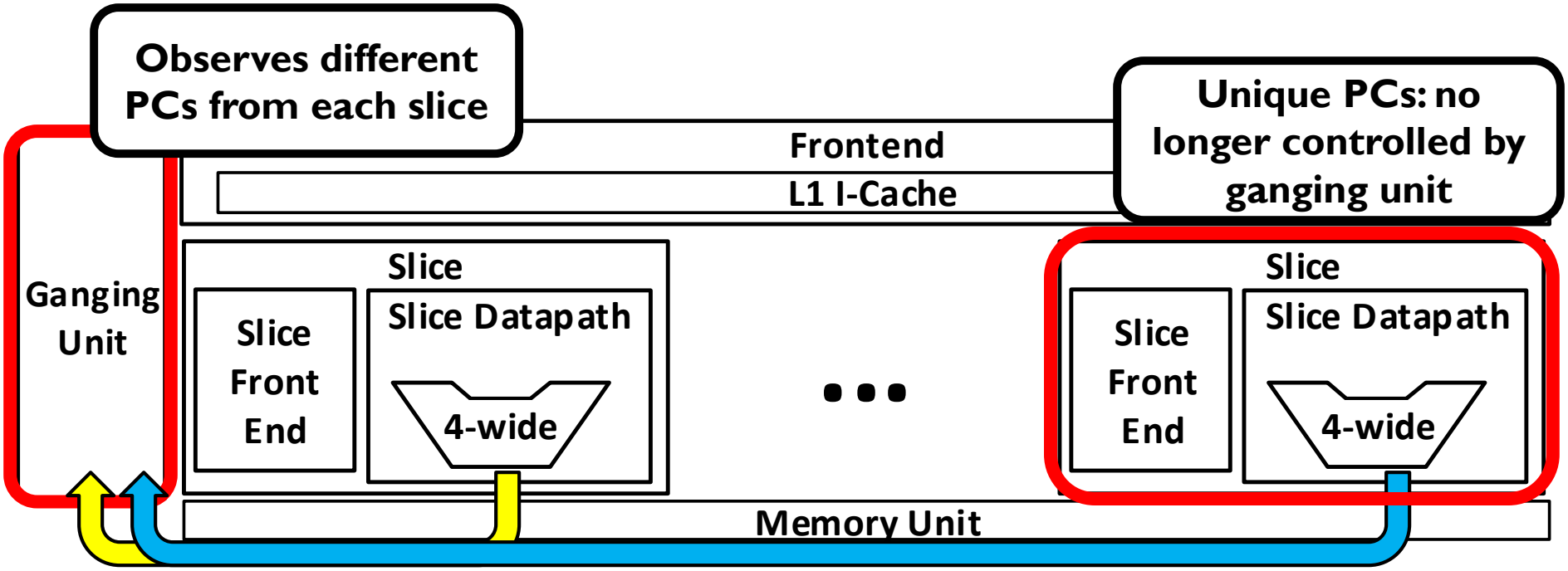
Initial operation

- Slices begin execution in ganged mode
  - Mirrors the baseline 32-wide warp system
- Question: When to break the gang?



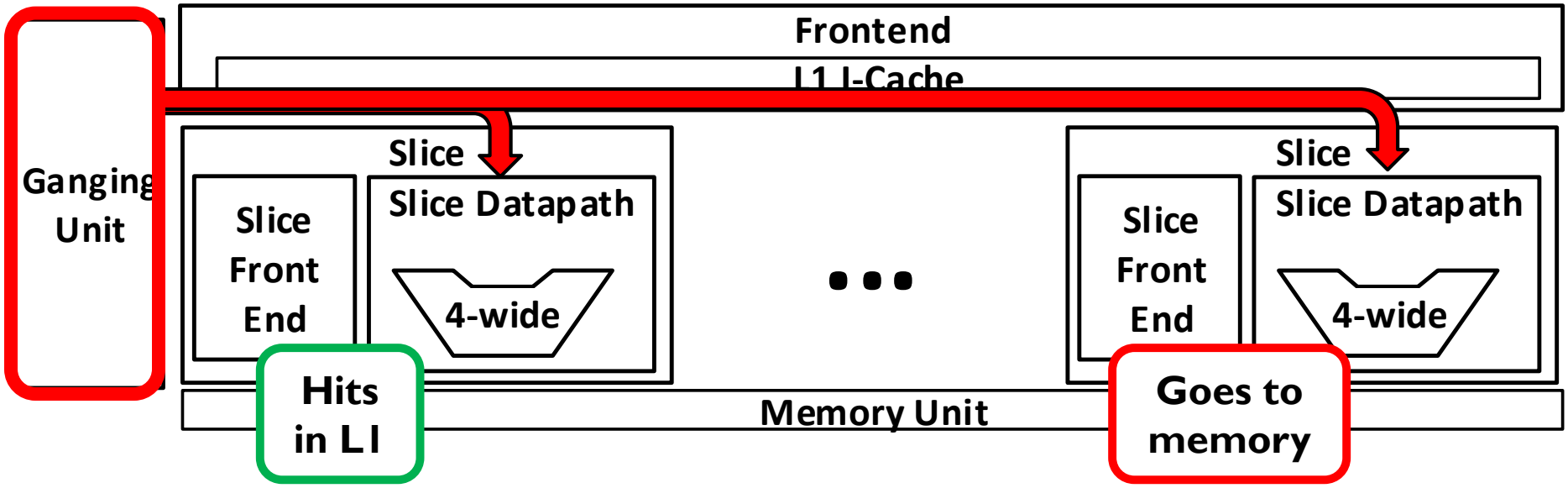
Breaking Gangs on Control Flow Divergence

- PCs common to more than one slice form a new gang
- Slices that follow a unique PC in the gang are transferred to independent control



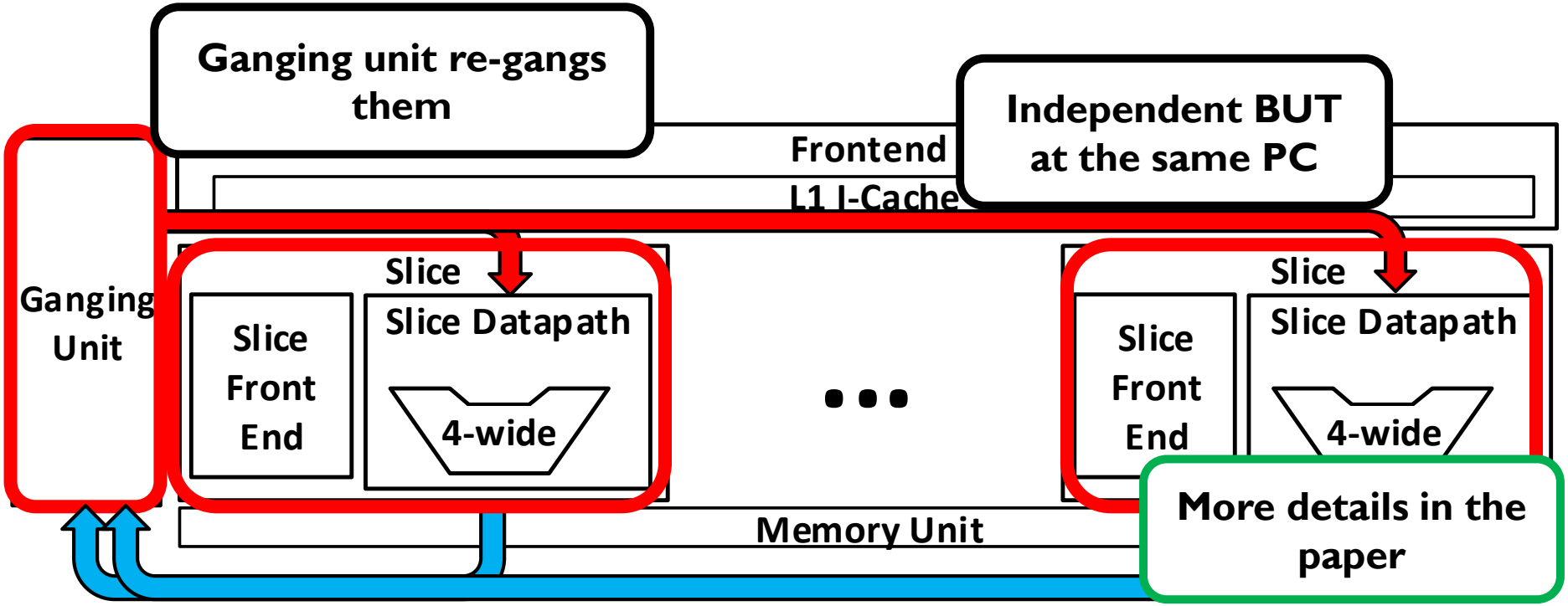
Breaking Gangs on Memory Divergence

- Latency of accesses from each slice can differ
- Evaluated several heuristics on breaking the gang when this occurs



Gang Reformation

- Performed opportunistically
  - Ganging unit checks for gangs or independent slices at the same PC
  - Forms them into a gang



Methodology

- In House, Cycle-Level Streaming Multiprocessor Model
  - 1 In-order core
  - 64KB L1 Data cache
  - 128KB L2 Data Cache (One SM's worth)
  - 48KB Shared Memory
  - Texture memory unit
  - Limited BW memory system
  - Greedy-Then-Oldest (GTO) Issue Scheduler

Configurations

- Warp Size 32 (WS 32)
- Warp Size 4 (WS 4)

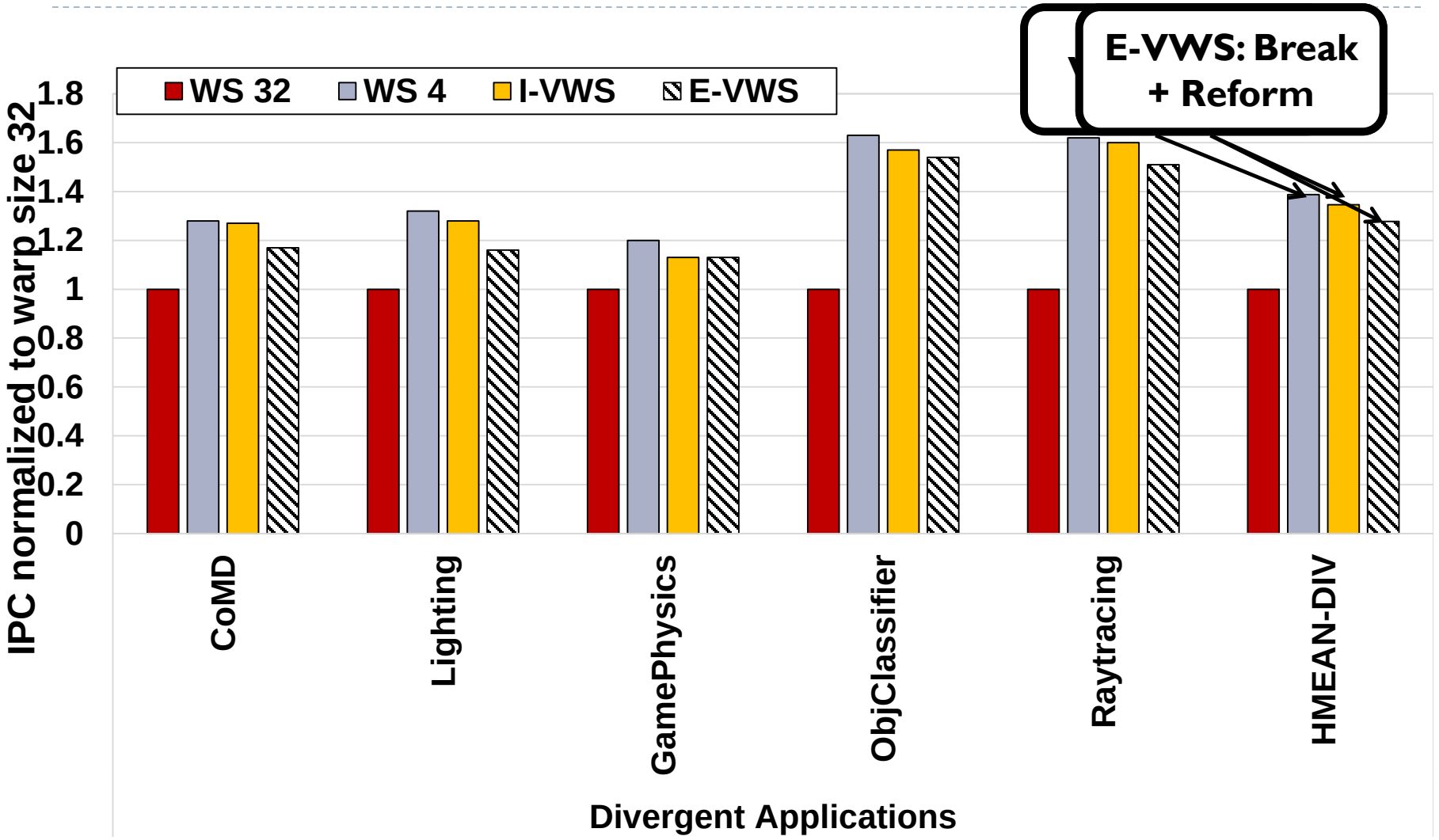
- Inelastic Variable Warp Sizing (I-VWS)
  - Gangs break on control flow divergence
  - Are not reformed

- Elastic Variable Warp Sizing (E-VWS)
  - Like I-VWS, except gangs are opportunistically reformed

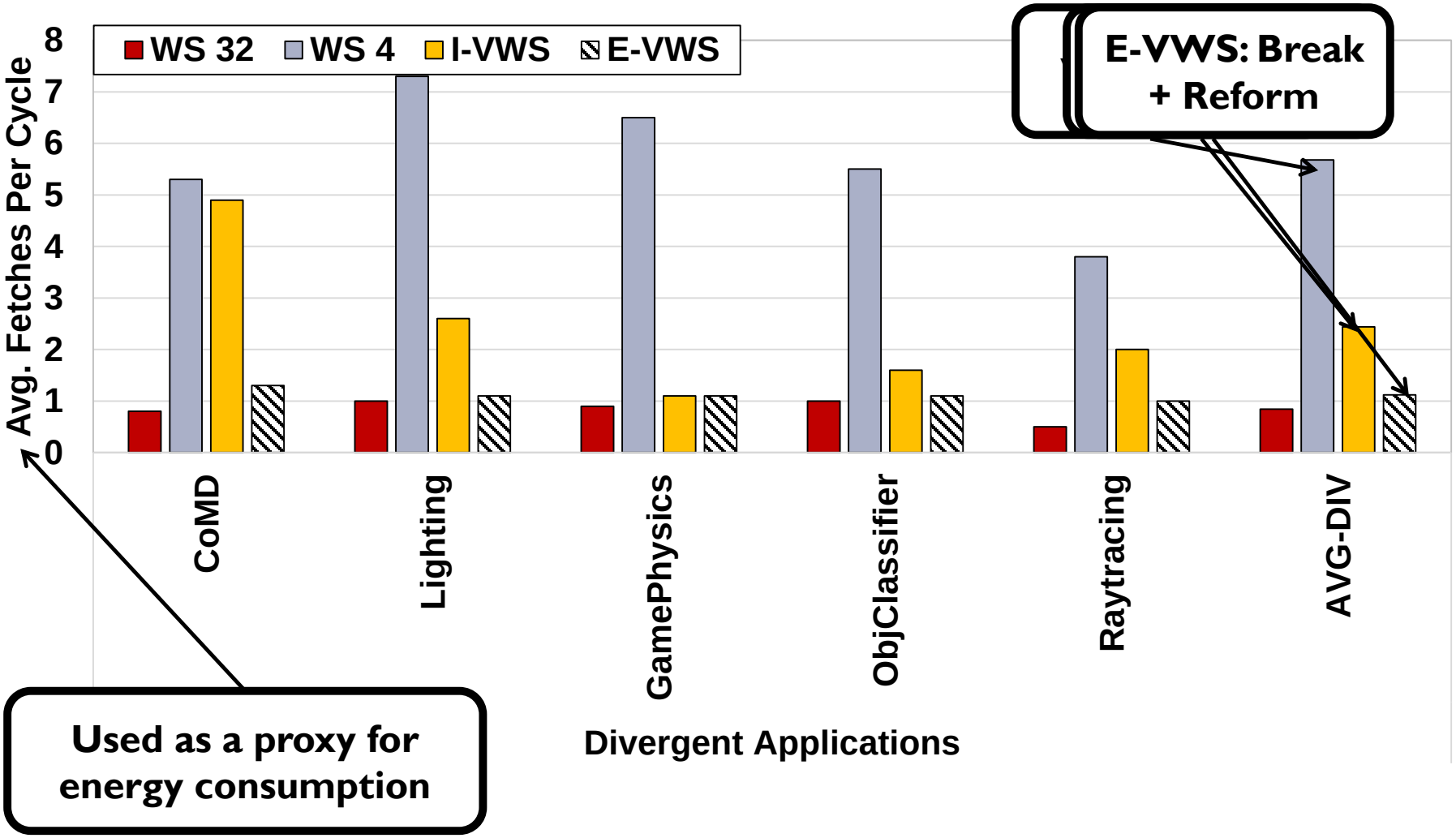
- Studied 5 applications from each categ

Paper Explores Many More Configurations

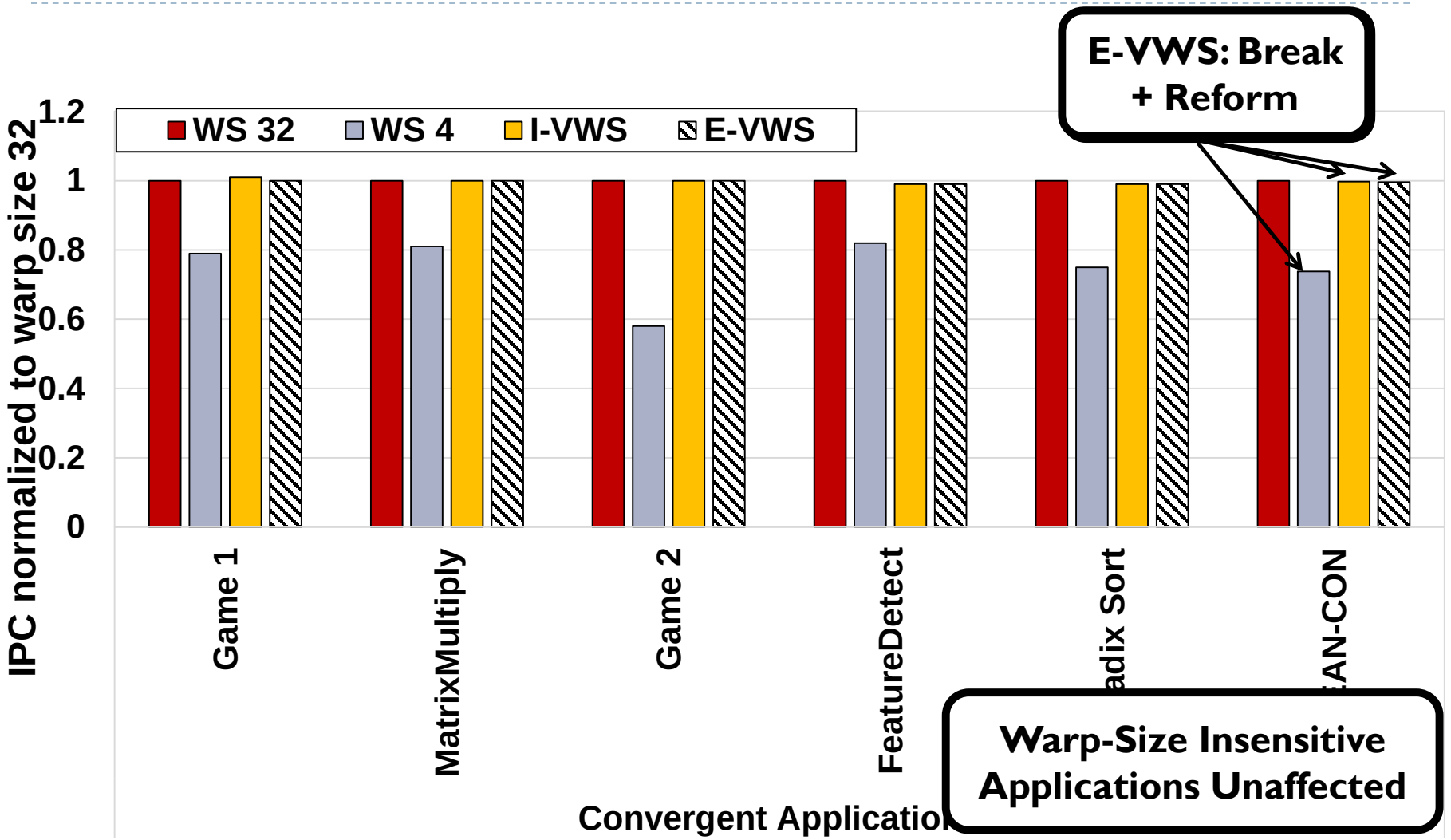
Divergent Application Performance



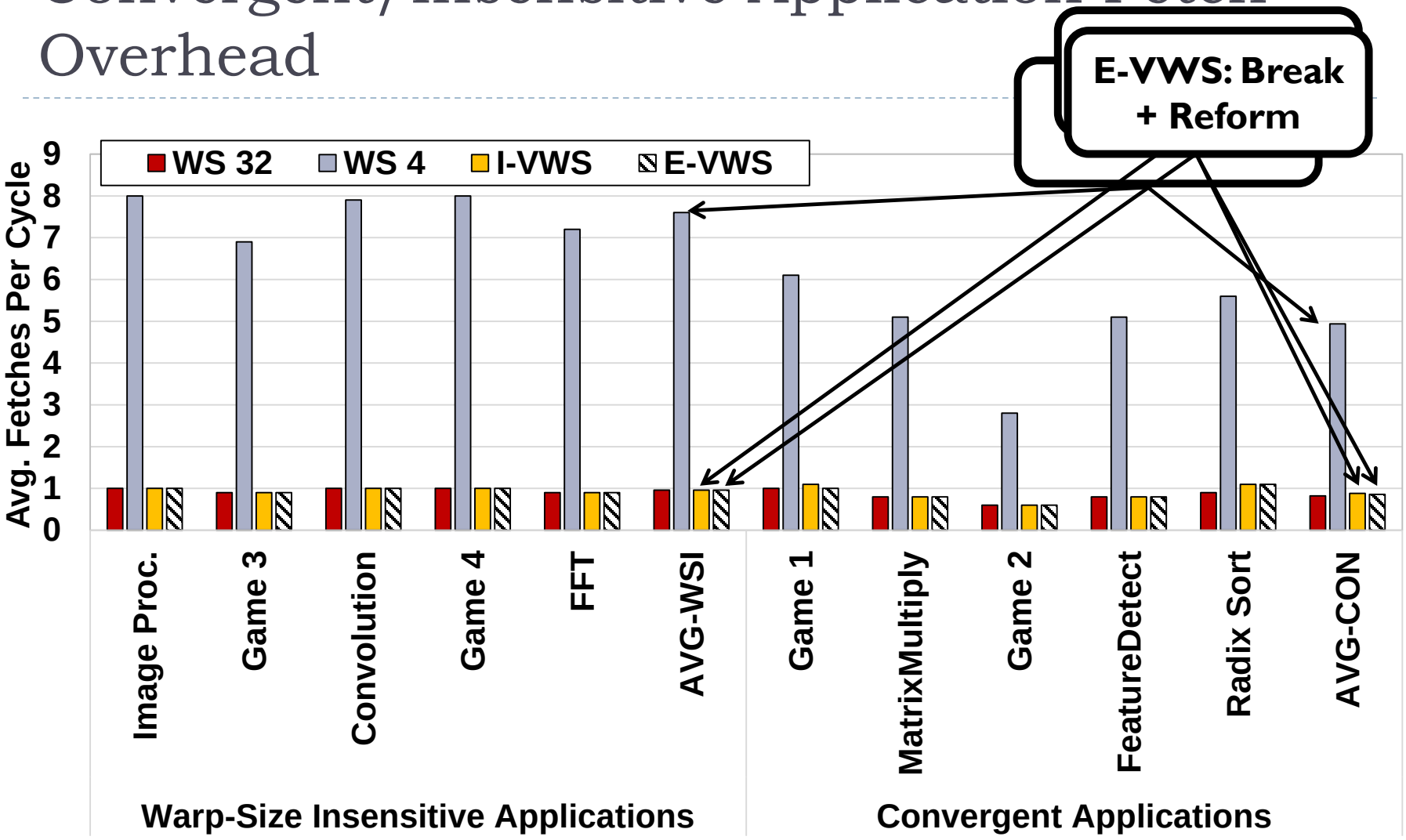
Divergent Application Fetch Overhead



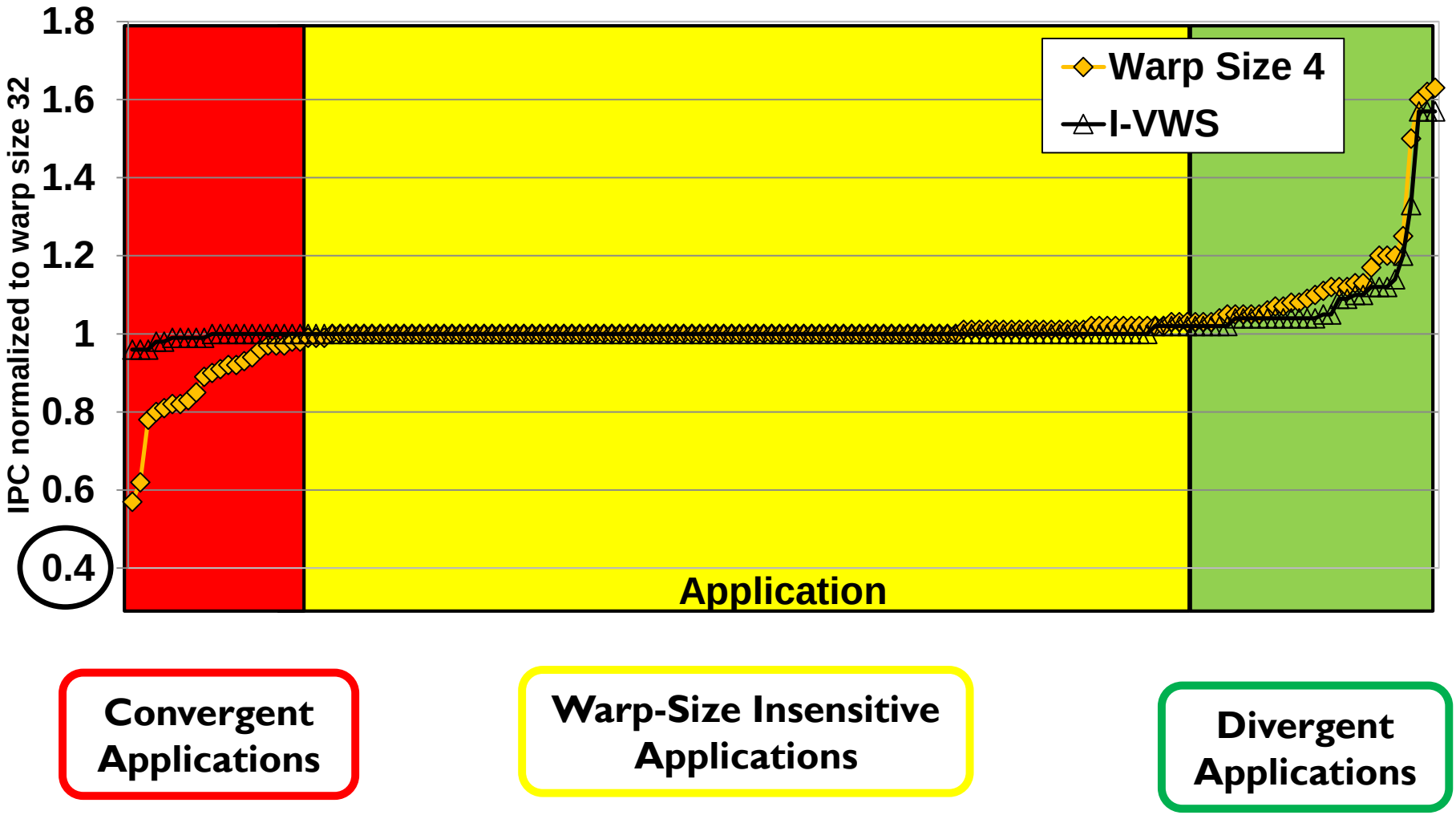
Convergent Application Performance



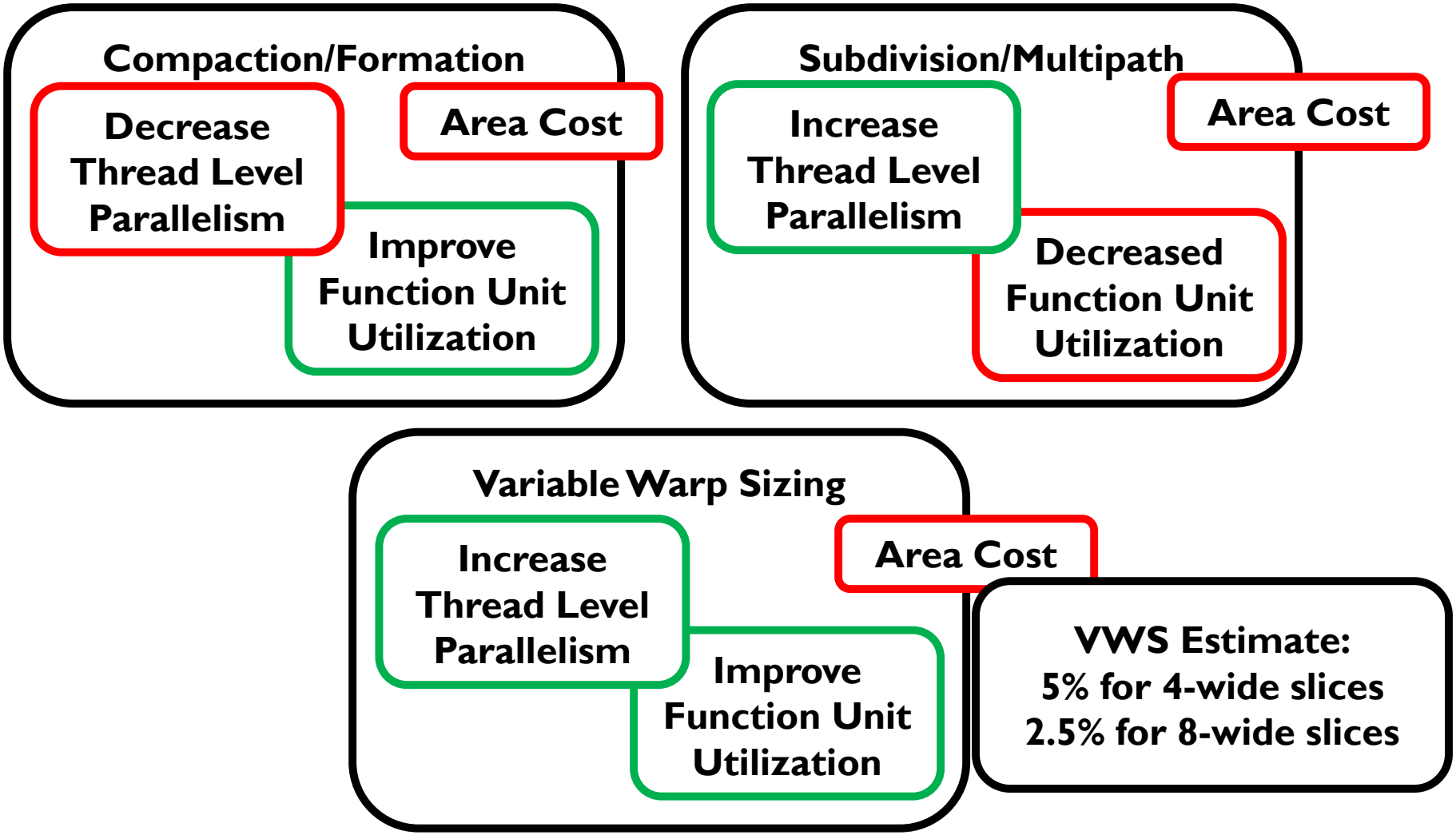
Convergent/Insensitive Application Fetch Overhead



165 Application Performance



Related Work



Conclusion

- Explored space surrounding warp size and perfromance
- Vary the size of the warp to meet the depends of the workload
  - 35% performance improvement on divergent apps
  - No performance degradation on convergent apps
- Narrow slices with ganged execution
  - Improves both SIMD efficiency and thread-level parallelism

Questions?

Managing DRAM Latency Divergence in Irregular GPGPU Applications

Niladrish Chatterjee

Mike O'Connor

Gabriel H. Loh

Nuwan Jayasena

Rajeev Balasubramonian

Irregular GPGPU Applications

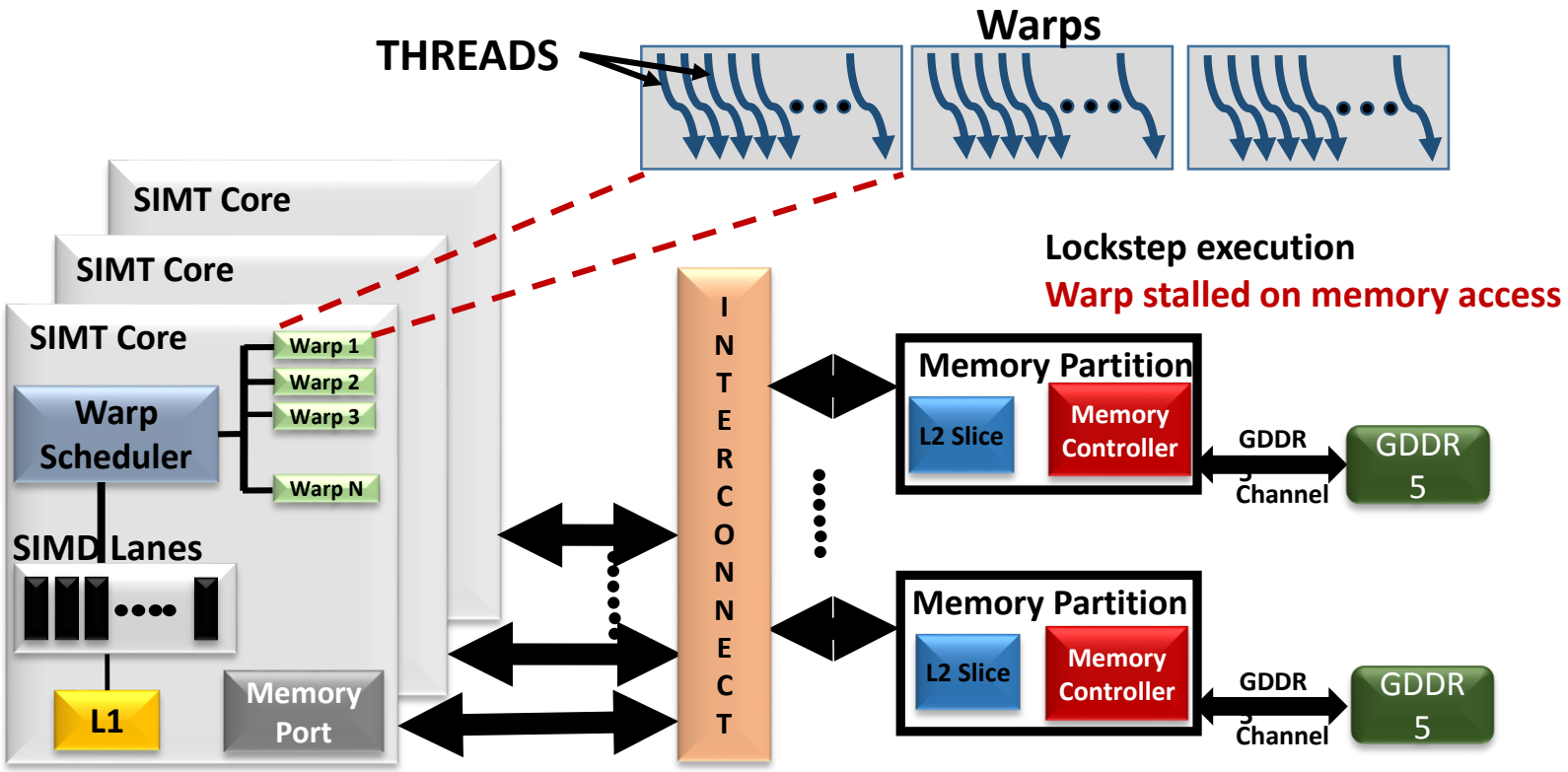
- Conventional GPGPU workloads access vector or matrix-based data structures
  - Predictable strides, large data parallelism

- Emerging Irregular Workloads
  - Pointer-based data-structures & data-dependent memory accesses
  - **Memory Latency Divergence on SIMT platforms**

Warp-aware memory scheduling to reduce DRAM latency divergence

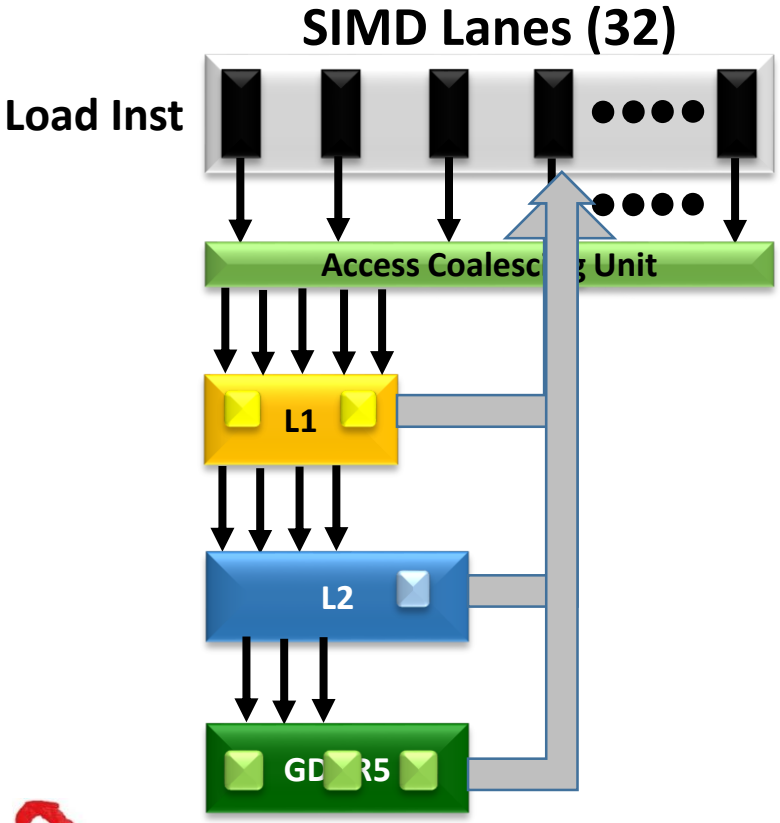
UTAH ARCH

SIMT Execution Overview



UTAH ARCH

Memory Latency Divergence



- Coalescer has limited efficacy in irregular workloads
- Partial hits in L1 and L2
  - 1st source of latency divergence
- DRAM requests can have varied latencies
  - Warp stalled for last request
- DRAM Latency Divergence

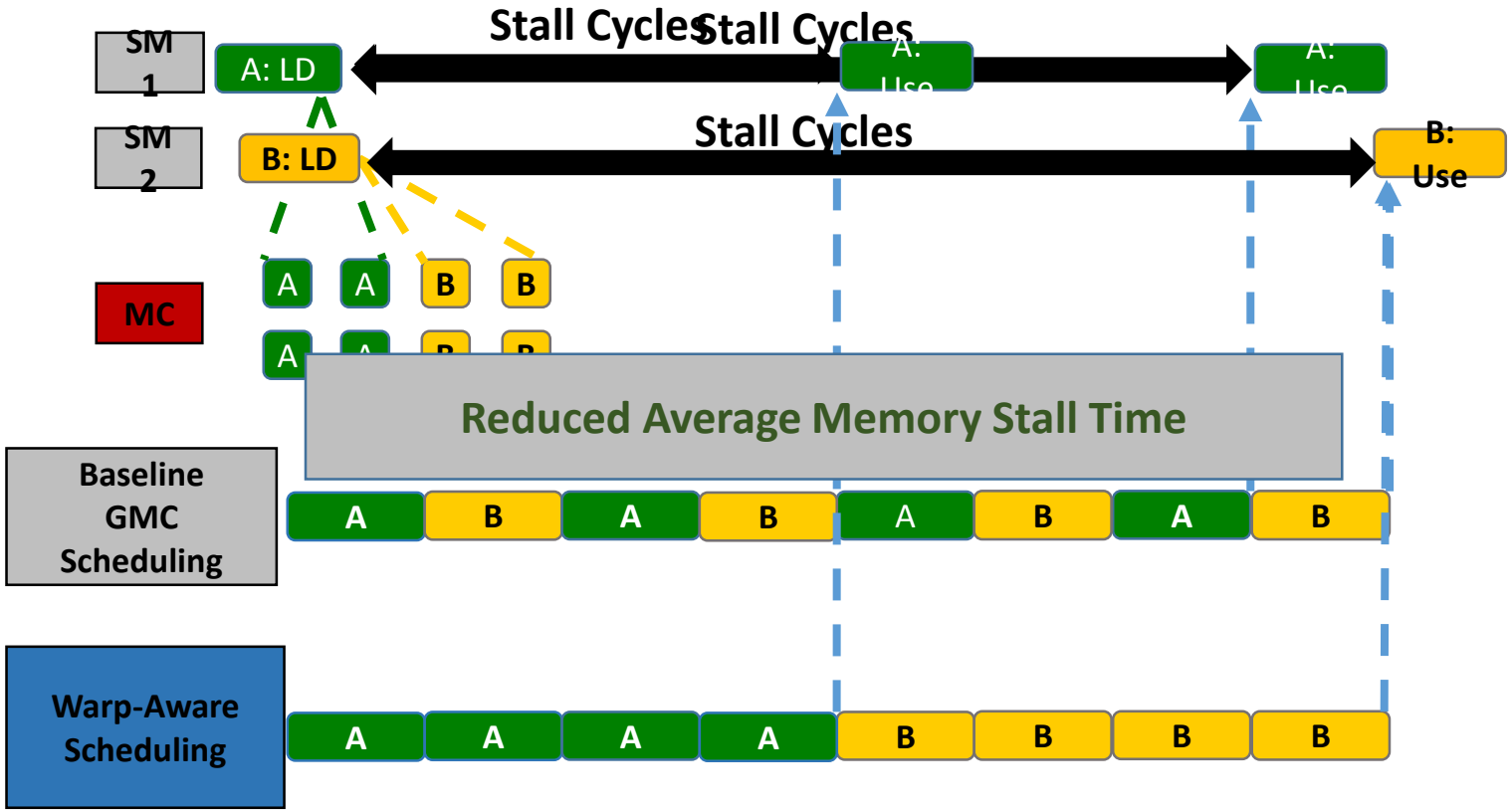
UTAH ARCH

GPU Memory Controller (GMC)

- Optimized for high throughput
- Harvest **channel and bank parallelism**
  - Address mapping to spread cache-lines across channels and banks.
- Achieve **high row-buffer hit rate**
  - Deep queuing
  - Aggressive reordering of requests for row-hit batching
- Not cognizant of the need to service requests from a warp together
  - **Interleave requests from different warps leading to latency divergence**

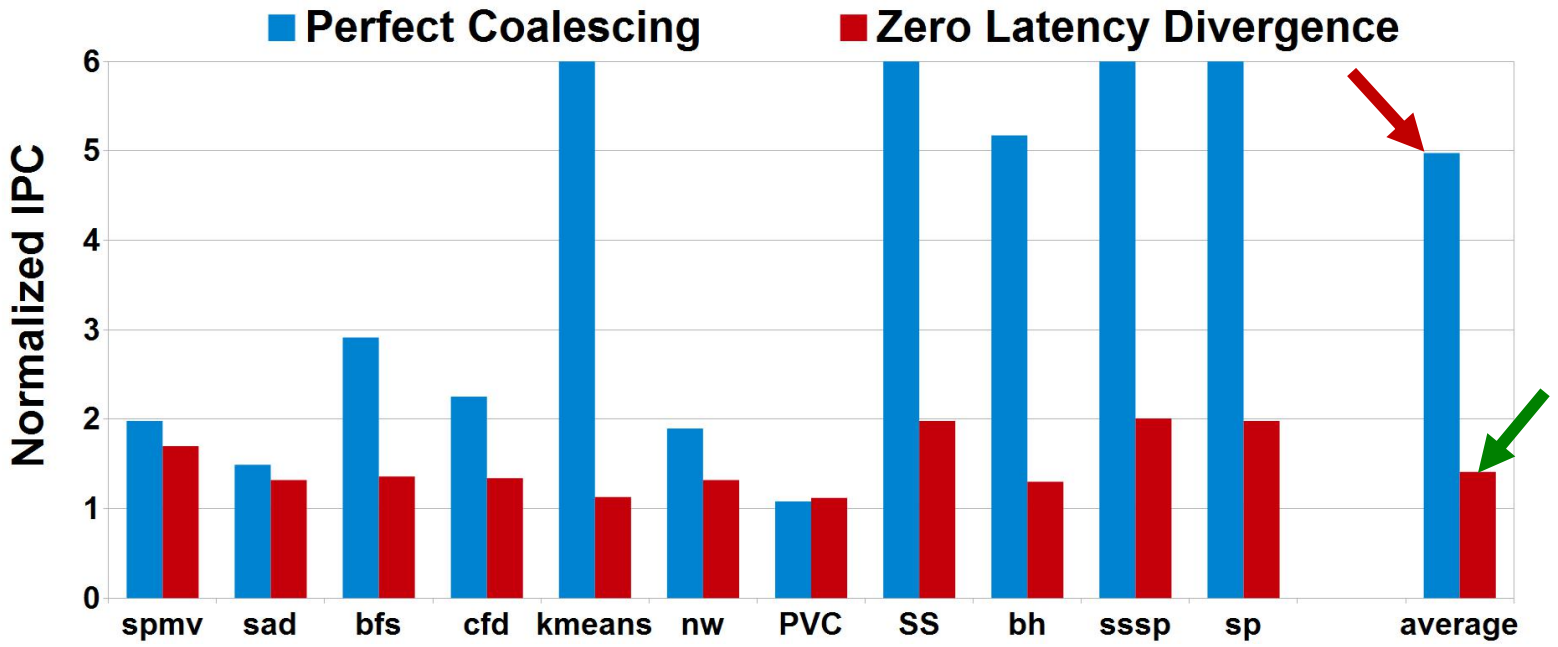
UTAH ARCH

Warp-Aware Scheduling



UTAH ARCH

Impact of DRAM Latency Divergence



If there was only 1 request per warp – 5X improvement.

If all requests from a warp were to be returned in perfect sequence from the DRAM – ~40% improvement.

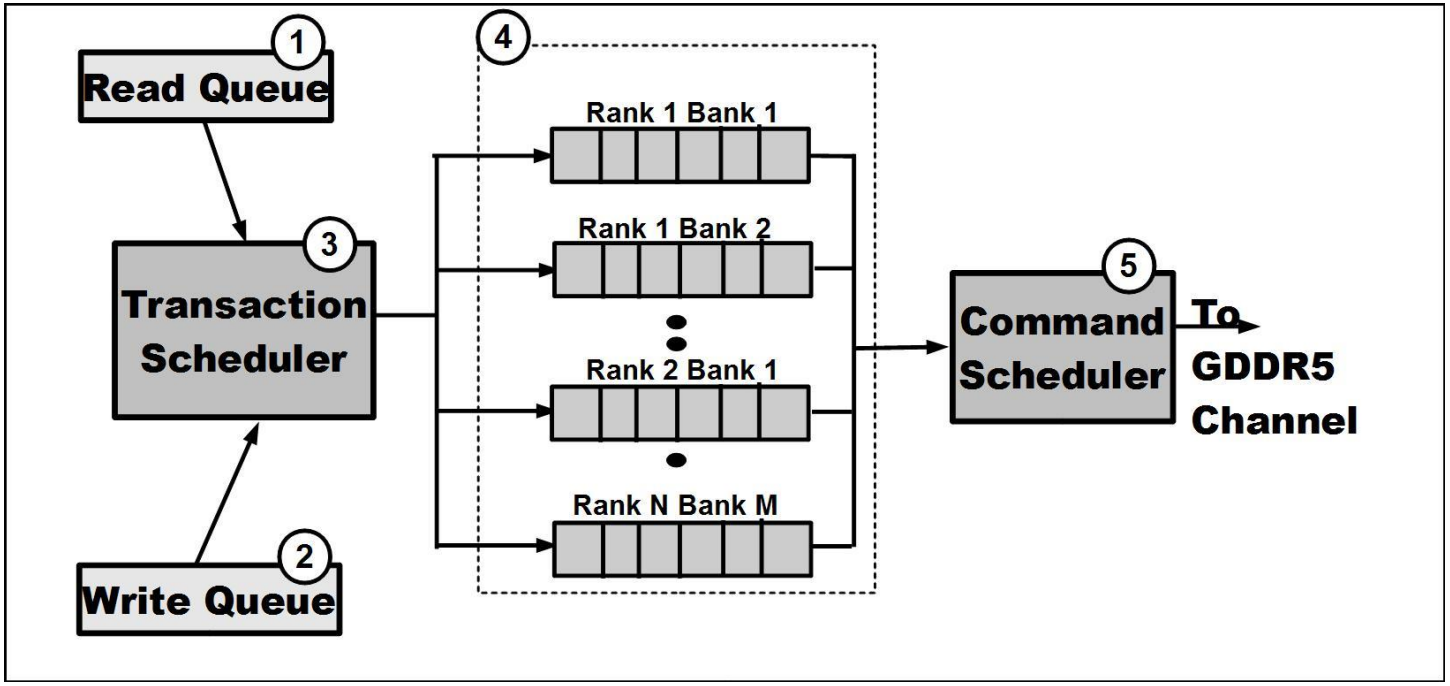
UTAH ARCH

Key Idea

- Form batches of requests from each warp
  - *warp-group*
- Schedule all requests from a warp-group together
- Scheduling algorithm arbitrates between warp-groups to minimize average stall-time of warps

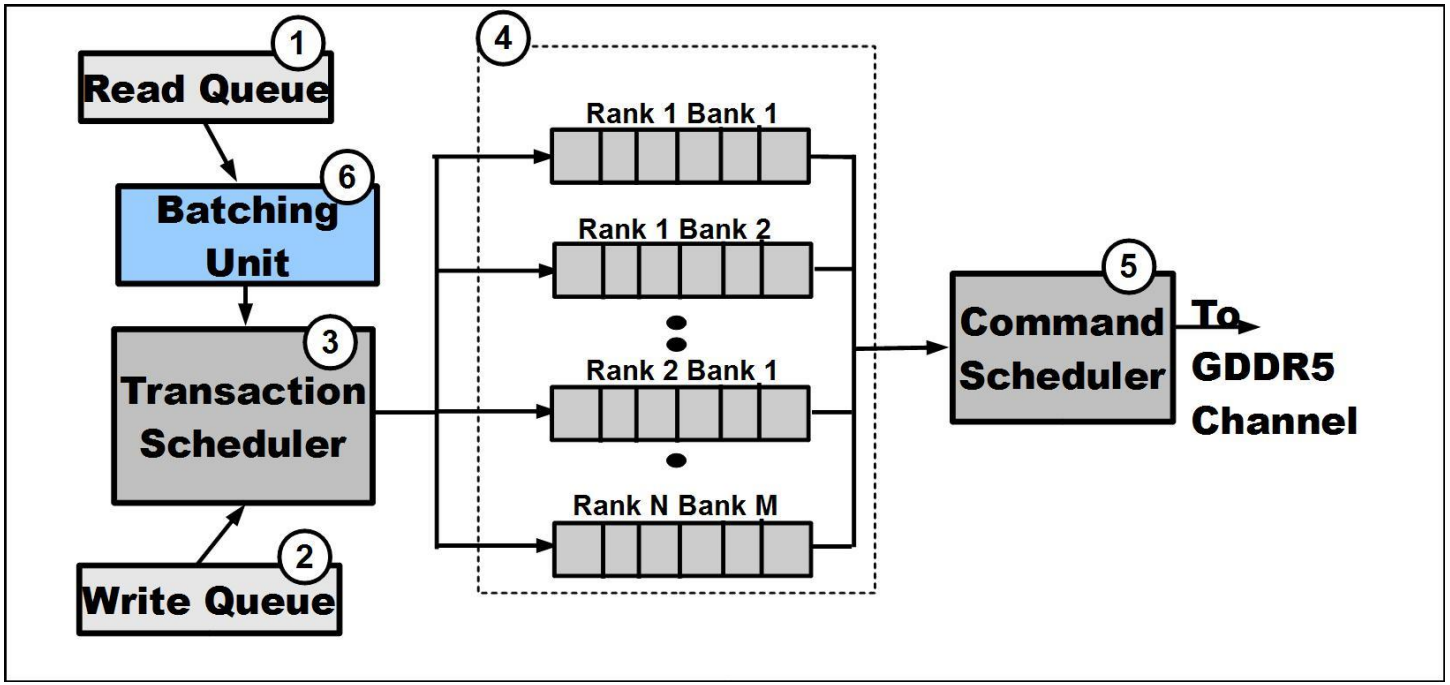
UTAH ARCH

Controller Design



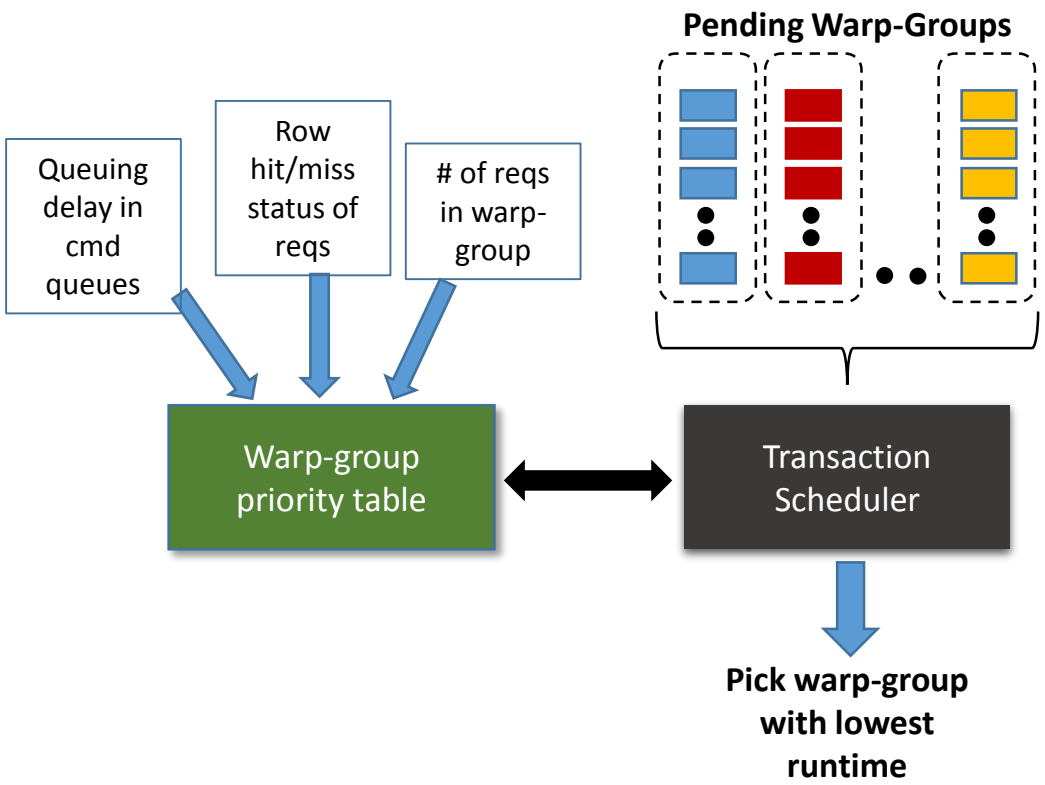
UTAH ARCH

Controller Design



UTAH ARCH

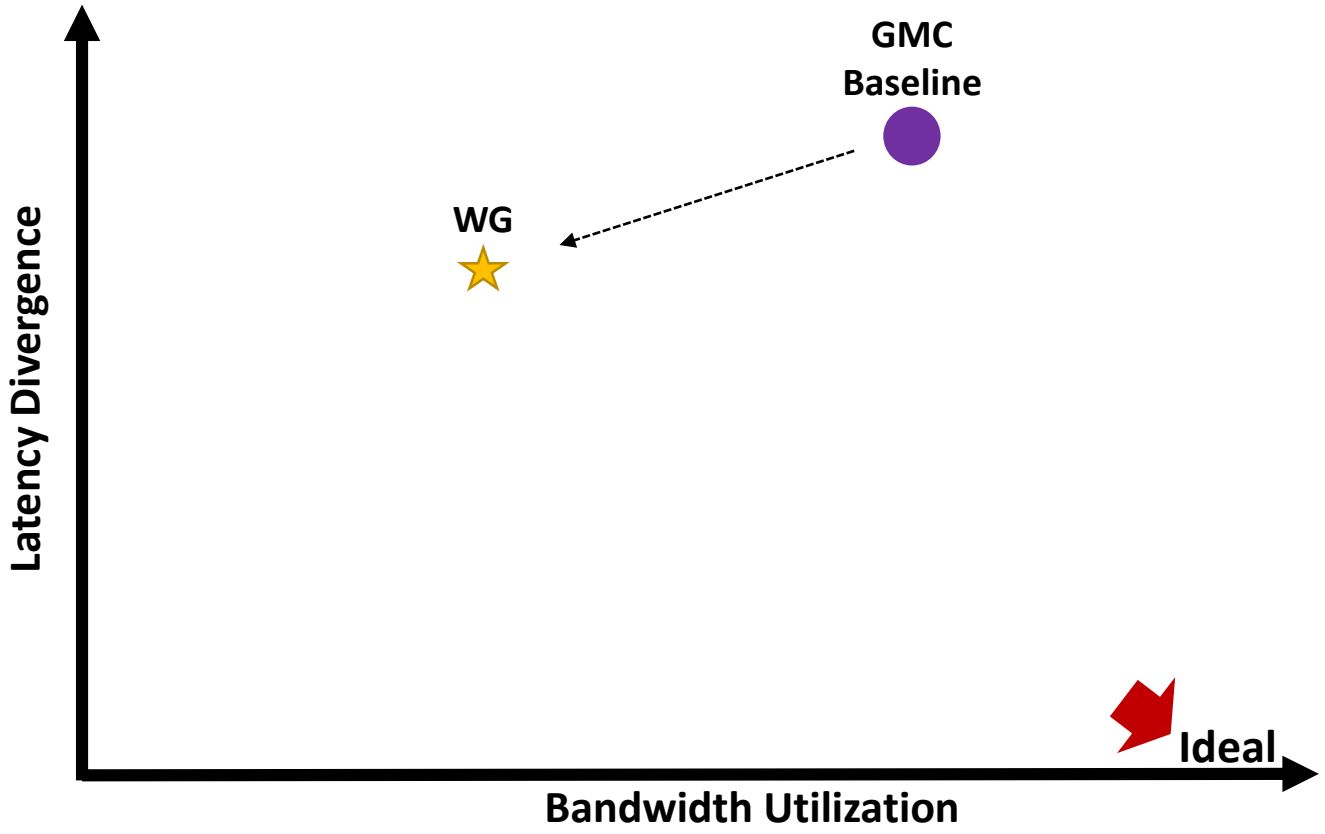
Warp-Group Scheduling : Single Channel



- Each Warp-Group assigned a priority
  - Reflects completion time of last request
- Higher Priority to
  - Few requests
  - High spatial locality
  - Lightly loaded banks
- Priorities updated dynamically
- Transaction Scheduler picks warp-group with lowest run-time
  - **Shortest-job-first based on actual service time**

UTAH ARCH

WG-scheduling



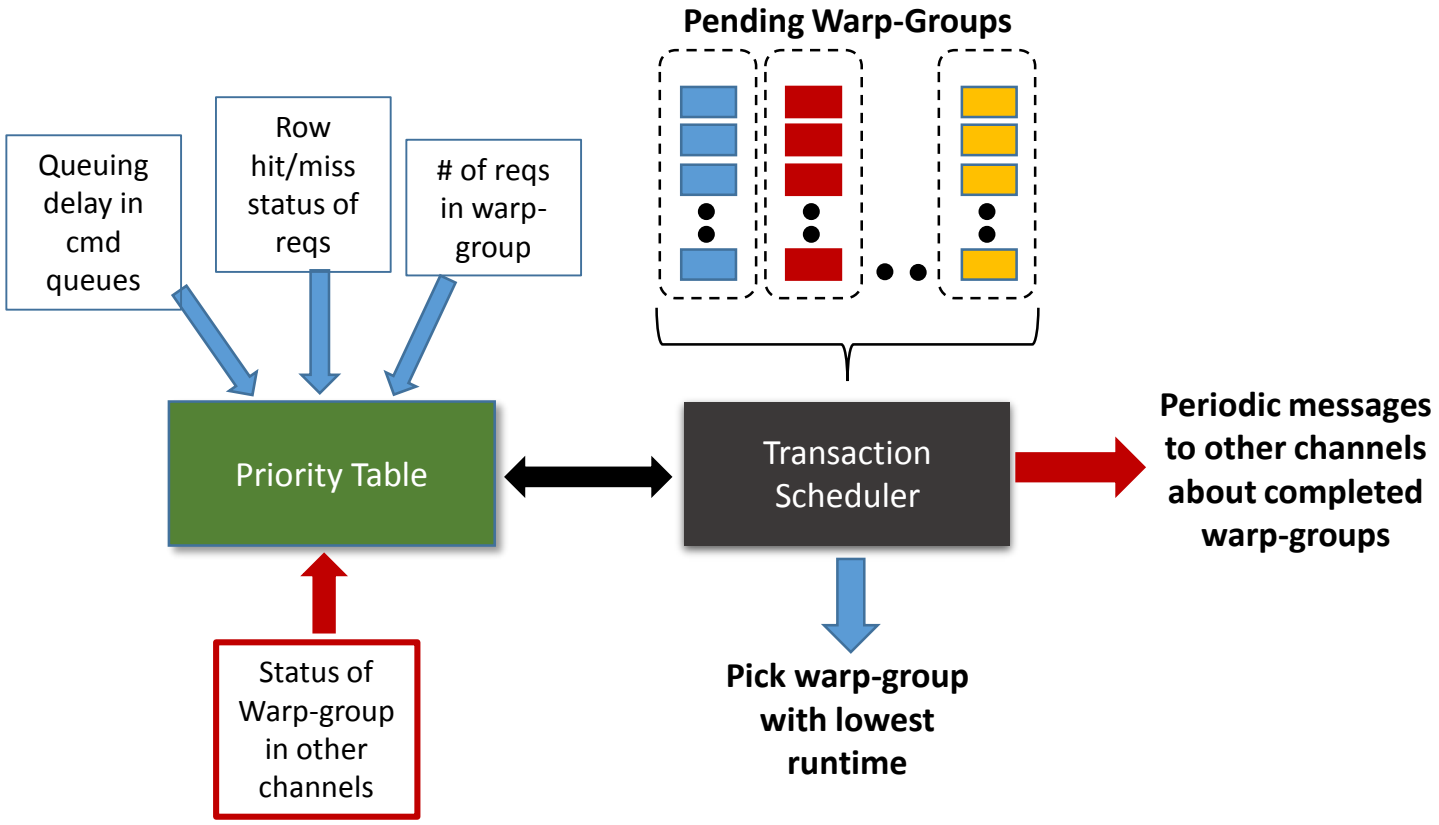
UTAH ARCH

Multiple Memory Controllers

- Channel level parallelism
  - Warp's requests sent to multiple memory channels
  - Independent scheduling at each controller
- Subset of warp's requests can be delayed at one or few memory controllers
- **Coordinate scheduling between controllers**
  - Prioritize warp-group that has already been serviced at other controllers
  - Coordination message broadcast to other controllers on completion of a warp-group.

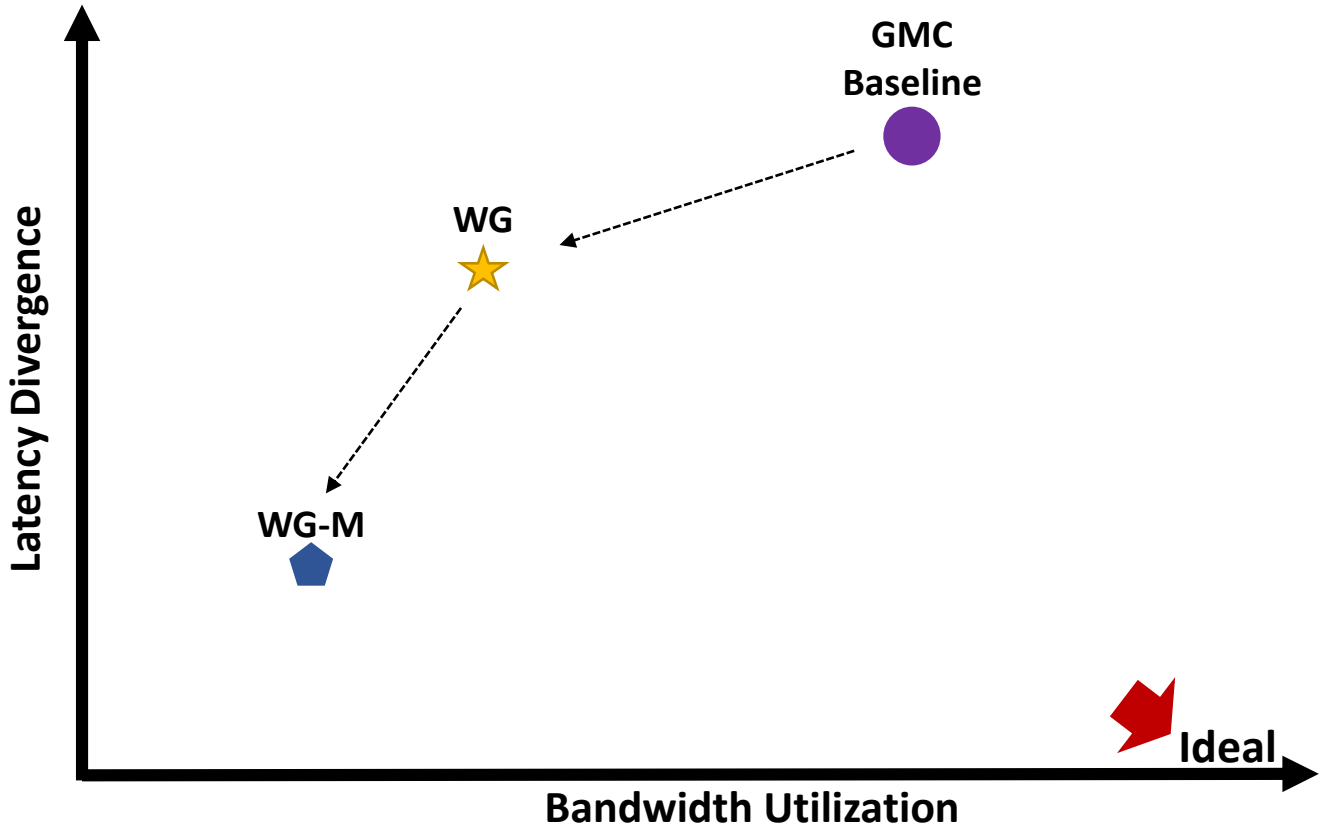
UTAH ARCH

Warp-Group Scheduling : Multi-Channel



UTAH ARCH

WG-M Scheduling



UTAH ARCH

Bandwidth-Aware Warp-Group Scheduling

- Warp-group scheduling negatively affects bandwidth utilization
  - Reduced row-hit rate
- Conflicting objectives
  - Issue row-miss request from current warp-group
  - Issue row-hit requests to maintain bus utilization
- Activate and Precharge idle cycles
  - Hidden by row-hits in other banks
- **Delay row-miss request to find the right slot**

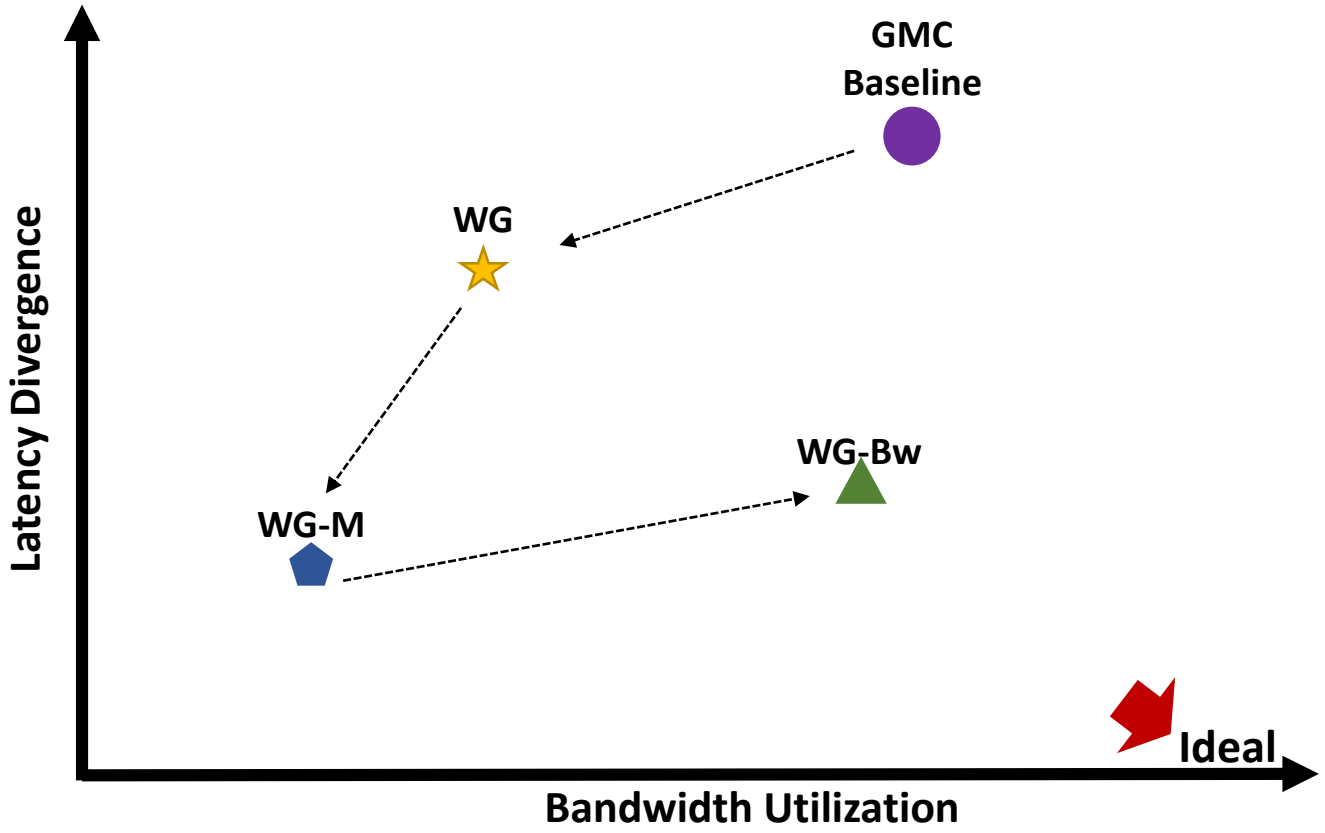
UTAH ARCH

Bandwidth-Aware Warp-Group Scheduling

- The minimum number of row-hits needed in other banks to overlap (*tRTP+tRP+tRCD*)
  - Determined by GDDR timing parameters
  - *Minimum efficient row burst (MERB)*
- Stored in a ROM looked up by Transaction Scheduler
- More banks with pending row-hits
  - smaller MERB
- Schedule row-miss after MERB row-hits have been issued to bank

UTAH ARCH

WG-Bw Scheduling



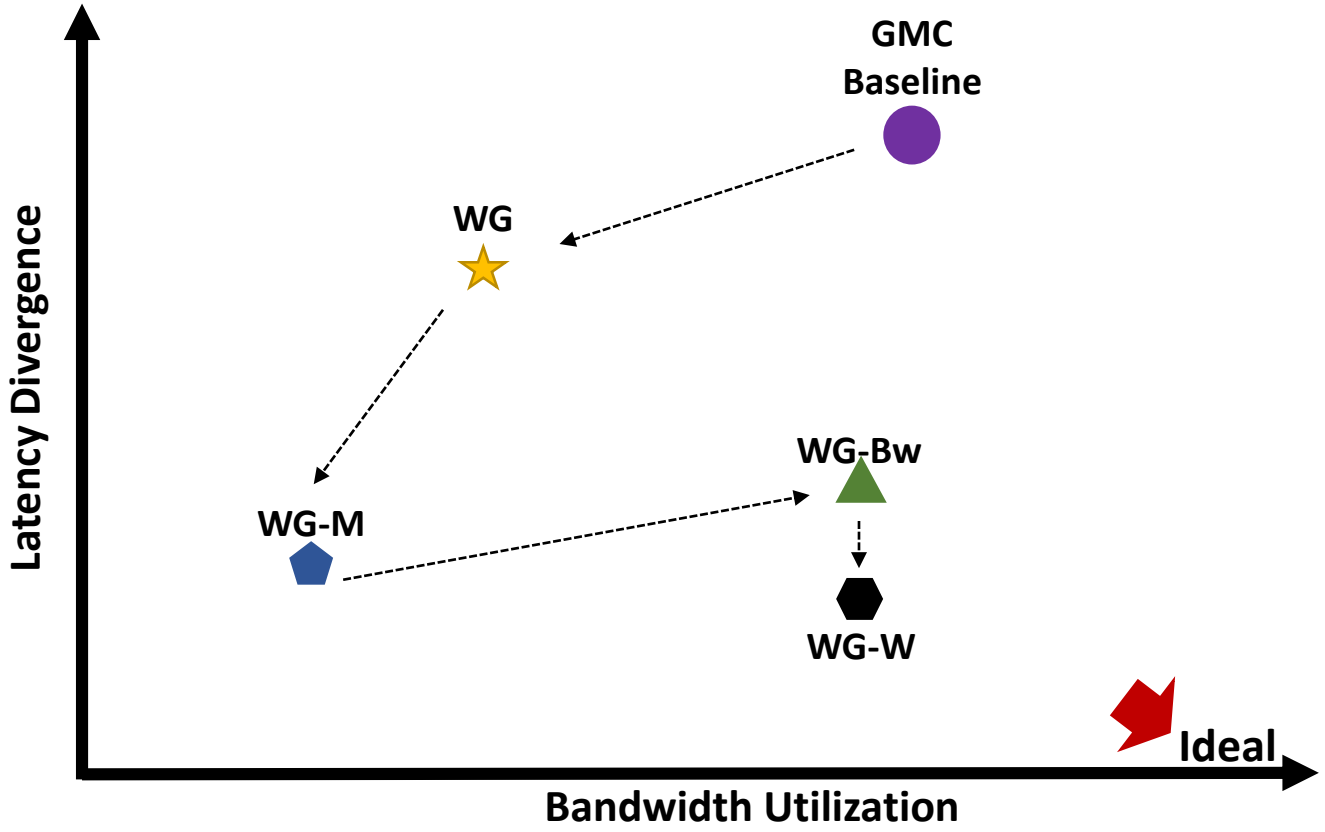
UTAH ARCH

Warp-Aware Write Draining

- Writes drained in batches
  - starts at High_Watermark
- Can stall small warp-groups
- When WQ reaches a threshold (lower than High_Watermark)
  - Drain singleton warp-groups only
- Reduce write-induced latency

UTAH ARCH

WG-scheduling



UTAH ARCH

Methodology

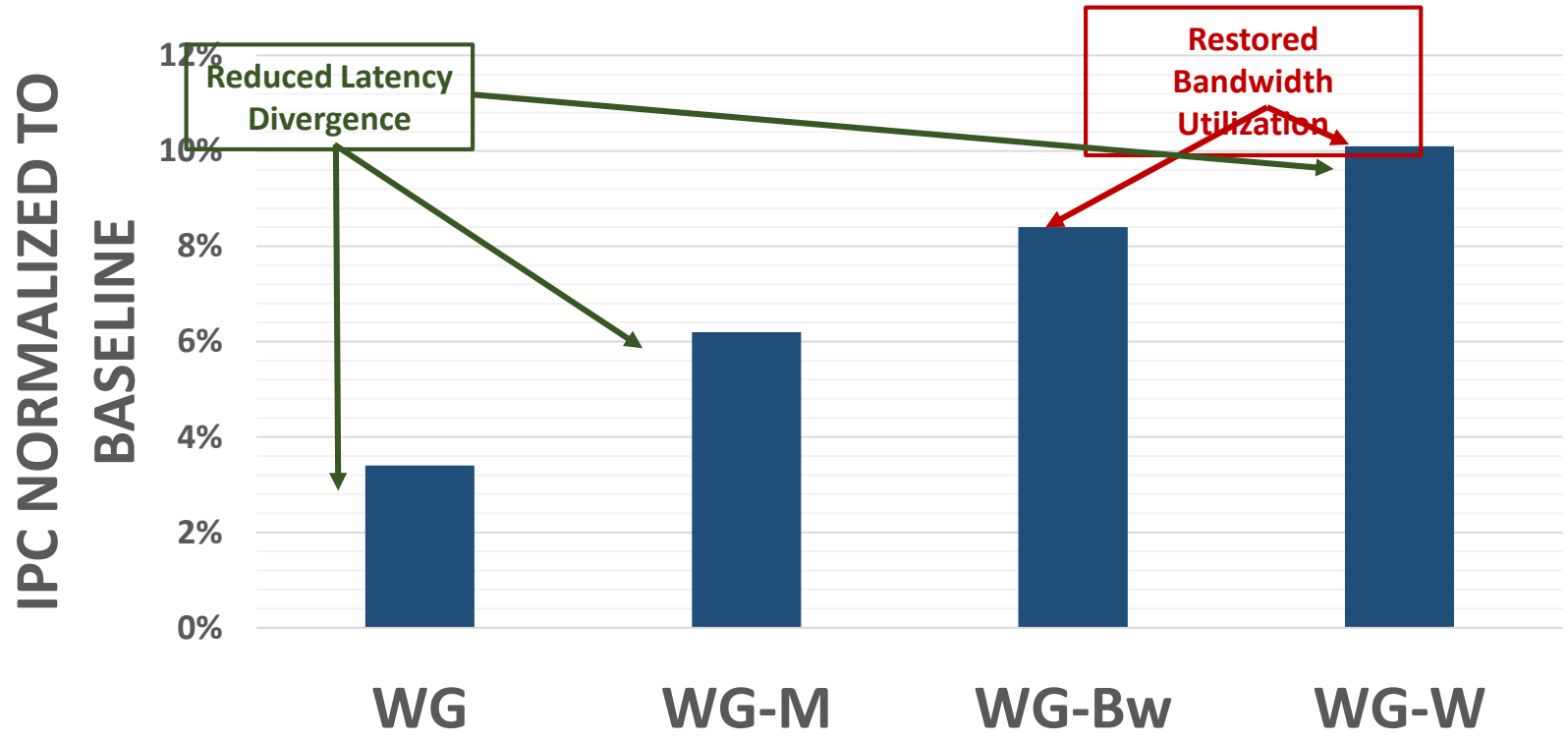
- GPGPUSim v3.1 : Cycle Accurate GPGPU simulator

SM Cores	30
Max Threads/Core	1024
Warp Size	32 Threads/warp
L1 / L2	32KB / 128 KB
DRAM	6Gbps GDDR5
DRAM Channels Banks	6 Channels 16 Banks/channel

- USIMM v1.3 : Cycle Accurate DRAM Simulator
  - modified to model GMC-baseline & GDDR5 timings
- Irregular and Regular workloads from Parboil, Rodinia, Lonestar, and MARS.

UTAH ARCH

Performance Improvement



UTAH ARCH

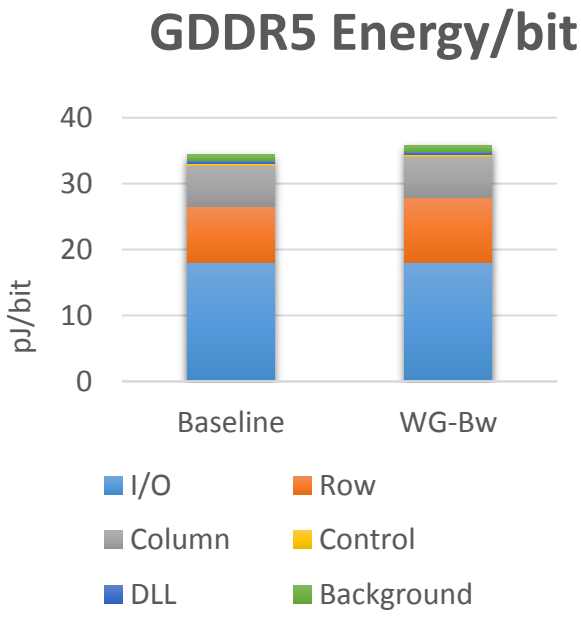
Impact on Regular Workloads

- Effective coalescing
- High spatial locality in warp-group
- WG scheduling works similar to GMC-baseline
  - No performance loss
- WG-Bw and WG-W provide
  - Minor benefits

UTAH ARCH

Energy Impact of Reduced Row Hit-Rate

- Scheduling Row-misses over Row-hits
  - Reduces the row-buffer hit rate 16%
- In GDDR5, power consumption dominated by I/O.
- Increase in DRAM power negligible compared to execution speed-up
  - **Net improvement in system energy**



UTAH ARCH

Conclusions

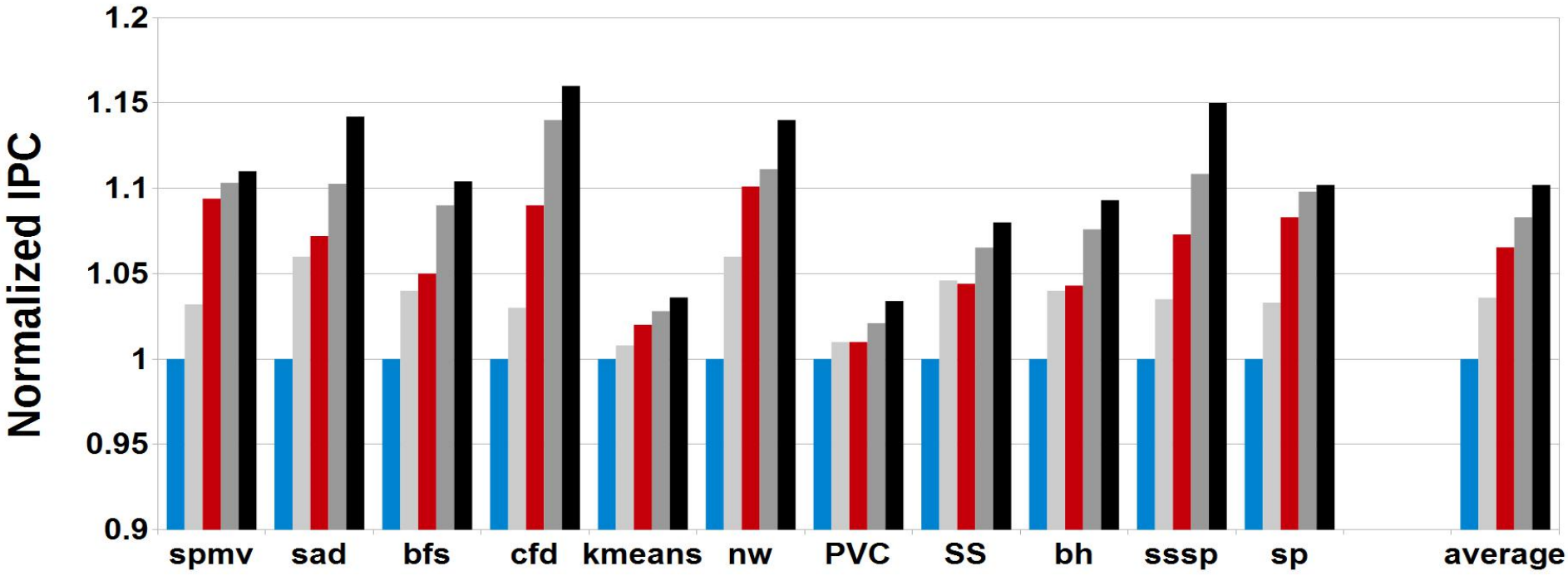
- Irregular applications place new demands on the GPU's memory system
- Memory scheduling can alleviate the issues caused by latency divergence
- Carefully orchestrating the scheduling of commands can help regain the bandwidth lost by warp-aware scheduling
- Future techniques must also include the cache-hierarchy in reducing latency divergence

UTAH ARCH

Thanks !

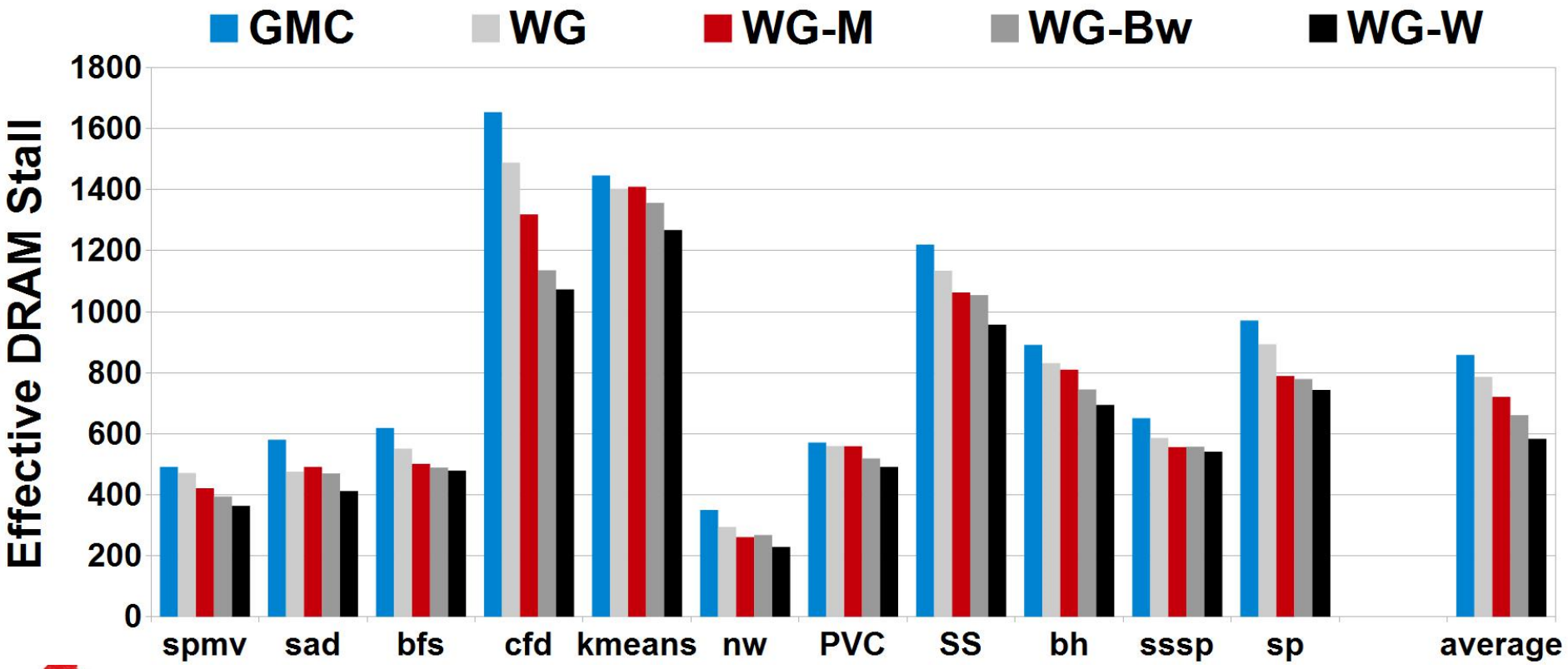
Backup Slides

Performance Improvement : IPC



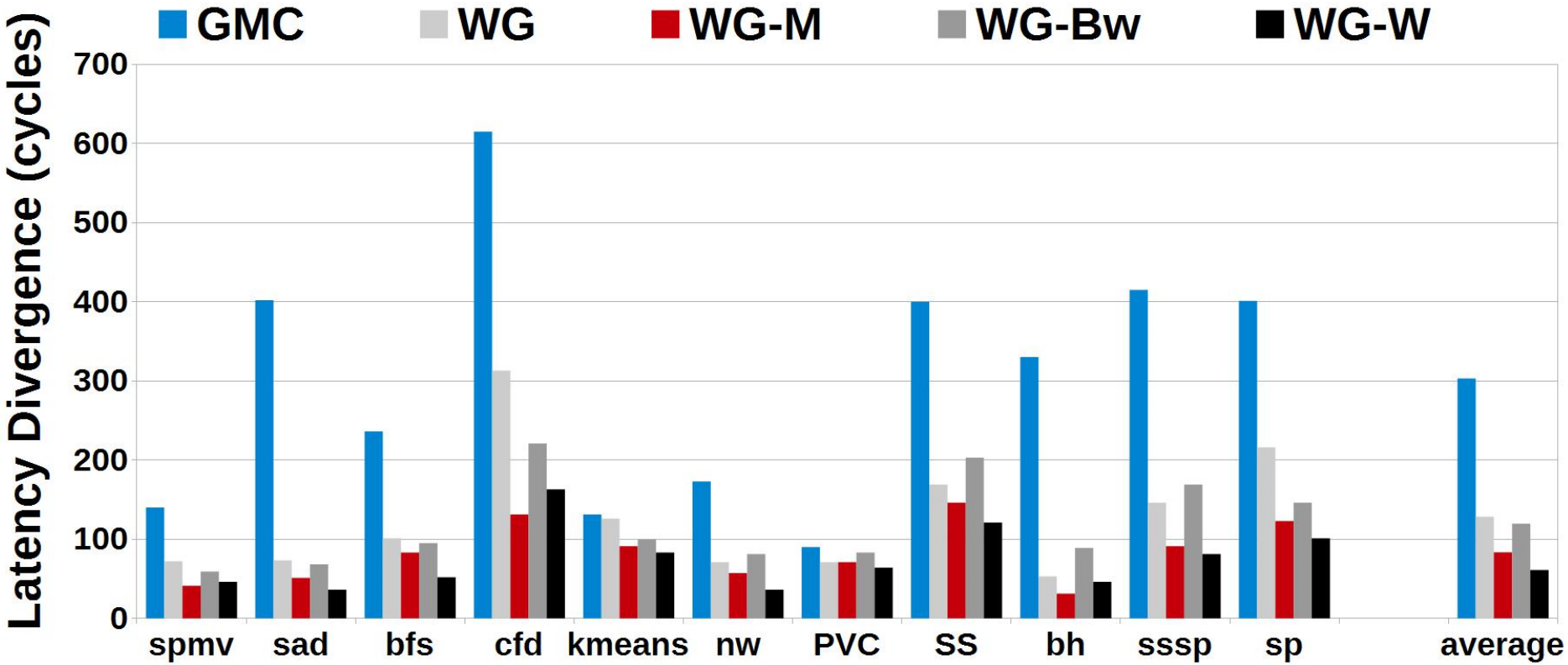
UTAH ARCH

Average Warp Stall Latency



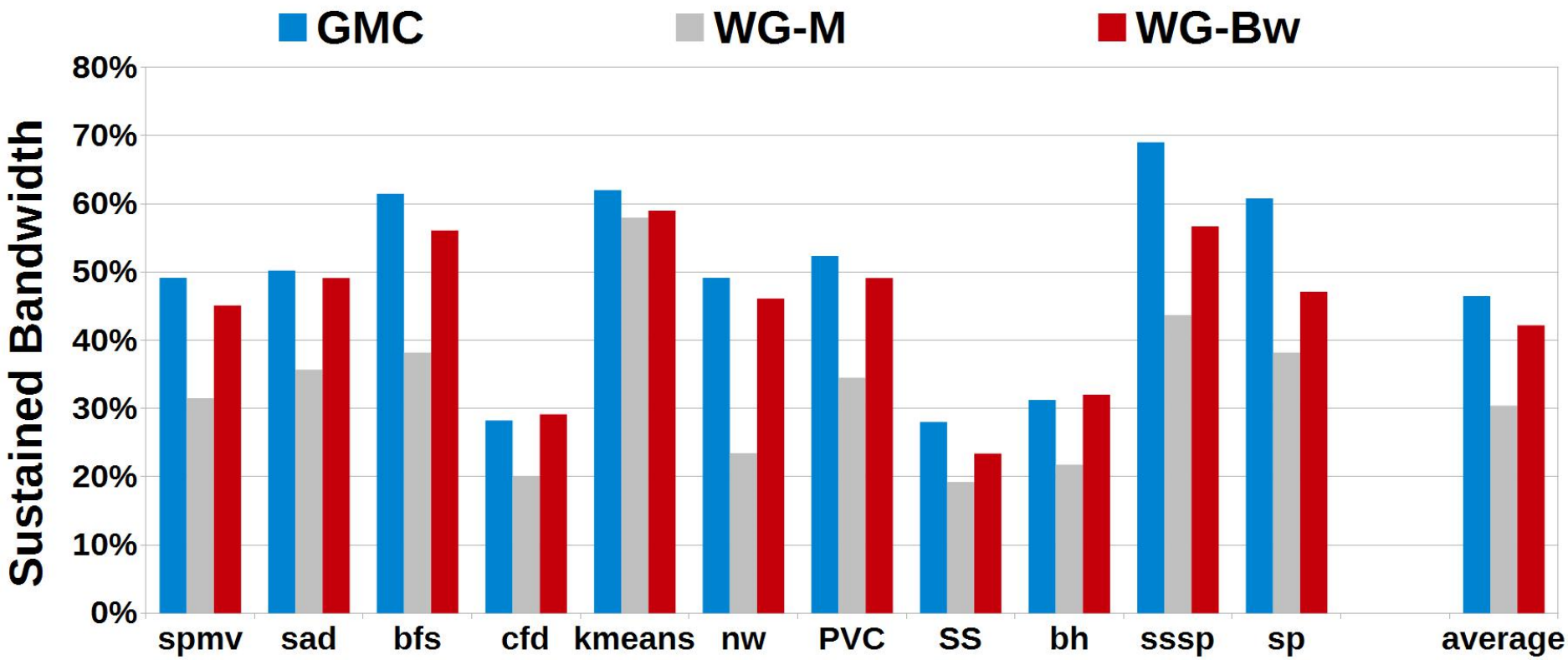
UTAH ARCH

DRAM Latency Divergence



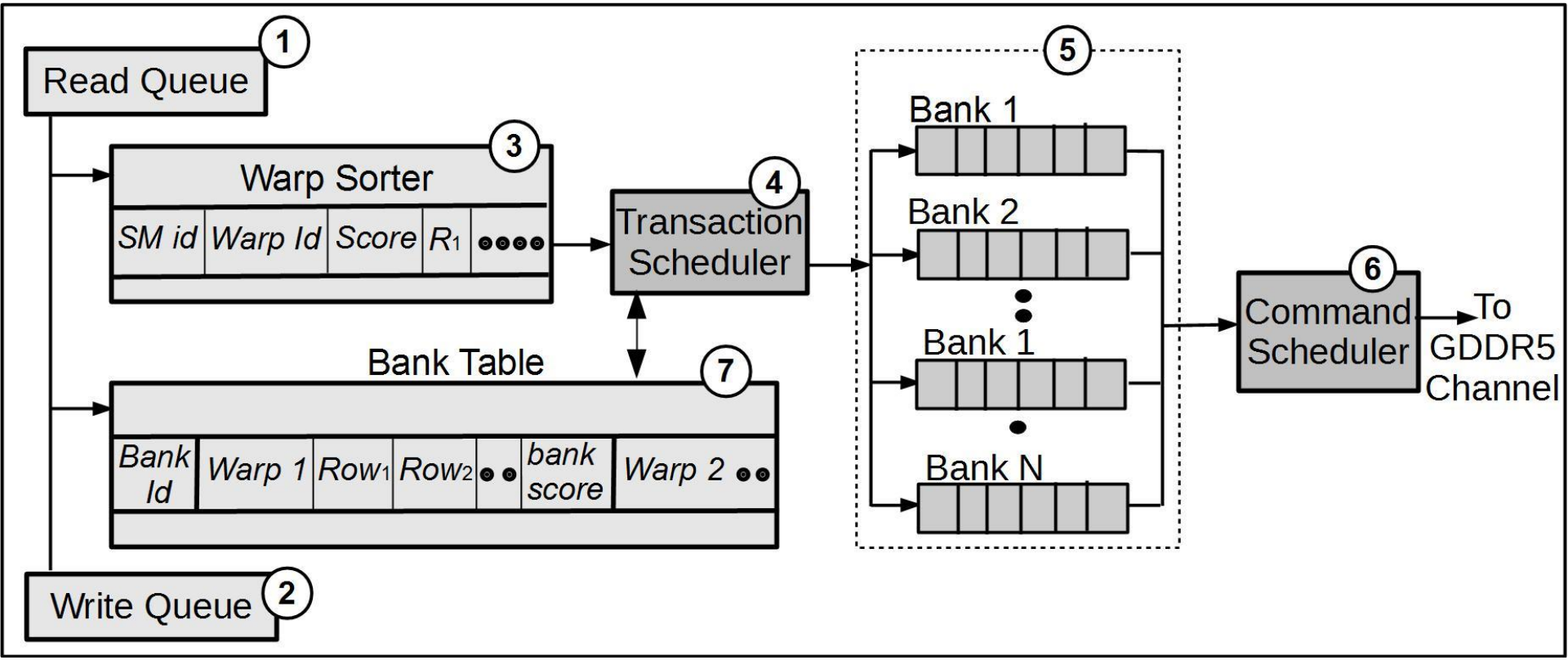
UTAH ARCH

Bandwidth Utilization



UTAH ARCH

Memory Controller Microarchitecture



UTAH ARCH

Warp-Group Scheduling

- Every batch assigned a priority-score
  - completion time of the longest request
- Higher priority to warp groups with
  - Few requests
  - High spatial locality
  - Lightly loaded banks
- Priorities updated after each warp-group scheduling
- Warp-group with lowest service time selected
  - Shortest-job-first based on *actual service time*, not number of requests

Priority-Based Cache Allocation in Throughput Processors

Dong Li[*†], Minsoo Rhu[*§†], Daniel R. Johnson[§], Mike O'Connor[§†], Mattan Erez[†], Doug Burger[‡] , Donald S. Fussell[†] and Stephen W. Keckler[§†]

† THE UNIVERSITY OF TEXAS AT AUSTIN

§ NVIDIA

‡ Microsoft

*First authors Li and Rhu have made equal contributions to this work and are listed alphabetically

Introduction

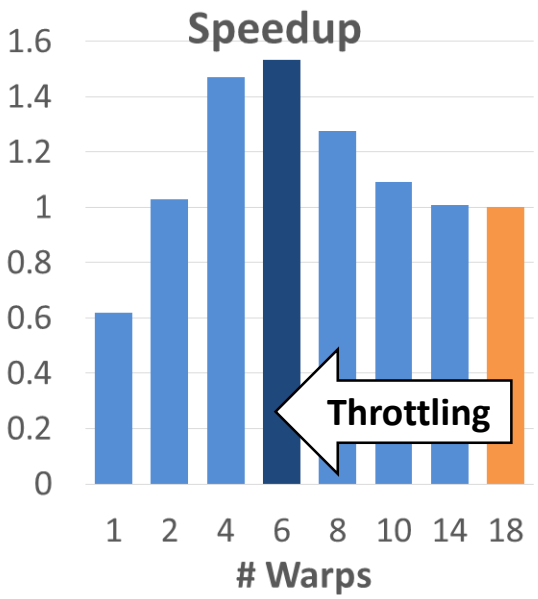
- Graphics Processing Units (GPUs)
  - General purpose throughput processors
  - Massive thread hierarchy, warp=32 threads

- **Challenge**:
  - Small caches + many threads → contention + cache thrashing

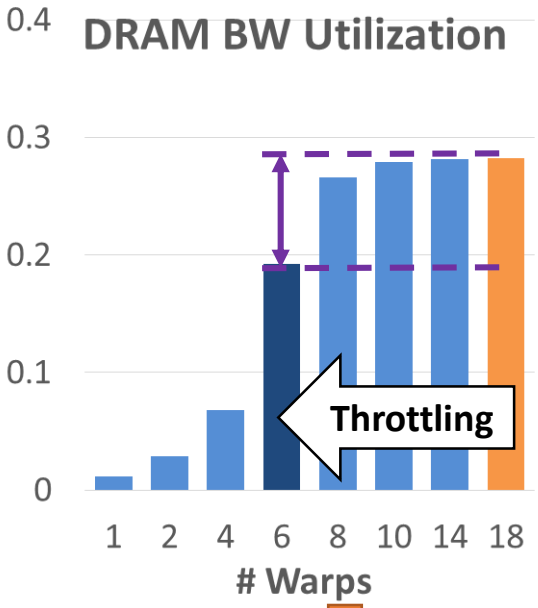
- Prior work: throttling thread level parallelism (TLP)
  - **Problem 1:** under-utilized system resources
  - **Problem 2:** unexplored cache efficiency

Motivation: Case Study (CoMD)

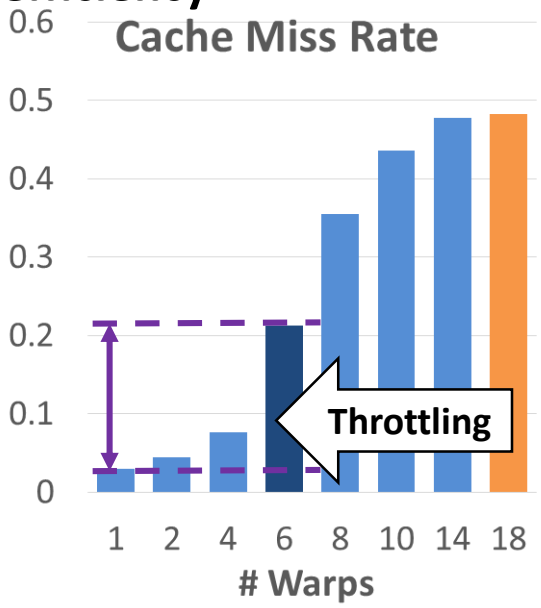
Throttling improves performance



Observation 1: Resource under-utilized



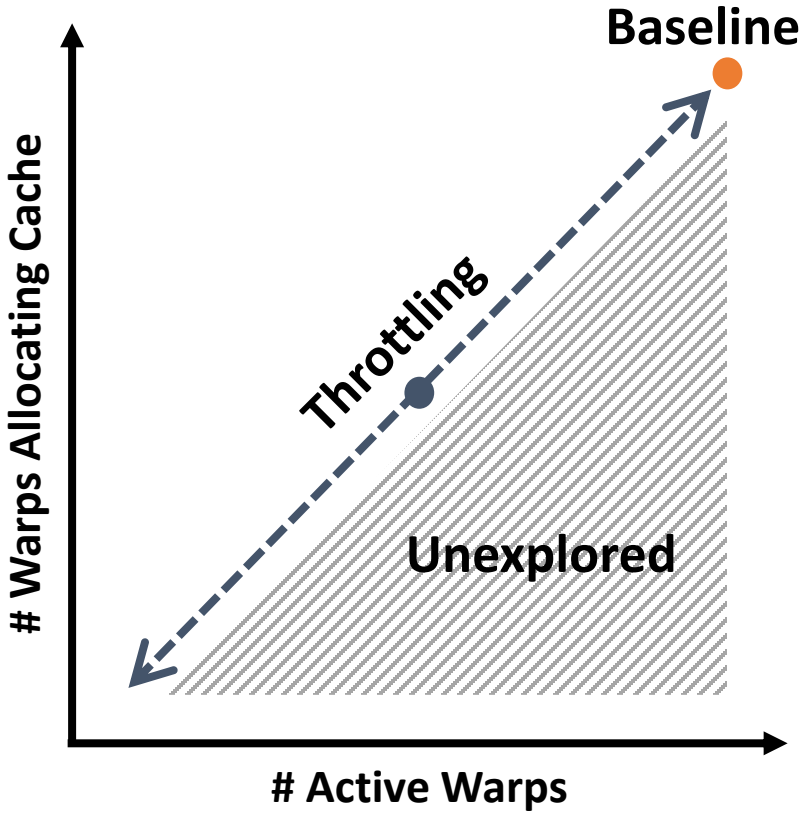
Observation 2: Unexplored cache efficiency



Best Throttled max TLP (default)

Motivation

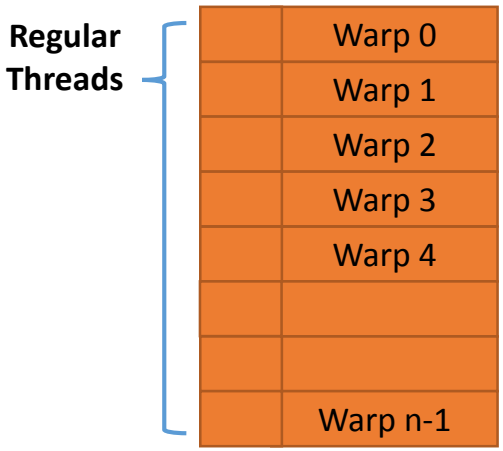
- Throttling
  - Tradeoff between cache efficiency and parallelism.
  - # Active Warps = # Warps Allocating Cache
- Idea
  - Decoupling cache efficiency from parallelism



Our Proposal: Priority Based Cache Allocation (PCAL)

Baseline

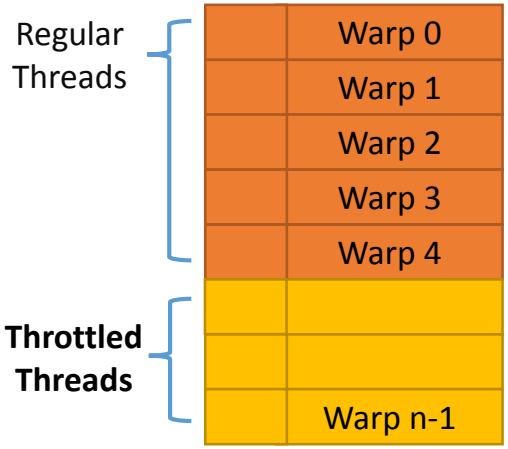
Standard operation



- Regular threads: threads have all capabilities, operate as usual (full access to the L1 cache)

TLP Throttling

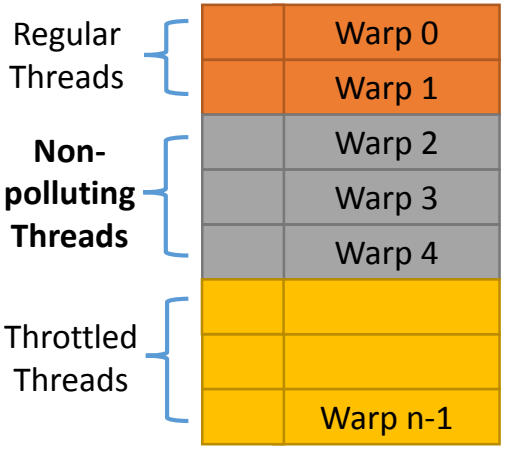
TLP reduced to improve performance



- Regular threads: threads have all capabilities, operate as usual (full access to the L1 cache)
- Throttled threads: throttled (not runnable)

Proposed Solution: PCAL

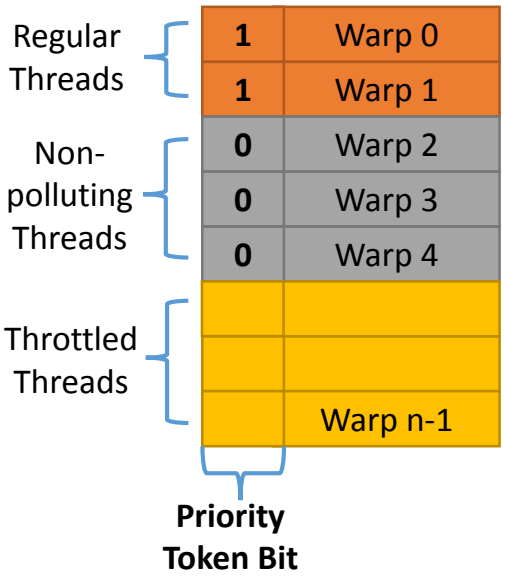
Three subsets of threads



- Regular threads: threads have all capabilities, operate as usual (full access to the L1 cache)
- Non-polluting threads: runnable, but prevented from *polluting* L1 cache
- Throttled threads: throttled (not runnable)

Proposed Solution: PCAL

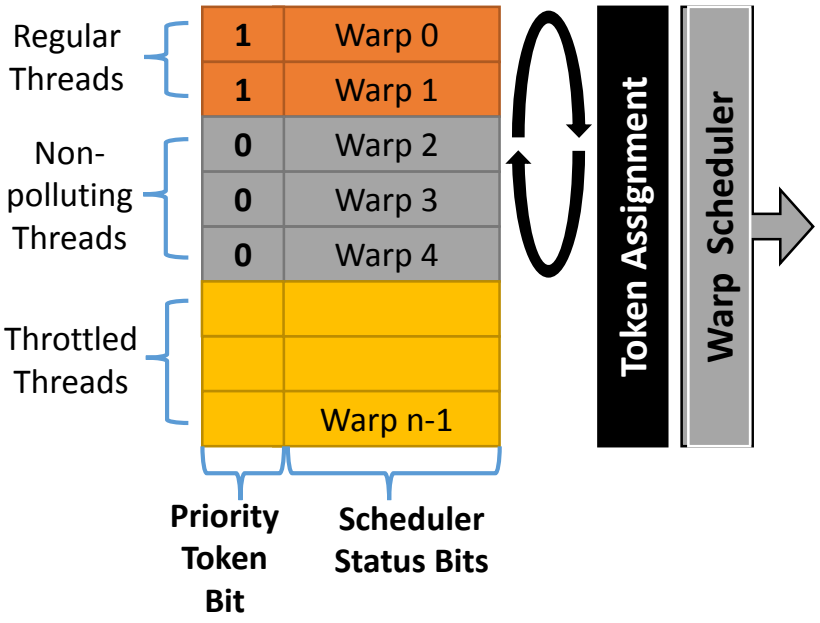
Tokens for privileged cache access



- **Tokens** grant capabilities or privileges to some threads
- Threads with tokens are allowed to ***allocate and evict*** L1 cache lines
- Threads without tokens can read and write, but ***not allocate or evict***, L1 cache lines

Proposed solution: Static PCAL

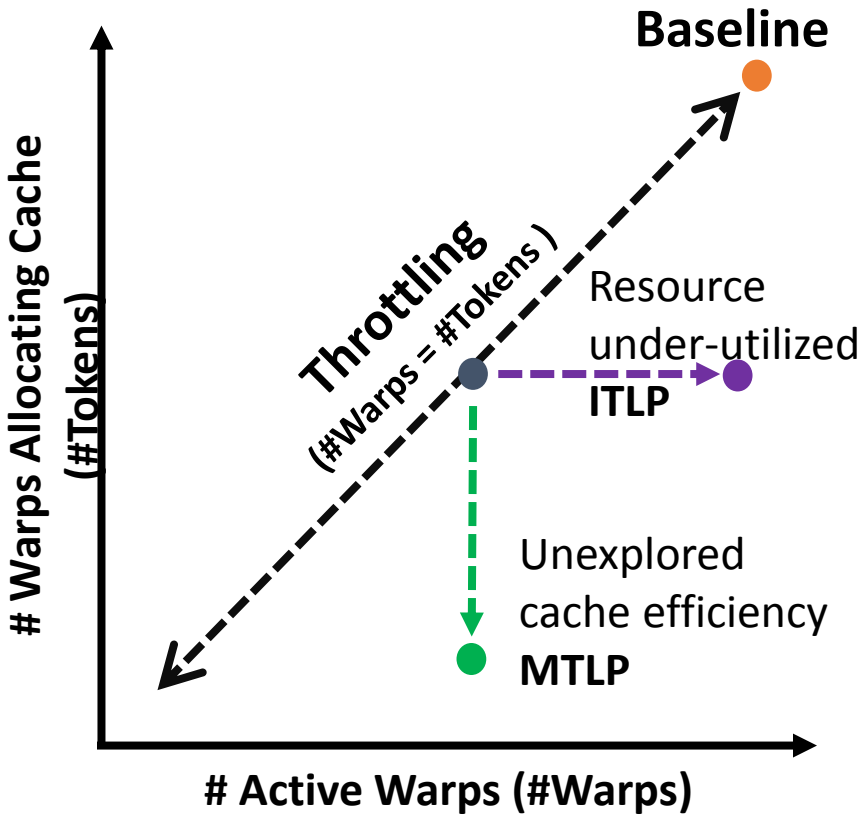
Enhanced scheduler operation



- HW implementation: simplicity
  - Scheduler manages per-warp token bits
  - Token bits sent with memory requests
- Policies: token assignment
  - Assign at warp launch, release at termination
  - Re-assigned to oldest token-less warp
- Parameters supplied by software prior to kernel launch

Two Optimization Strategies

to exploit the 2-D (#Warps, #Tokens) search space



- **1. Increasing TLP (ITLP)**
  - Adding non-polluting warps
  - Without hurting regular warps
- **2. Maintaining TLP (MTLP)**
  - Reduce #Token to increase the efficiency
  - Without decreasing TLP

Proposed Solution: Dynamic PCAL

- Decide parameters at run time
  - Choose MTLP/ITLP based on resource usage performance counter
  - For MTLP: search **#Tokens** in parallel
  - For ITLP: search **#Warps** in sequential

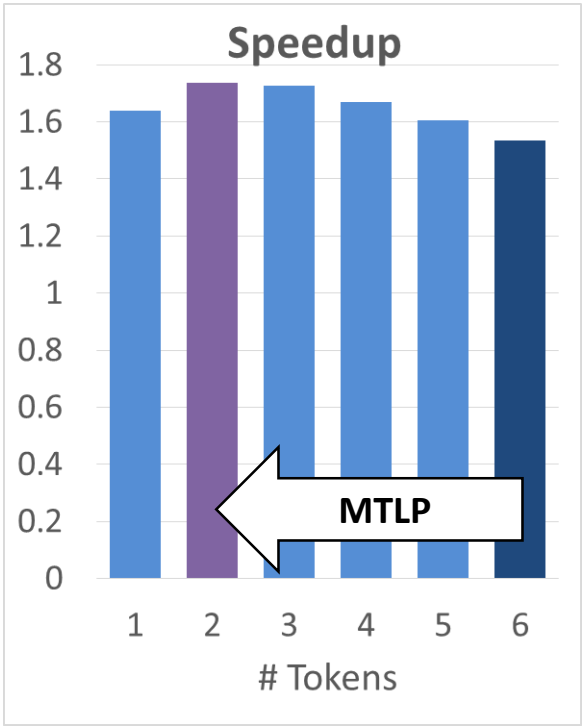
- Please refer to our paper for more details

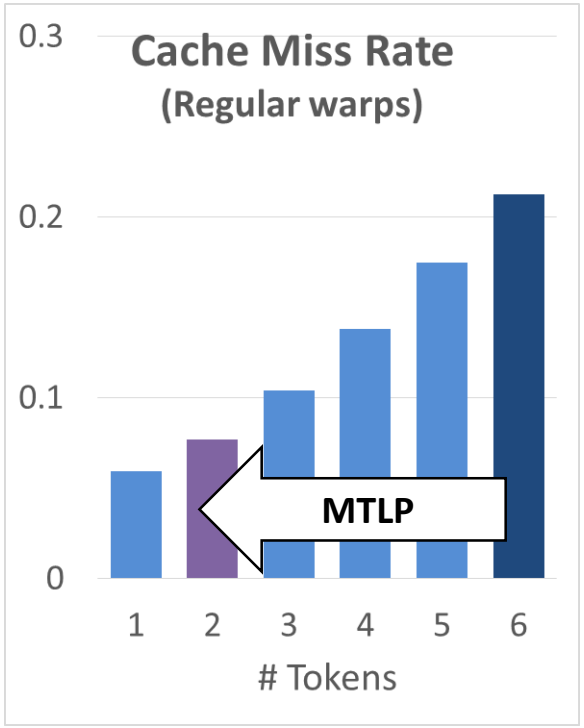
Evaluation

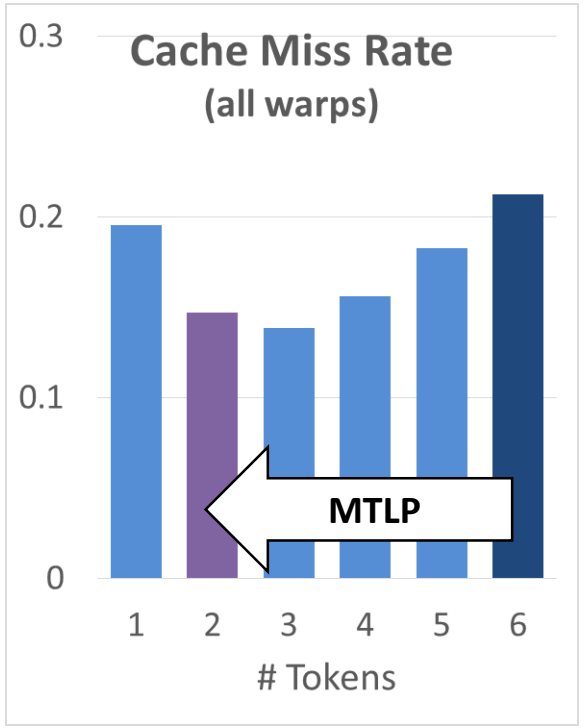
- Simulation: GPGPU-Sim
  - Open source academic simulator, configured similar to Fermi (full-chip)
- Benchmarks
  - Open source GPGPU suites: Parboil, Rodinia, LonestarGPU. etc
  - Selected benchmark subset shows *sensitivity to cache size*

MTLP Example (CoMD)

MTLP: reducing #Tokens, keeping #Warps =6



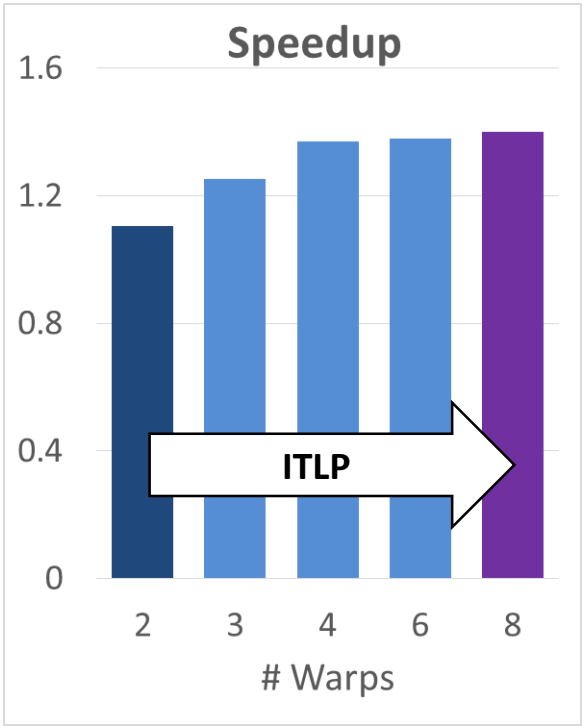


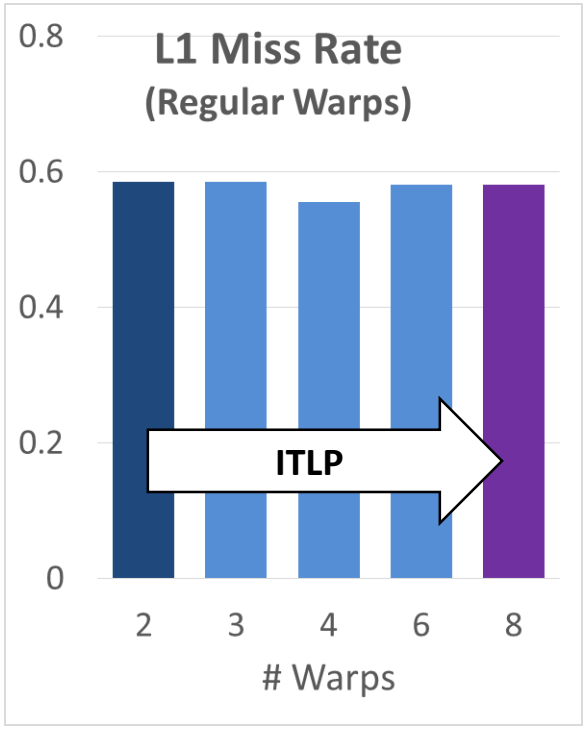


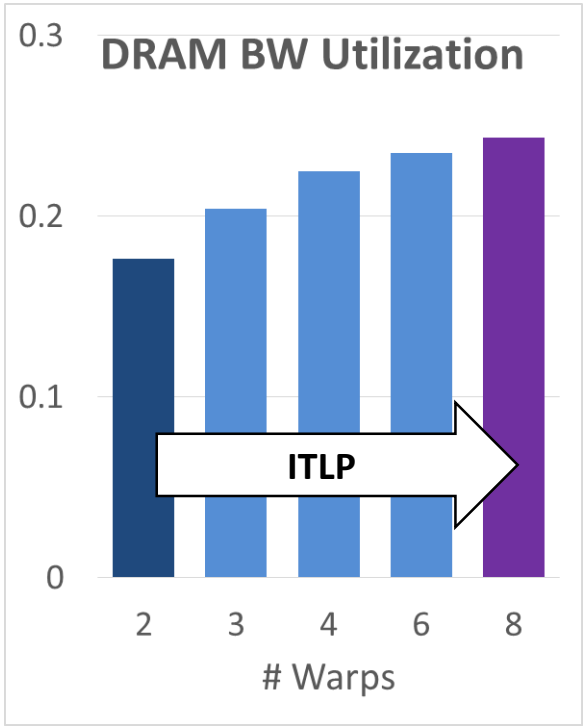
PCAL with MTLP Best Throttled

ITLP Example (Similarity-Score)

ITLP: increasing #Warps, keeping #Tokens=2

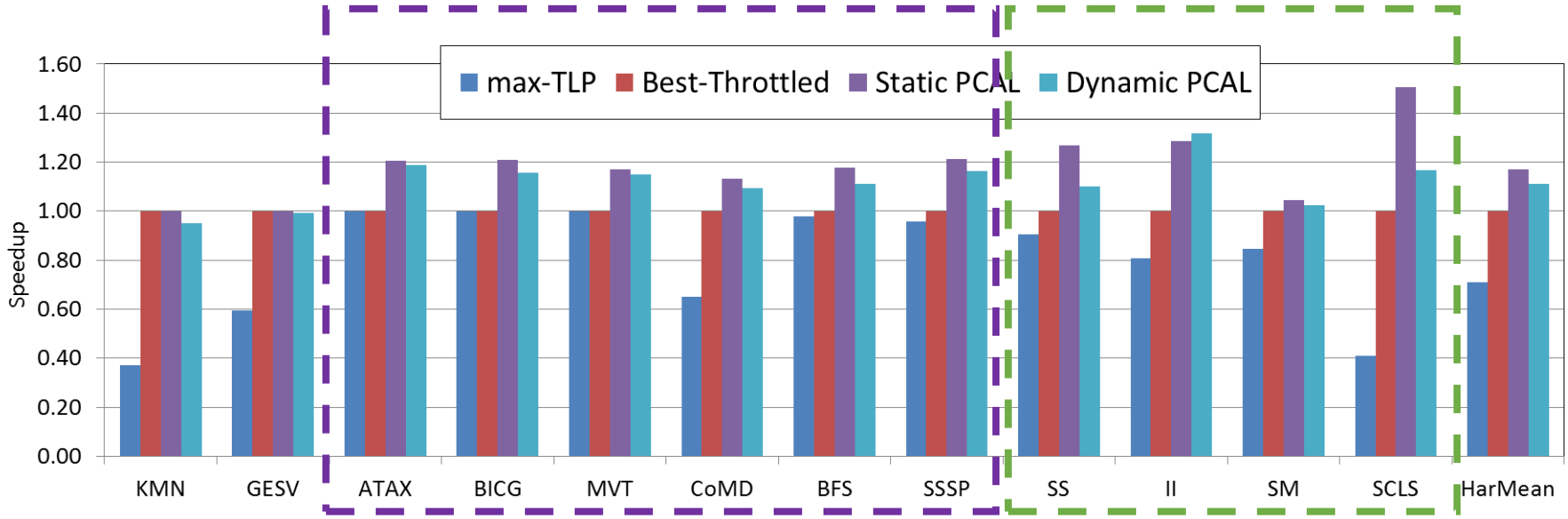






PCAL with ITLP Best Throttled

PCAL Results Summary

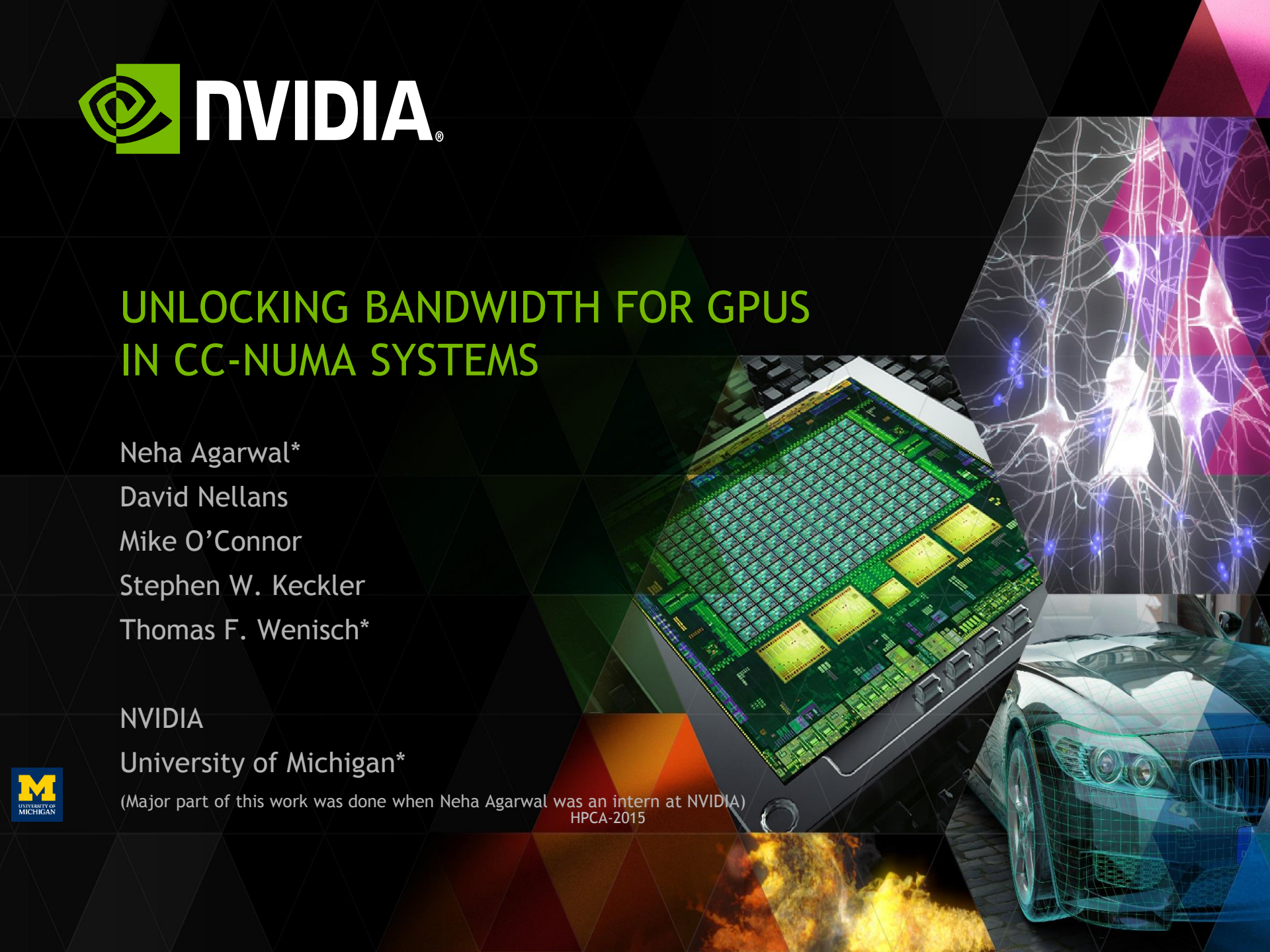


- Baseline: best throttled results
- Performance improvement: Static-PCAL: 17%, Dynamic-PCAL: 11%

Conclusion

- Existing throttling approach may lead to two problems:
  - Resources underutilization
  - Unexplored cache efficiency
- We propose PCAL, a simple mechanism to rebalance cache efficiency and parallelism
  - ITLP: increases parallelism without hurting regular warps
  - MTLP: alleviates cache thrashing while maintaining parallelism
- Throughput improvement over best throttled results
  - Static PCAL: 17%
  - Dynamic PCAL: 11%

Questions



UNLOCKING BANDWIDTH FOR GPUS IN CC-NUMA SYSTEMS

Neha Agarwal*

David Nellans

Mike O'Connor

Stephen W. Keckler

Thomas F. Wenisch*

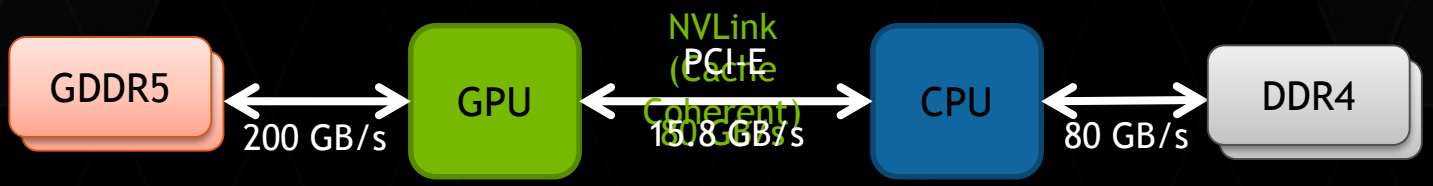
NVIDIA

University of Michigan*

(Major part of this work was done when Neha Agarwal was an intern at NVIDIA)

HPCA-2015

EVOLVING GPU MEMORY SYSTEM



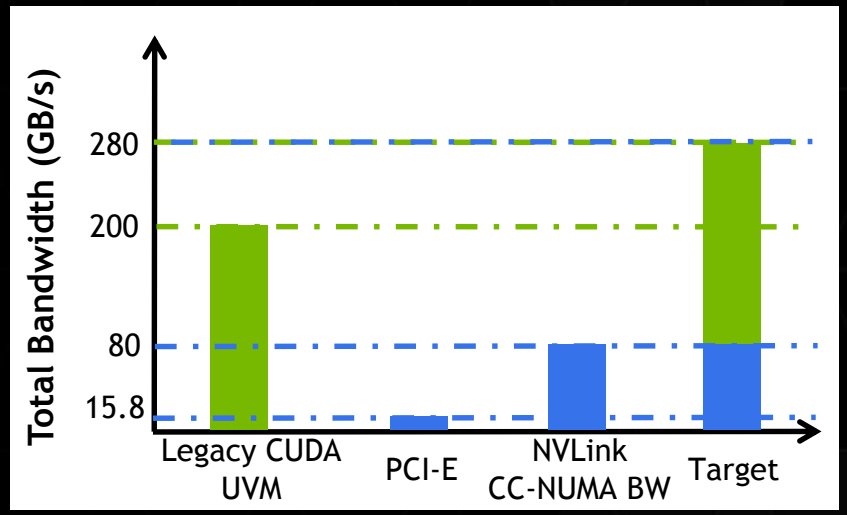
Roadmap

CUDA 1.0-5.x	CUDA 6.0+: Current	Future
cudaMemcpy	Unified virtual memory	CPU-GPU cache-coherent high BW interconnect
Programmer controlled copying to GPU memory	Run-time controlled copying → Better productivity	How to best exploit full BW while maintaining programmability?

HPCA-2015

DESIGN GOALS & OPPORTUNITIES

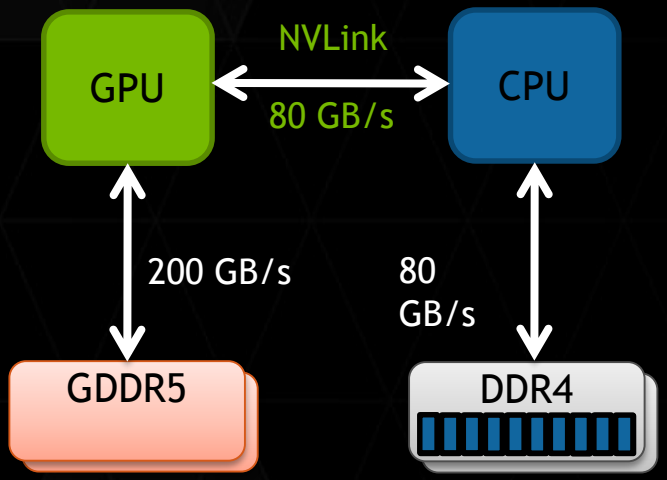
- Simple programming model:
  - No need for explicit data copying
- Exploit full DDR + GDDR BW
  - Additional 30% BW via NVLink
  - Crucial to BW sensitive GPU apps

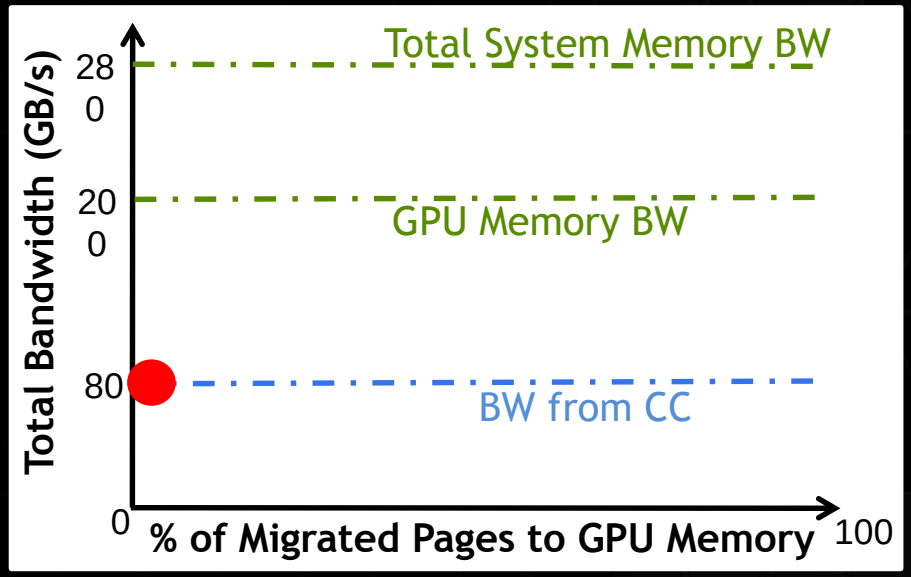


Design intelligent dynamic page migration policies to achieve both these goals

HPCA-2015

BANDWIDTH UTILIZATION

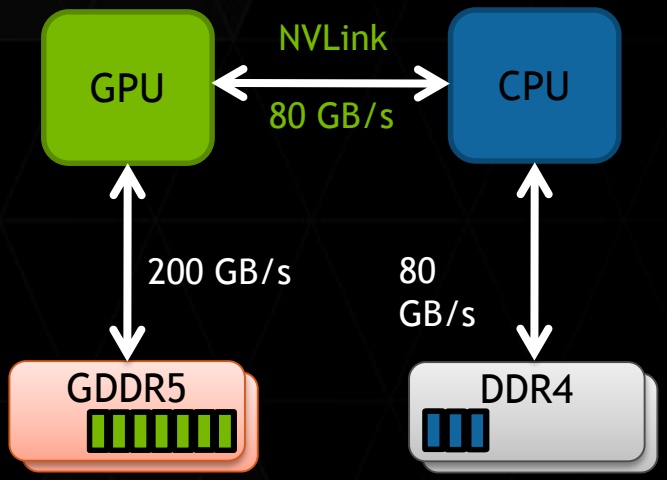


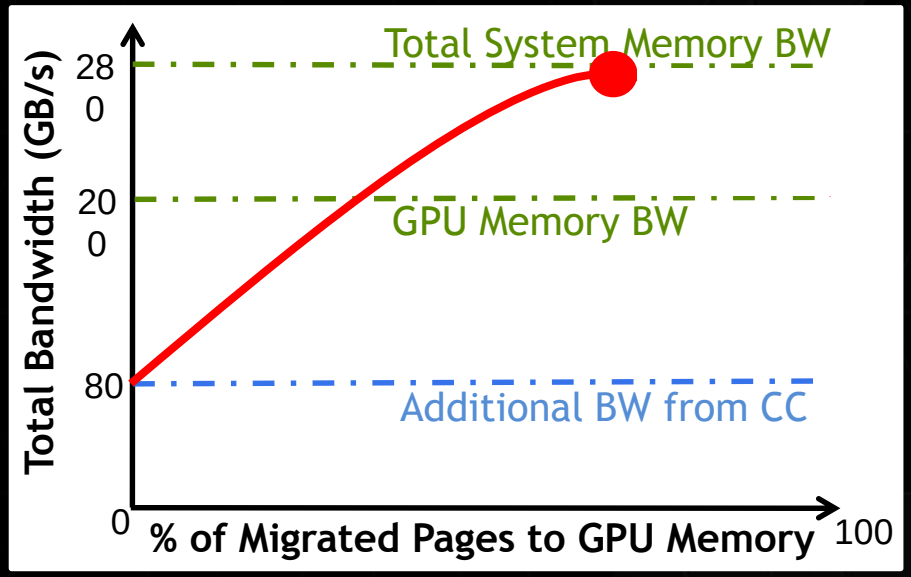


- Coherence-based accesses, no page migration
- Wastes GPU memory BW

HPCA-2015

BANDWIDTH UTILIZATION



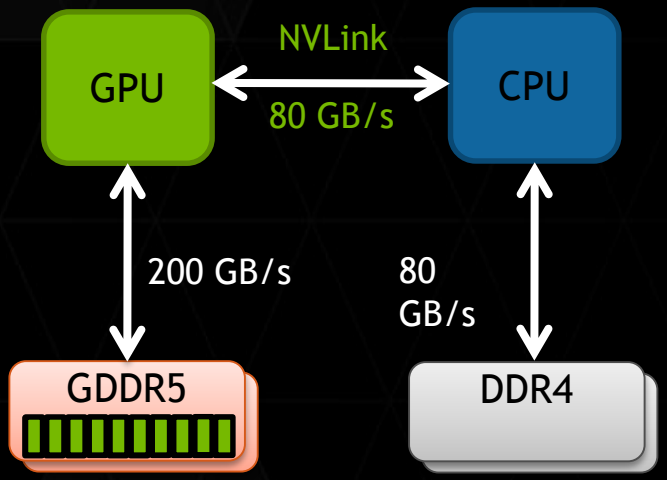


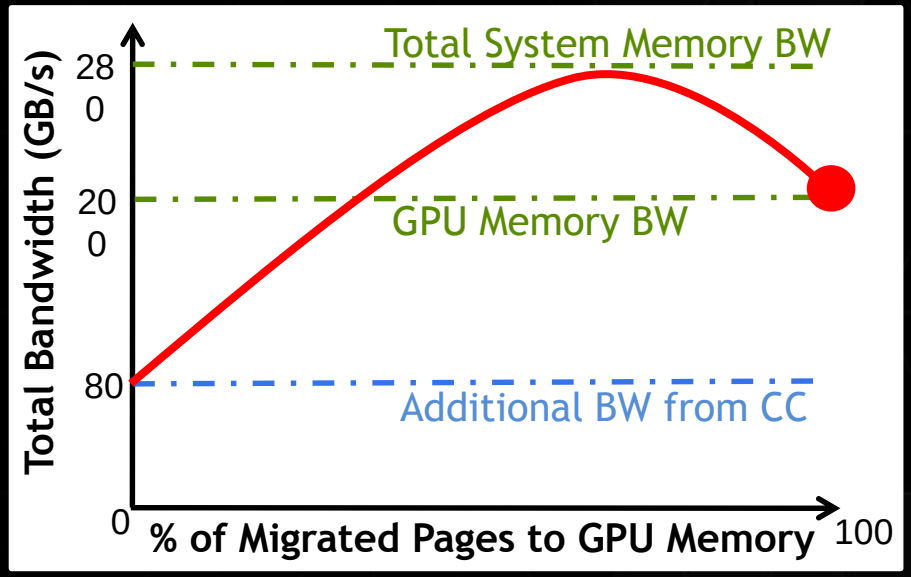
- Static Oracle: Place data in the ratio of memory bandwidths [ASPLOS'15]

Dynamic migration can exploit the full system memory BW

HPCA-2015

BANDWIDTH UTILIZATION





- Excessive migration leads to under-utilization of DDR BW

Migrate pages for optimal BW utilization

HPCA-2015

CONTRIBUTIONS

Intelligent Dynamic Page Migration

- Aggressively migrate pages upon First-Touch to GDDR memory
- Pre-fetch neighbors of touched pages to reduce TLB shootdowns
- Throttle page migrations when nearing peak BW

Dynamic page migration performs 1.95x better than no migration
Comes within 28% of the static oracle performance
6% better than Legacy CUDA

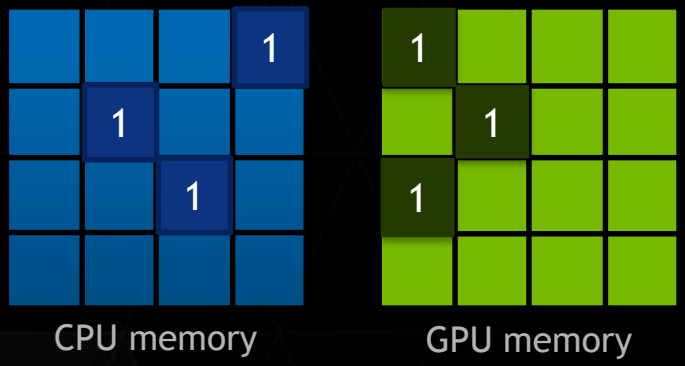
HPCA-2015

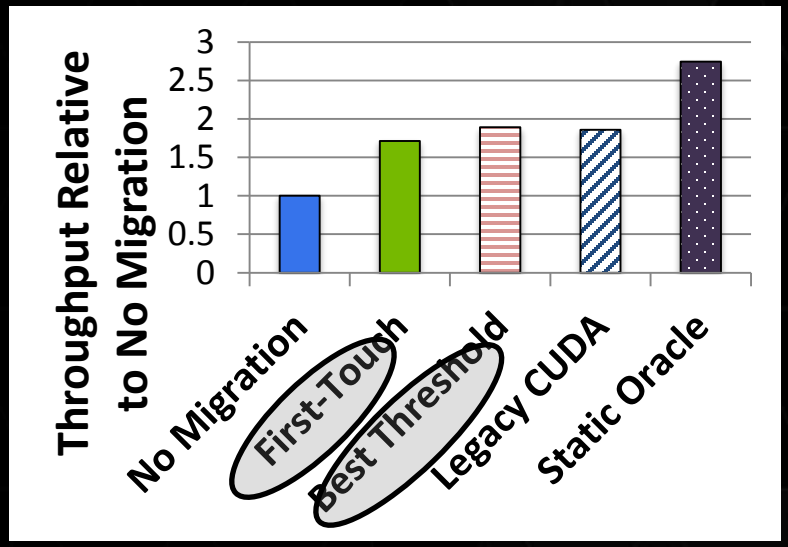
OUTLINE

- Page Migration Techniques
  - First-Touch page migration
  - Range-Expansion to save TLB shootdowns
  - BW balancing to stop excessive migrations
- Results & Conclusions

FIRST-TOUCH PAGE MIGRATION

- Naive: Migrate pages that are touched



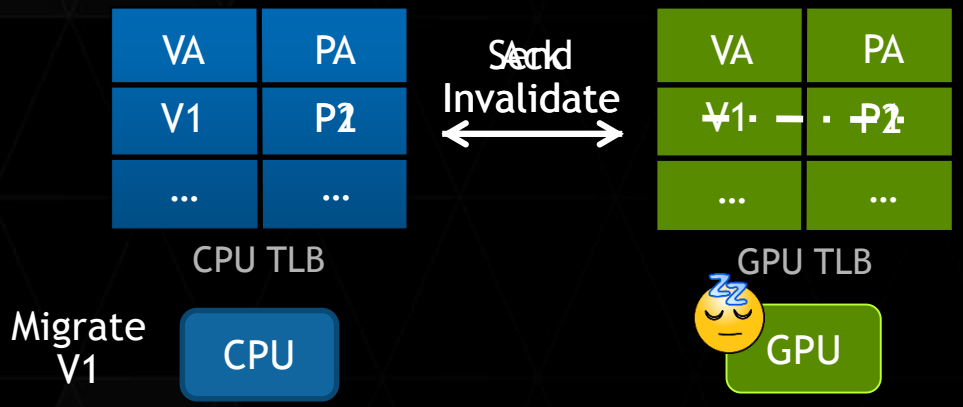


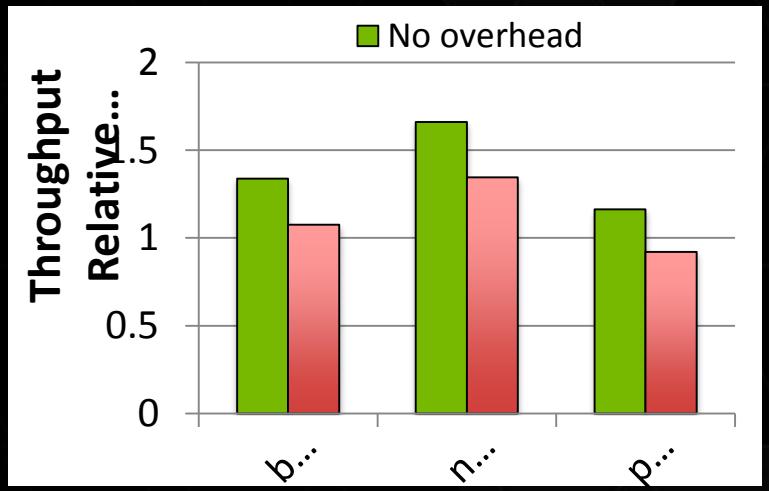
- First-Touch migration approaches Legacy CUDA

First-Touch migration is cheap, no hardware counters required

HPCA-2015

PROBLEMS WITH FIRST-TOUCH MIGRATION





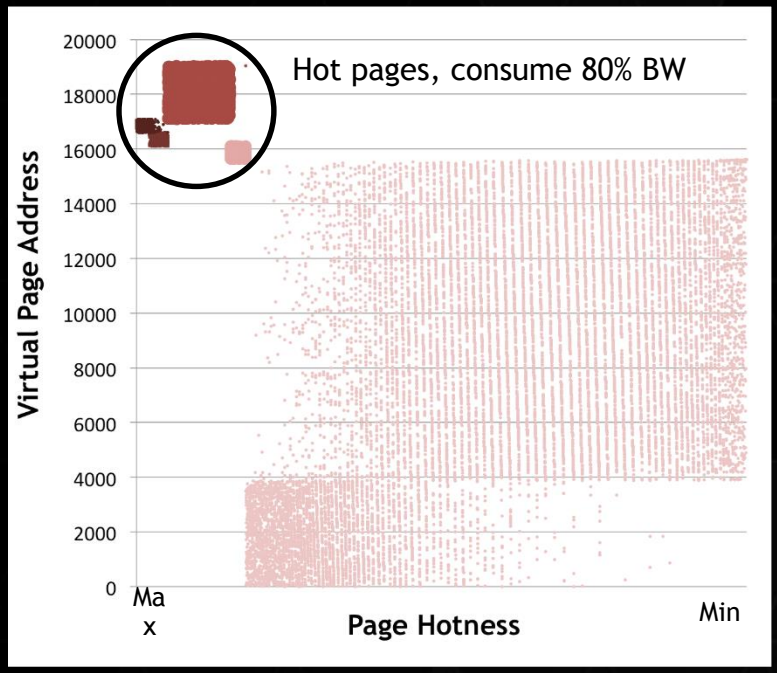
- TLB shootdowns may negate benefits of page migration

How to migrate pages without incurring shootdown cost?

HPCA-2015

PRE-FETCH TO AVOID TLB SHOOTDOWNS

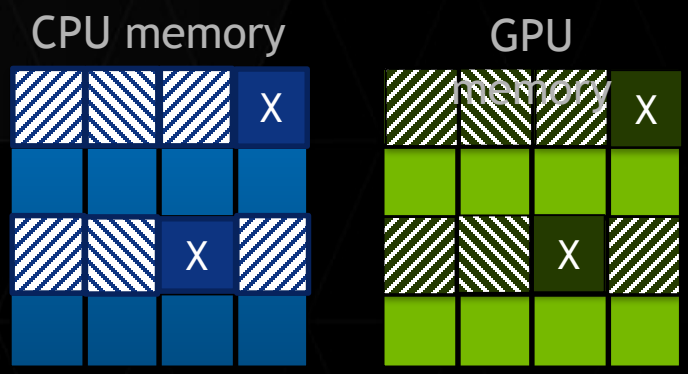
- Intuition: Hot virtual addresses are clustered
- Pre-fetch pages before access by the GPU

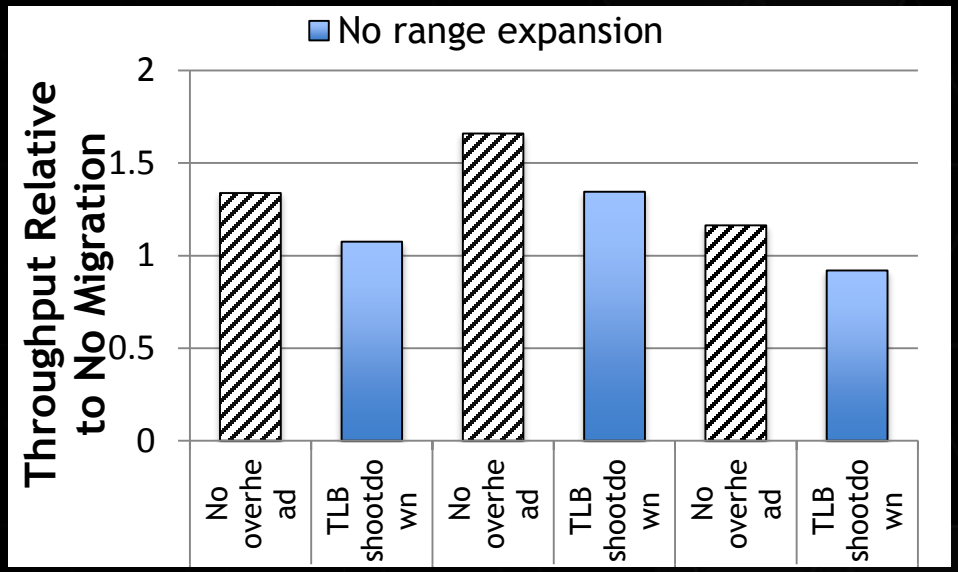


No shootdown cost for pre-fetched pages

HPCA-2015

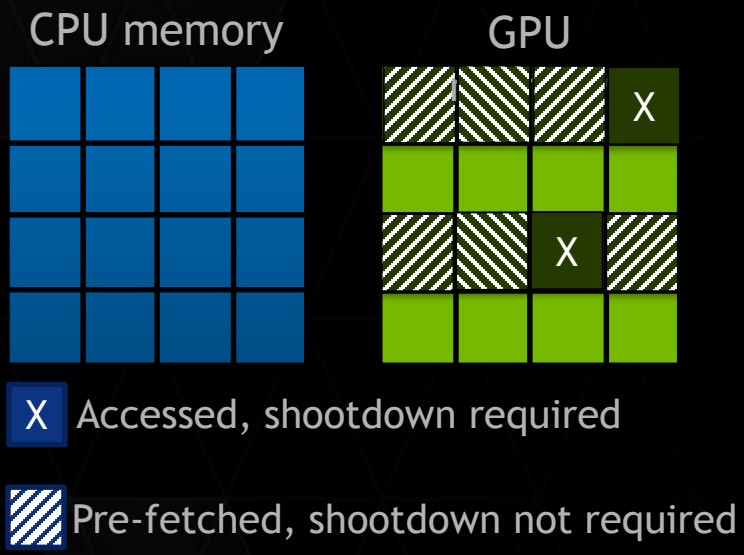
MIGRATION USING RANGE-EXPANSION

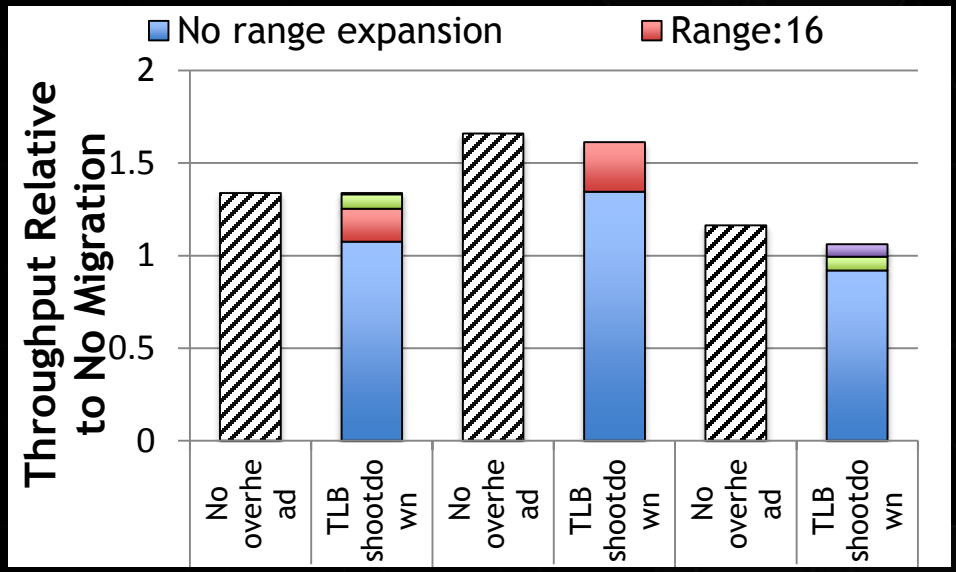




- Pre-fetch pages in spatially contiguous range

MIGRATION USING RANGE-EXPANSION



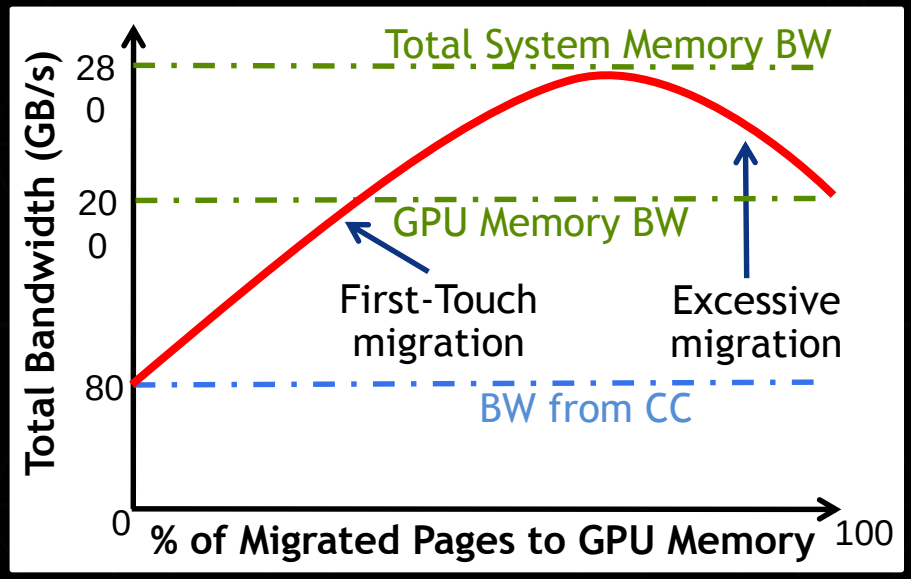


- Pre-fetch pages in spatially contiguous range

Range-Expansion hides TLB shootdown overhead

HPCA-2015

REVISITING BANDWIDTH UTILIZATION

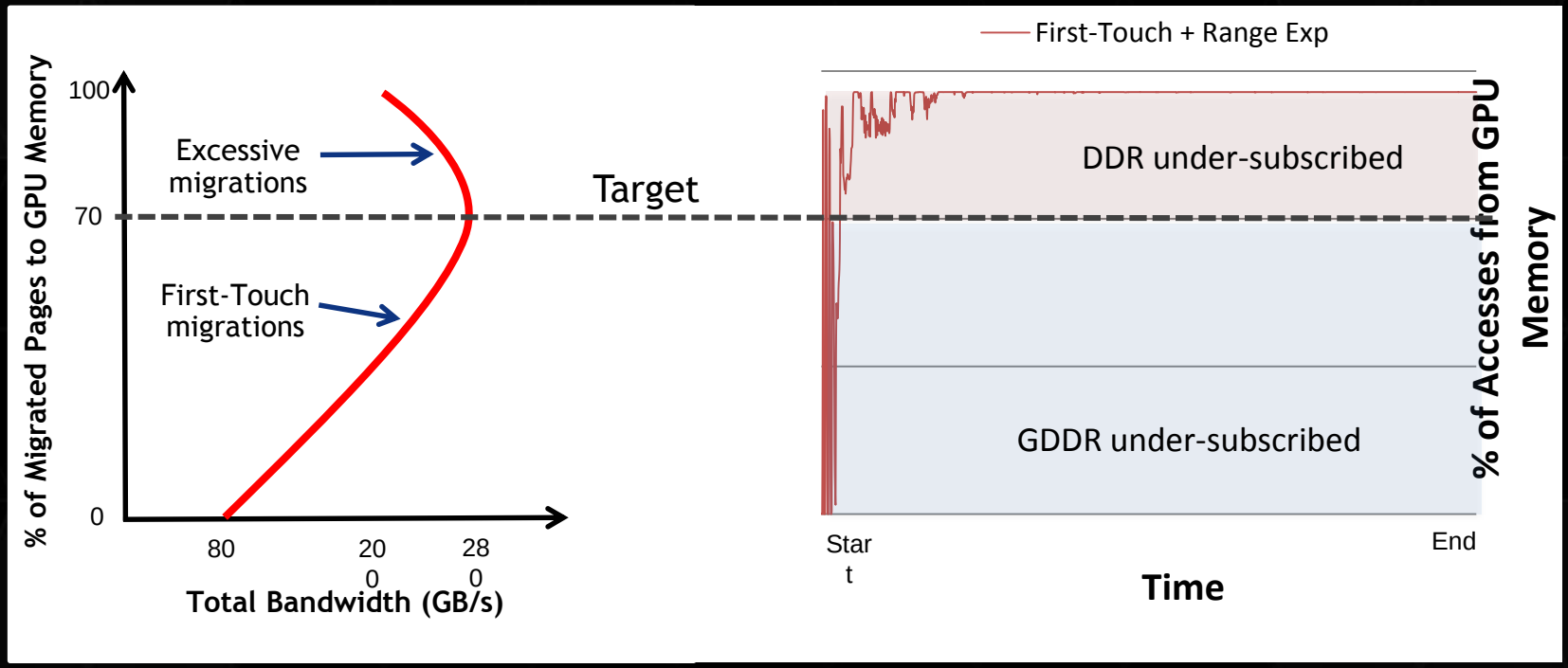


- First-Touch + Range-Expansion aggressively unlocks GDDR BW

How to avoid excessive page migrations?

HPCA-2015

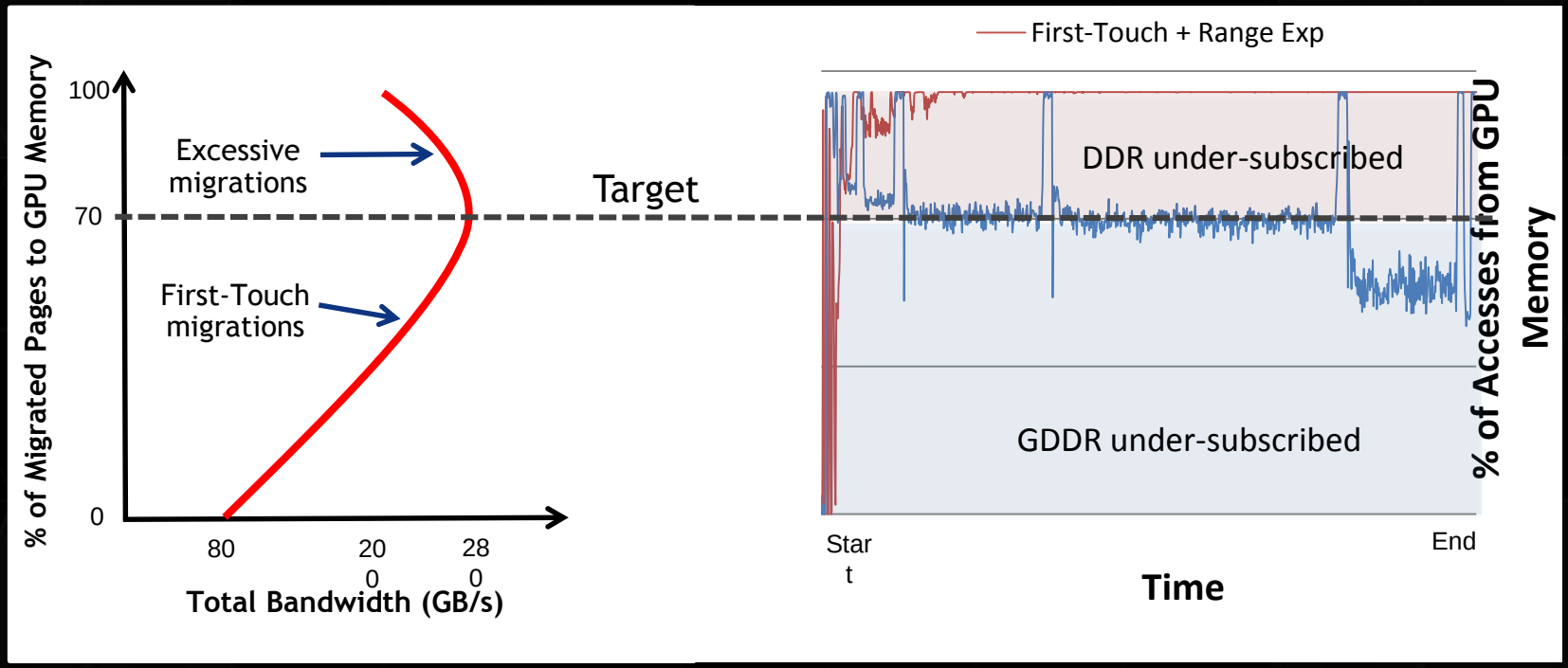
BANDWIDTH BALANCING



Throttle migrations when nearing peak BW

HPCA-2015

BANDWIDTH BALANCING

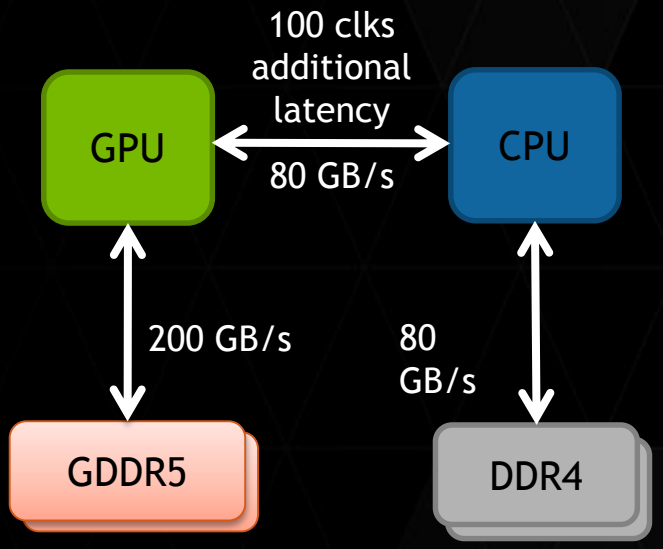


Throttle migrations when nearing peak BW

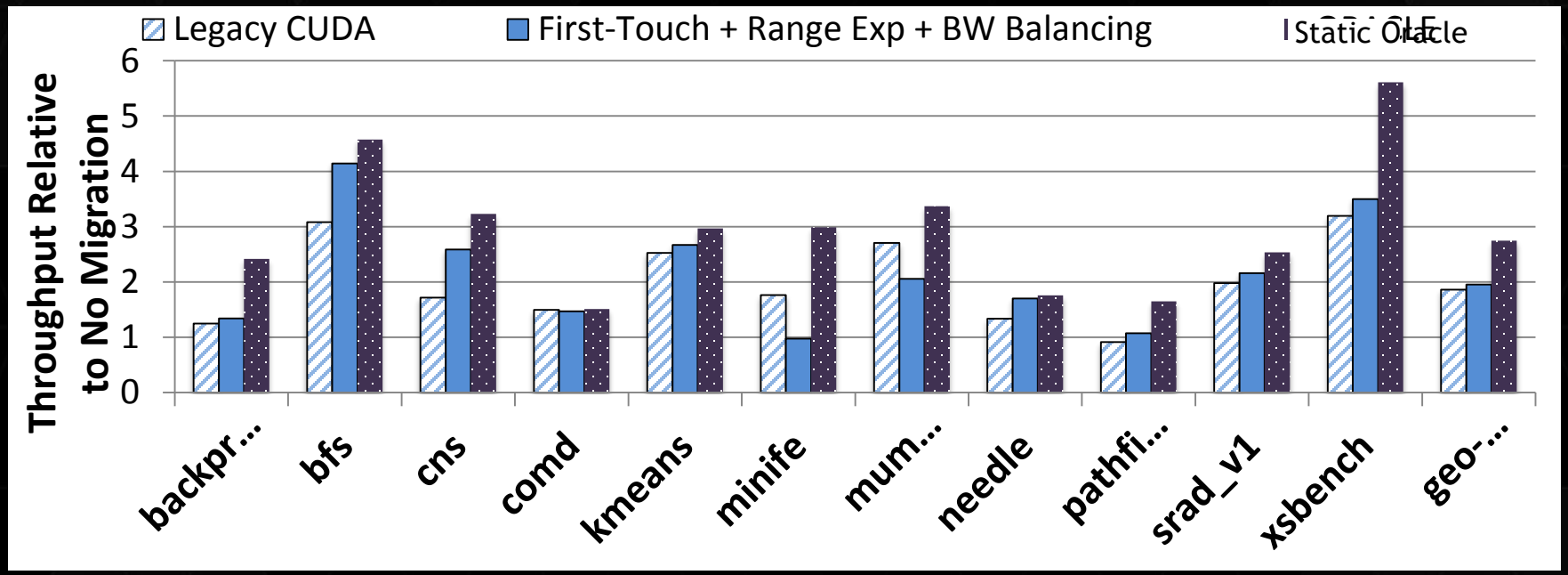
HPCA-2015

SIMULATION ENVIRONMENT

- Simulator: GPGPU-Sim 3.x
- Heterogeneous 2-level memory
  - GDDR5 (200GB/s, 8-channels)
  - DDR4 (80GB/s, 4-channels)
- GPU-CPU interconnect
  - Latency: 100 GPU core cycles
- Workloads:
  - Rodinia applications [Che'IISWC2009]
  - DoE mini apps [Villa'SC2014]

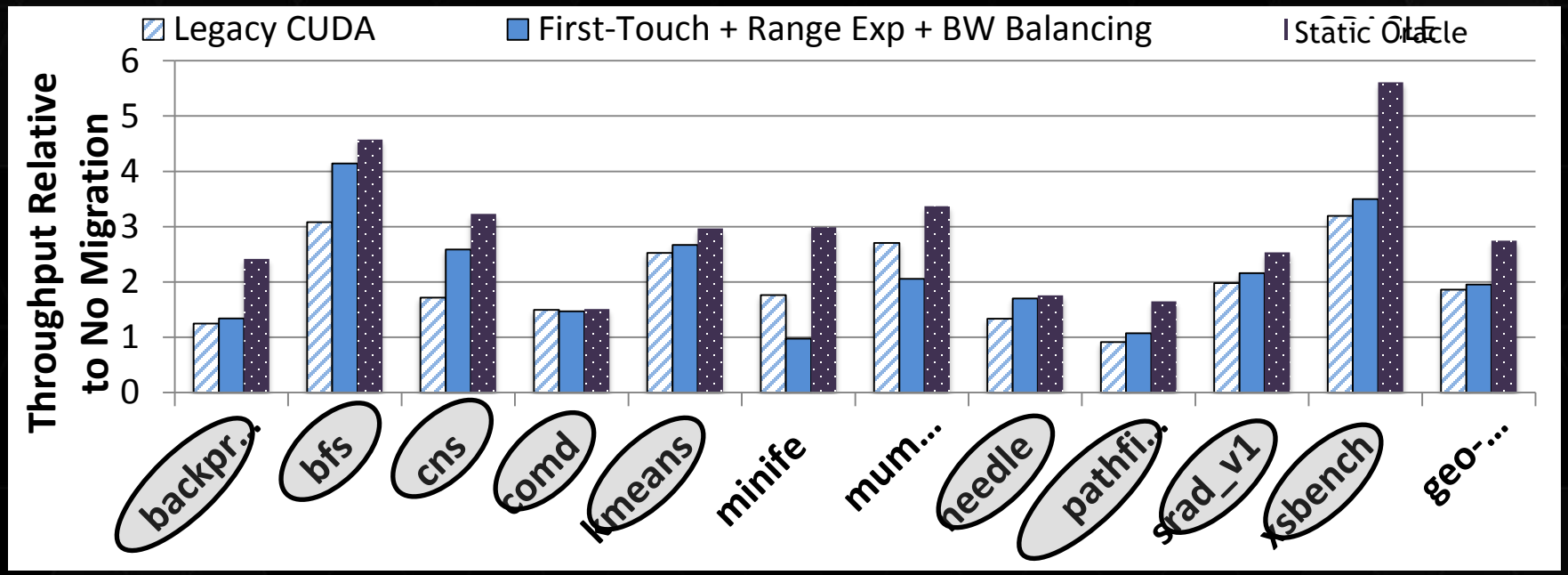


RESULTS



HPCA-2015

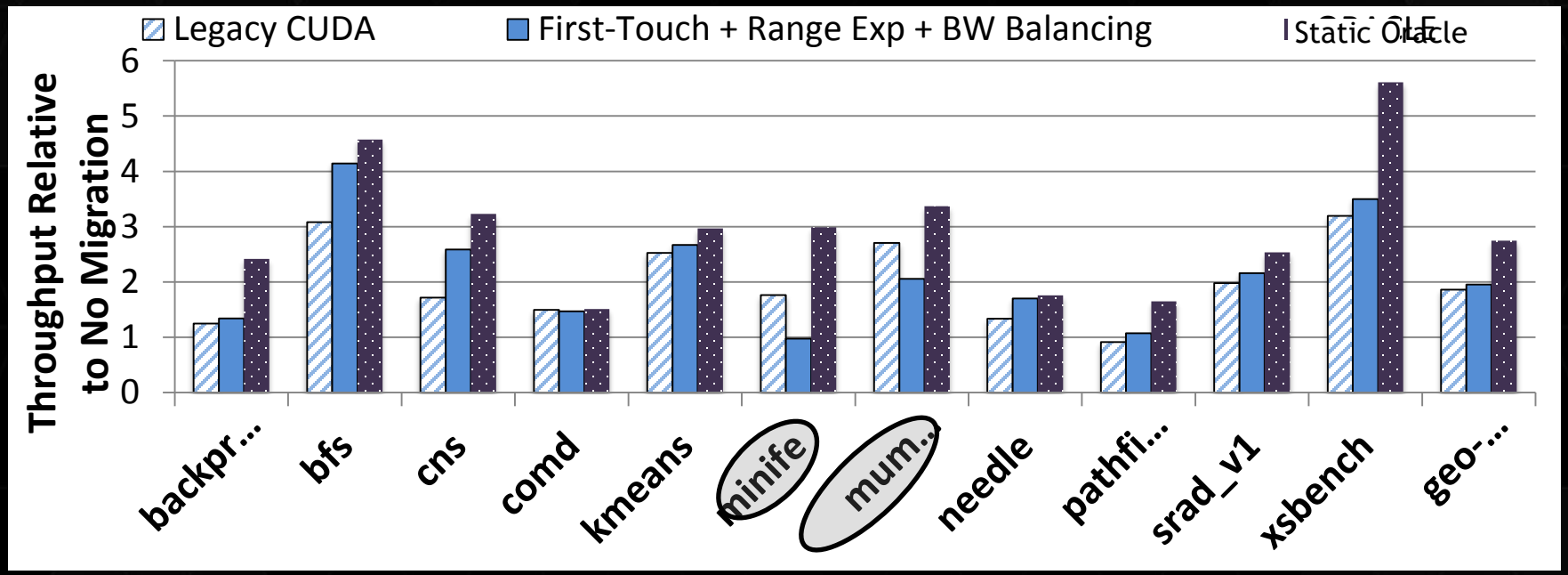
RESULTS



First-Touch + Range-Expansion + BW Balancing outperforms Legacy CUDA

HPCA-2015

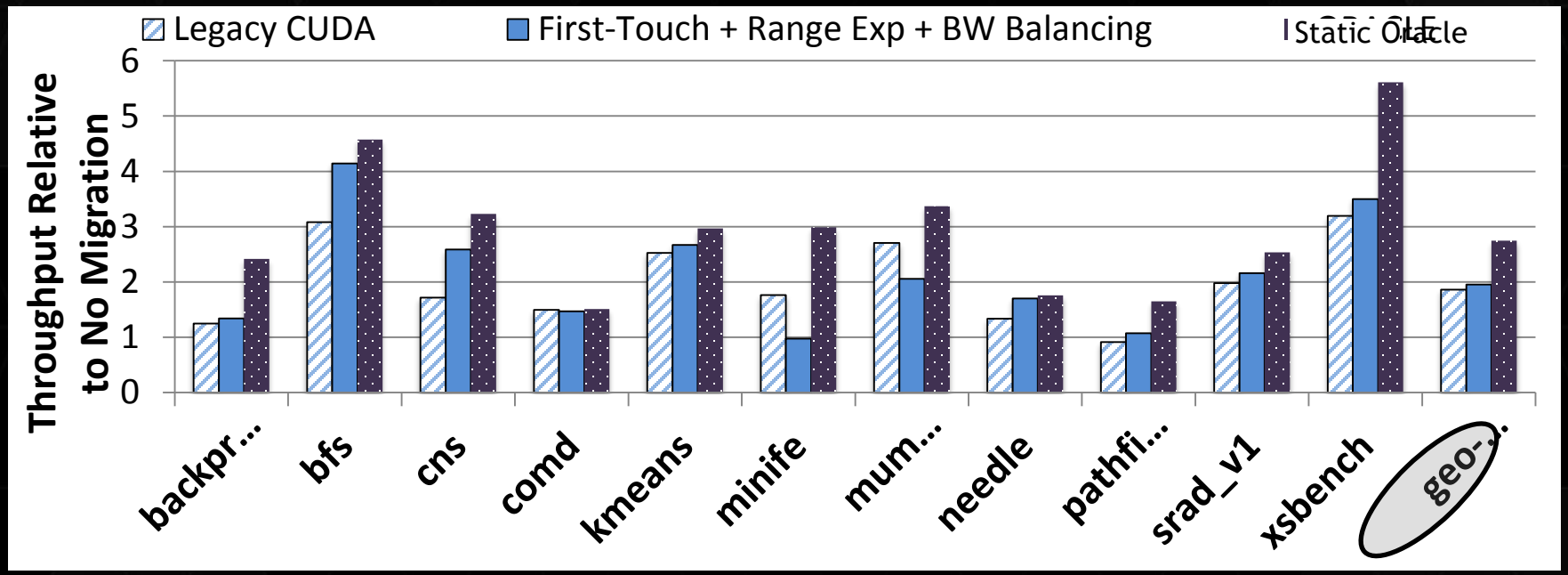
RESULTS



Streaming accesses, no-reuse after migration

HPCA-2015

RESULTS



Dynamic page migration performs 1.95x better than no migration
Comes within 28% of the static oracle performance
6% better than Legacy CUDA

HPCA-2015

CONCLUSIONS

- Developed migration policies without any programmer involvement
- First-Touch migration is cheap but has high TLB shootdowns
- First-Touch + Range-Expansion technique unlocks GDDR memory BW
- BW balancing maximizes BW utilization, throttles excessive migrations

These 3 complementary techniques effectively unlock full system BW

HPCA-2015

THANK YOU

HPCA-2015

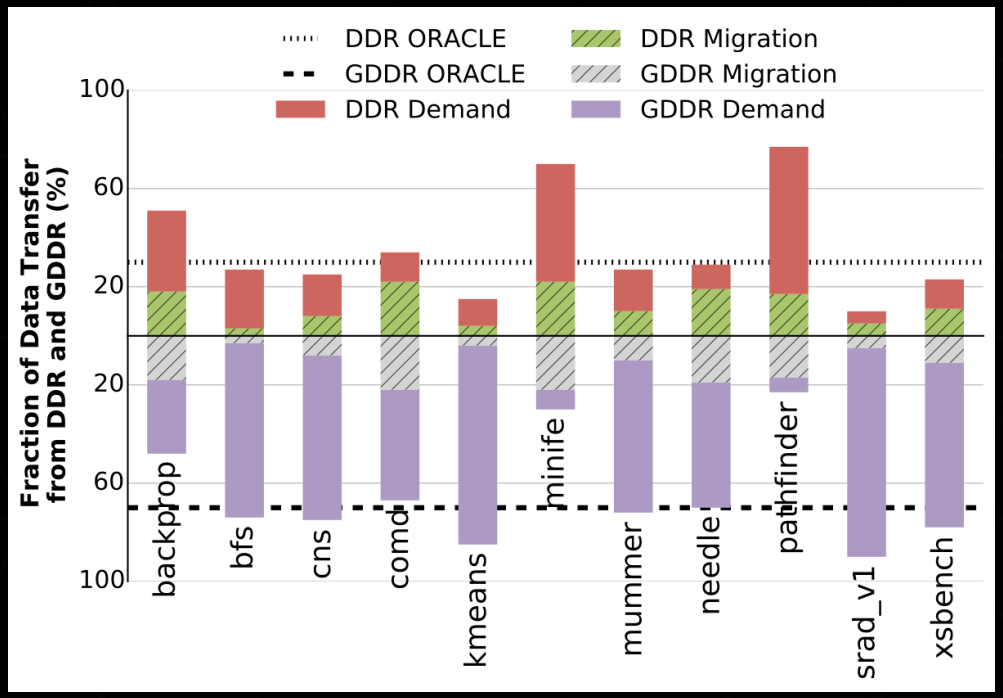
EFFECTIVENESS OF RANGE-EXPANSION

Benchmark	Execution Overhead of TLB Shootdown	% Migrations Without Shootdown	Execution Runtime Saved
Backprop	29.1%	26%	7.6%
Pathfinder	25.9%	10%	2.6%
Needle	24.9%	55%	2.75%
Mummer	21.15%	13%	2.75%
Bfs	6.7%	12%	0.8%

Range-Expansion can save up to 45% TLB shootdowns

HPCA-2015

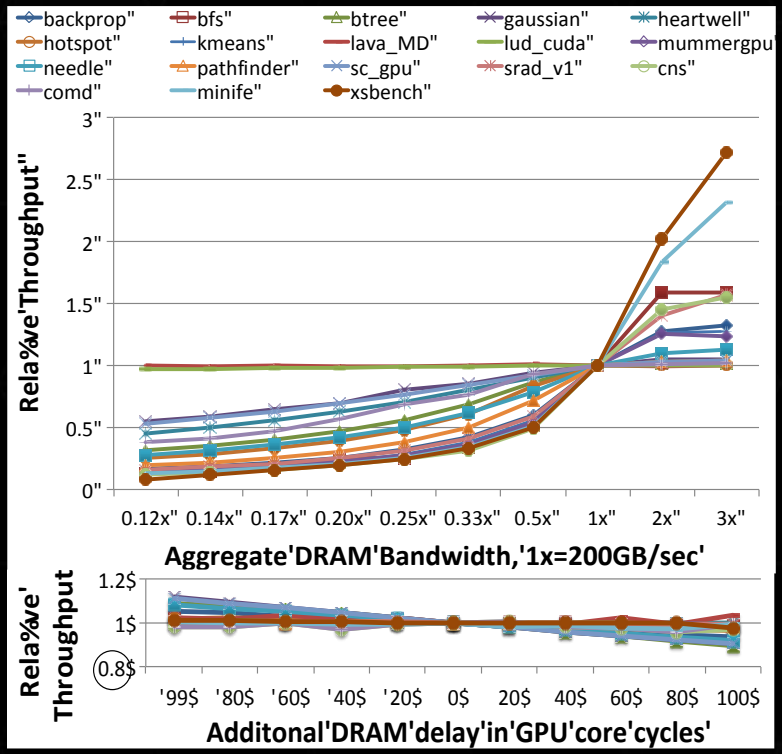
DATA TRANSFER RATIO



Performance is low when GDDR/DDR ratio is away from optimal

HPCA-2015

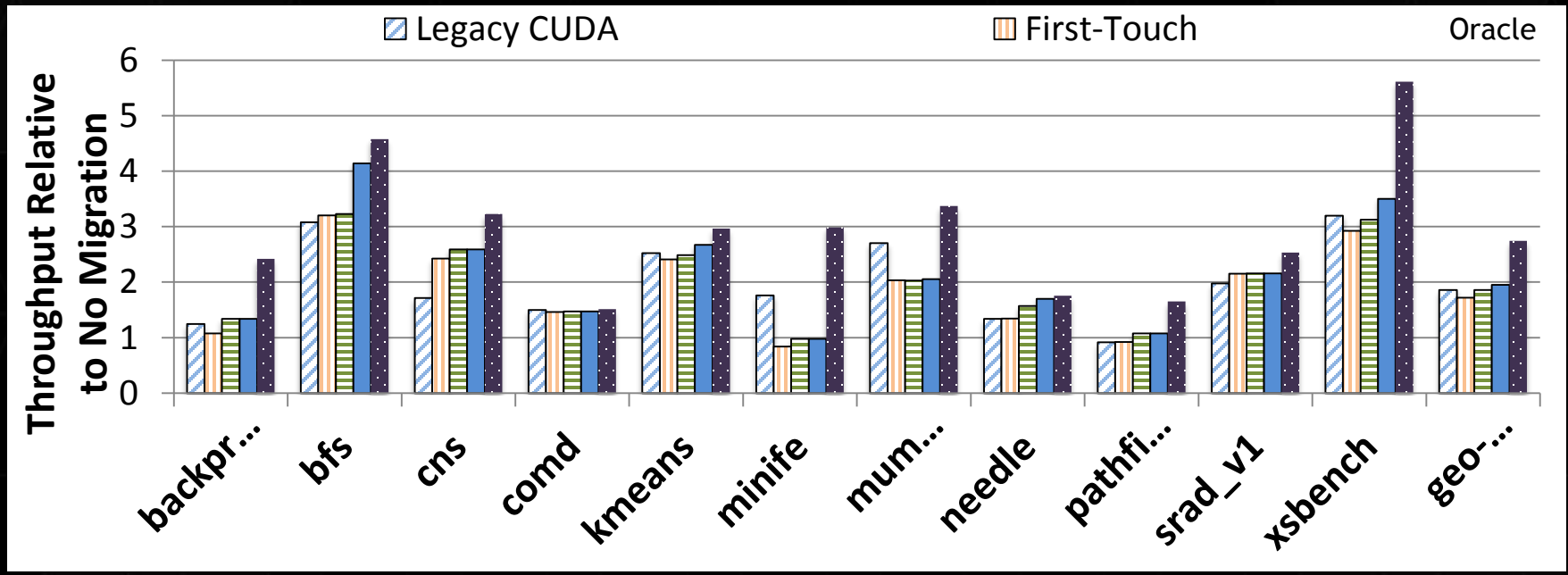
GPU WORKLOADS: BW SENSITIVITY



GPU workloads are highly BW sensitive

HPCA-2015

RESULTS



Dynamic page migration performs 1.95x better than no migration
Comes within 28% of the static oracle performance
6% better than Legacy CUDA

HPCA-2015