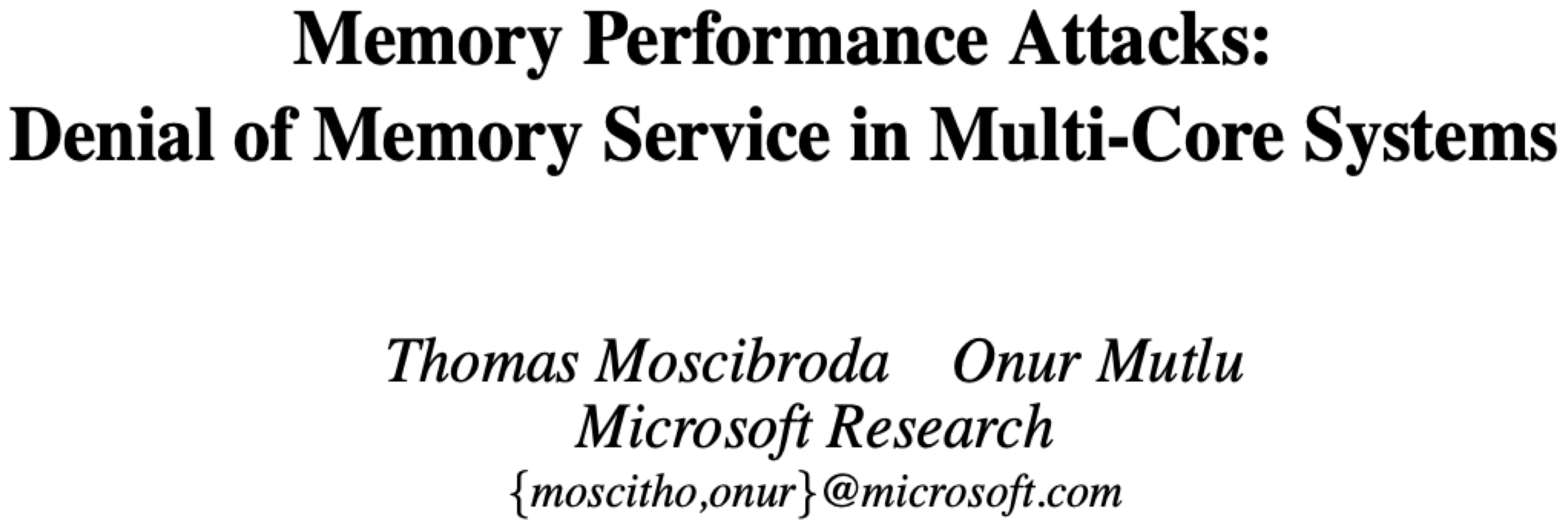
ETH zürich

Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems

Anlin Yan
ETH Zurich

Fall 2019

28 November 2019



Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems

Thomas Moscibroda *Onur Mutlu*
Microsoft Research
{*moscitho,onur*}@*microsoft.com*

Proceedings of the 16th USENIX Security Symposium (USENIX SECURITY), pages 257-274, Boston, MA, August 2007

Summary

- **Problem:**
  - DRAM memory scheduler designed for single-core system
  - In multi-core system: unfair when threads with certain access pattern are present

- **Goal:** fair memory scheduler

- **Key Ideas:**
  - Approximate unfairness by computing thread slowdown
  - Prioritize unfairly slowed-down threads

- **Result:** FairMem removes denial of service threat

Structure

- Background
  - Denial of Service
  - DRAM
- Memory Performance Attacks: Denial of Memory Service in Multi-Core System
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - Summary
- Analysis
  - Strengths
  - Weaknesses
  - Takeaways
  - Further ideas
  - Discussion

Structure

- **Background**
  - Denial of Service
  - DRAM
- Memory Performance Attacks: Denial of Memory Service in Multi-Core System
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - Summary
- Analysis
  - Strengths
  - Weaknesses
  - Takeaways
  - Further ideas
  - Discussion

Structure

Denial of Service (DoS)

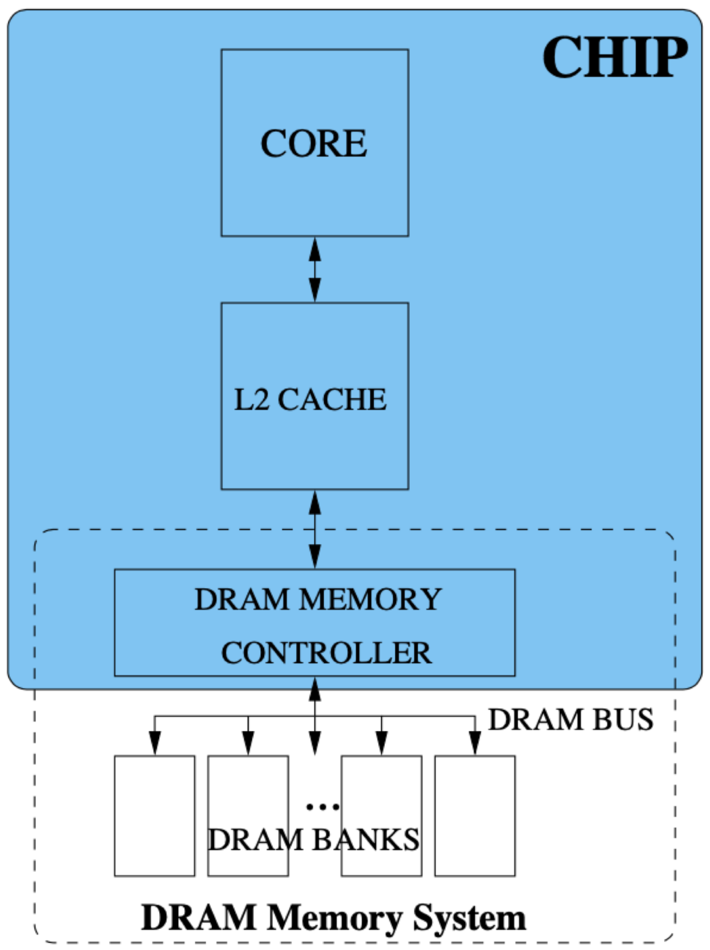
- Any type of attack where the attacker prevents legitimate users from accessing some resource

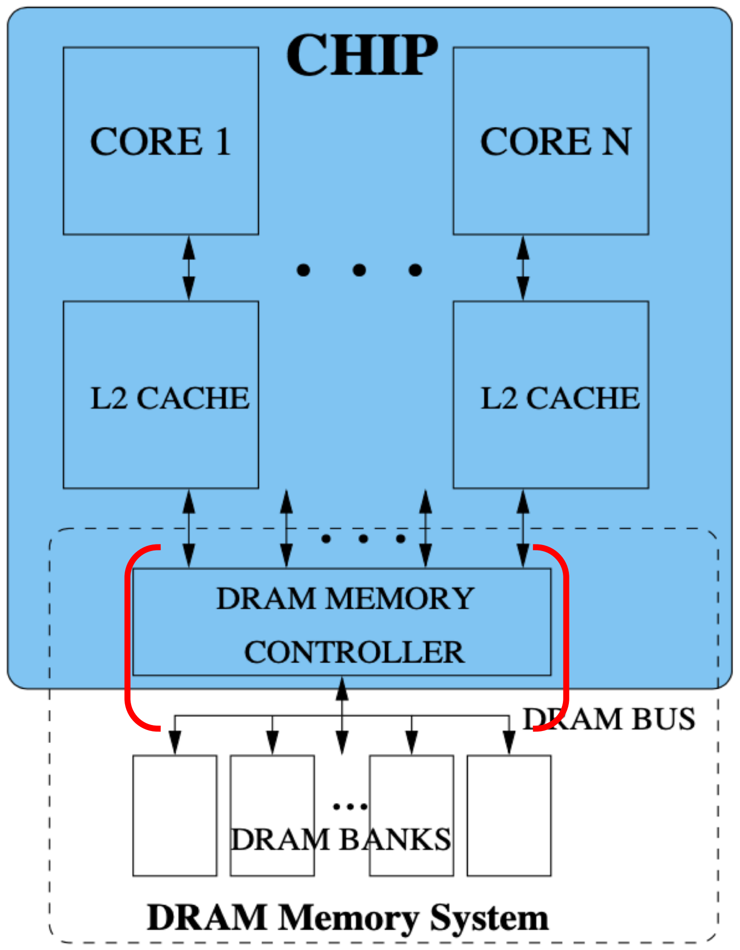
Structure

- **Background**
  - Denial of Service
  - **<u>DRAM</u>**
- Memory Performance Attacks: Denial of Memory Service in Multi-Core System
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - Summary
- Analysis
  - Strengths
  - Weaknesses
  - Takeaways
  - Further ideas
  - Discussion

Multi-Core Systems

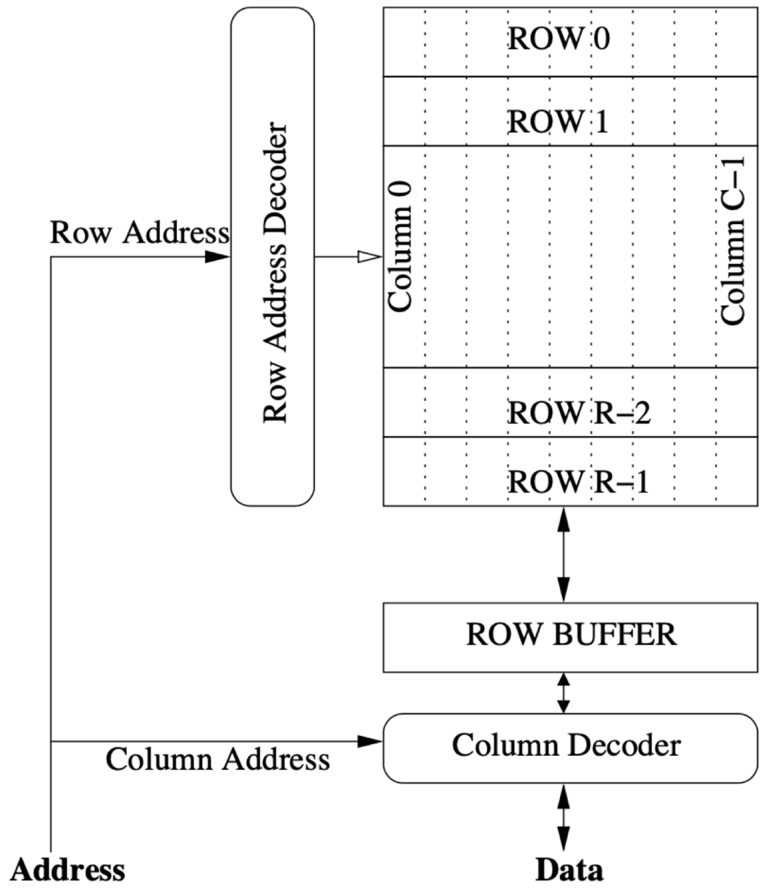
- Separate caches
- Shared DRAM memory system
- DRAM receives requests at L2 granularity



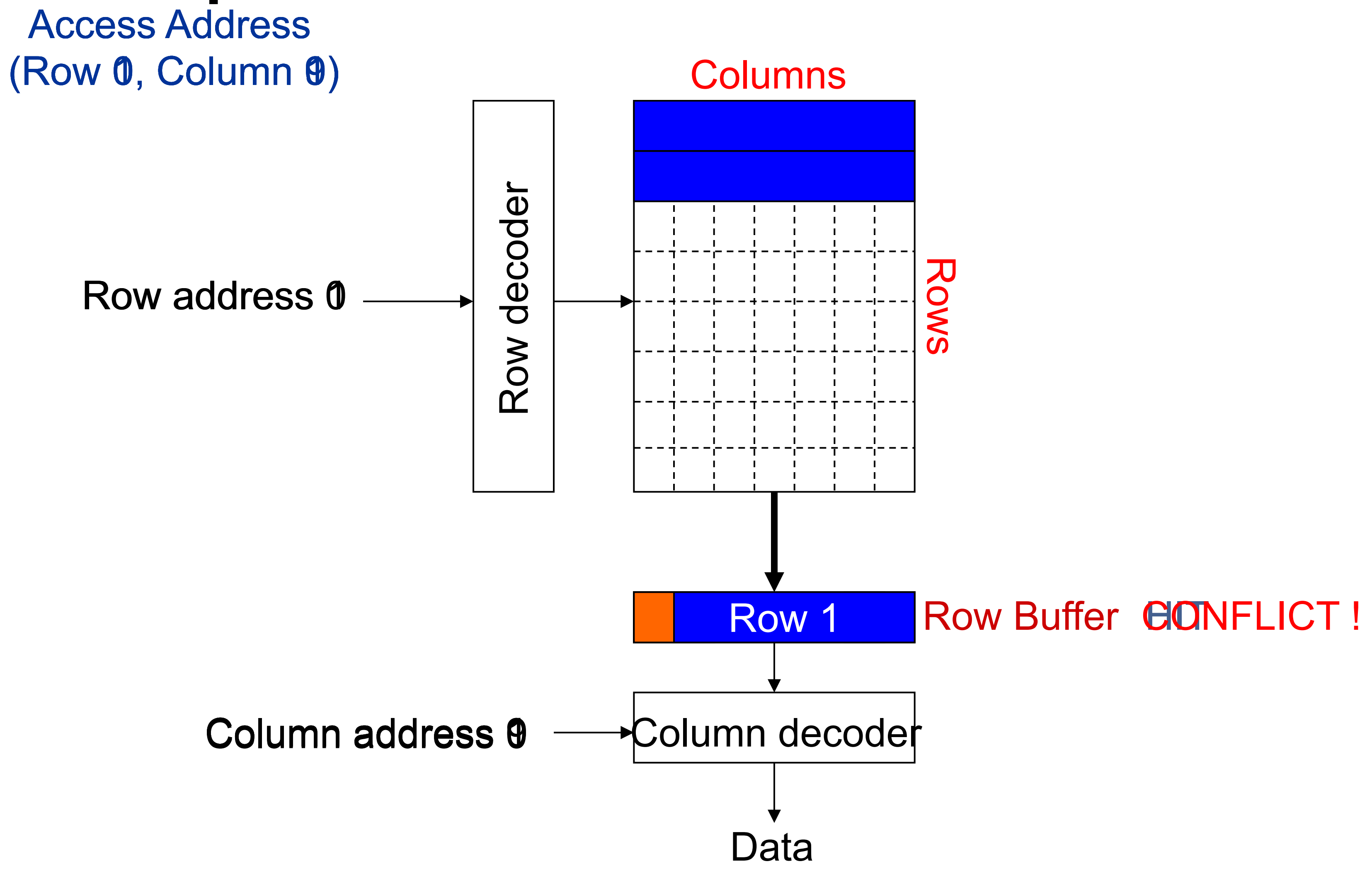


DRAM Bank

- Banks store data
- Each bank has row buffer
- Access memory through row buffer
- Row buffer contains at most 1 row
- Request hits or misses row buffer content
→ Row hit/ row miss
- Row miss has large latency
- Multiple banks
→ Bank-level parallelism



DRAM Bank Operations

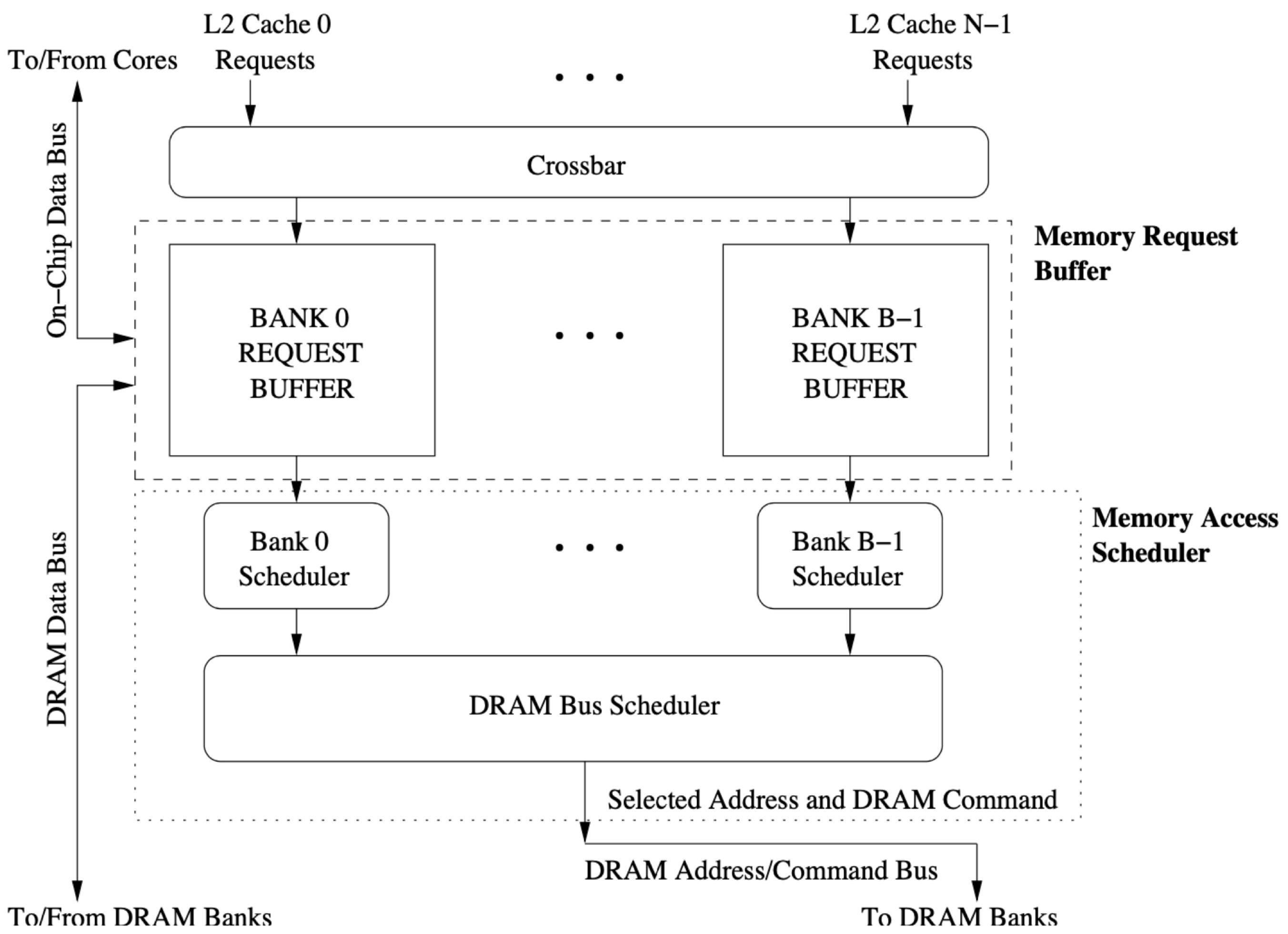


ETH zürich

Multi-Core DRAM Memory Systems

- Seperate bank request buffers
→ Seperate per bank scheduler

- Single, shared bus
→ Single, shared bus scheduler



DRAM Memory Access Scheduling Mechanism: FR-FCFS

Bank scheduler:

1. Row-hit-first
2. Oldest-within-bank-first

Bus scheduler:

- Oldest-across-banks-first (among all requests proposed by individual bank scheduler)

Structure

- Background
  - Denial of Service
  - DRAM
- **Memory Performance Attacks: Denial of Memory Service in Multi-Core System**
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - Summary
- Analysis
  - Strengths
  - Weaknesses
  - Takeaways
  - Further ideas
  - Discussion

Structure

FR-FCFS: Bank Scheduler's Vulnerability To Denial of Service

Bank scheduler:

- Row-hit-first

→ Low row-buffer locality threads have low priority

- Oldest-within-bank-first

→ Prioritizes threads that generate requests fast

FR-FCFS: Bus Scheduler’s Vulnerability To Denial of Service

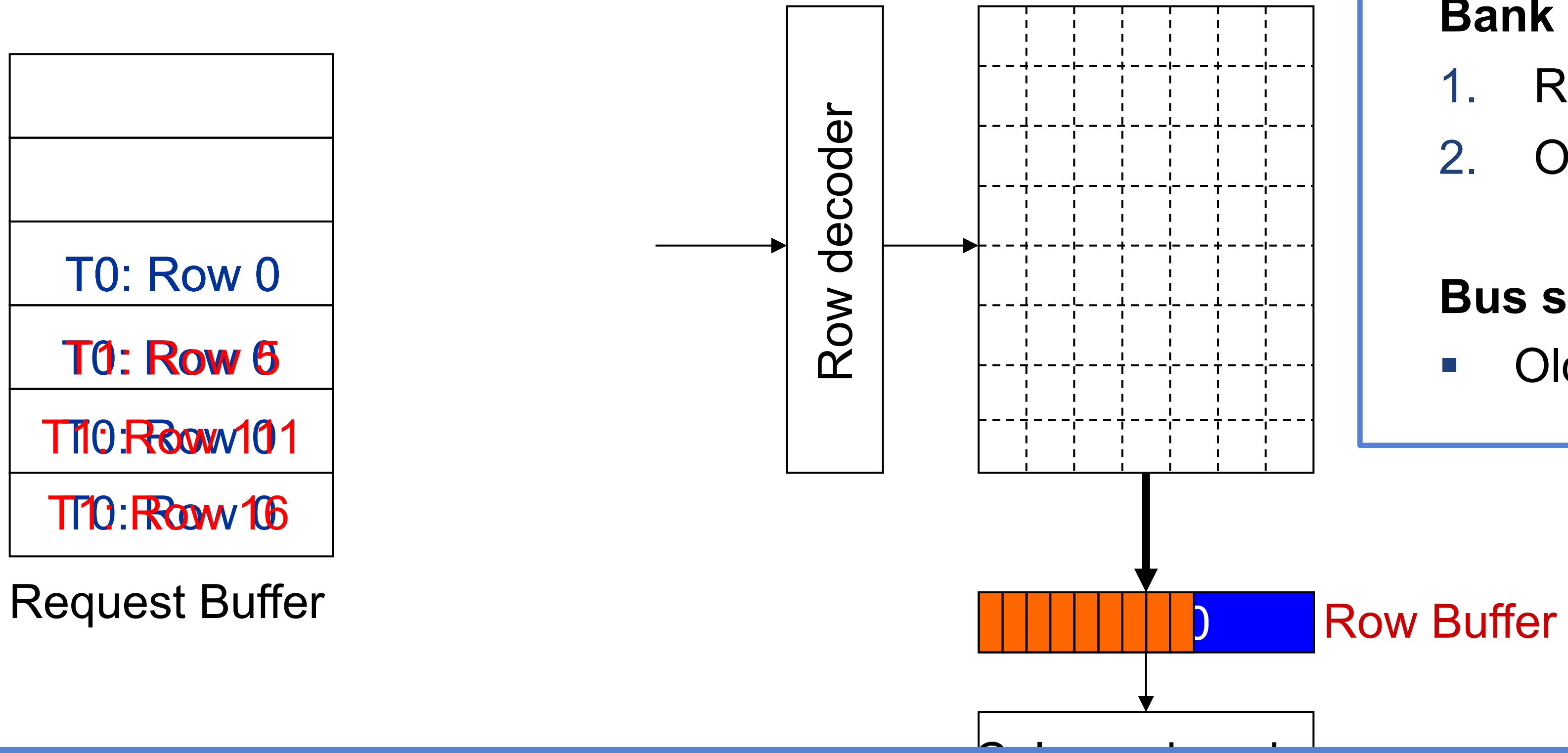
Bus scheduler:

- Oldest-across-banks-first (among all requests proposed by individual bank scheduler)

→ First level selects requests in a row hit maximizing way, no regard for time of arrival of request

→ Aggressive threads serviced more

What Does An Aggressive, High Row-Buffer Locality Thread Do?



Bank scheduler:

1. Row-hit-first
2. Oldest-within-bank-first

Bus scheduler:

- Oldest-across-banks-first

It hogs DRAM!

DoS Threat: Memory Performance Hog (MPH)

Is an aggressive, high row-buffer locality thread…

- Hogs shared resources
- Significantly reduces performance of other threads
- No significant performance reduction itself

“Any attack where the attacker prevents legitimate users from accessing some resource”

DoS Threat: Memory Performance Hog (MPH)

An aggressive, high row-buffer locality thread…

- Hogs shared resources
- Significantly reduces performance of other threads
- No significant performance reduction itself

MPH

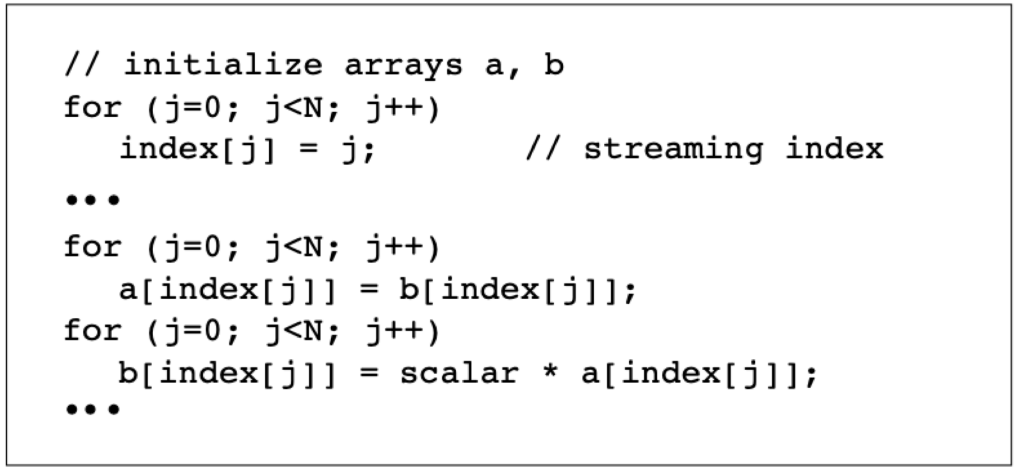
Slow, low row-buffer locality threads

"Any attack where the attacker prevents legitimate users from accessing some resource" ← DRAM

MPH is DoS threat!

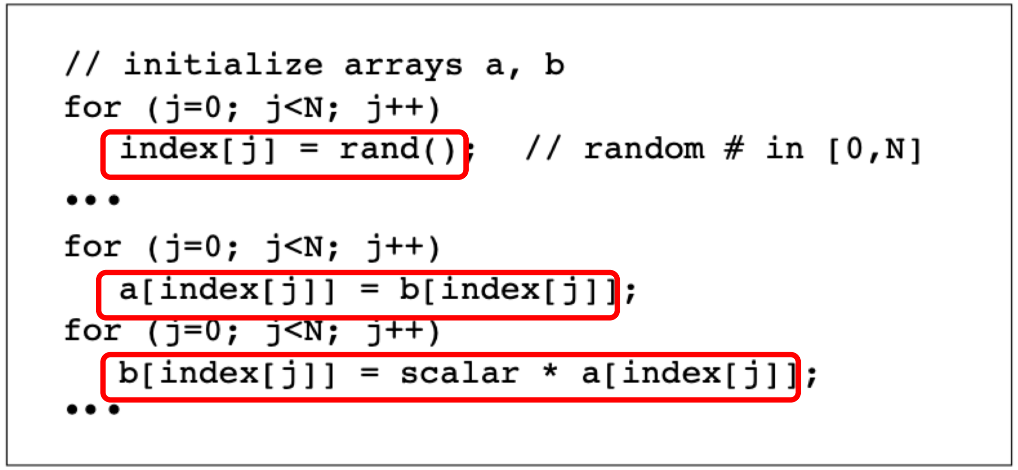
Memory Performance Hog (MPH) Example

- Intel Pentium D 930, dual-core system
- N>> L2 cache



```
// initialize arrays a, b
for (j=0; j<N; j++)
    index[j] = j;        // streaming index
...
for (j=0; j<N; j++)
    a[index[j]] = b[index[j]];
for (j=0; j<N; j++)
    b[index[j]] = scalar * a[index[j]];
...
```

(a) STREAM



```
// initialize arrays a, b
for (j=0; j<N; j++)
    index[j] = rand();   // random # in [0,N]
...
for (j=0; j<N; j++)
    a[index[j]] = b[index[j]];
for (j=0; j<N; j++)
    b[index[j]] = scalar * a[index[j]];
...
```

(b) RDARRAY

- High row-buffer locality
- Low row-buffer locality

Memory Performance Hog (MPH) Example: Result

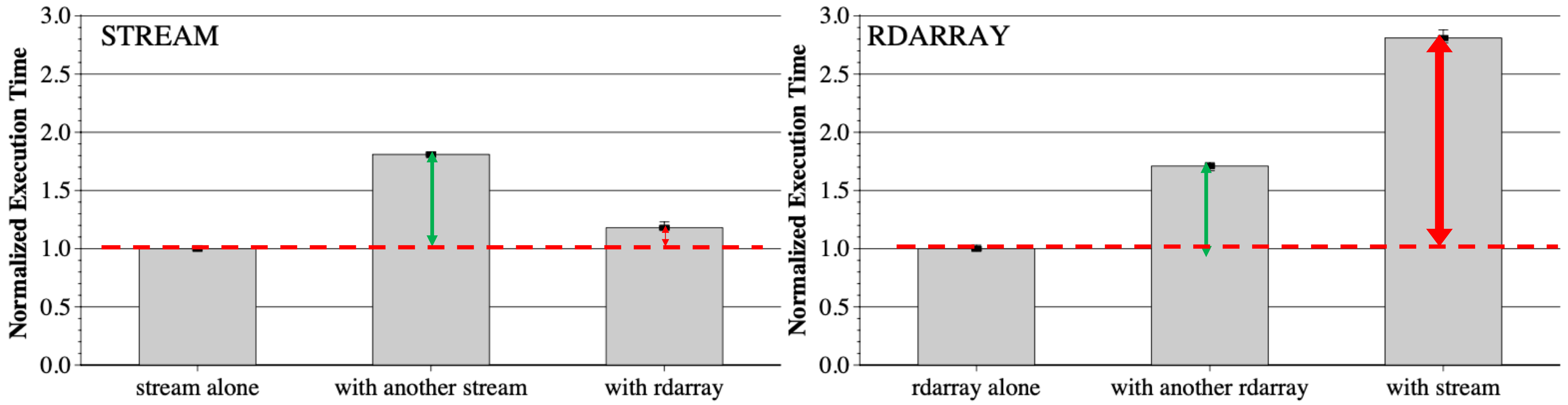


Figure 4: Normalized execution time of (a) *stream* and (b) *rdarray* when run alone/together on a dual-core system

Key Problem:
Memory Access Scheduler Unfair In Multi-Core Systems

- MPH can destroy other threads‘ performance
- Memory system cannot distinguish between erroneous programming, necessary memory behavior of the application or malicious attack

Multi-core system needs new scheduling policy!

Structure

Fairness

- N applications receive 1/N of system resources, progress at 1/N

→ Does not consider bank has "state"

→ Disregards row-buffer locality

  → Punishes high row-buffer locality threads

Need new definition of fairness in DRAM sense!

Fairness in DRAM Memory Sense

- Latency
  - Thread inherent latency (depends on row-buffer locality)
  - Contention caused latency

- Fairness
  - Thread inherent latency unchanged
  - Contention caused latency proportionally distributed across all threads

Structure

Fair Memory Scheduling Model

Cumulated latency of thread *i*: L_{i_shared}

- Ideal single-core cumulated latency: L_{i_alone}
- Ratio of experienced slowdown: $\mathrm{s_i} := \frac{L_{i\_shared}}{L_{i\_alone}}$
- DRAM fairness index: $F := \frac{\max_{s_i} s_i}{\min_{s_i} s_i}$

→ F=1: every thread experiences the same relative slowdown

Fair Memory Scheduling Model: Short-term vs long-term fairness

- Cumulated latency of thread *i:* L_{i_shared} increases with thread life
- Short-term unfairness increasingly little effect on s_i

→ Introduce time interval *T*

Fair Memory Scheduling Model Over Interval T

- Cumulated latency of thread *i* over *T*: $L_{i\_shared}(T)$
- Ideal single-core cumulated latency: $L_{i\_alone}(T)$
- Ratio of experienced slowdown: $\mathrm{s_i(T)}$
- DRAM fairness index: $F(T)$
- Good $F(T)$ for large T $\nRightarrow$ good $F(T')$ for small T'

A Fair Memory Scheduling Algorithm: FairMem

Unfairness threshold

If $F(T) \geq \alpha$

    $then\ A_{fair}$

Else

    $A_{FR-FCFS}$

A_{fair}

Bank scheduler:

1. Highest slowdown-index $s_i(T)$ first
2. Highest FR-FCFS first

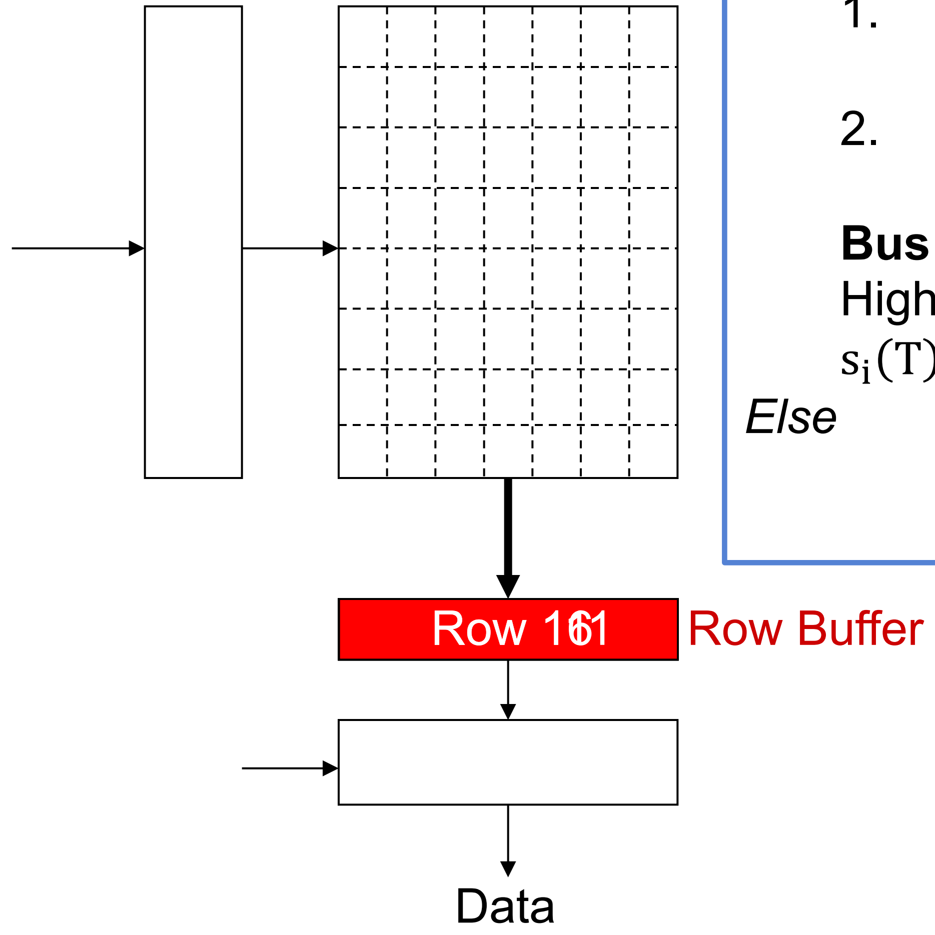
Bus scheduler:

- Highest slowdown-index $s_i(T)$ first

How Does FairMem Prevent DoS?

T0: Row 0
T1: Row 5
T0: Row 0
T1: Row 111
T0: Row 0
T1: Row 16

T0 Slowdown	1.04
T1 Slowdown	1.06
Unfairness	1.06
α	1.05



If $F(T) \geq \alpha$

Bank scheduler:

1. Highest slowdown-index $\mathrm{s_i(T)}$
2. Highest FR-FCFS

Bus scheduler:

Highest slowdown-index $\mathrm{s_i(T)}$

Else

$A_{FR-FCFS}$

FairMem

- Prioritize most slowed-down thread

→ Limits adverse effect of MPH

- Throughput maximizing under fairness constraint

```
If $F(T) \geq \alpha$
    **Bank scheduler:**
    1. Highest slowdown-index $s_i(T)$
    2. Highest FR-FCFS

    **Bus scheduler:**
    Highest slowdown-index $s_i(T)$
Else
        $A_{FR-FCFS}$
```

Structure

- Background
  - Denial of Service
  - DRAM
- **Memory Performance Attacks: Denial of Memory Service in Multi-Core System**
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - **Hardware implementation**
  - Results
  - Sensitivity analysis
  - Summary
- Analysis
  - Strengths
  - Weaknesses
  - Takeaways
  - Further ideas
  - Discussion

Hardware Implementation of FairMem

- $L_{i\_shared}(T)$: count #memory cycles where ready request of thread *i* buffered, for each bank

- $L_{i\_alone}(T)$: simulate thread *i* running alone on single-core with FR-FCFS
  - Maintain what would be in row-buffer
  - Ignore all requests by threads *j*, *j≠i*

→ O(#cores x #banks) counters

Economical Hardware Implementation of FairMem: Reduce Counters by Sampling

Random sample of subset of requests by thread *i* to some bank *b*

→ Does *i* request the same row in its next request to *b*?
  → Approximate row hit rate
    → Approximate latency

→ O(#cores) counters

ETH zürich

Economical Hardware Implementation of FairMem

$$\mathrm{s_i}(T) := \frac{L_{i_{shared}}(T)}{L_{i_{alone}}(T)}, \quad F := \frac{\max\limits_{s_i} s_i(T)}{\min\limits_{s_i} s_i(T)}$$

- Dividers have high energy consumption

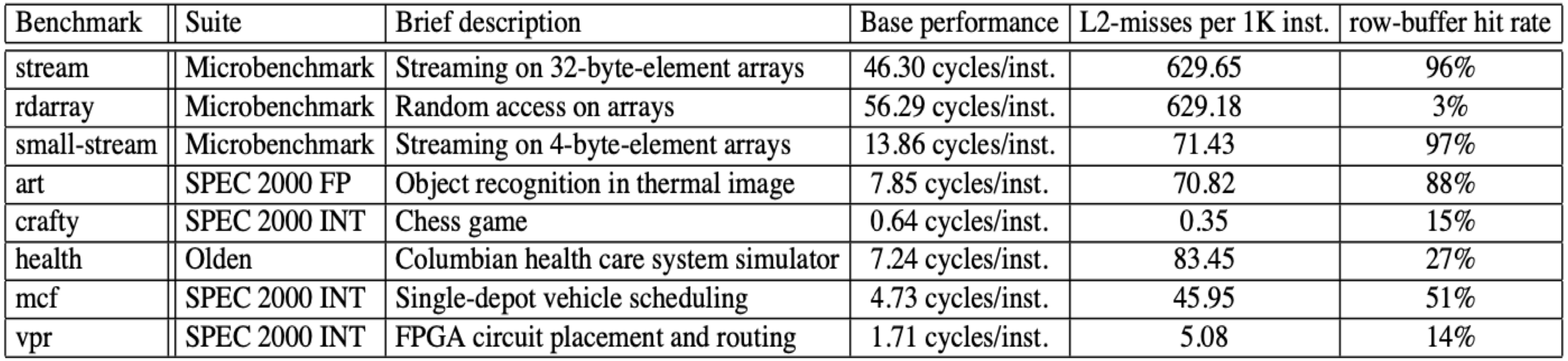
→1 divider, reused in round robin to compute in intervals

Structure

FairMem: Set up

- Processor and memory system simulator based on Pin dynamic binary instrumentation tool
- DRAM simulation based on *DRAMsim*
- Instruction-level performance simulator for simulating applications compiled for x86

→ Mimic dual-core, based on Intel Pentium M



Benchmark	Suite	Brief description	Base performance	L2-misses per 1K inst.	row-buffer hit rate
stream	Microbenchmark	Streaming on 32-byte-element arrays	46.30 cycles/inst.	629.65	96%
rdarray	Microbenchmark	Random access on arrays	56.29 cycles/inst.	629.18	3%
small-stream	Microbenchmark	Streaming on 4-byte-element arrays	13.86 cycles/inst.	71.43	97%
art	SPEC 2000 FP	Object recognition in thermal image	7.85 cycles/inst.	70.82	88%
crafty	SPEC 2000 INT	Chess game	0.64 cycles/inst.	0.35	15%
health	Olden	Columbian health care system simulator	7.24 cycles/inst.	83.45	27%
mcf	SPEC 2000 INT	Single-depot vehicle scheduling	4.73 cycles/inst.	45.95	51%
vpr	SPEC 2000 INT	FPGA circuit placement and routing	1.71 cycles/inst.	5.08	14%

Table 2: Evaluated applications and their performance characteristics on the baseline processor

FairMem: Results Microbenchmarks

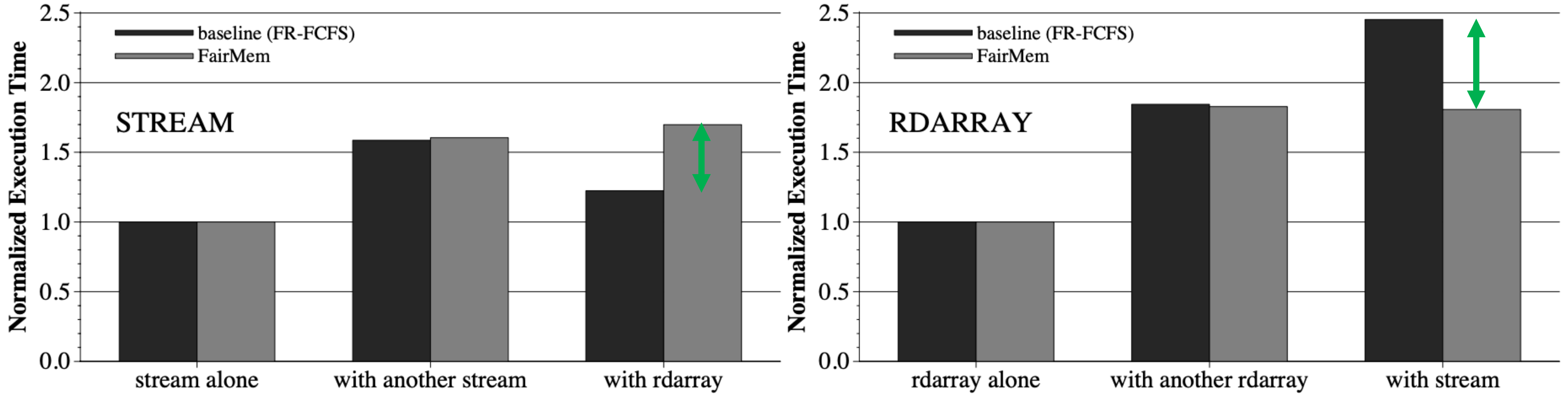


Figure 7: Slowdown of (a) *stream* and (b) *rdarray* benchmarks using FR-FCFS and our FairMem algorithm

FairMem is successful in containing the effect of MPH in Microbenchmarks!

ETH zürich

FairMem: Results Real Applications

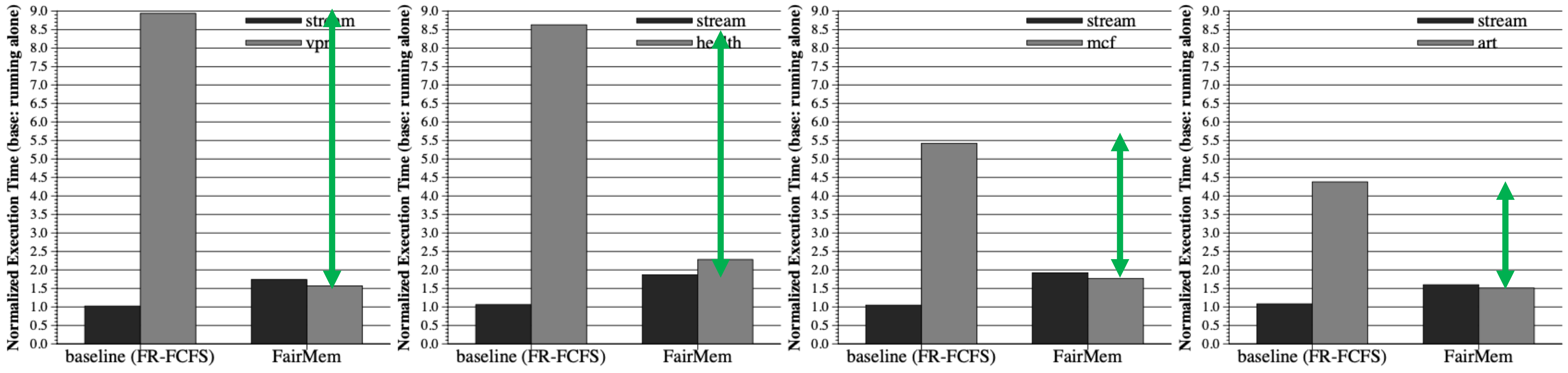
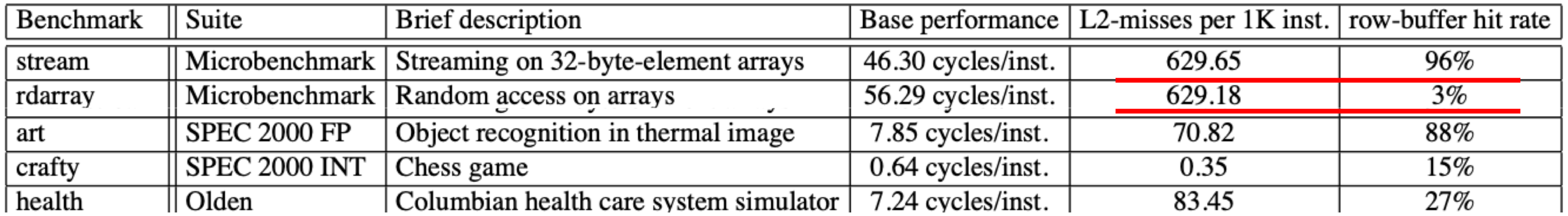


Figure 8: Slowdown of different application combinations using FR-FCFS and our FairMem algorithm

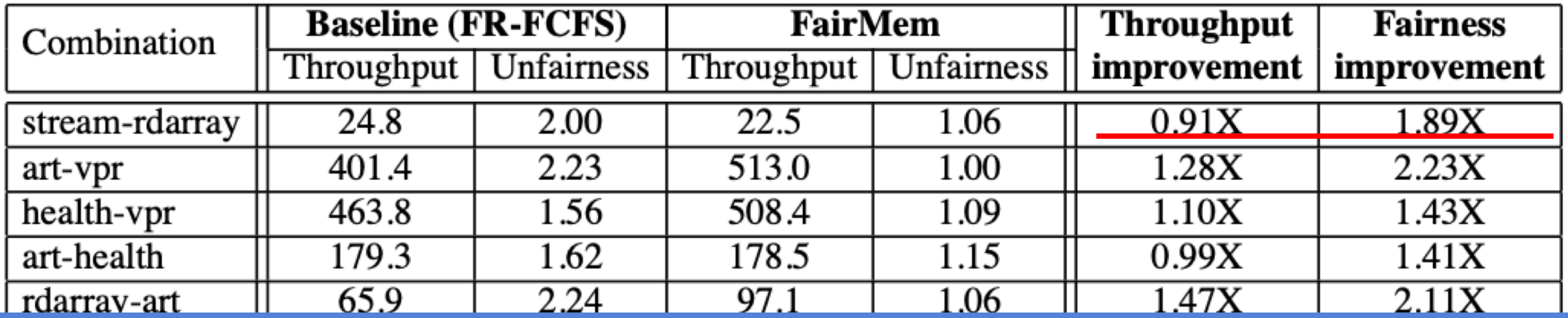
FairMem is successful in containing the effect of MPH in real applications!

FairMem: Results Real Applications



Benchmark	Suite	Brief description	Base performance	L2-misses per 1K inst.	row-buffer hit rate
stream	Microbenchmark	Streaming on 32-byte-element arrays	46.30 cycles/inst.	629.65	96%
rdarray	Microbenchmark	Random access on arrays	56.29 cycles/inst.	629.18	3%
art	SPEC 2000 FP	Object recognition in thermal image	7.85 cycles/inst.	70.82	88%
crafty	SPEC 2000 INT	Chess game	0.64 cycles/inst.	0.35	15%
health	Olden	Columbian health care system simulator	7.24 cycles/inst.	83.45	27%

Table 2: Evaluated applications and their performance characteristics on the baseline processor



Combination	Baseline (FR-FCFS)		FairMem		Throughput improvement	Fairness improvement
	Throughput	Unfairness	Throughput	Unfairness		
stream-rdarray	24.8	2.00	22.5	1.06	0.91X	1.89X
art-vpr	401.4	2.23	513.0	1.00	1.28X	2.23X
health-vpr	463.8	1.56	508.4	1.09	1.10X	1.43X
art-health	179.3	1.62	178.5	1.15	0.99X	1.41X
rdarray-art	65.9	2.24	97.1	1.06	1.47X	2.11X

Throughput decreases when running applications with high L2-miss rate!

Structure

Effect of Number of Cores

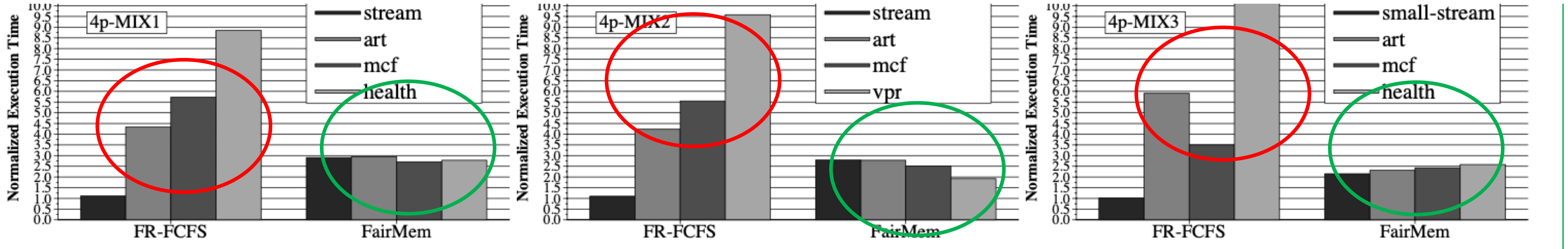


Figure 12: Effect of FR-FCFS and FairMem scheduling on different application mixes in a 4-core system

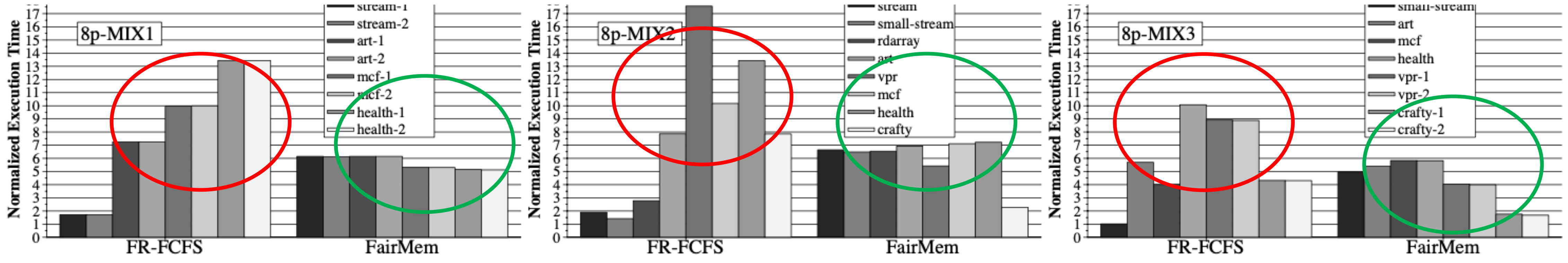


Figure 13: Effect of FR-FCFS and FairMem scheduling on different application mixes in an 8-core system

Effect of Row-Buffer Size

Increase row-buffer size
→ Increase row hits for MPH
→ Exacerbate problem

Decrease row-buffer size
→ Decrease performance of non-interfering high row-buffer locality threads
→ Decrease bandwidth

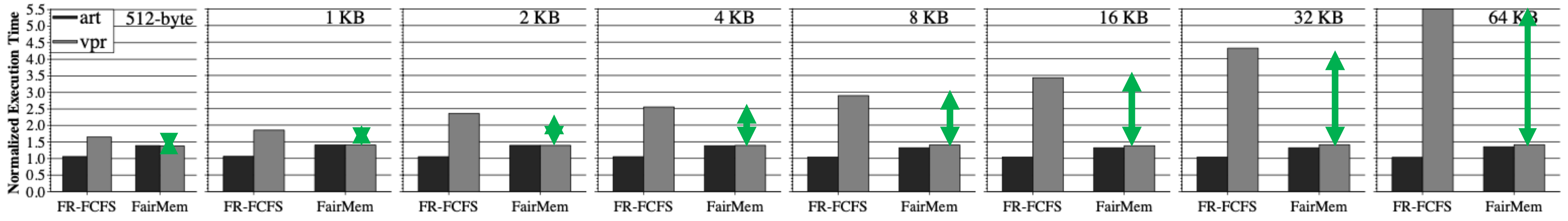


Figure 9: Normalized execution time of *art* and *vpr* when run together on processors with different row-buffer sizes. *Execution time is independently normalized to each machine with different row-buffer size.*

Effect of Number of Banks

- Parallelism proportional to number of banks

→ Less thread conflicts

- Large number of banks expensive

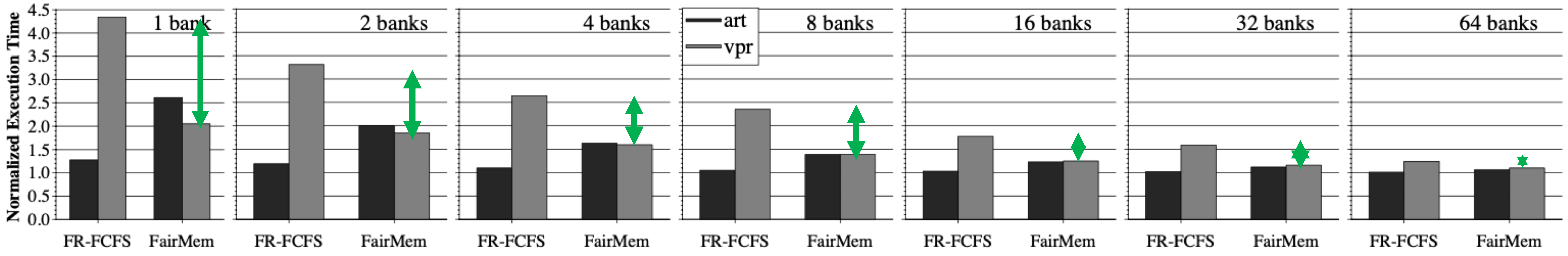
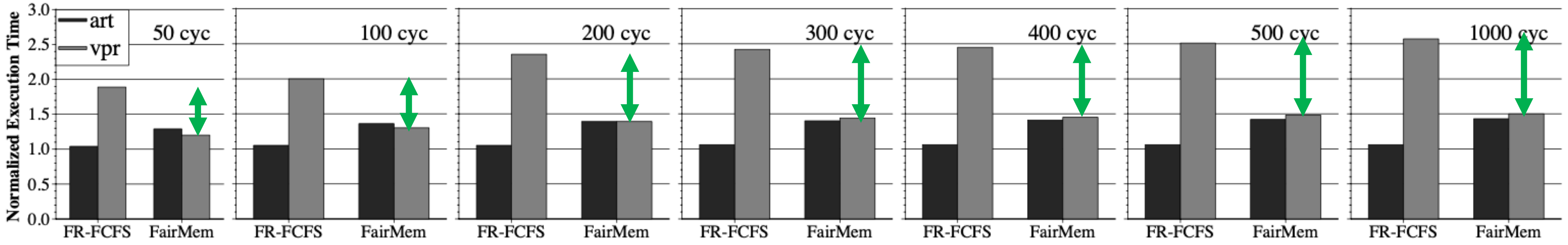


Figure 10: Slowdown of *art* and *vpr* when run together on processors with various number of DRAM banks. *Execution time is independently normalized to each machine with different number of banks.*

Effect of Memory Latency

Increase row hit/ row conflict latency

→ Increases impact of MPH on other threads' performances



FairMem successful in containing the effects of MPHs for various number of cores, number of banks, size of banks and memory latencies!

Structure

- Background
  - Denial of Service
  - DRAM
- **Memory Performance Attacks: Denial of Memory Service in Multi-Core System**
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - **Summary**
- Analysis
  - Strengths
  - Weaknesses
  - Takeaways
  - Further ideas
  - Discussion

Summary

- **Problem:** DRAM memory scheduler denies service to low row-buffer locality threads when MPH present due to prioritizing row-hit requests

- **Goal:** fair memory scheduler equalizing relative performance slow-down

- **Key Ideas:**
  - Maintain experienced latency
  - Simulate latency of running alone
  - Approximate individual threads‘ slowdown, system fairness
  - Prioritize slowed-down threads according to defined threshold

- **Result:** FairMem contains effect of MPH, removes DoS threat

ETH zürich

Structure

Structure

Strengths

- Hard guarantee of containing the effect of MPHs
- Forward-looking; Novel in identifying MPH as DoS threat
- Motivation: tested on “real HW”
- Problem is fundamental, relevant
- Gives good foundation to improve upon
- Flexibility: user defines unfairness threshold α
- Comprehensible: well-structured, thorough background, well-explained

ETH zürich

Structure

Weaknesses

- Might decrease throughput

- HW modification and complexity

→ Additional power consumption

- Fairness provided only when “sufficiently unfair”

- No hard guarantee of staying below threshold α

Structure

- Background
  - Denial of Service
  - DRAM
- Memory Performance Attacks: Denial of Memory Service in Multi-Core System
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - Summary
- **Analysis**
  - Strengths
  - Weaknesses
  - **Takeaways**
  - Further ideas
  - Discussion

Takeaways

- Maximising one variable often comes at cost of another
  - Fairness and throughput
- Novel technology can introduce novel threats
  - Multi-core systems with DRAM memory access scheduler designed for single-core system

Structure

- Background
  - Denial of Service
  - DRAM
- Memory Performance Attacks: Denial of Memory Service in Multi-Core System
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - Summary
- **Analysis**
  - Strengths
  - Weaknesses
  - Takeaways
  - **Further ideas**
  - Discussion

Further Ideas

- Other ways of computing $L_{i\_alone}(T)$?
  - $L_{i\_alone}(T) \coloneqq$ Ideal single-core cumulated latency
  - Simulated in FairMem (simulate row-hit rate)

  $\rightarrow$Approximate interference instead, subtract from latency

  $\rightarrow$Simulate by giving highest priority to a thread *i*

- How to give specific threads more importance?

$\rightarrow$ Include weights in slowdown index

Structure

- Background
  - Denial of Service
  - DRAM
- Memory Performance Attacks: Denial of Memory Service in Multi-Core System
  - Vulnerabilities of current memory scheduling algorithm
  - DRAM fairness
  - Fair memory scheduling model
  - Hardware implementation
  - Results
  - Sensitivity analysis
  - Summary
- **Analysis**
  - Strengths
  - Weaknesses
  - Takeaways
  - Further ideas
  - **Discussion**

Discussion Starters

- When/ why is fairness relevant?
- Is the problem expected to become worse?
- Is per-core DRAM a feasible solution?
  - Achievable through HW vs SW

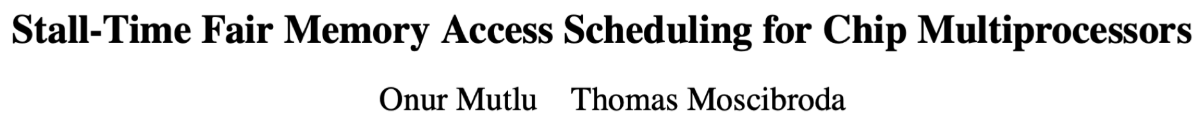
```
If $F(T) \geq \alpha$
    Bank scheduler:
    1. Highest slowdown-index $s_i(T)$
    2. Highest FR-FCFS

    Bus scheduler:
    Highest slowdown-index $s_i(T)$

Else
        $A_{FR-FCFS}$
```

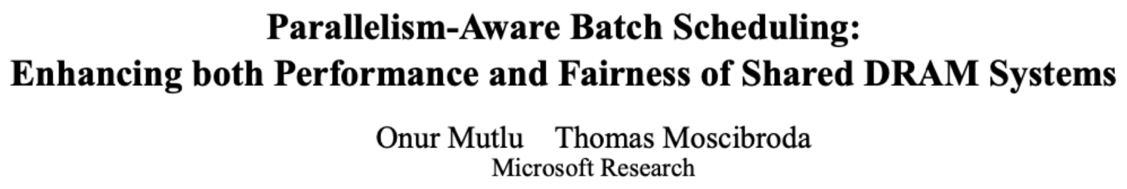
- Could highest-slowdown-index-first in banks and FR-FCFS across banks be enough?
- What might be the advantage/disadvage of a solution that provides fairness from the start?
- Can you recall/ propose a software solution?

Follow Up Work



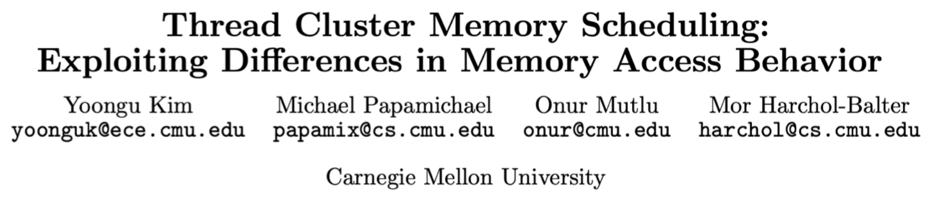
Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors

Onur Mutlu Thomas Moscibroda



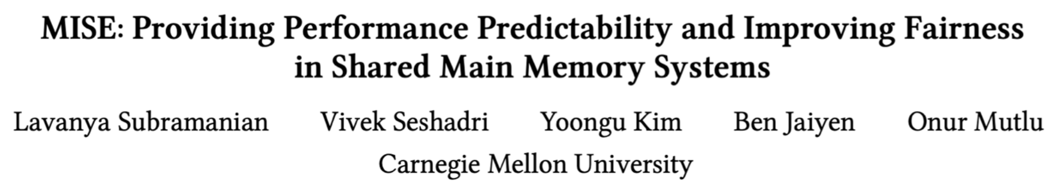
Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems

Onur Mutlu Thomas Moscibroda
Microsoft Research



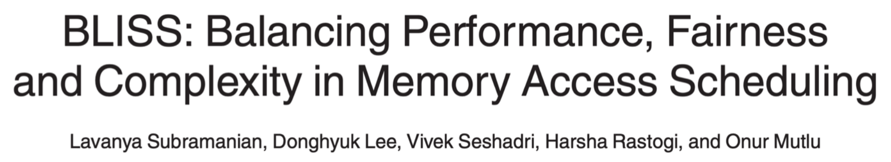
Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior

Yoongu Kim yoonguk@ece.cmu.edu
Michael Papamichael papamix@cs.cmu.edu
Onur Mutlu onur@cmu.edu
Mor Harchol-Balter harchol@cs.cmu.edu
Carnegie Mellon University



MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems

Lavanya Subramanian Vivek Seshadri Yoongu Kim Ben Jaiyen Onur Mutlu
Carnegie Mellon University



BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling

Lavanya Subramanian, Donghyuk Lee, Vivek Seshadri, Harsha Rastogi, and Onur Mutlu

ETH zürich

Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems

Anlin Yan
ETH Zurich

Fall 2019

28 November 2019

Further Ideas

Ratio of experienced slowdown: $\mathrm{s_i} := \frac{L_{i\_shared}}{L_{i\_alone}}$

DRAM fairness index: $F := \frac{\max_{s_i} s_i}{\min_{s_i} s_i}$

- Better quantification of unfairness?

→IPC, for instance, better characterizes a benchmark’s behavior than the total execution time

Further Ideas

- TCM(Thread cluster memory): classify thread into memory-intensive and non-intensive groups
- deprioritized high-memory-intensity applications might be slowed down
- Per group individual scheduling policy

→ low-mem-intensity: ranking based on intensity

→ high: shuffle ranks to provide fairness

ETH zürich

Further Ideas

- ATLAS: observation: low memory service receiving applications experience interference from high memory service receiving applications -> adaptive per-thread least-attained-service memory scheduling, multiple memory schedulers controlling different channels of main memory, schedule: in each time period controllers coordinate to determine a consistent ranking of threads, least serviced in past have highest ranking -> preserve bank-level parallelism

→ thread's requests all serviced from start to finish without other threads' request being serviced? (gives bank-level parallelism and starvation freedom)

Further Ideas: Bank Partitioning

- Combine with software bank partitioning (ameliorate low row-buffer hit ratio and delay of re-ordering due to tasks being mapped to same bank and thus interfering each other): dedicate specific physical pages and thus also specific DRAM bank to each core

→ #DRAM banks grows much slower than #cores

-> across bank, ie bus level contention

(Bounding Memory Interference Delay in COTS-based Multi-core Systems, Kim)

Independent channels -> separate data buses and independent memory controllers (MC)

Further Ideas

- STFM (very similar to proposed FairMem)
- Maintain T_alone by estimating T_interference: T_alone = T_shared – T_interference (heuristics based! Compute T_alone by computing how much it is interefered)
- T_interference
  - DRAM bus interference (otherwise scheduled command (any ready command in request buffer)stalled for t_bus cycles)
  - DRAM bank interference: row-buffer locality interference and waiting for another thread to be serviced (cannot however simply sum up bank interferences, since these are serviced in parallel)

Further Ideas

- PAR-BS(inter-thread interference destroys bank-level access parallelism, overlapped latencies become serialized): form a batch of outstanding requests and prioritize all requests within batch, form ranking within batch based on estimated stall time (finish time), avoid reordering in batch, highest ranking: lowest number of requests to any bank, lowest ranking: highest number of requests to any bank (not parallelisable) –> improve thoughput an bank-level-parallelism

->not scalable due to significant coordination between memory controllers

Optimization problem of achieving high bank-level parallelism and high row hit rate is NP-complete, no efficient algorithmic solutions are expected to exist

Further Ideas

- MISE: providing performance predicatability and improving fairness in shared main memory systems
- Request-service-rate as proxy for performance in memory intensive applications
- Alone-request-service-rate (memory controller gives each thread in round robin highest priority, ie very little interference from other threads)
- MISE for instance much higher accuracy than STFM (when compared to measured worst-case)

Further Ideas

- BLISS(Blacklisting Memory Scheduler): Balancing Performance, Fairness and Complexity in Memory Access Scheduling (L.Subramanian)
- Shortcomings of so far Application-aware memory schedulers: have high H/W complexity (individual total order ranking), total order unfair to low-ranking applications
- Group into vulnerable-to-interference and interference-causing, computed by counting #consecutive requests served from each application

Further Ideas

- Fairness via souce throttling
- Estimate application slowdown due to inter-application interference at cache and memory as ratio of uninterfered to interfered exectuiton itime
- Thread weights to give different thread slowdowns more or less importance, their slowdown more or less tolerable