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

Thomas Moscibroda Onur Mutlu
Microsoft Research

Presented by Florian Ettinger

ETH Zürich
17 October 2018

Problem

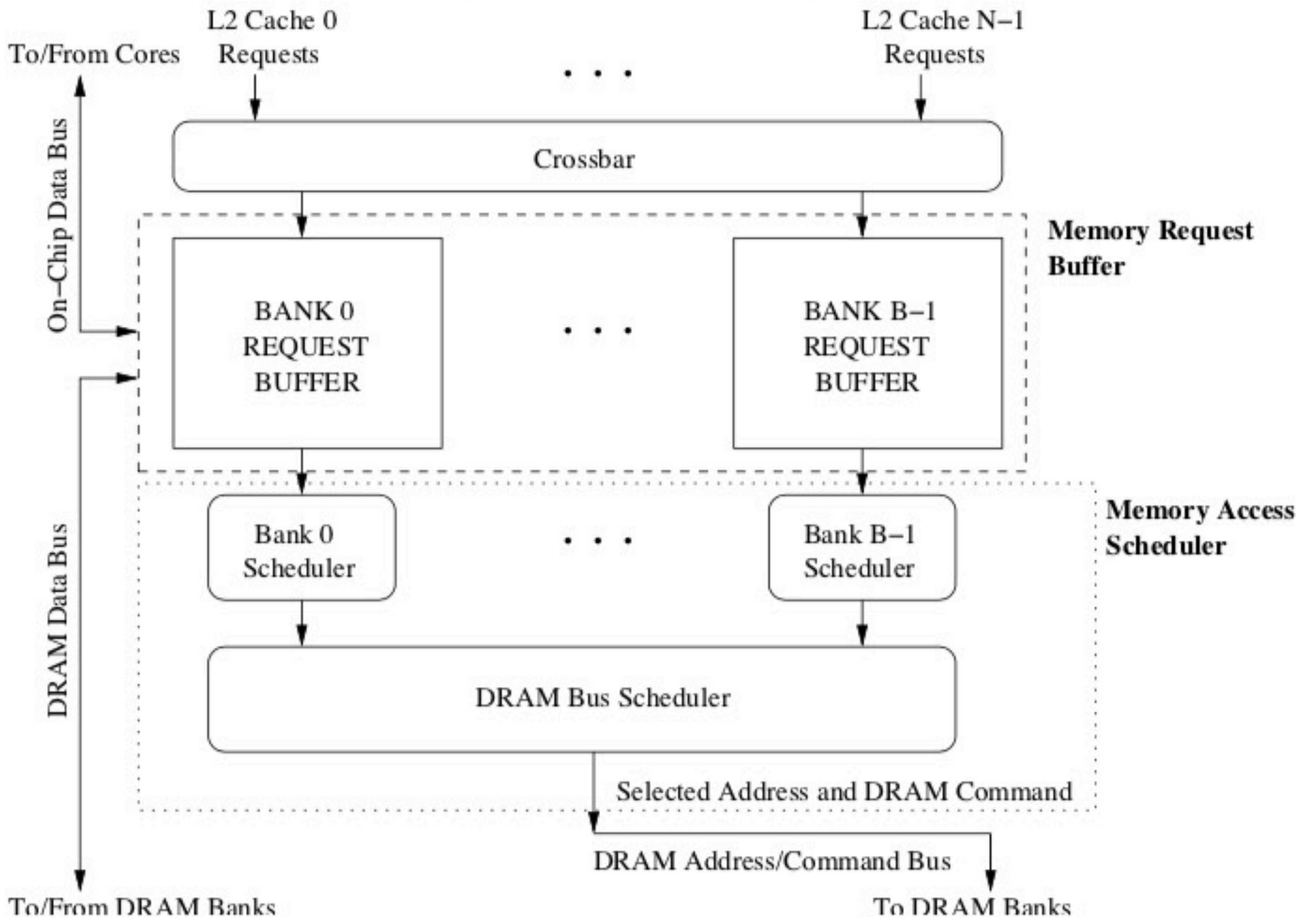
Problem

- The shared DRAM memory system can be used to attack the performance of other programs on a multi-core system

- No efficient way to solve in software
  - OS or other applications have no direct control over the way DRAM requests are scheduled

Background

DRAM controller



Memory Access Scheduling Algorithm

- First-Ready First-Come-First-Serve (FR-FCFS)
  - Bank scheduler
    1. Row-hit-first
    2. Oldest-within-bank-first
  - Across-bank scheduler
    1. Oldest-across-banks-first

- Problems:
  - Row-hit-first scheduling prioritises high row-buffer locality
  - Oldest-first scheduling prioritises threads that generate memory requests at a faster rate

Memory Performance Hog (MPH)

- A program that exploits unfairness in FR-FCFS
  - DoS in a multi-core memory system

- No efficient solution in software to defend against MPH
  - The software has no direct control over memory requests scheduling

- Regular application can unintentionally behave like an MPH
  - A memory-intensive application can cause severe performance degradations for other threads

Example of MPH

- STREAM(MPH):
  - High L2 miss rate
  - High row buffer locality
- RDARRAY:
  - High L2 miss rate
  - How row buffer locality

```
// 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]];
•••
```

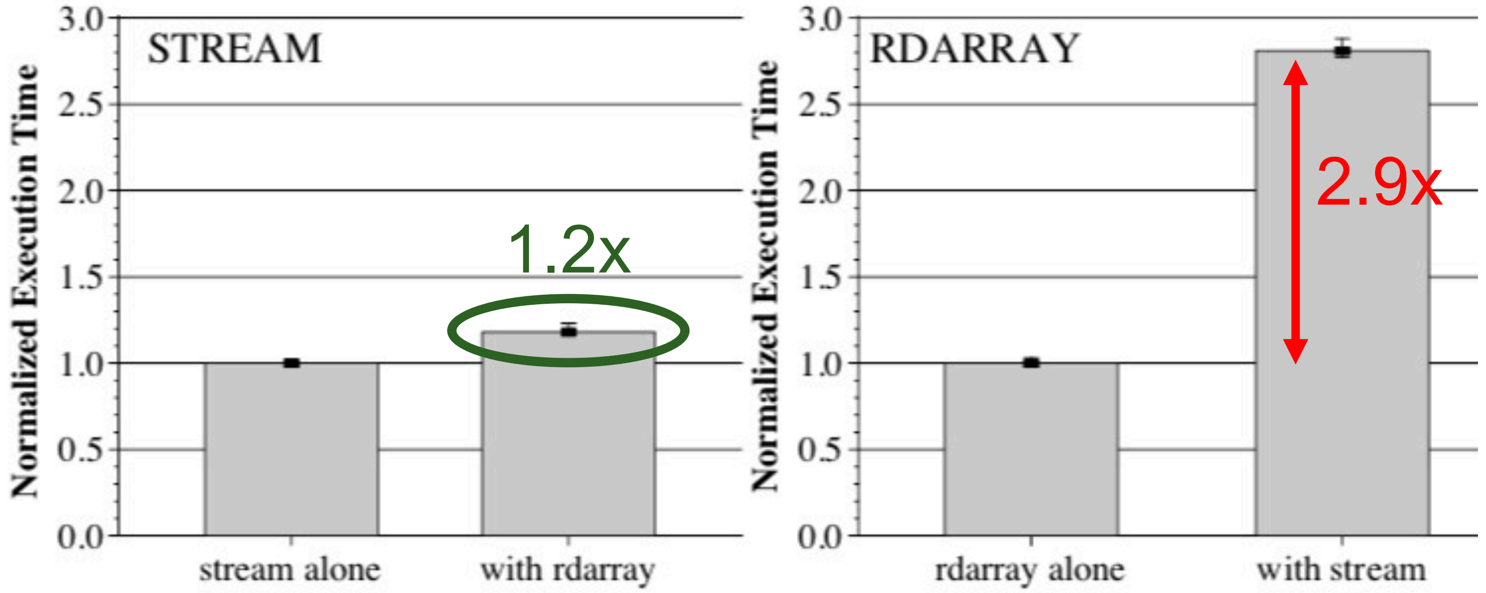
(b) RDARRAY

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%

Example of MPH I

- Running STREAM and RDARRAY together causes
  - Slowdown of RDARRAY by 2.9x
  - Only a slowdown of STREAM by 1.2x

- A result of the row hit first scheduler the bank uses



Goal

Goal

- A new algorithm to schedule memory requests on a multi-core shared DRAM memory system
    - Every thread should have “fair” access to the memory
    - Overall system throughput should not be reduced

Novelty, Key Approach, and Ideas

Approach

- *In a multi-core system with* N *threads, no thread should suffer more relative performance slowdown — compared to the performance it gets if it used the same memory system by itself — than any other thread*

Fairness

- Slowdown index $\chi_i := L_i / \widetilde{L}_i$
  - Captures the price a thread pays because of other threads using the shared memory
  - *Cumulated latency across all banks* L_i
  - *Ideal single core cumulated latency across all banks* $\widetilde{L}_i$

- System fairness $\Psi := \dfrac{\max_i \chi_i}{\min_j \chi_j}$
  - Captures the overall fairness of the system

Thread i, j

Mechanisms

Fair Memory Scheduling Algorithm

- Important considerations
  - How much unfairness is allowed to optimize for throughput?

- FairMem Scheduling Algorithm
  - Bank scheduler
    1. Two candidate requests from each bank
       - Highest FR-FCFS priority
       - Request by threat with highest slowdown index
    2. Fairness-oriented selection
       - If overall system unfairness is greater than the limit use request by threat with highest slowdown index
  - Across-bank scheduler
    1. Highest-DRAM-slowdown-index-first across banks

DRAM changes to enable FairMem

Implementation

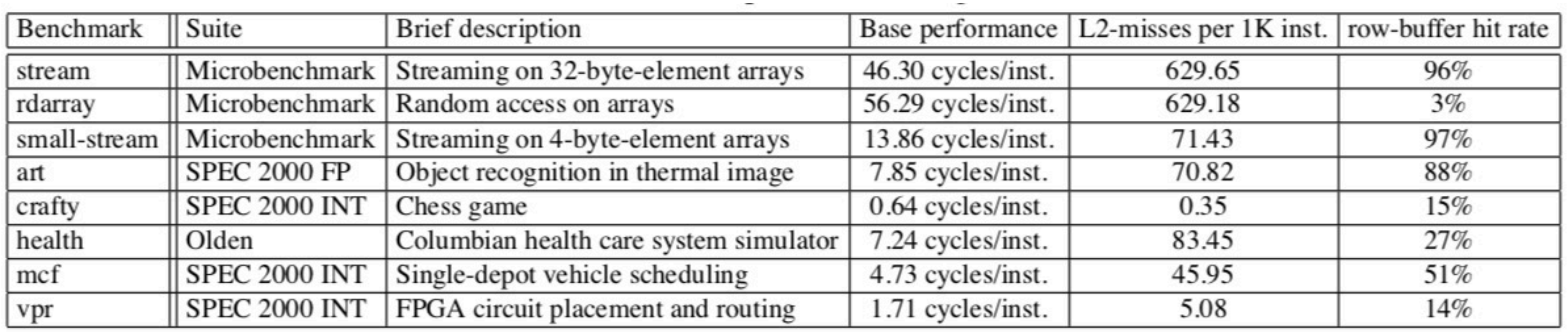
- Calculating L_i
  - For each active thread, a counter maintains the number of memory cycles during which one request is buffered for each bank
- Calculating $\widetilde{L}_i$
  - Simulating an FR-FCFS priority scheme to get ideal latency

- High hardware overhead
  - Reusing dividers and approximating $\widetilde{L}_i$ can reduce overhead

Key Results: Methodology and Evaluation

Methodology

- Simulated dual-core processor and memory system
  - DRAM: 8 banks 2K-byte row-buffer
  - DRAM latency:
    - Row-buffer hit 50ns (200 cycles)
    - Closed 75ns (300 cycles)
    - Conflict 100ns (400 cycles)
- Evaluated applications

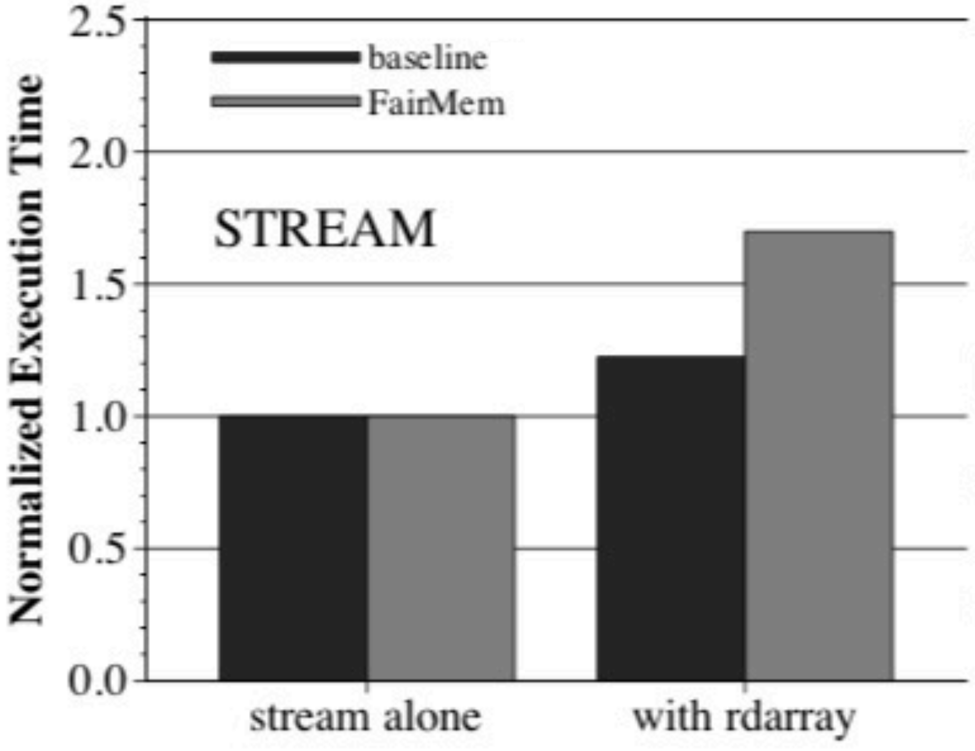


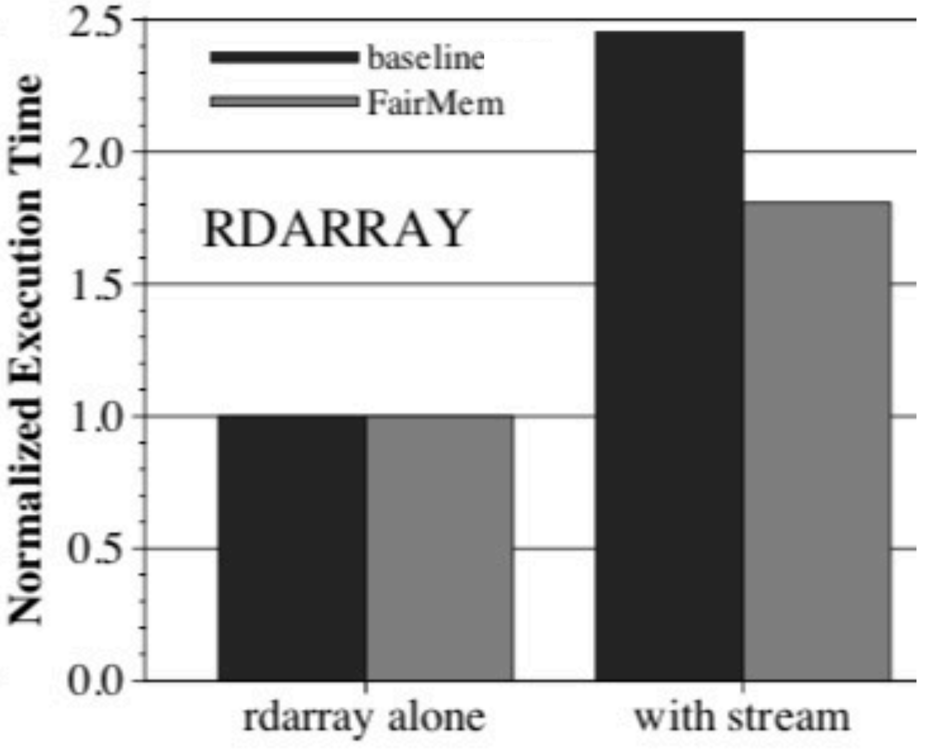
Benchmark	Suite	Brief description	Base performance	L2-misses per 1K inst.	row-buffer hit rate
stream	Microbenchmark	Streaming on 32-byte-element arrays	46.30 cycles/inst.	629.65	96%
rdarray	Microbenchmark	Random access on arrays	56.29 cycles/inst.	629.18	3%
small-stream	Microbenchmark	Streaming on 4-byte-element 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%

- Metrics
  - Execution time
  - Throughput (executed instructions per 1000 cycles)

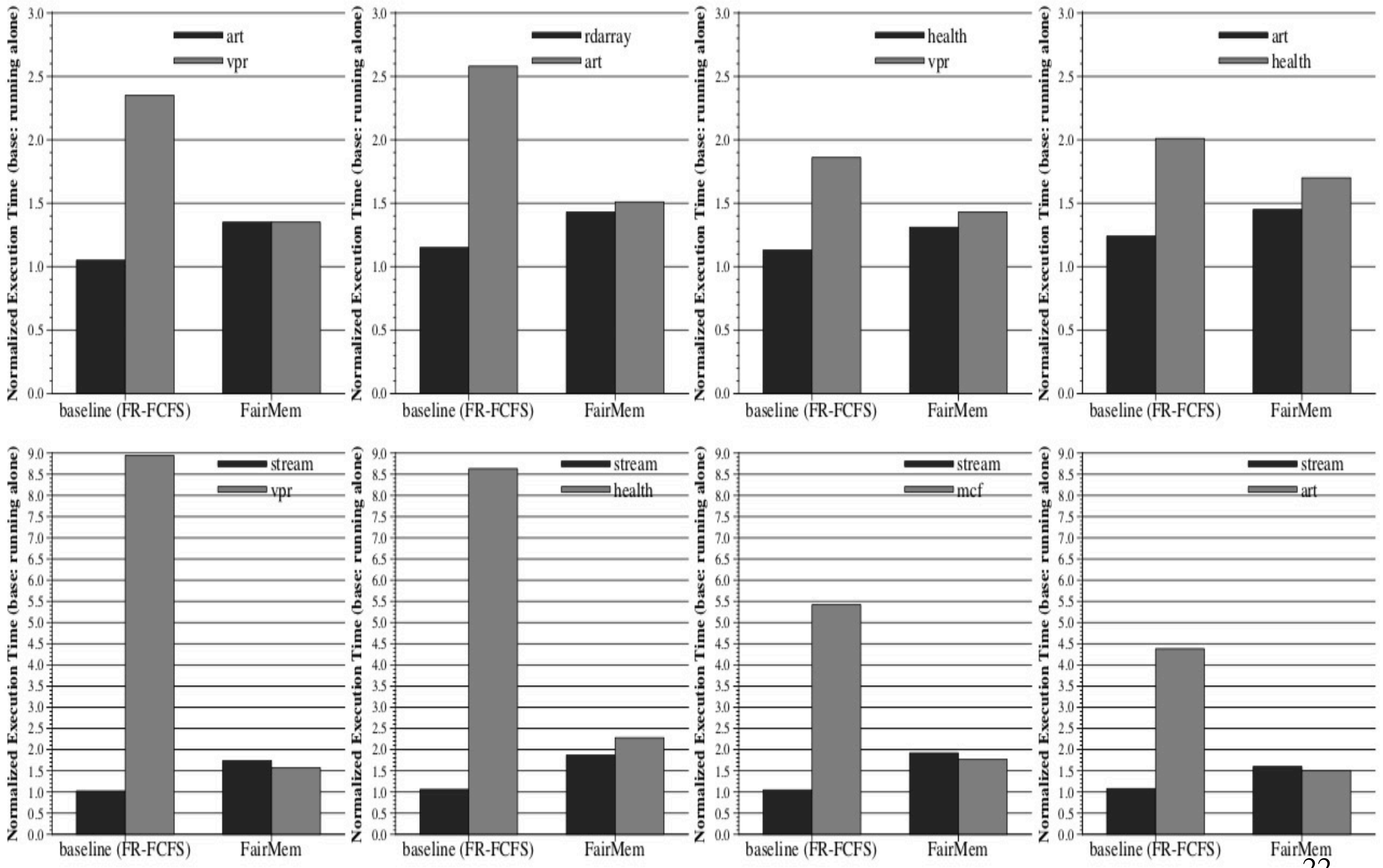
Results

- Baseline(FR-FCFS):
  - *stream* slowdown of 1.22x
  - *rdarray* slowdown of 2.45x
- FairMem:
  - *stream and rdarray slowdown of 1.8x*





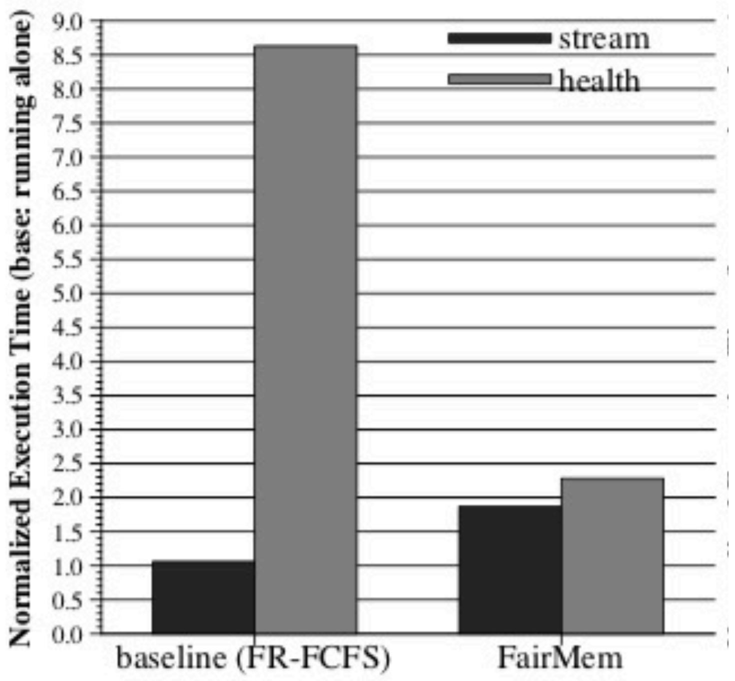
Results

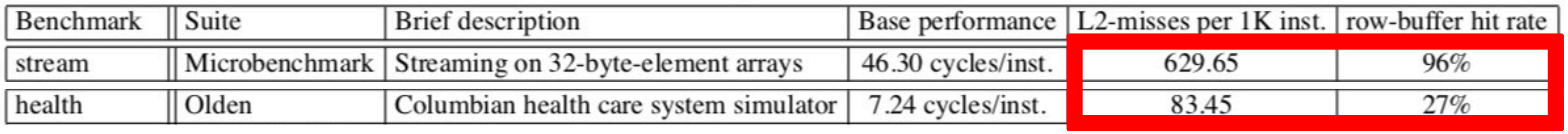


Normalized Execution Time (base: running alone)
art
vpr
rdarray
art
health
vpr
art
health
stream
vpr
stream
health
stream
mcf
stream
art
baseline (FR-FCFS)
FairMem

stream/health

- With FR-FCFS
  - health slowdown of 8.5x
  - stream slowdown of 1.05x
- Inequality due to
  - 7 times higher L2 miss rate
  - High row-buffer hit rate
- FairMem splits slowdown to 2.28x(health) and 1.8x(stream)



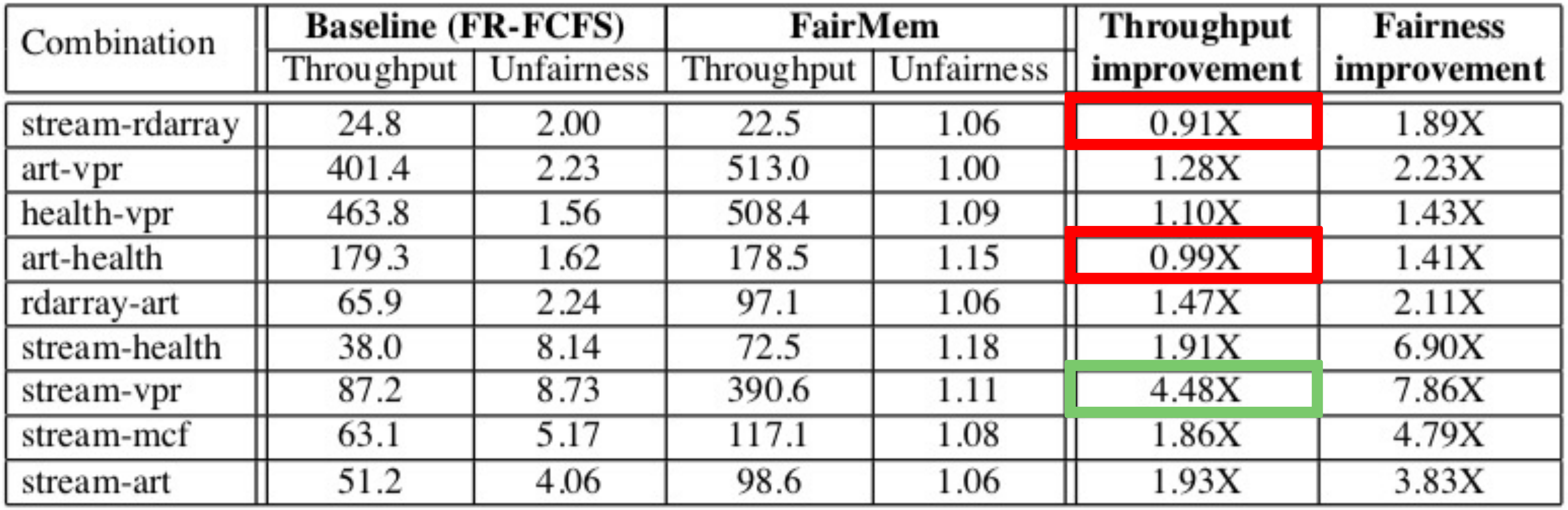


Benchmark	Suite	Brief description	Base performance	L2-misses per 1K inst.	row-buffer hit rate
stream	Microbenchmark	Streaming on 32-byte-element arrays	46.30 cycles/inst.	629.65	96%
health	Olden	Columbian health care system simulator	7.24 cycles/inst.	83.45	27%

Throughput

- Improvement up to 4.4x!

- But throughput reduced up to 9% when two extremely memory-intensive applications run together



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
stream-health	38.0	8.14	72.5	1.18	1.91X	6.90X
stream-vpr	87.2	8.73	390.6	1.11	4.48X	7.86X
stream-mcf	63.1	5.17	117.1	1.08	1.86X	4.79X
stream-art	51.2	4.06	98.6	1.06	1.93X	3.83X

Summary

Summary

- Due to unfairness in the memory system of multi-core architectures, applications can destroy the memory-related performance of other applications

- FairMem
  - Uses a novel definition of fairness in shared memory DRAM to track the level of unfairness and counters it
  - Needs hardware implementation

- Switching to FairMem greatly improves the fairness of shared memory DRAM with only small losses in overall system throughput

Strengths

Strengths

- Early examination of a problem that is still relevant today with the rise of multi-core processors in the last years
- Novel definition of fairness that is easy to understand and can serve as a great basis to further work on
- Sparked a lot of papers further examining the problem
  - E.g. STFM
- Well-written, easy to understand paper

Weaknesses

Weaknesses

- Requires change in hardware by the manufacturer
    - Introduces more overhead
- Slight system throughput decreases for certain workloads
- No direct measure of DRAM possible
    - Only hypothesis of what algorithm is used in DRAM today
- Problem is approached on a high level that leaves low level consideration open
    - No consideration about the scaling of energy consumption when the core count increases

Thoughts and Ideas

Thoughts and Ideas

- Could we incorporate other ideas to help with his problem?
  - E.g. splitting memory intensive threads from low memory intensive thread
- Should we allow a thread to be prioritized in the DRAM memory system to make sure it experiences no delay?
  - Is it possible to combine it with the FairMem algorithm?
- Are there other metrics we could track to reduce the overhead?

Takeaways

Key Takeaways

- Memory performance hogs can exploit the scheduling of DRAM requests to destroy the memory-related performance of other applications
- A security risk that will become more significant with the increased use of multi-core processors
- FairMem can reduce the unfairness of the system and stop this attacks by tracking the slowdown a thread suffers
- Easy to read and understand paper

Questions/Open Discussion

Discussion

- Where can the proposed attack do the most harm?
  - How dangerous is this attack in a real-world scenario?

- Why is this new definition of fairness necessary?
  - Is it possible to share the DRAM memory system in a different way?

- Could we use private DRAM memory for each core?

Additional Slides

Additional papers

- STFM [Onur Mutlu ; Thomas Moscibroda, MICRO 2007]
- ATLAS [Yoongu Kim ; Dongsu Han ; Onur Mutlu ; Mor Harchol-Balter, HPCA 2010]
- TCM [Yoongu Kim ; Michael Papamichael ; Onur Mutlu ; Mor Harchol-Balter, MICRO 2010]