MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency

Rachata Ausavarungnirun

Vance Miller Joshua Landgraf Saugata Ghose

Jayneel Gandhi Adwait Jog Christopher J. Rossbach Onur Mutlu

Carnegie Mellon

TEXAS
The University of Texas at Austin

ETH zürich

vmware

SAFARI

Executive Summary

Problem: Address translation overheads
limit the latency hiding capability of a GPU

High contention at the shared TLB

Low L2 cache utilization

Large performance loss vs. no translation

Key Idea

Prioritize **address translation requests** over **data requests**

MASK: a GPU memory hierarchy that

A. Reduces shared TLB contention
B. Improves L2 cache utilization
C. Reduces page walk latency

MASK **improves system throughput by** **57.8%** on average over state-of-the-art address translation mechanisms

Outline

- Executive Summary
- **Background, Key Challenges and Our Goal**
- **MASK:** A Translation-aware Memory Hierarchy
- Evaluation
- Conclusion

Why Share Discrete GPUs?

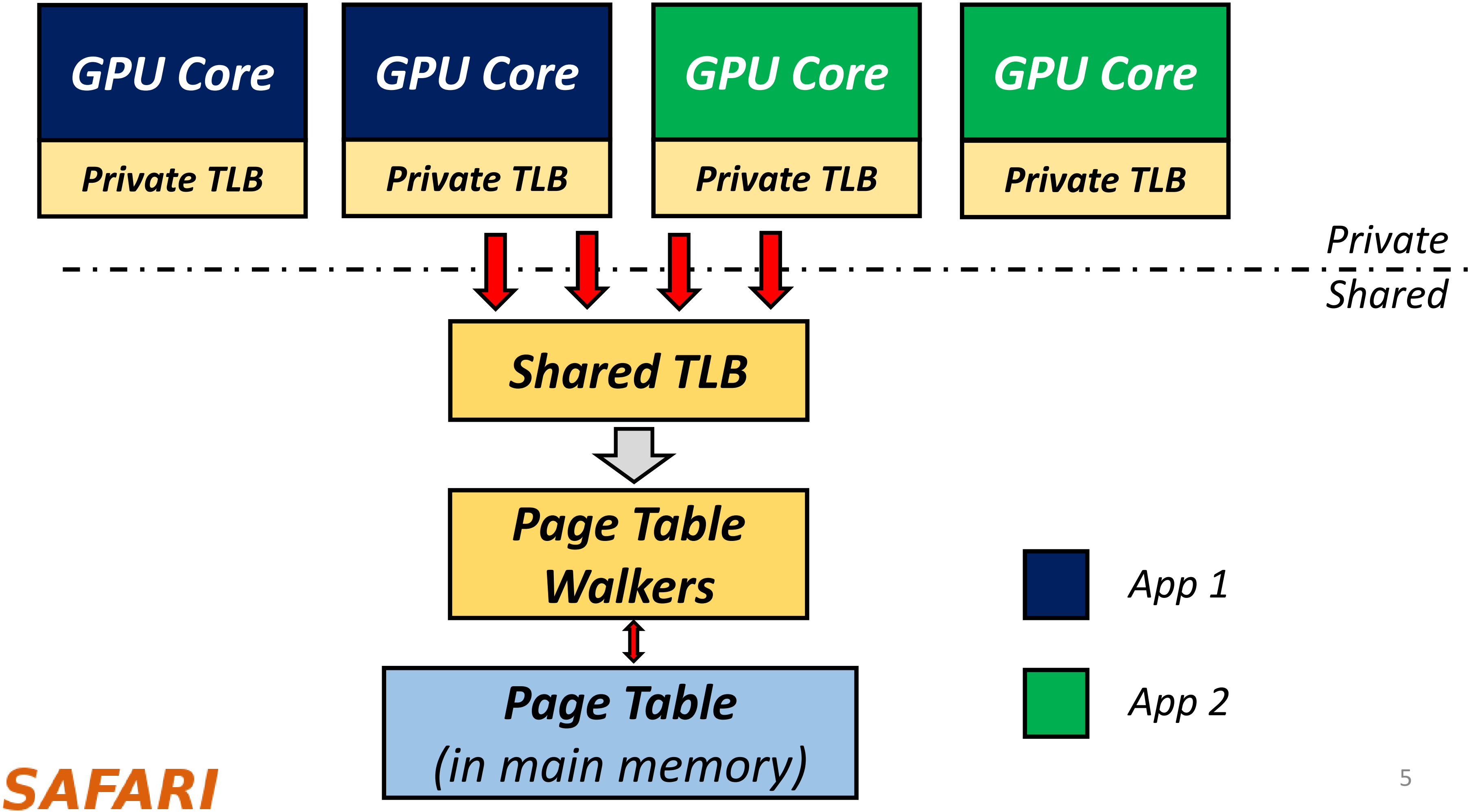
- Enables multiple GPGPU applications to run concurrently

- Better resource utilization
  - An application often cannot utilize an entire GPU
  - Different compute and bandwidth demands

- Enables GPU sharing in the cloud
  - Multiple users spatially share each GPU

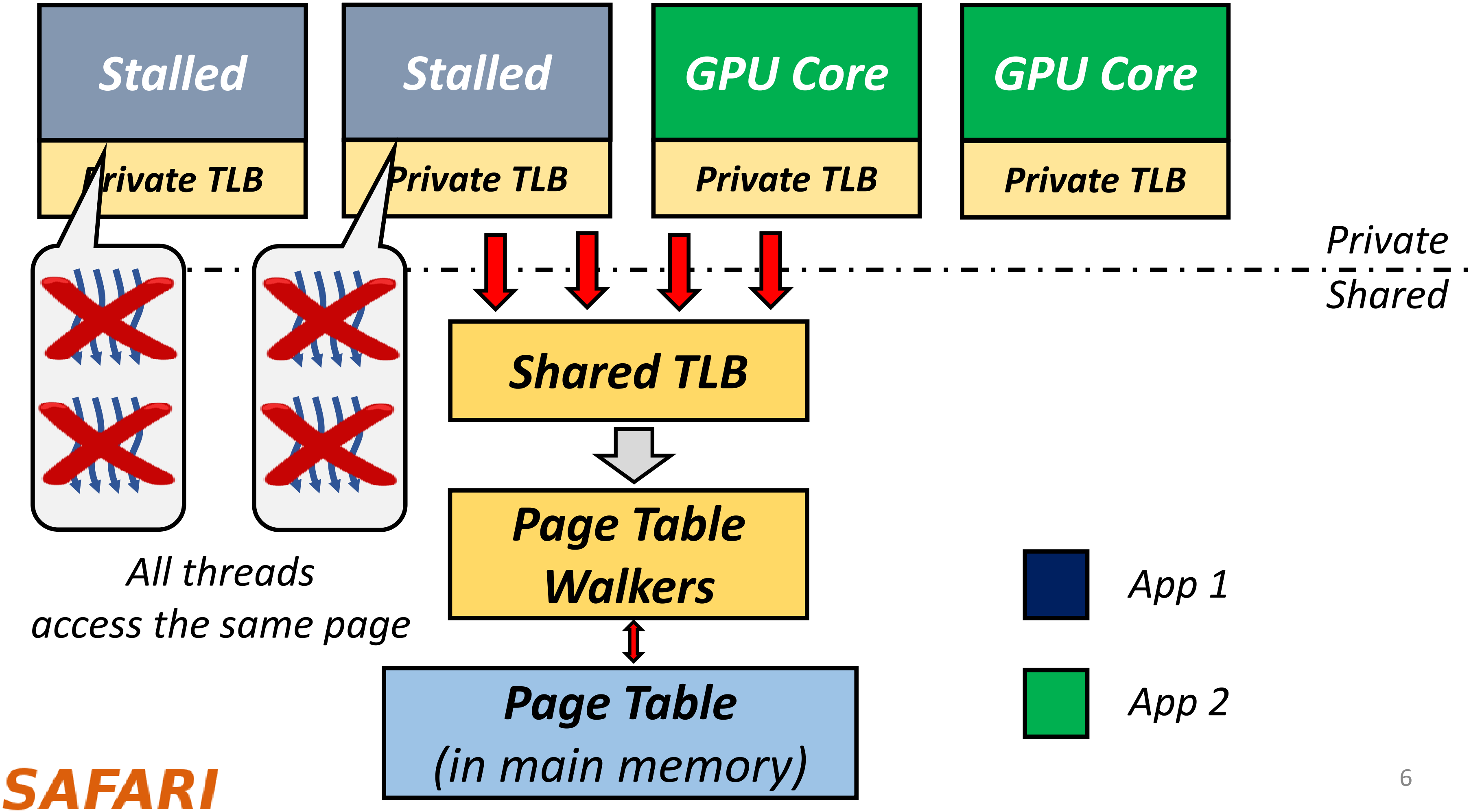
Key requirement: fine-grained memory protection

State-of-the-art Address Translation in GPUs



A TLB Miss Stalls Multiple Warps

Data in a page is shared by many threads



Stalled

Stalled

GPU Core

GPU Core

Private TLB

Private TLB

Private TLB

Private TLB

Private

Shared

Shared TLB

Page Table Walkers

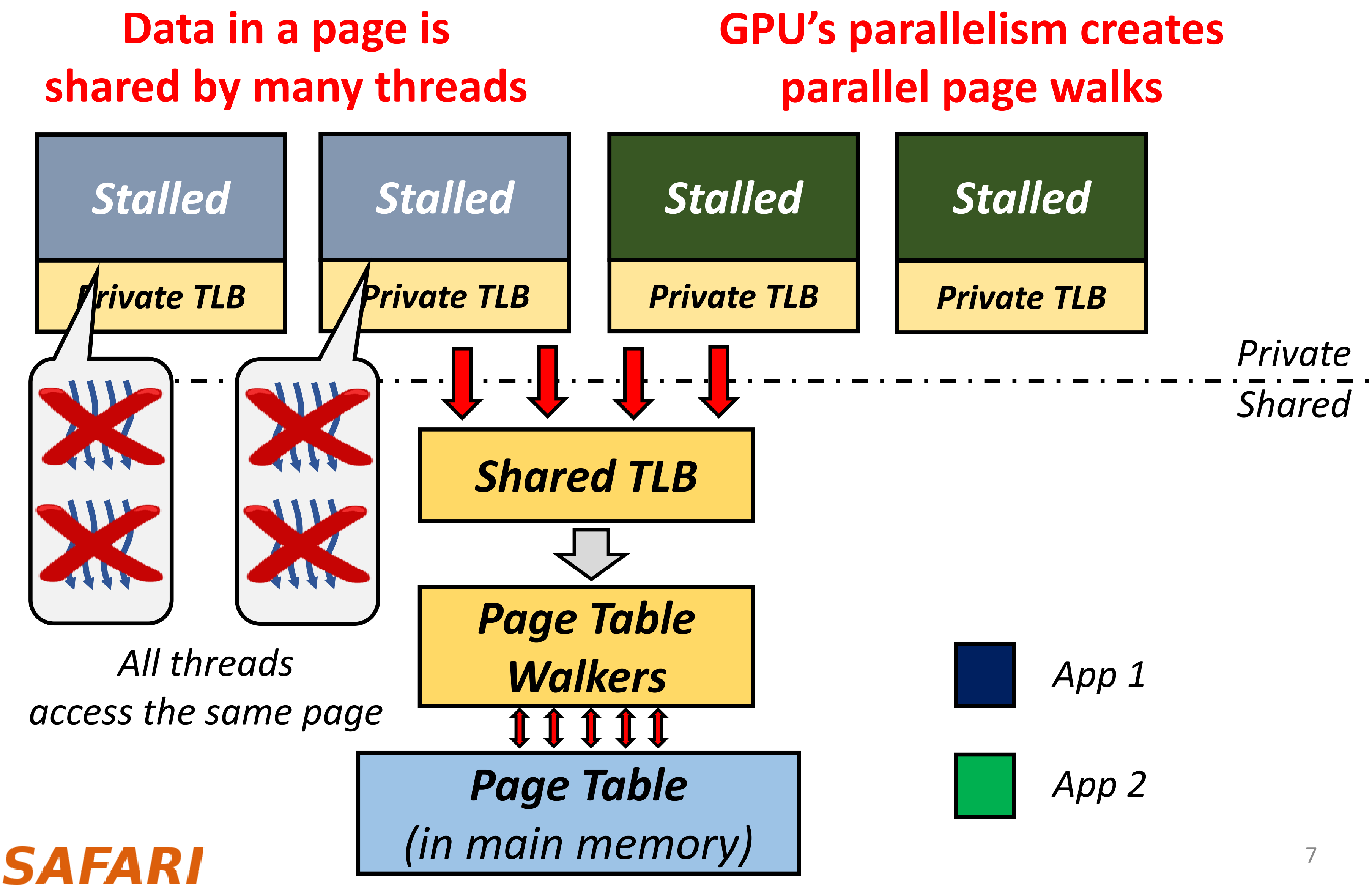
Page Table (in main memory)

All threads access the same page

🟦 App 1

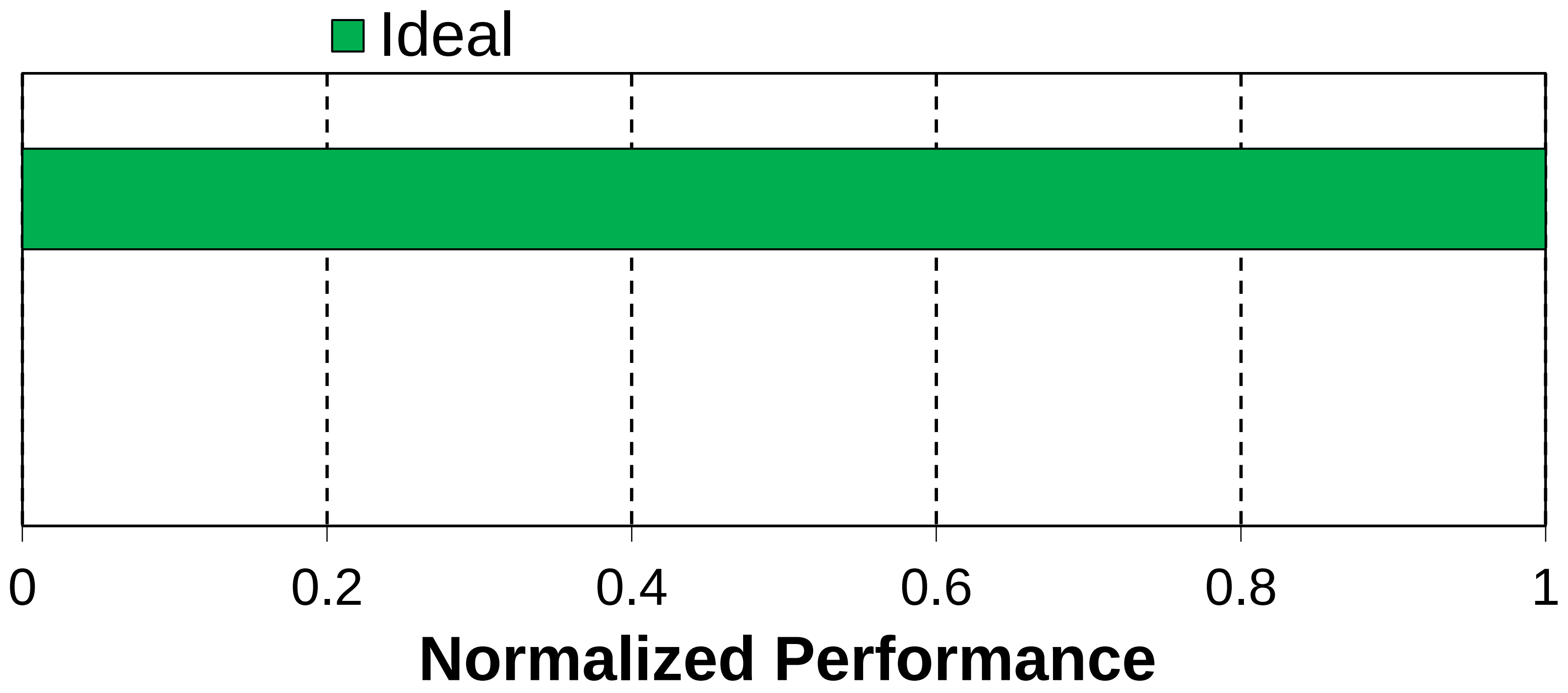
🟩 App 2

Multiple Page Walks Happen Together


Data in a page is shared by many threads
GPU's parallelism creates parallel page walks
Stalled
Stalled
Stalled
Stalled
Private TLB
Private TLB
Private TLB
Private TLB
Private
Shared
Shared TLB
Page Table Walkers
Page Table (in main memory)
All threads access the same page
App 1
App 2

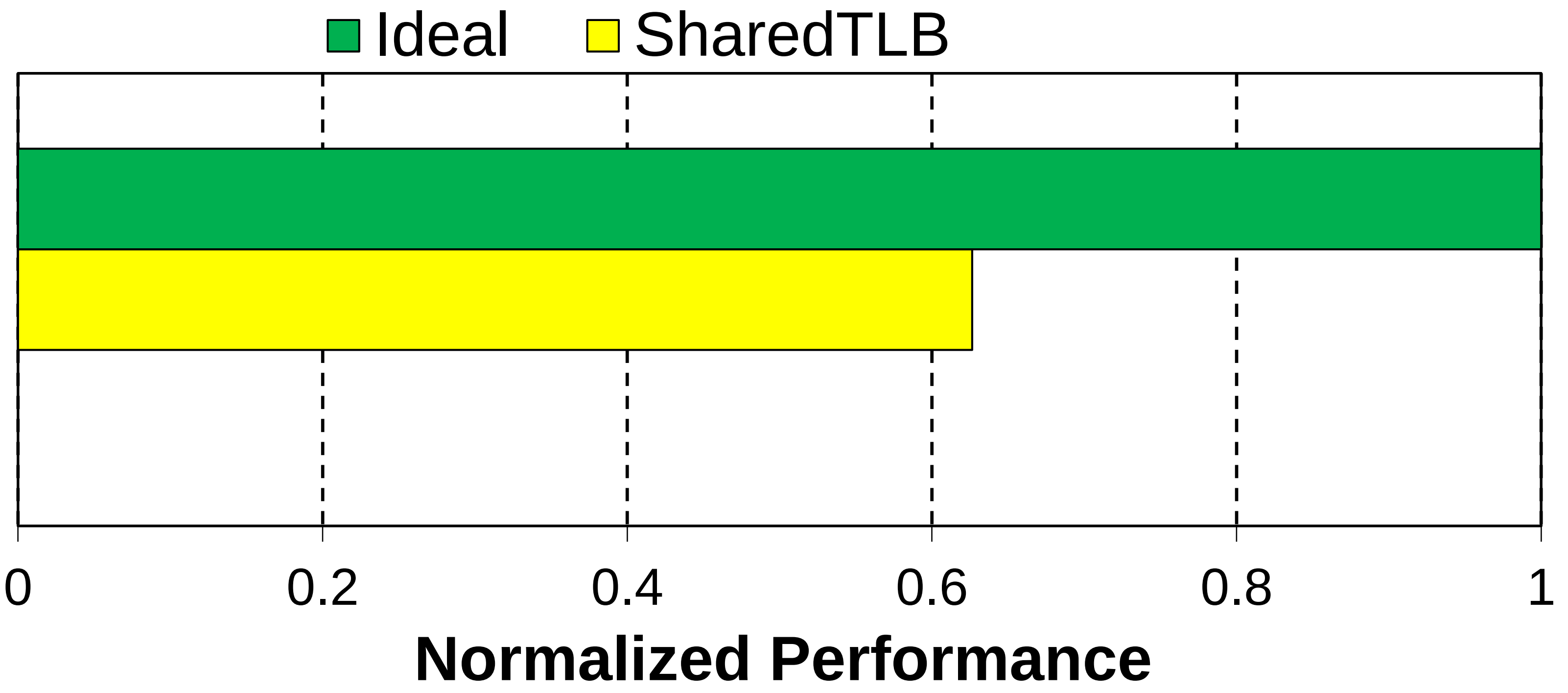
SAFARI

Effect of Translation on Performance

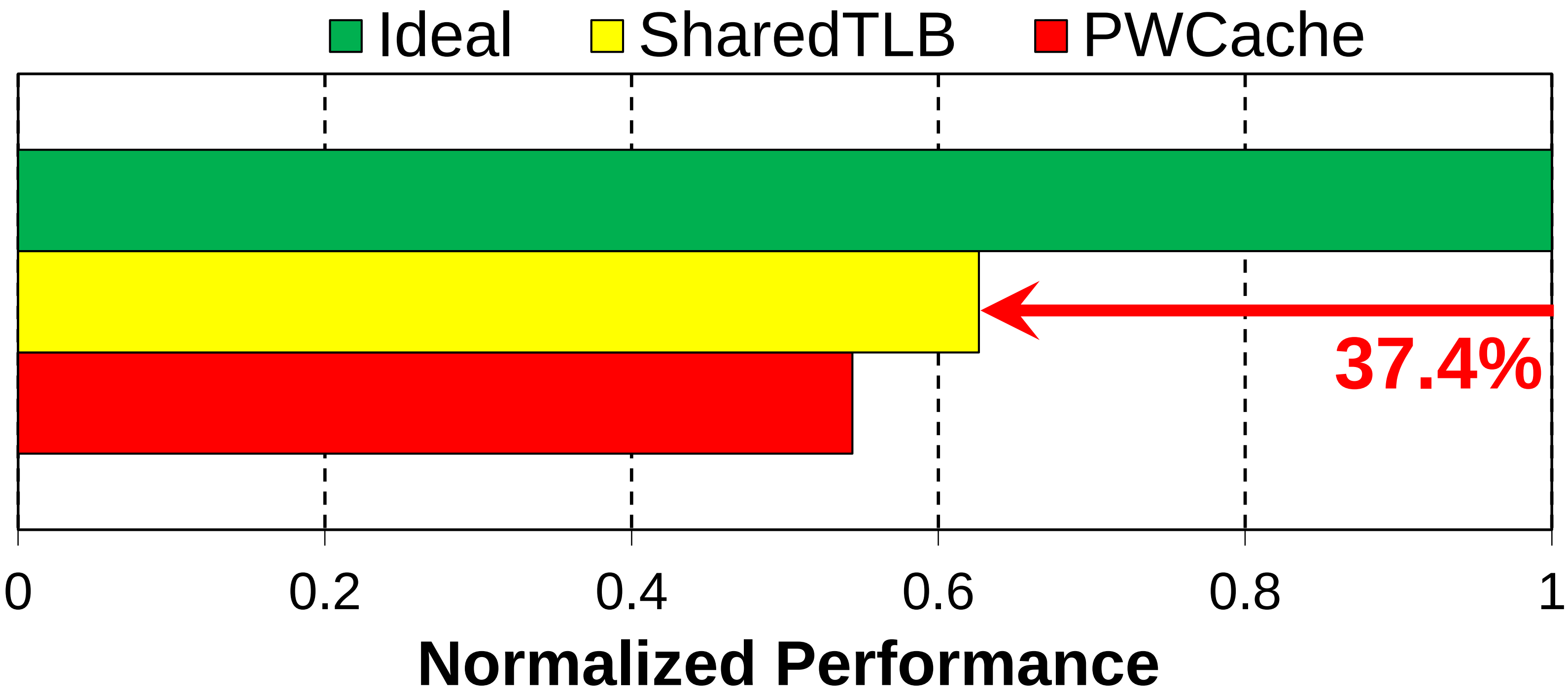


SAFARI

Effect of Translation on Performance



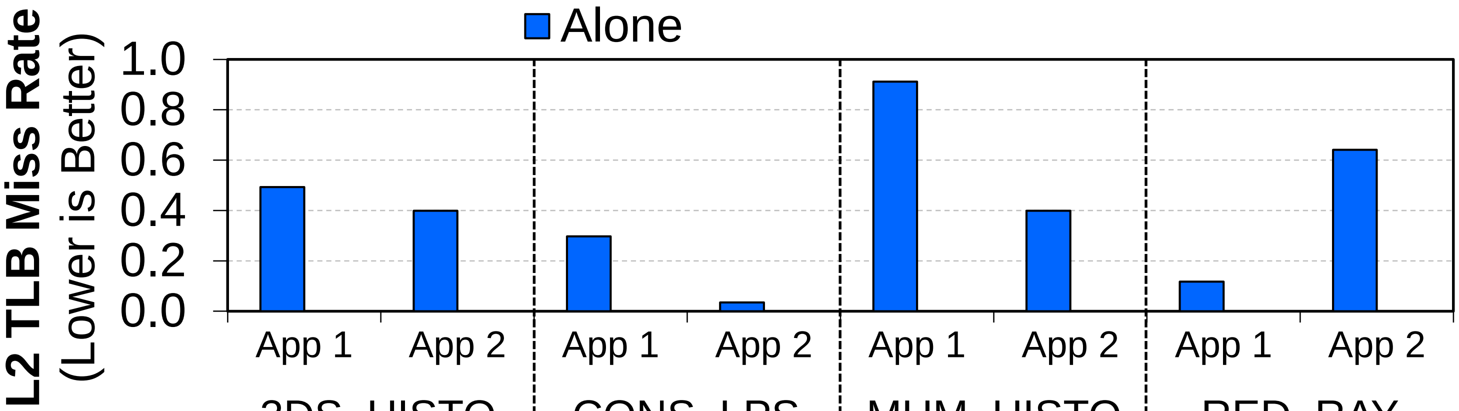
Effect of Translation on Performance



What causes the large performance loss?

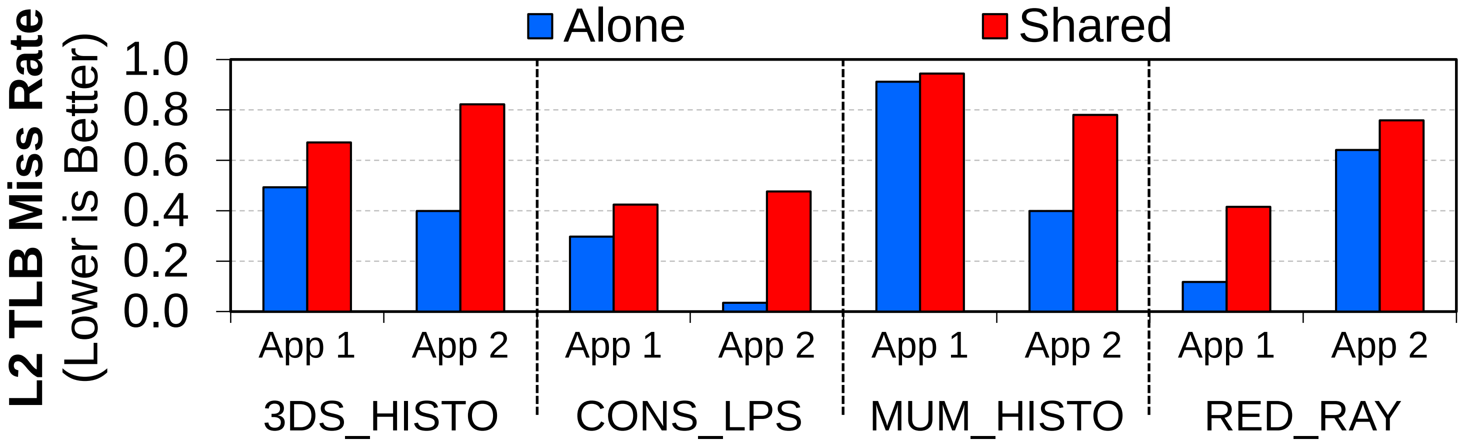
Problem 1: Contention at the Shared TLB

- Multiple GPU applications contend for the TLB



Problem 1: Contention at the Shared TLB

- Multiple GPU applications contend for the TLB



Contention at the shared TLB leads to lower performance

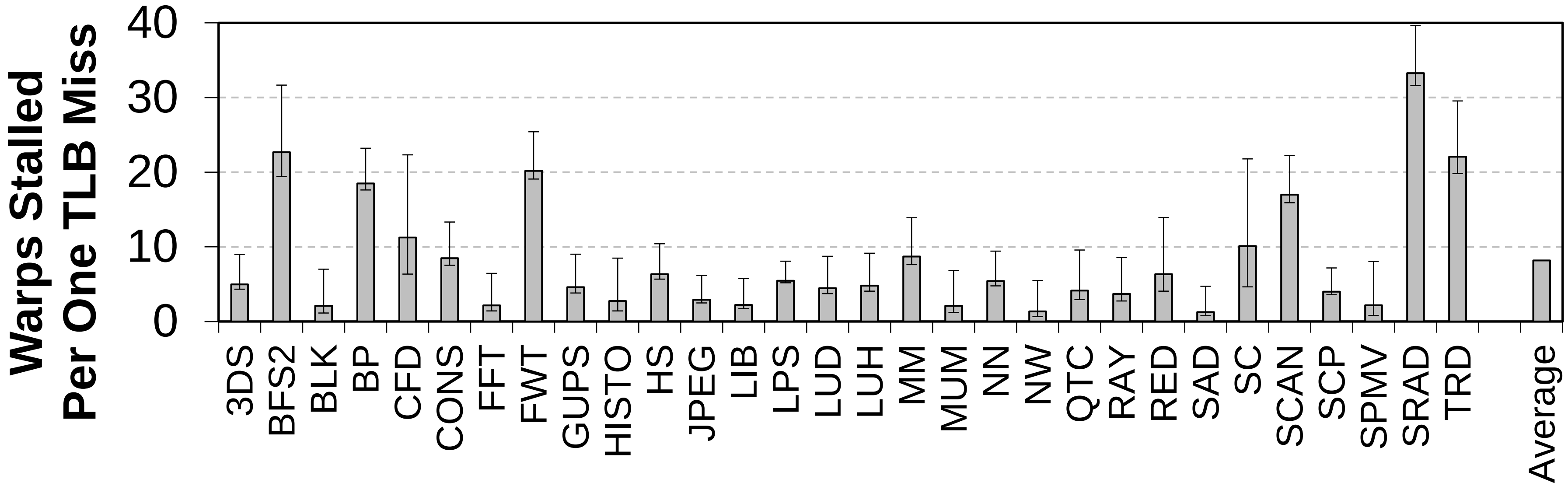
Problem 2: Thrashing at the L2 Cache

- L2 cache can be used to reduce page walk latency
  - → **Partial translation data can be cached**
- **Thrashing Source 1:** Parallel page walks
  - → **Different address translation data evicts each other**
- **Thrashing Source 2:** GPU memory intensity
  - → **Demand-fetched data evicts address translation data**

L2 cache is **ineffective** at reducing page walk latency

Observation: Address Translation Is Latency Sensitive

- Multiple warps share data from a single page



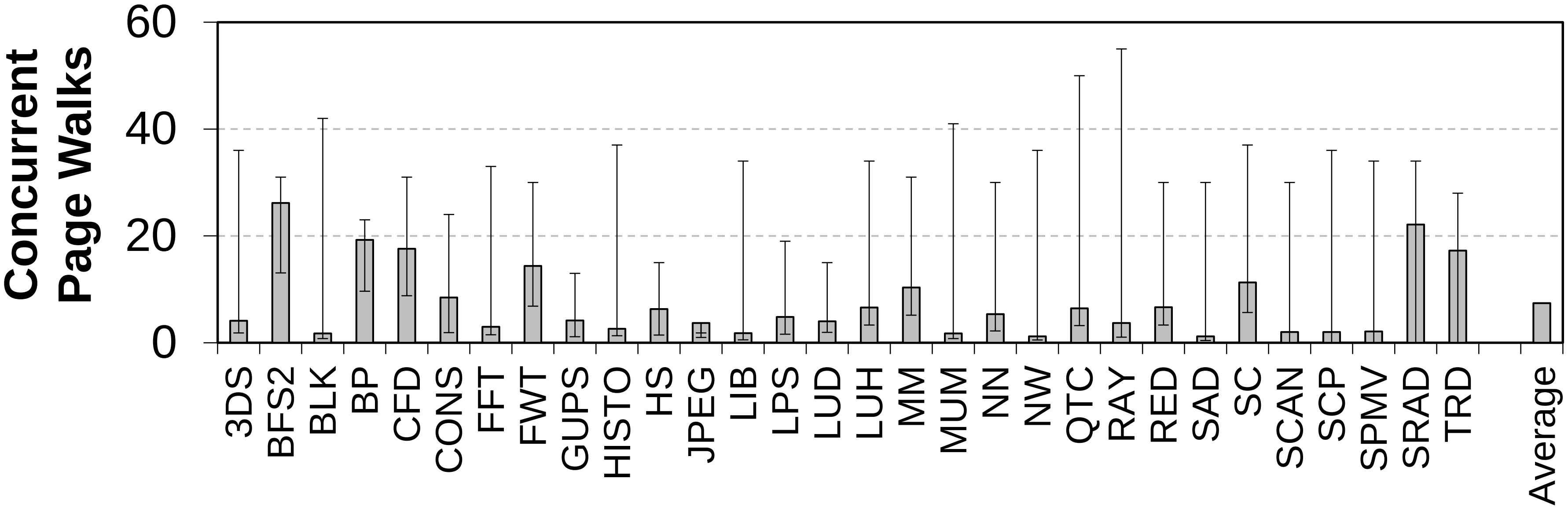
A single TLB miss causes **8 warps** to stall on average

SAFARI

Observation: Address Translation Is Latency Sensitive

- Multiple warps share data from a single page

- GPU's parallelism causes multiple concurrent page walks



High address translation latency → **More stalled warps**

SAFARI

Our Goals

- **Reduce shared TLB contention**

- **Improve L2 cache utilization**

- **Lower page walk latency**

Outline

- Executive Summary
- Background, Key Challenges and Our Goal
- **MASK:** A Translation-aware Memory Hierarchy
- Evaluation
- Conclusion

MASK: A Translation-aware Memory Hierarchy

- **Reduce shared TLB contention**
  - **A. TLB-fill Tokens**
- **Improve L2 cache utilization**
  - **B. Translation-aware L2 Bypass**
- **Lower page walk latency**
  - **C. Address-space-aware Memory Scheduler**

A: TLB-fill Tokens

- **Goal:** Limit the number of warps that can fill the TLB
  - → A warp with **a token** fills the **shared TLB**
  - → A warp with **no token** fills a very small **bypass cache**

- Number of tokens changes based on TLB miss rate
  - → Updated every epoch

- Tokens are assigned based on warp ID

Benefit: Limits contention at the shared TLB

MASK: A Translation-aware Memory Hierarchy

- **Reduce shared TLB contention**

  A. TLB-fill Tokens

- **Improve L2 cache utilization**

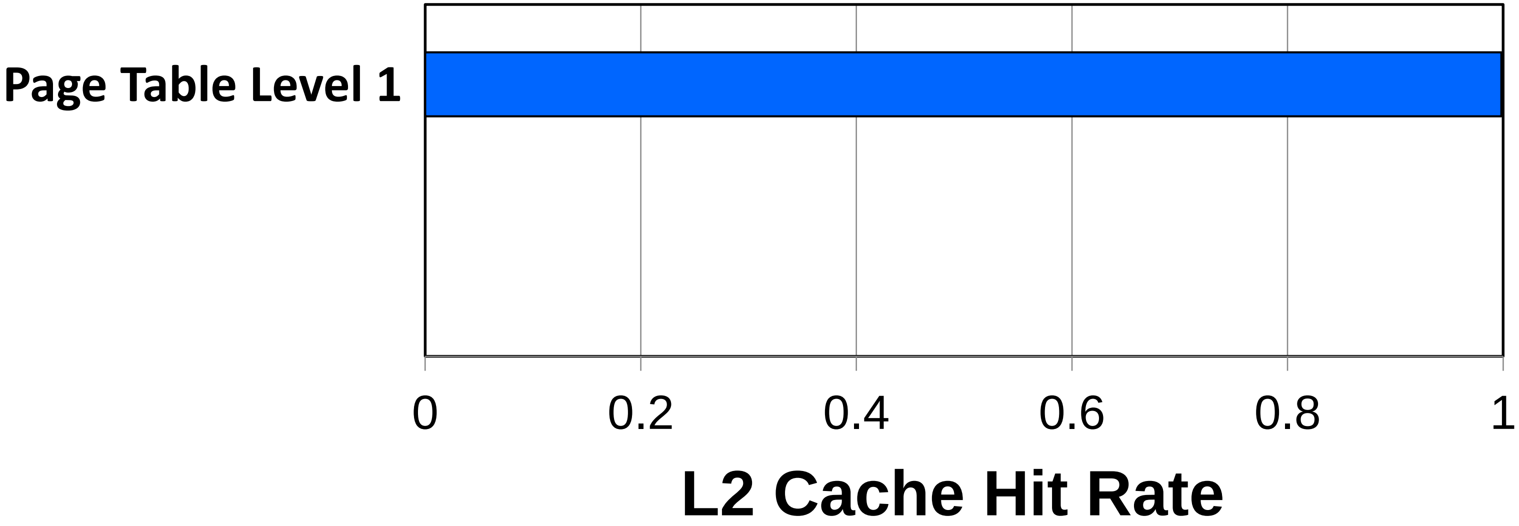
  B. Translation-aware L2 Bypass

- **Lower page walk latency**

  C. Address-space-aware Memory Scheduler

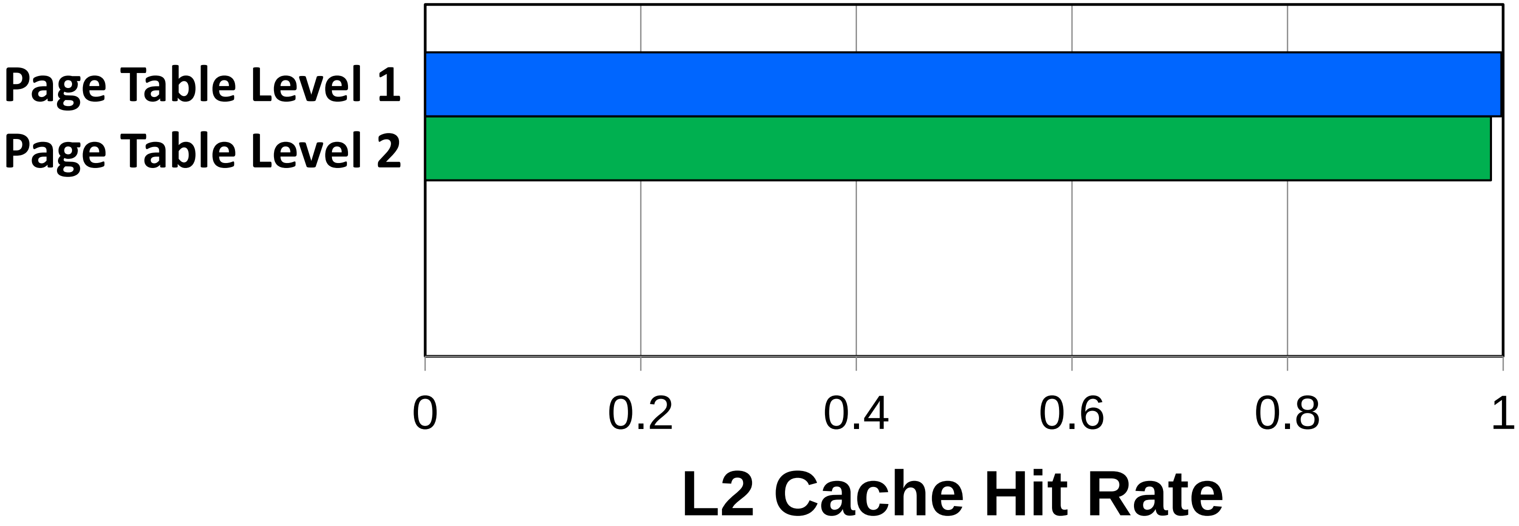
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



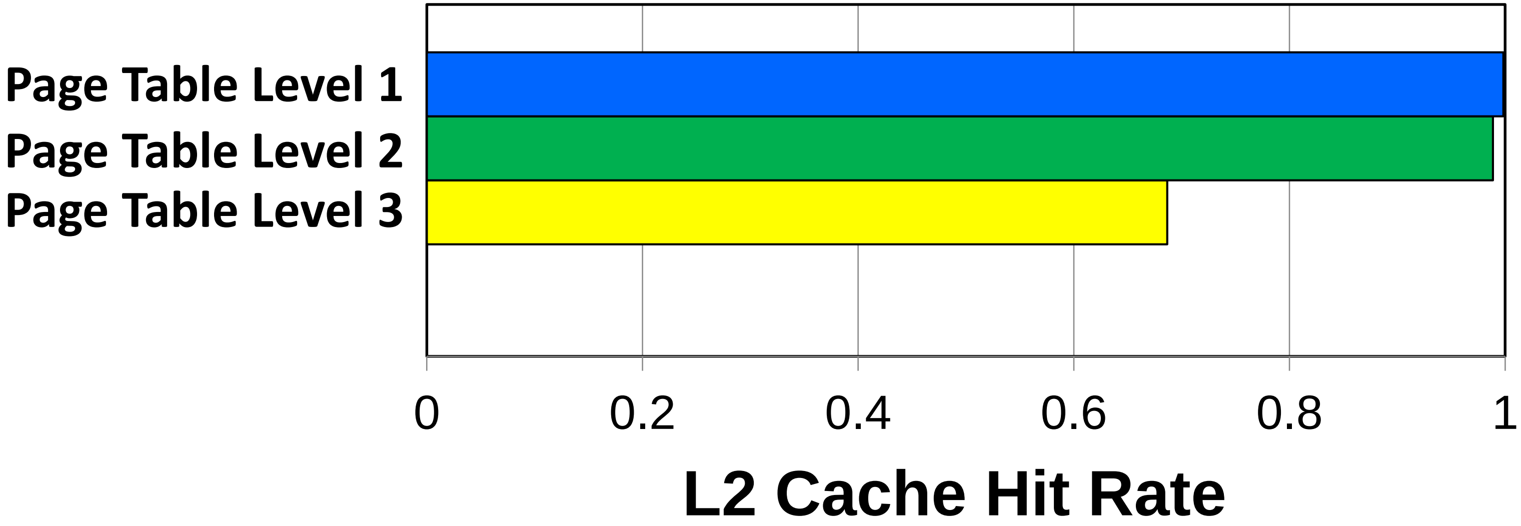
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



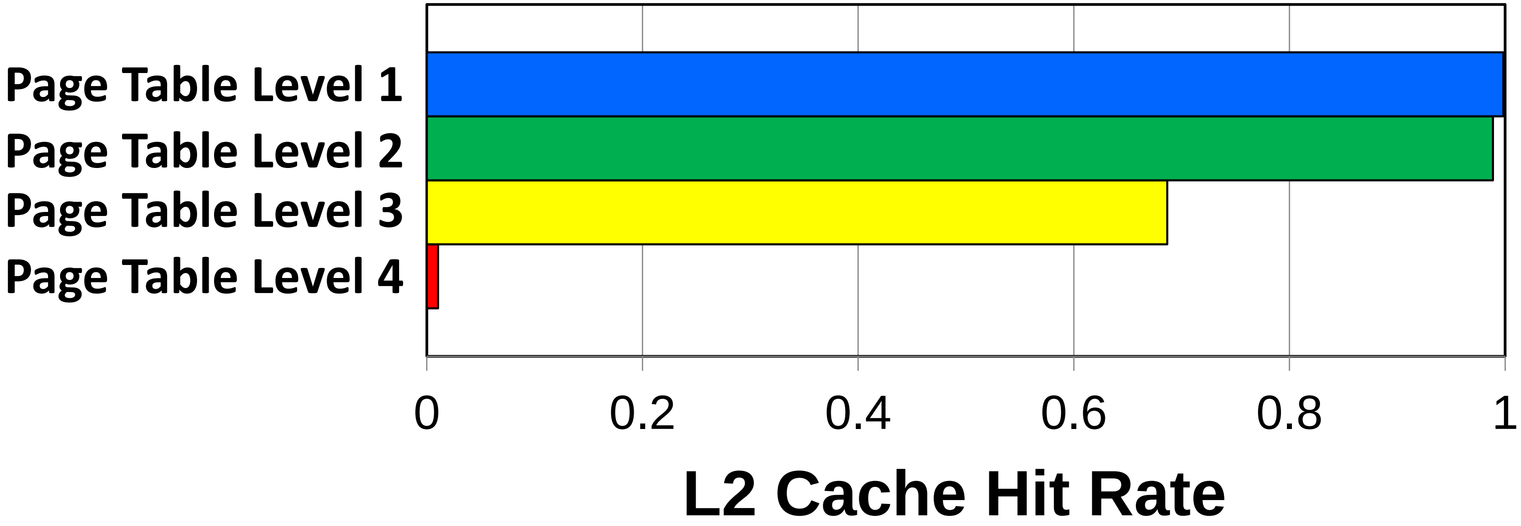
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



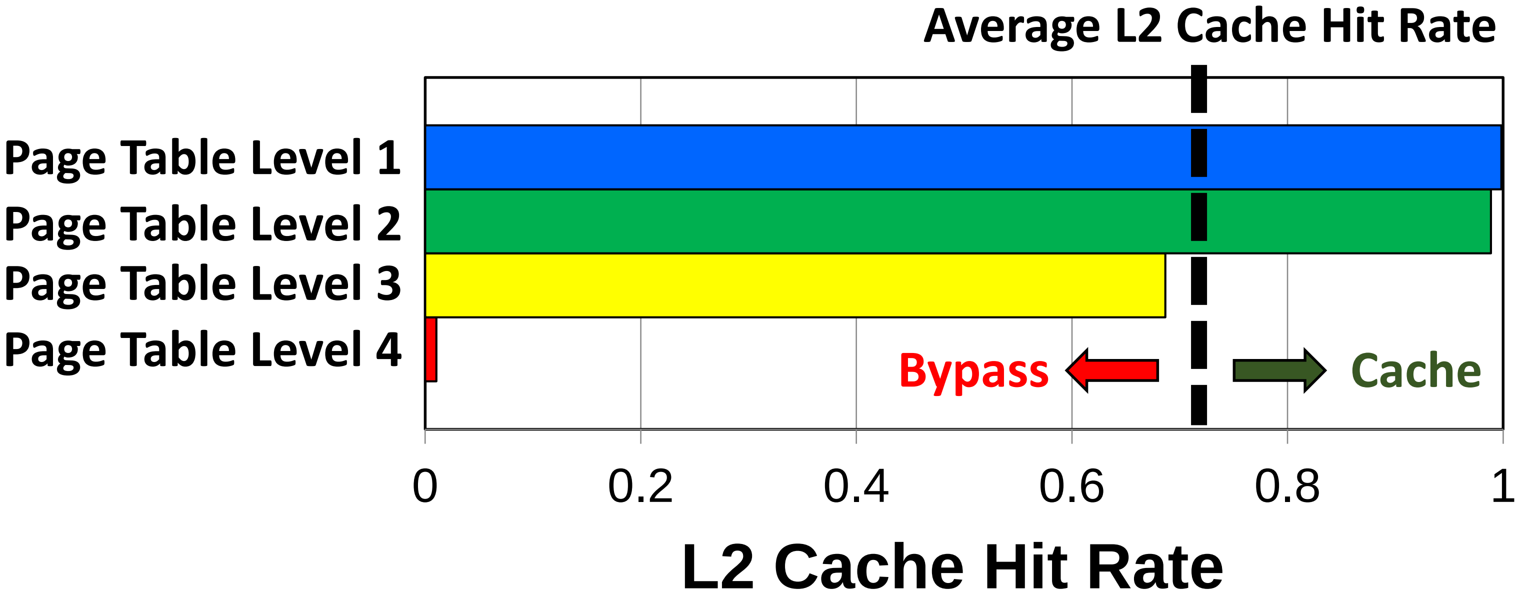
Some address translation data **does not benefit from caching**

Only cache address translation data with high hit rate

SAFARI

B: Translation-aware L2 Bypass

- **Goal:** Cache address translation data with high hit rate



Benefit 1: **Better L2 cache utilization** for translation data

Benefit 2: Bypassed requests → **No L2 queuing delay**

MASK: A Translation-aware Memory Hierarchy

- Reduce shared TLB contention

  A. TLB-fill Tokens

- Improve L2 cache utilization

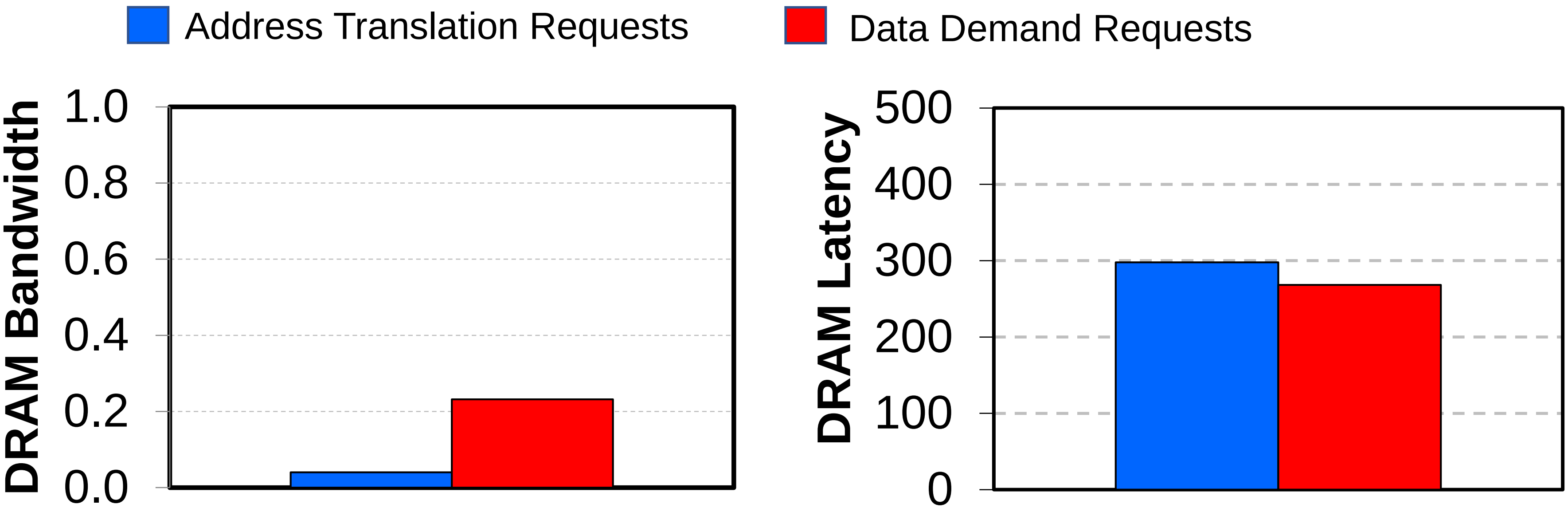
  B. Translation-aware L2 Bypass

- Lower page walk latency

  C. Address-space-aware Memory Scheduler

C: Address-space-aware Memory Scheduler

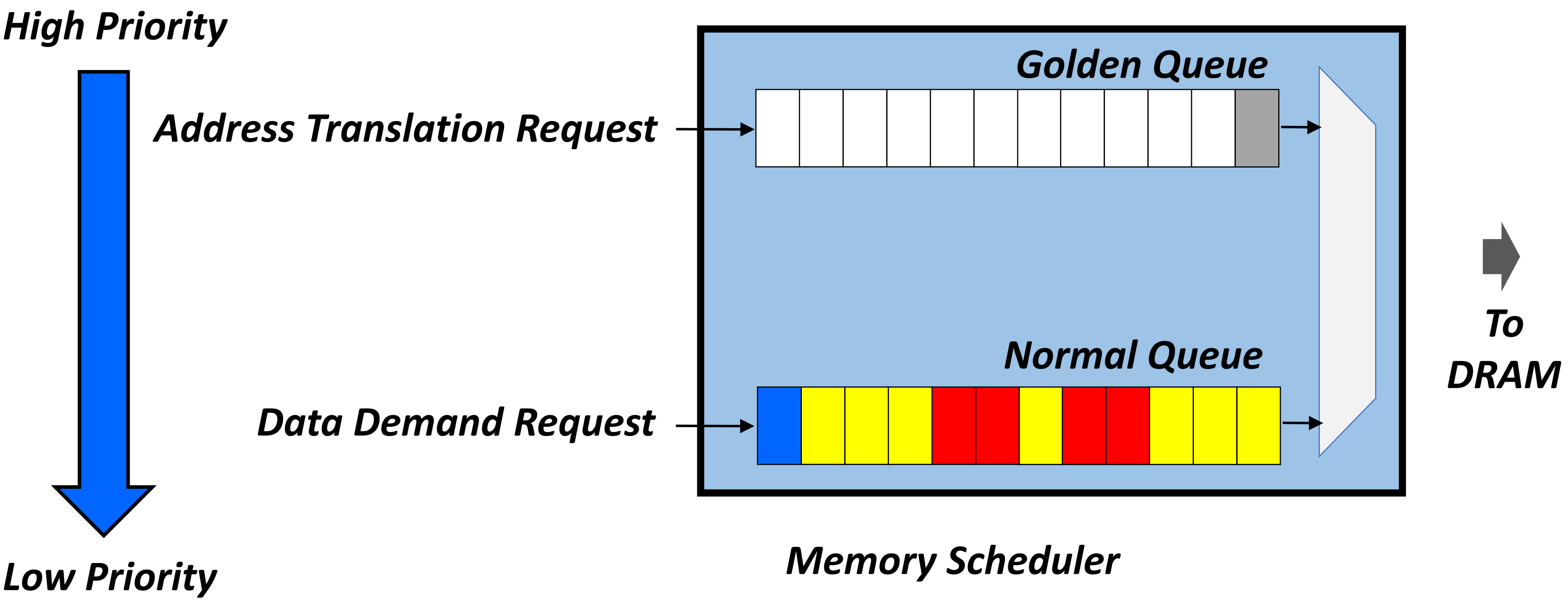
- **Cause:** Address translation requests **are treated similarly** to data demand requests



Idea: **Lower address translation request** latency

C: Address-space-aware Memory Scheduler

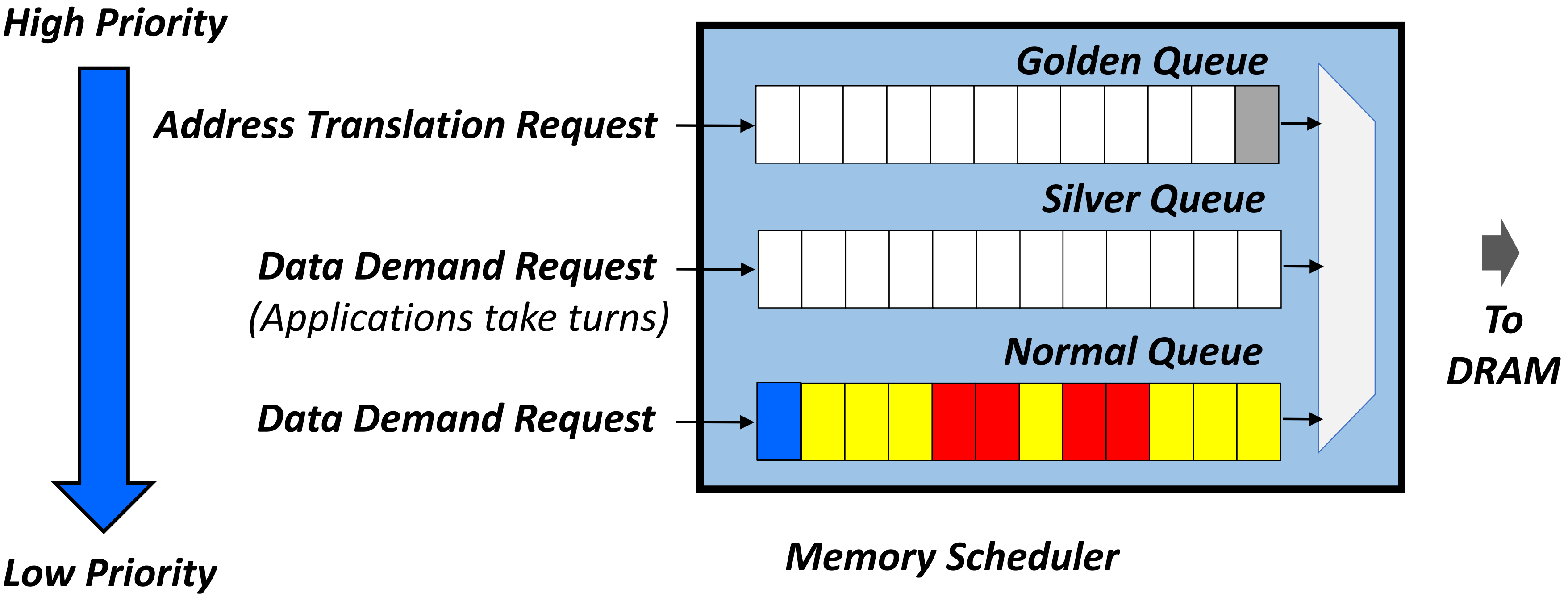
- **Idea 1: Prioritize address translation requests** over data demand requests



SAFARI

C: Address-space-aware Memory Scheduler

- **Idea 1: Prioritize address translation requests** over data demand requests
- **Idea 2: Improve quality-of-service** using the Silver Queue



Each application takes turn injecting into the Silver Queue

SAFARI

Outline

- **Executive summary**
- **Background, Key Challenges and Our Goal**
- **MASK:** A Translation-aware Memory Hierarchy
- **Evaluation**
- **Conclusion**

Methodology

- Mosaic simulation platform [MICRO '17]
  - Based on GPGPU-Sim and MAFIA [Jog et al., MEMSYS '15]
  - Models page walks and virtual-to-physical mapping
  - Available at **https://github.com/CMU-SAFARI/Mosaic**

- NVIDIA GTX750 Ti

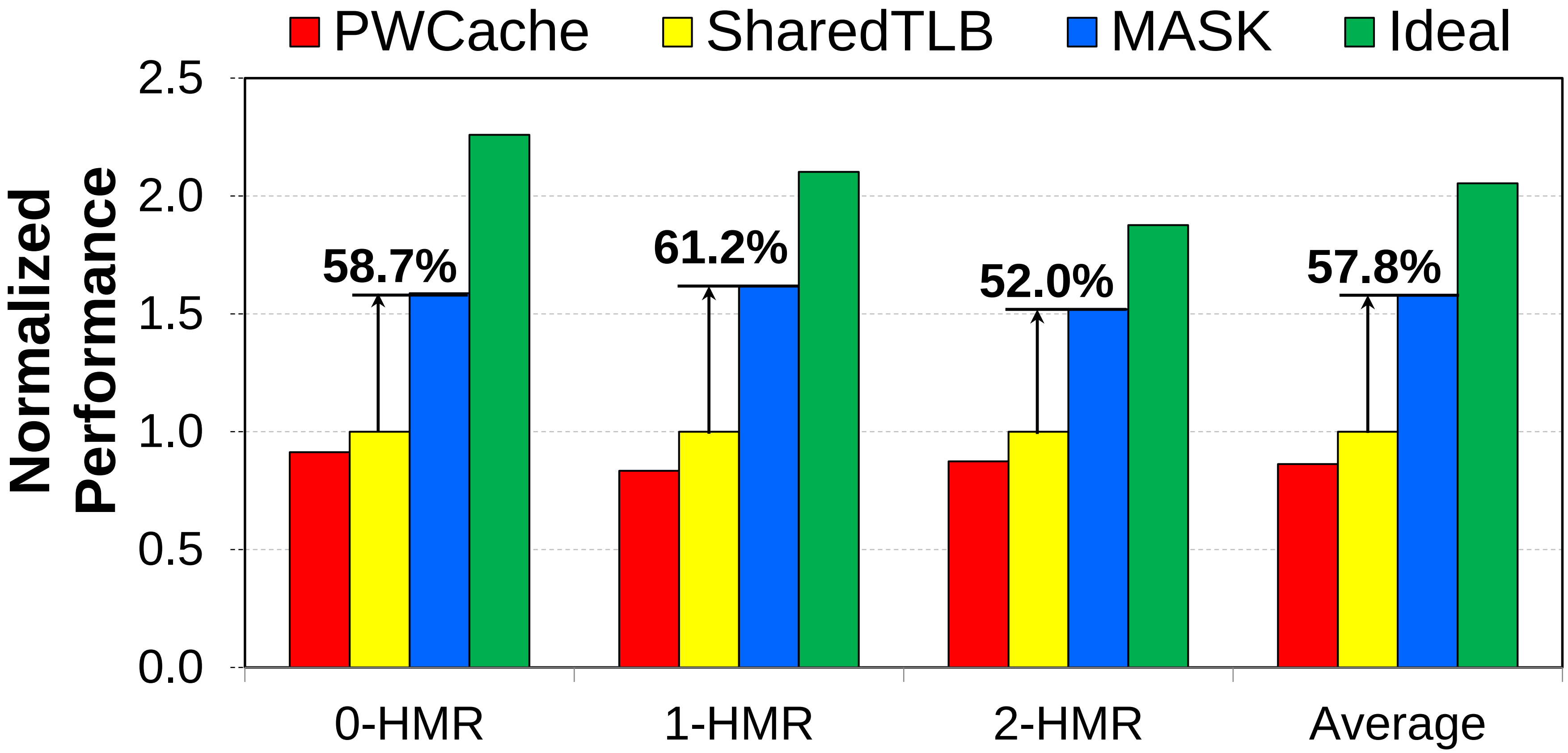
- Two GPGPU applications execute concurrently

- CUDA-SDK, Rodinia, Parboil, LULESH, SHOC suites
  - 3 workload categories based on TLB miss rate

Comparison Points

- State-of-the-art CPU–GPU memory management [Power et al., HPCA '14]
  - → **PWCache:** Page Walk Cache GPU MMU design
  - → **SharedTLB:** Shared TLB GPU MMU design
- **Ideal:** Every TLB access is an L1 TLB hit

Performance



MASK outperforms state-of-the-art design for **every workload**

SAFARI

Other Results in the Paper

- MASK reduces unfairness
- Effectiveness of each individual component
  - **All three MASK components are effective**
- Sensitivity analysis over multiple GPU architectures
  - **MASK improves performance on all evaluated architectures,** including CPU–GPU heterogeneous systems
- Sensitivity analysis to different TLB sizes
  - **MASK improves performance on all evaluated sizes**
- Performance improvement over different memory scheduling policies
  - **MASK improves performance over other state-of-the-art memory schedulers**

Conclusion

Problem: Address translation overheads
limit the latency hiding capability of a GPU

High contention at the shared TLB

Low L2 cache utilization

Large performance loss vs. no translation

Key Idea
Prioritize **address translation requests** over **data requests**

MASK: a translation-aware GPU memory hierarchy

A. TLB-fill Tokens **reduces shared TLB contention**
B. Translation-aware L2 Bypass **improves L2 cache utilization**
C. Address-space-aware Memory Scheduler **reduces page walk latency**

MASK **improves system throughput by 57.8%** on average over state-of-the-art address translation mechanisms

MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency

Rachata Ausavarungnirun

Vance Miller Joshua Landgraf Saugata Ghose

Jayneel Gandhi Adwait Jog Christopher J. Rossbach Onur Mutlu

Carnegie Mellon

TEXAS
The University of Texas at Austin

ETH zürich

vmware

SAFARI

Backup Slides

Other Ways to Manage TLB Contention

- **Prefetching:**
  - Stream prefetcher is ineffective for multiple workloads
  - GPU's parallelism makes it hard to predict which translation data to prefetch
  - GPU's parallelism causes thrashing on the prefetched data

- **Reuse-based technique:**
  - Lowers TLB hit rate
  - Most pages have similar TLB hit rate

Other Ways to Manage L2 Thrashing

- **Cache Partitioning**
  - Performs ~3% worse on average compared to Translation-aware L2 Bypass
  - Multiple address translation requests still thrash each other
  - Can lead to underutilization
  - Lowers hit rate of data requests

- **Cache Insertion Policy**
  - Does not yield better hit rate for lower page table level
  - Does not benefit from lower queuing latency

Utilizing Large Page?

- **One single large page size**
  - → High demand paging latency
  - → > 90% performance overhead with demand paging
  - → All threads stall during large page PCIe transfer
- **Mosaic [Ausavarungnirun et al., MICRO'17]**
  - → Supports for multiple page sizes
  - → Demand paging happens on small page granularity
  - → Allocates data from the same application in large page granularity
  - → Opportunistically coalesces small page to reduce TLB contention

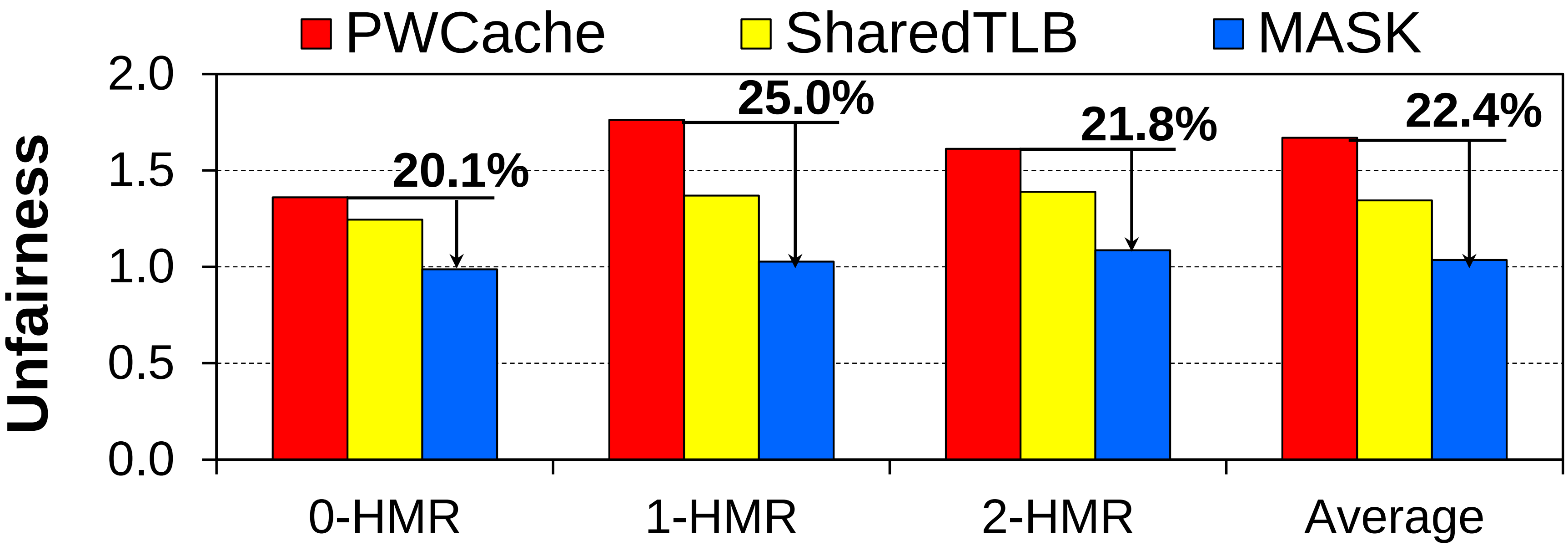
→ **MASK + Mosaic** performs within 5% of the Ideal TLB

Area and Storage Overhead

- **Area overhead**
  - <1% area of its original components (Shared TLB, L2$, Memory Scheduler)

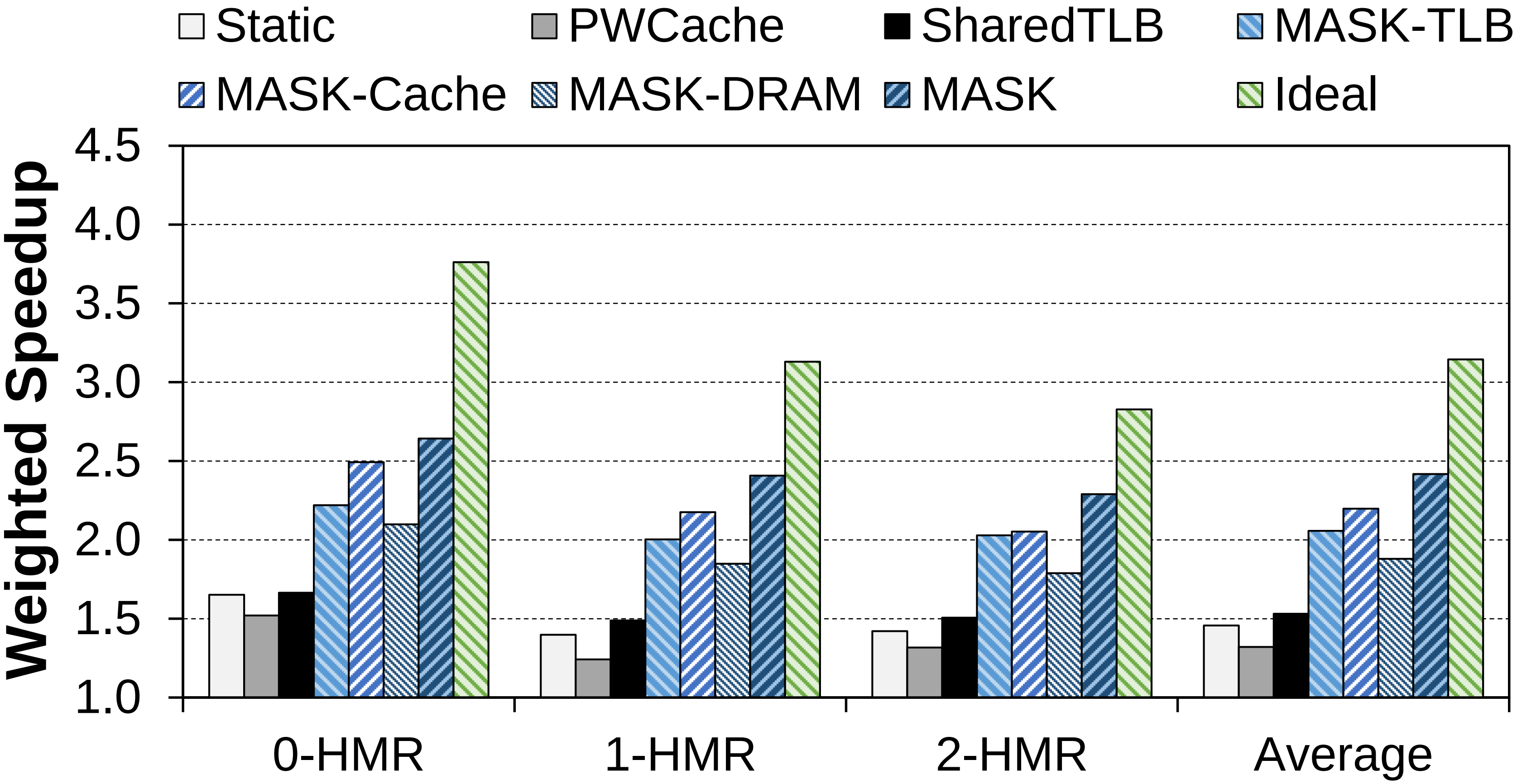
- **Storage overhead**
  - TLB-fill Tokens:
    - 3.8% extra storage on Shared TLB
  - Translation-aware L2 Bypass:
    - 0.1% extra storage on L2$
  - Address-space-aware Memory Scheduler:
    - 6% extra memory request buffer

Unfairness

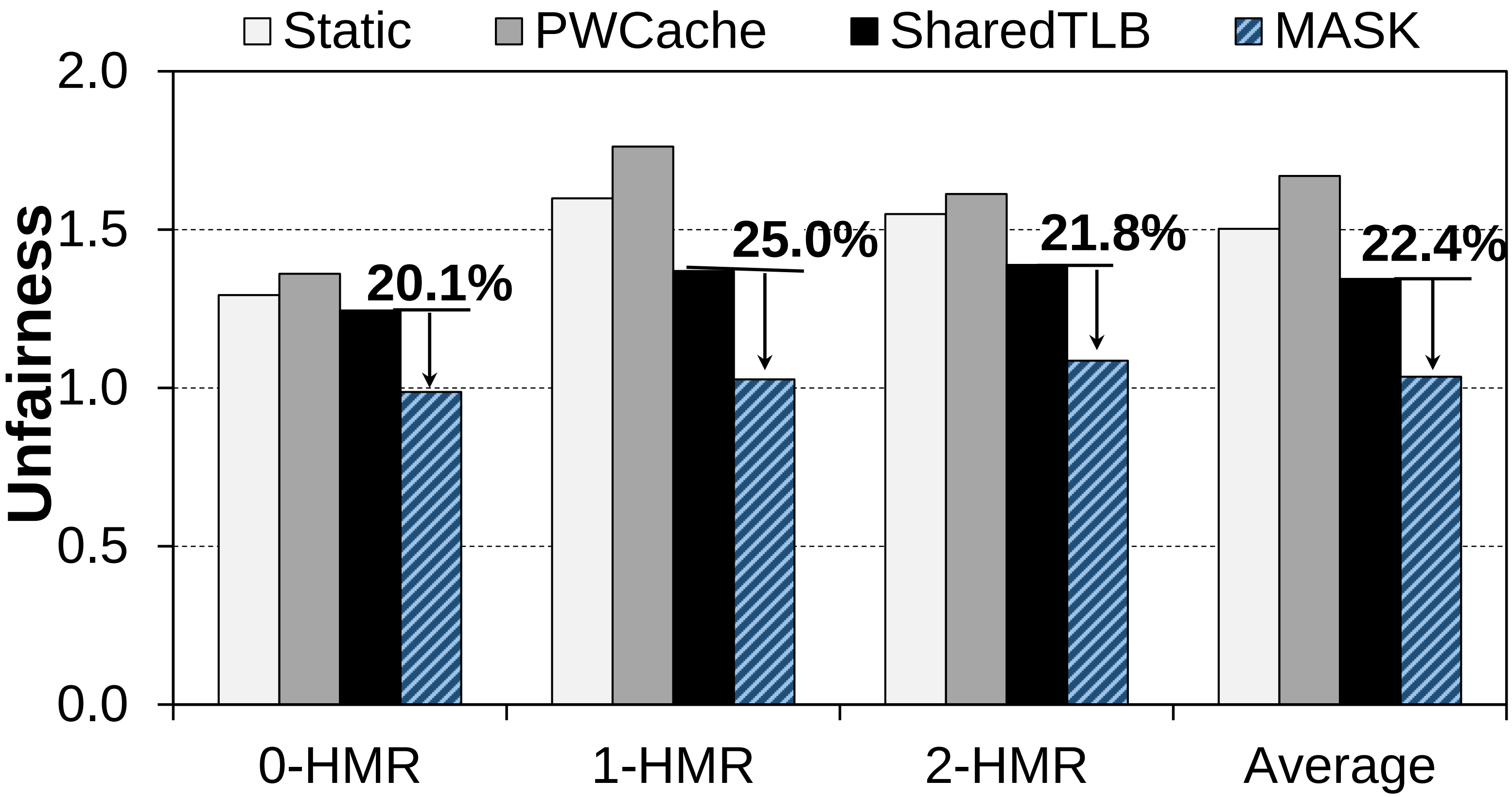


MASK is effective at improving fairness

Performance vs. Other Baselines

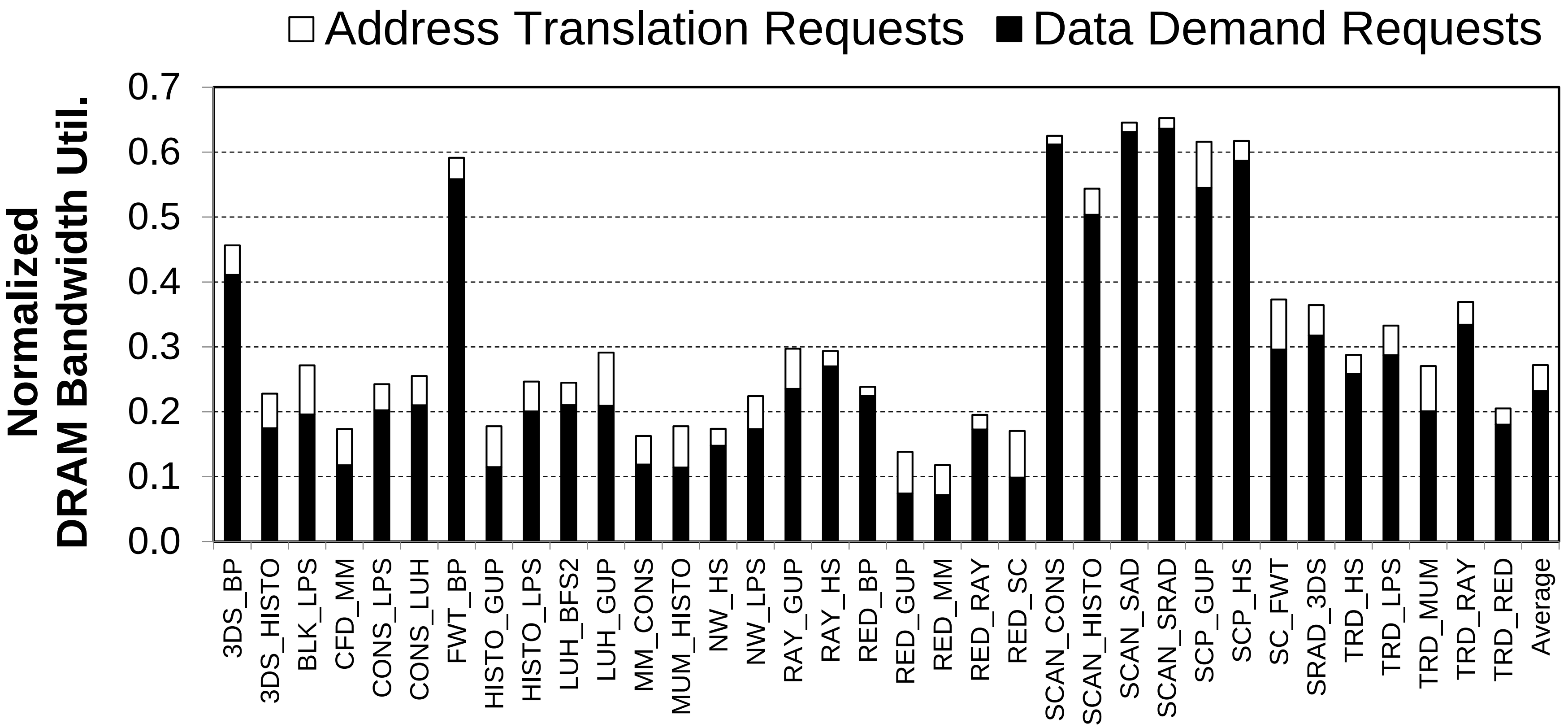


Unfairness

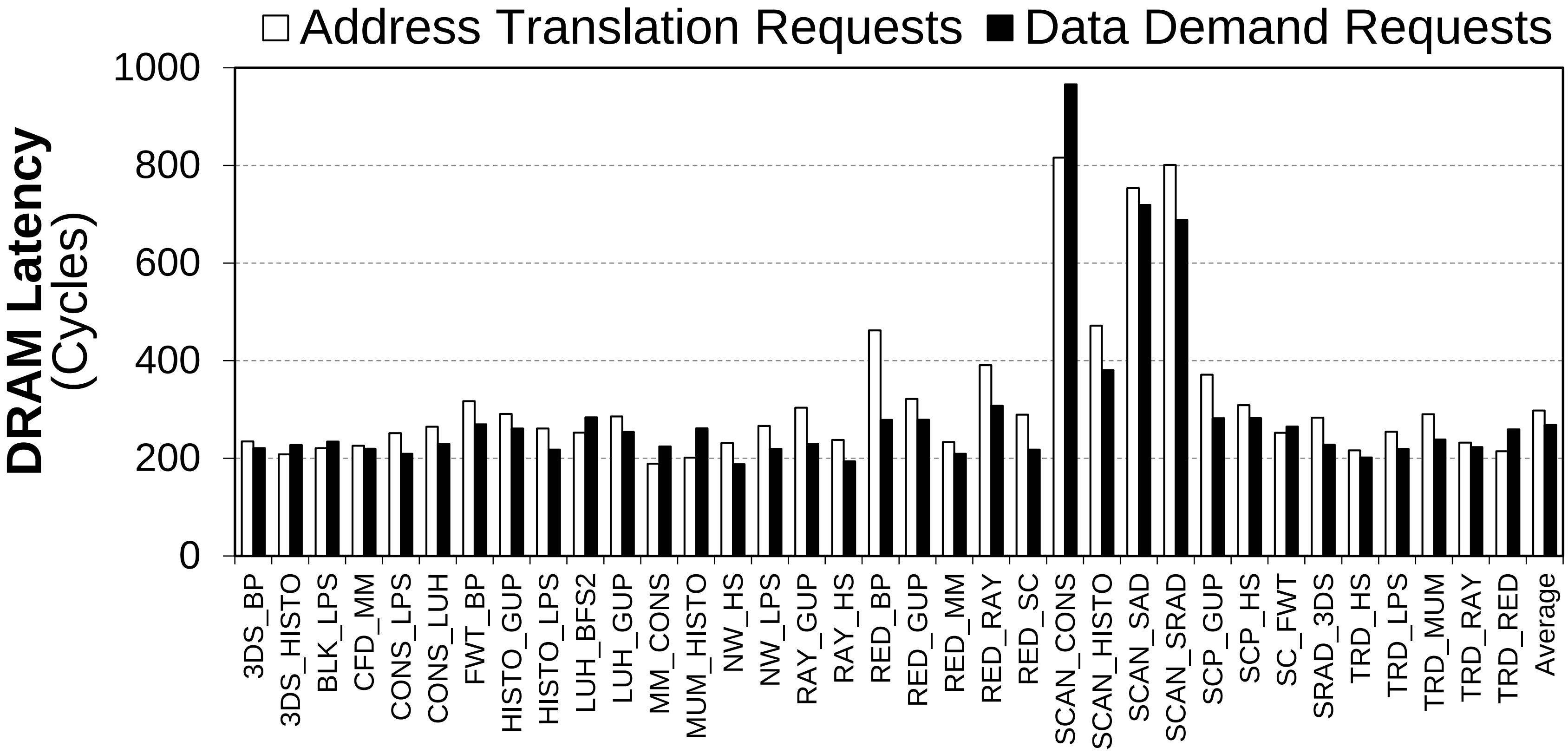


Static
PWCache
SharedTLB
MASK
Unfairness
2.0
1.5
1.0
0.5
0.0
20.1%
25.0%
21.8%
22.4%
0-HMR
1-HMR
2-HMR
Average

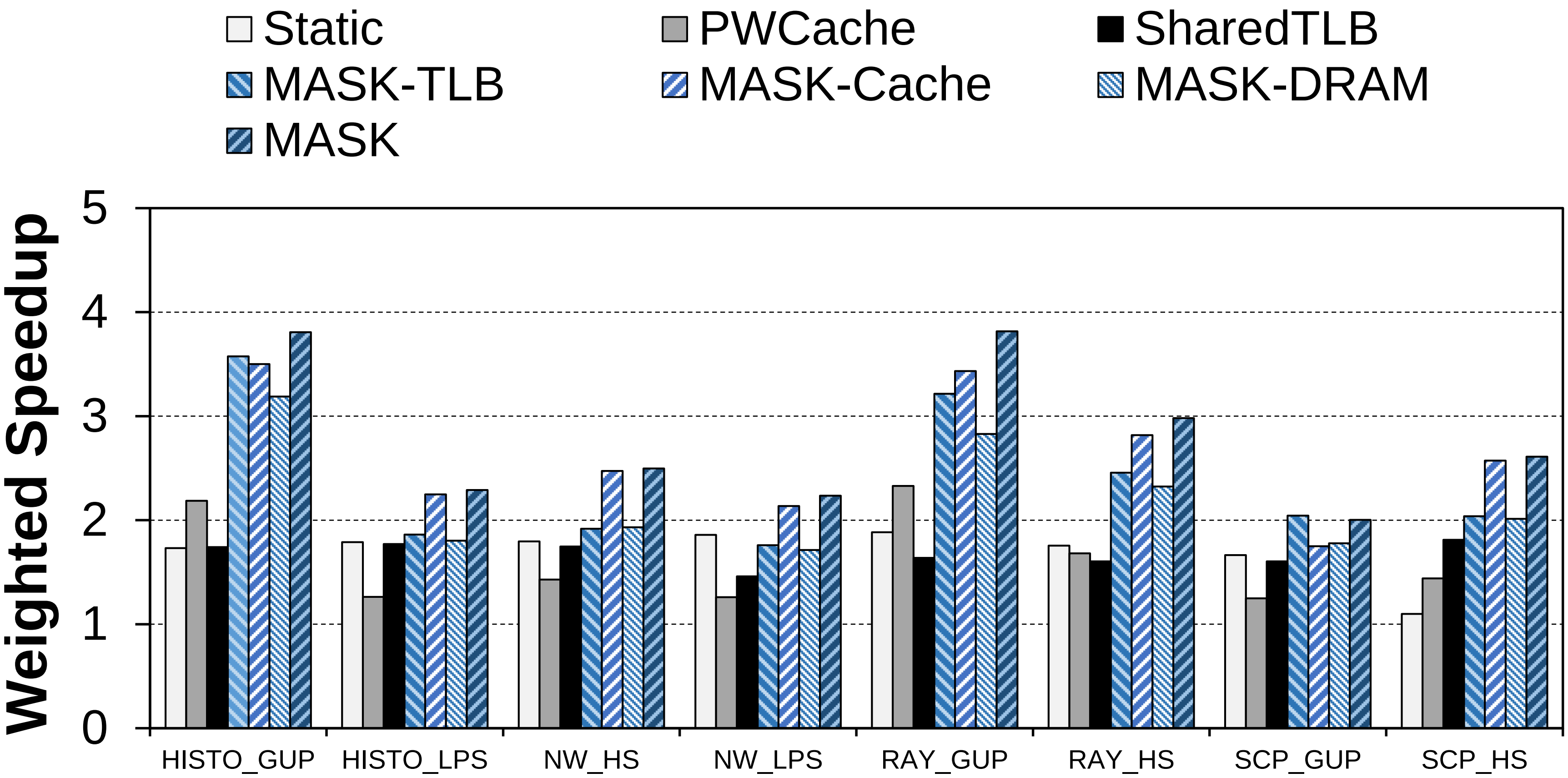
DRAM Utilization Breakdowns



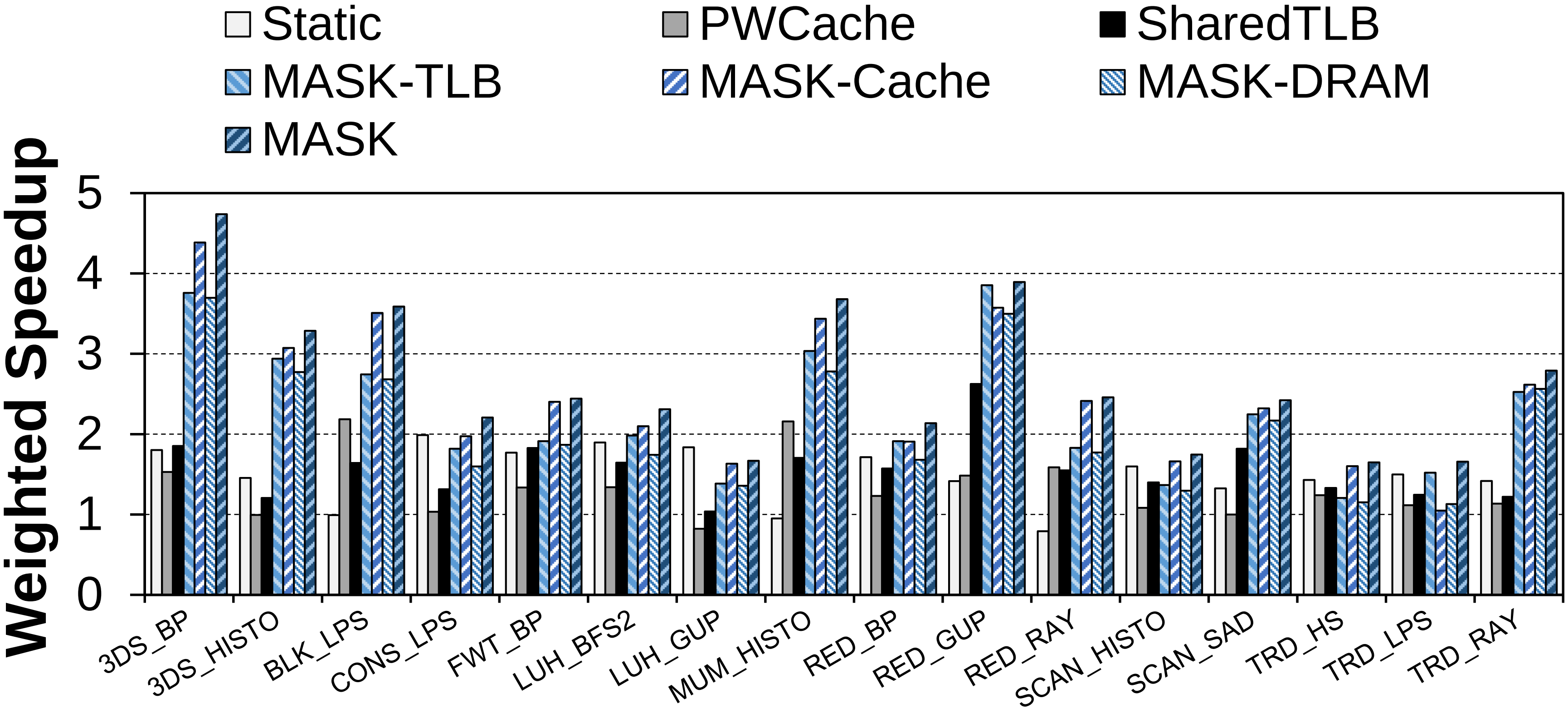
DRAM Latency Breakdowns



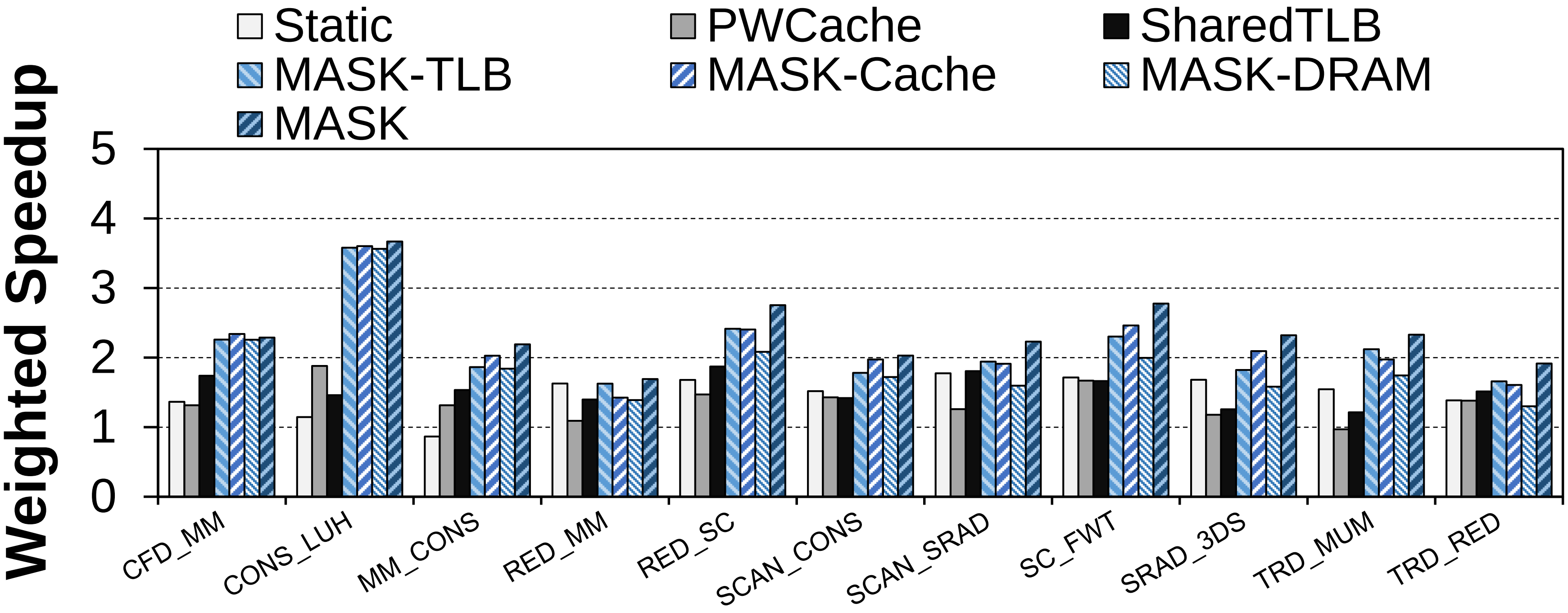
0-HMR Performance



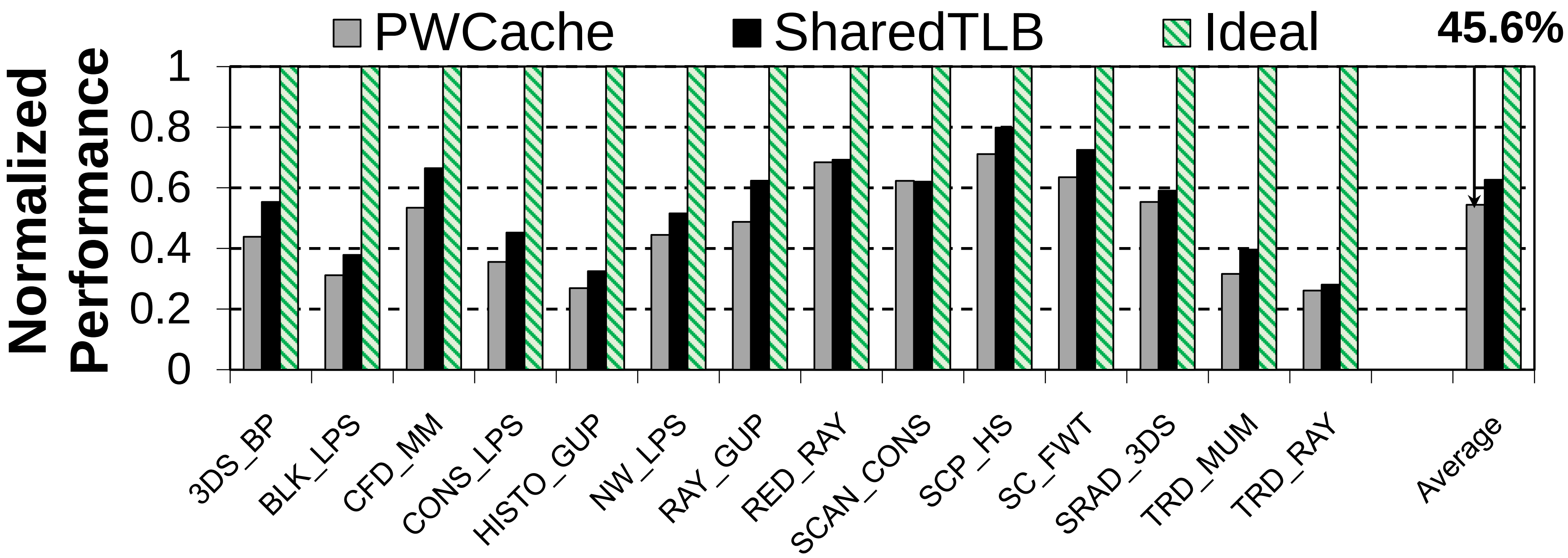
1-HMR Performance



2-HMR Performance



Additional Baseline Performance


PWCache
SharedTLB
Ideal
Normalized Performance
3DS_BP
BLK_LPS
CFD_MM
CONS_LPS
HISTO_GUP
NW_LPS
RAY_GUP
RED_RAY
SCAN_CONS
SCP_HS
SC_FWT
SRAD_3DS
TRD_MUM
TRD_RAY
Average