Seminar in Computer Architecture

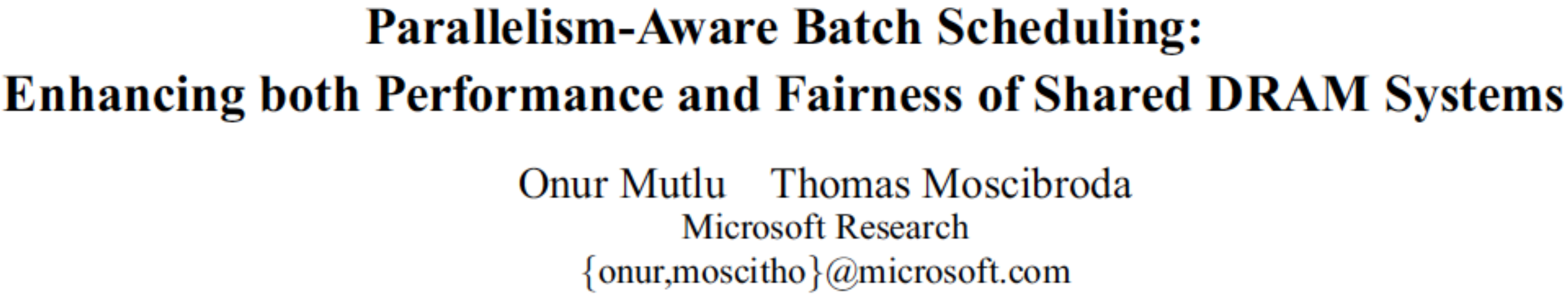
Meeting 4: PAR-BS Memory Scheduler

Prof. Onur Mutlu

ETH Zürich
Fall 2020
8 October 2020

More on PAR-BS

- Onur Mutlu and Thomas Moscibroda, **"Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems"** *Proceedings of the 35th International Symposium on Computer Architecture* (***ISCA***), pages 63-74, Beijing, China, June 2008. [Summary] [Slides (ppt)]
  One of the 12 computer architecture papers of 2008 selected as Top Picks by IEEE Micro.

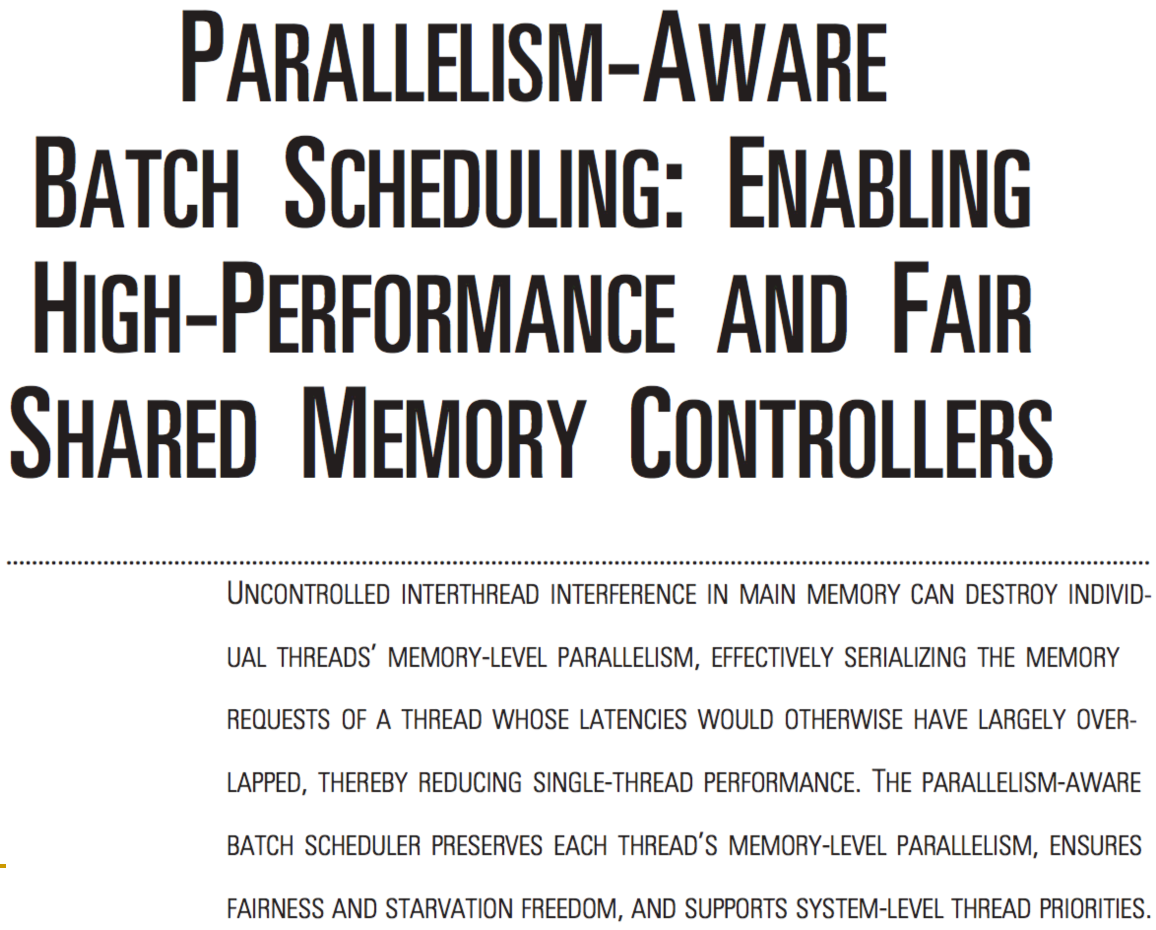


Parallelism-Aware Batch Scheduling:
Enhancing both Performance and Fairness of Shared DRAM Systems

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

More on PAR-BS

- Onur Mutlu and Thomas Moscibroda, **"Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Memory Controllers"** *IEEE Micro, Special Issue: Micro's Top Picks from 2008 Computer Architecture Conferences (**MICRO TOP PICKS**)*, Vol. 29, No. 1, pages 22-32, January/February 2009.



PARALLELISM-AWARE BATCH SCHEDULING: ENABLING HIGH-PERFORMANCE AND FAIR SHARED MEMORY CONTROLLERS

UNCONTROLLED INTERTHREAD INTERFERENCE IN MAIN MEMORY CAN DESTROY INDIVIDUAL THREADS' MEMORY-LEVEL PARALLELISM, EFFECTIVELY SERIALIZING THE MEMORY REQUESTS OF A THREAD WHOSE LATENCIES WOULD OTHERWISE HAVE LARGELY OVERLAPPED, THEREBY REDUCING SINGLE-THREAD PERFORMANCE. THE PARALLELISM-AWARE BATCH SCHEDULER PRESERVES EACH THREAD'S MEMORY-LEVEL PARALLELISM, ENSURES FAIRNESS AND STARVATION FREEDOM, AND SUPPORTS SYSTEM-LEVEL THREAD PRIORITIES.

We Will Do This Differently

- I will give a “conference talk”
- You can ask questions and analyze what I described

Parallelism-Aware Batch Scheduling

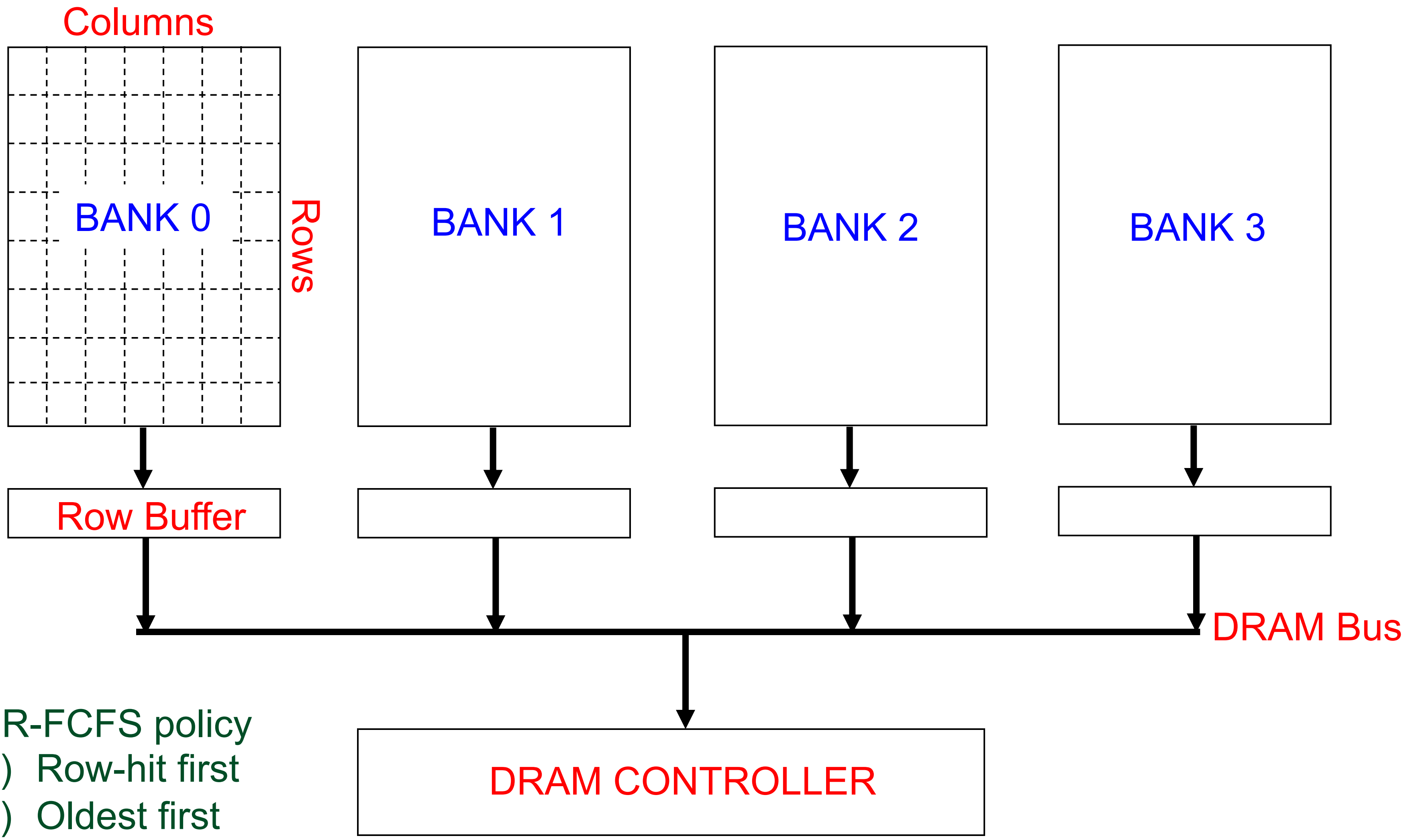
Enhancing both Performance and Fairness of Shared DRAM Systems

Onur Mutlu and Thomas Moscibroda
Computer Architecture Group
Microsoft Research

Outline

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

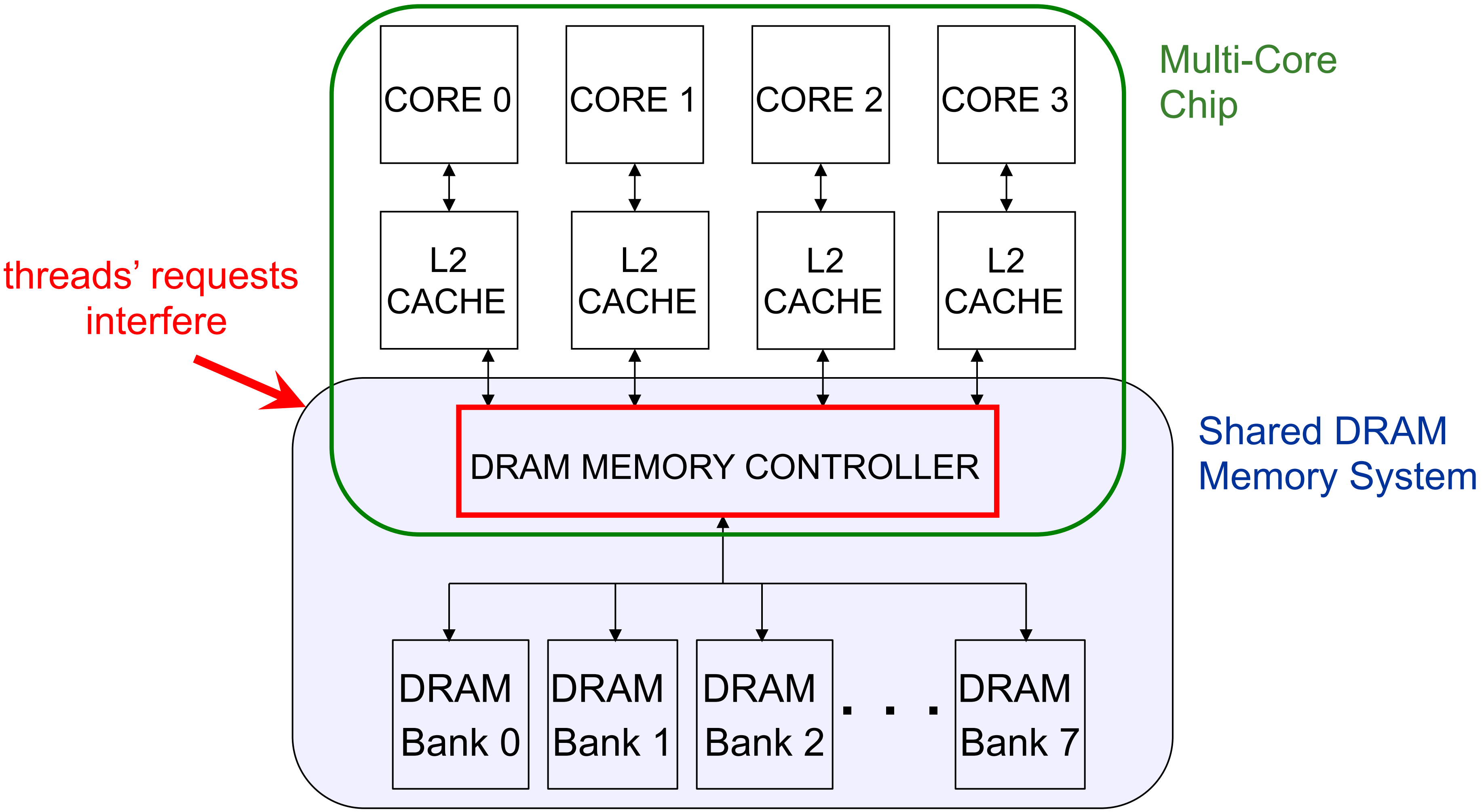
The DRAM System



FR-FCFS policy

1) Row-hit first
2) Oldest first

Multi-Core Systems

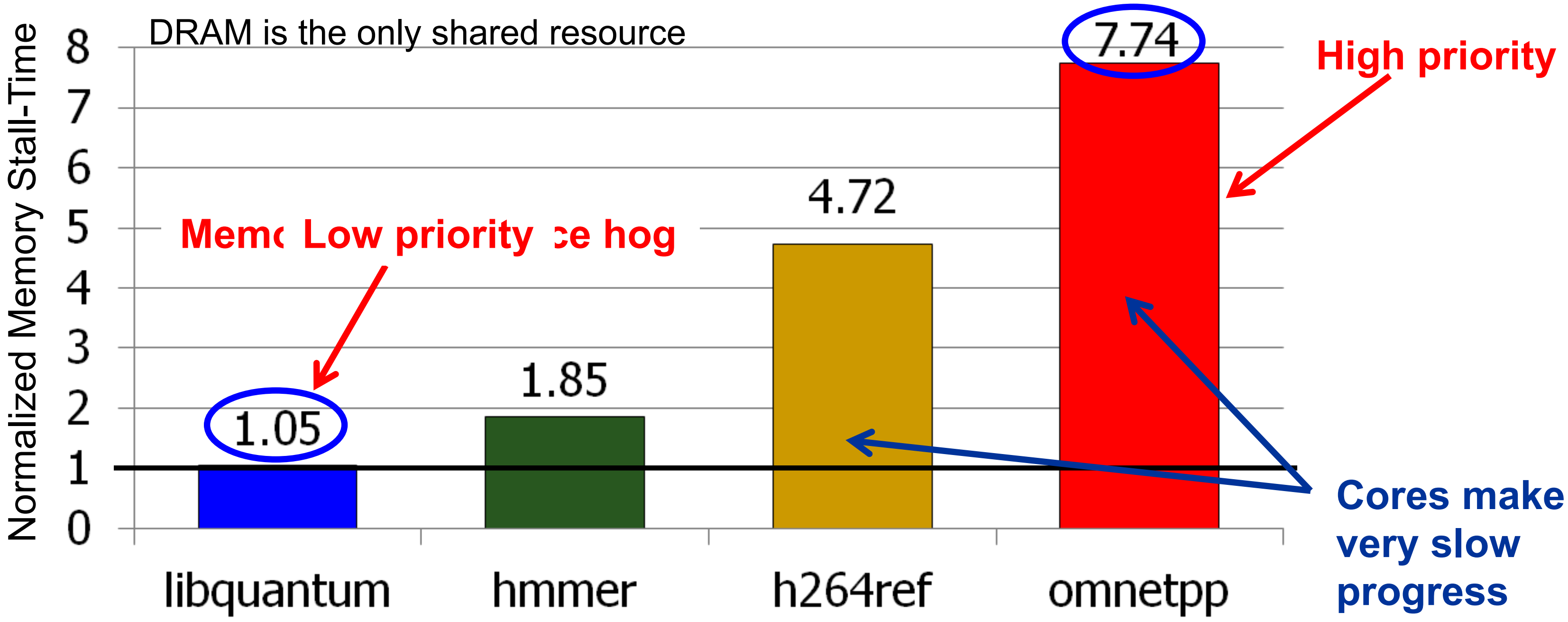


Inter-thread Interference in the DRAM System

- Threads delay each other by causing resource contention:
  - Bank, bus, row-buffer conflicts [MICRO 2007]
- Threads can also destroy each other's DRAM bank parallelism
  - Otherwise parallel requests can become serialized

- Existing DRAM schedulers are unaware of this interference
- They simply aim to maximize DRAM throughput
  - Thread-unaware and thread-unfair
  - No intent to service each thread's requests in parallel
  - FR-FCFS policy: 1) row-hit first, 2) oldest first
    - Unfairly prioritizes threads with high row-buffer locality

Consequences of Inter-Thread Interference in DRAM



Memc Low priority ce hog

High priority

Cores make very slow progress

- Unfair slowdown of different threads [MICRO 2007]
- System performance loss [MICRO 2007]
- Vulnerability to denial of service [USENIX Security 2007]
- Inability to enforce system-level thread priorities [MICRO 2007]

Our Goal

- Control inter-thread interference in DRAM

- Design a shared DRAM scheduler that
  - provides high system performance
    - preserves each thread's DRAM bank parallelism
  - provides fairness to threads sharing the DRAM system
    - equalizes memory-slowdowns of equal-priority threads
  - is controllable and configurable
    - enables different service levels for threads with different priorities

Outline

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

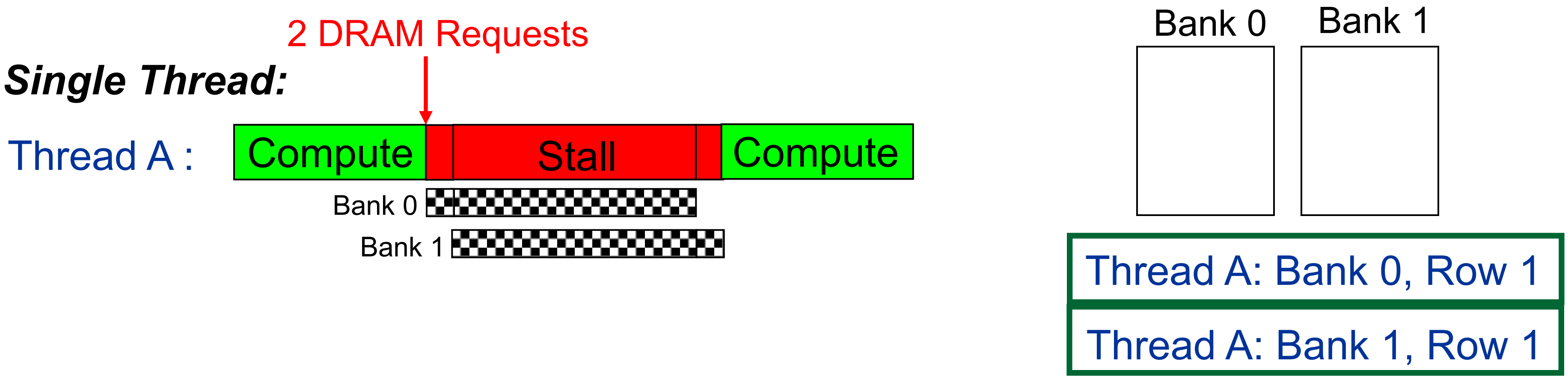
The Problem

- Processors try to tolerate the latency of DRAM requests by generating multiple outstanding requests
  - Memory-Level Parallelism (MLP)
  - Out-of-order execution, non-blocking caches, runahead execution

- Effective only if the DRAM controller actually services the multiple requests in parallel in DRAM banks

- Multiple threads share the DRAM controller
- DRAM controllers are not aware of a thread's MLP
  - Can service each thread's outstanding requests serially, not in parallel

Bank Parallelism of a Thread



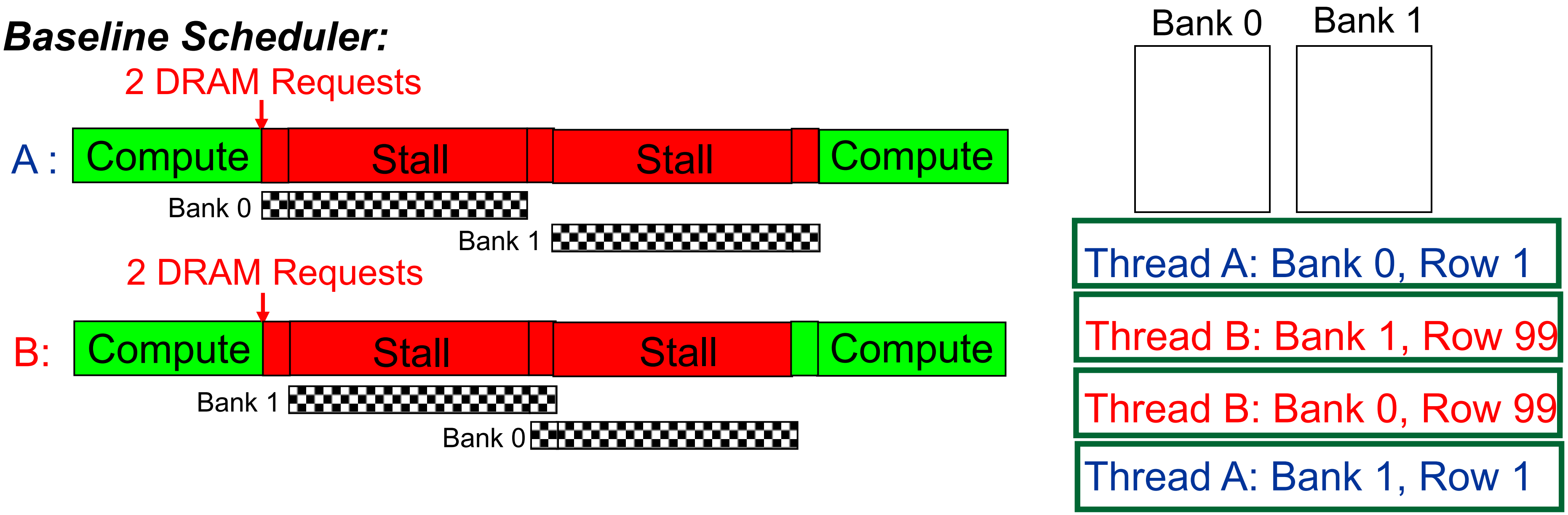
Thread A: Bank 0, Row 1

Thread A: Bank 1, Row 1

Bank access latencies of the two requests overlapped
Thread stalls for ~ONE bank access latency

Bank Parallelism Interference in DRAM

Baseline Scheduler:



2 DRAM Requests

A : Compute | Stall | Stall | Compute

Bank 0

Bank 1

2 DRAM Requests

B: Compute | Stall | Stall | Compute

Bank 1

Bank 0

Bank 0

Bank 1

Thread A: Bank 0, Row 1

Thread B: Bank 1, Row 99

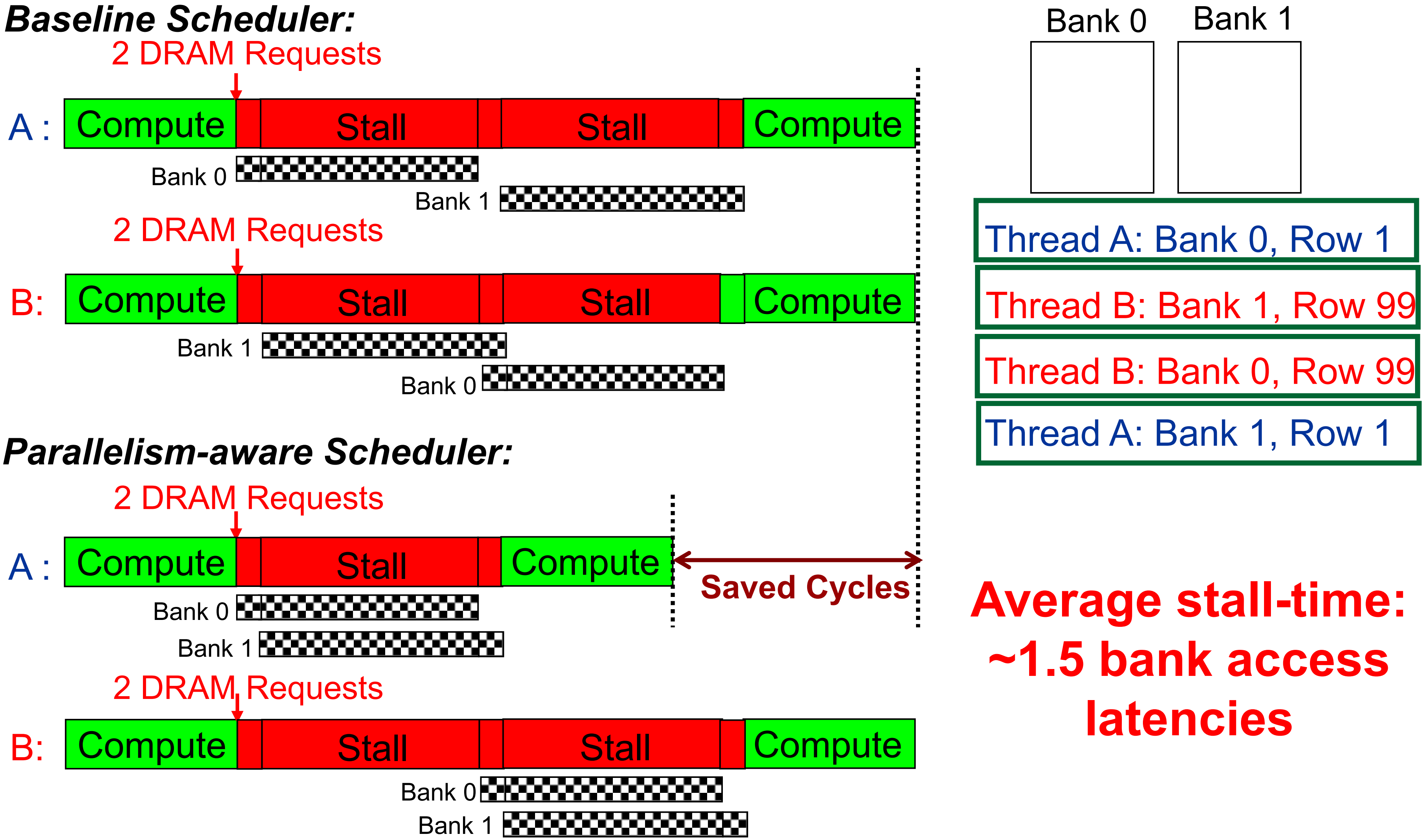
Thread B: Bank 0, Row 99

Thread A: Bank 1, Row 1

Bank access latencies of each thread serialized

Each thread stalls for ~TWO bank access latencies

Parallelism-Aware Scheduler



Baseline Scheduler:

Parallelism-aware Scheduler:

Thread A: Bank 0, Row 1

Thread B: Bank 1, Row 99

Thread B: Bank 0, Row 99

Thread A: Bank 1, Row 1

Average stall-time: ~1.5 bank access latencies

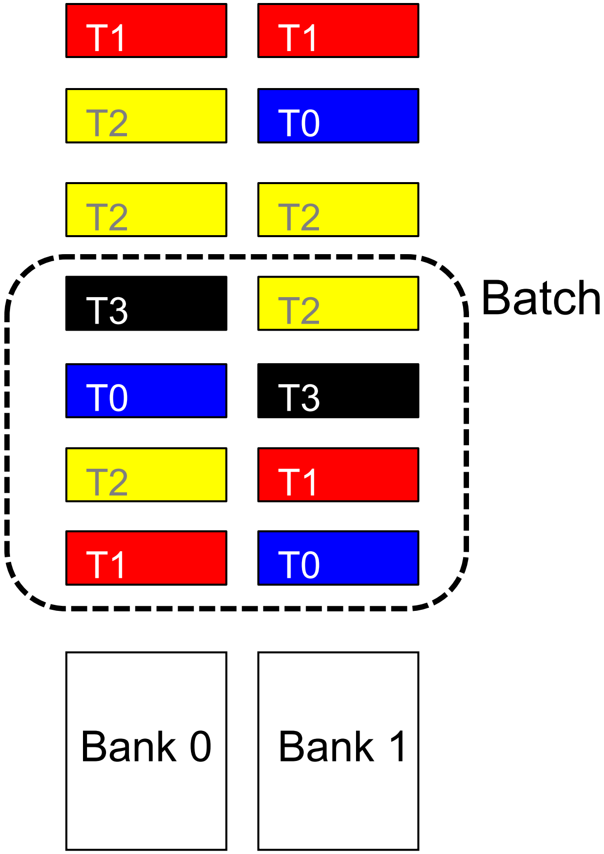
Outline

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling (PAR-BS)
  - Request Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

Parallelism-Aware Batch Scheduling (PAR-BS)

- Principle 1: Parallelism-awareness
  - Schedule requests from a thread (to different banks) back to back
  - Preserves each thread's bank parallelism
  - But, this can cause starvation…

- Principle 2: Request Batching
  - Group a fixed number of oldest requests from each thread into a "batch"
  - Service the batch before all other requests
  - Form a new batch when the current one is done
  - Eliminates starvation, provides fairness
  - Allows parallelism-awareness within a batch



PAR-BS Components

- Request batching

- Within-batch scheduling
  - Parallelism aware

Request Batching

- Each memory request has a bit (*marked)* associated with it

- Batch formation:
  - Mark up to *Marking-Cap* oldest requests per bank for each thread
  - Marked requests constitute the batch
  - Form a new batch when no marked requests are left

- Marked requests are prioritized over unmarked ones
  - No reordering of requests across batches: no starvation, high fairness

- How to prioritize requests within a batch?

Within-Batch Scheduling

- Can use any existing DRAM scheduling policy
  - FR-FCFS (row-hit first, then oldest-first) exploits row-buffer locality
- But, we also want to preserve intra-thread bank parallelism
  - Service each thread's requests back to back

- Scheduler computes a ranking of threads when the batch is formed

  HOW?
  - Higher-ranked threads are prioritized over lower-ranked ones
  - Improves the likelihood that requests from a thread are serviced in parallel by different banks
    - Different threads prioritized in the same order across ALL banks

How to Rank Threads within a Batch

- Ranking scheme affects system throughput and fairness

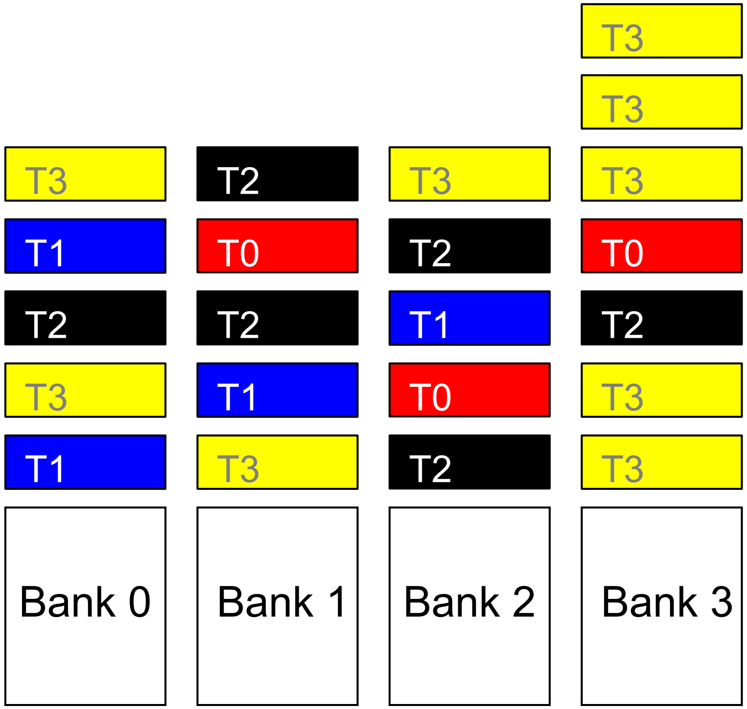
- Maximize system throughput
  - Minimize average stall-time of threads within the batch
- Minimize unfairness (Equalize the slowdown of threads)
  - Service threads with inherently low stall-time early in the batch
  - Insight: delaying memory non-intensive threads results in high slowdown

- Shortest stall-time first (shortest job first) ranking
  - Provides optimal system throughput [Smith, 1956]*
  - Controller estimates each thread’s stall-time within the batch
  - Ranks threads with shorter stall-time higher

* W.E. Smith, “Various optimizers for single stage production,” Naval Research Logistics Quarterly, 1956.

Shortest Stall-Time First Ranking

- Maximum number of marked requests to any bank (max-bank-load)
  - Rank thread with lower max-bank-load higher (~ low stall-time)
- Total number of marked requests (total-load)
  - Breaks ties: rank thread with lower total-load higher



T3
T1
T2
T3
T1
Bank 0

T2
T0
T2
T1
T3
Bank 1

T3
T2
T1
T0
T2
Bank 2

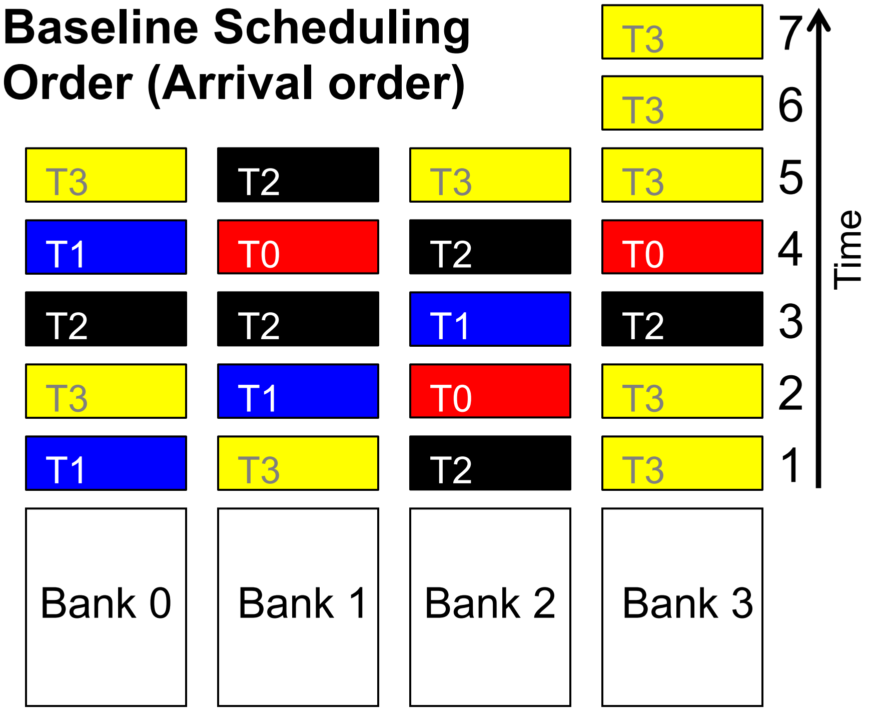
T3
T3
T3
T0
T2
T3
T3
Bank 3

	max-bank-load	total-load

Ranking:
T0 > T1 > T2 > T3

Example Within-Batch Scheduling Order

Baseline Scheduling Order (Arrival order)

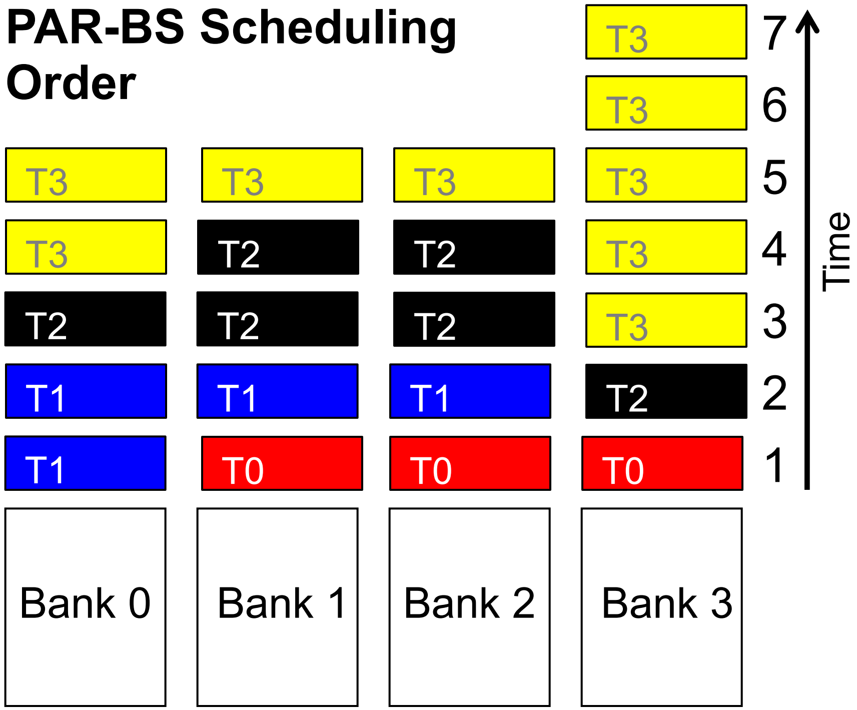


Time	Bank 0	Bank 1	Bank 2	Bank 3
7				T3
6				T3
5	T3	T2	T3	T3
4	T1	T0	T2	T0
3	T2	T2	T1	T2
2	T3	T1	T0	T3
1	T1	T3	T2	T3

	T0	T1	T2	T3
Stall times				

AVG: 5 bank access latencies

PAR-BS Scheduling Order



Time	Bank 0	Bank 1	Bank 2	Bank 3
7				T3
6				T3
5	T3	T3	T3	T3
4	T3	T2	T2	T3
3	T2	T2	T2	T3
2	T1	T1	T1	T2
1	T1	T0	T0	T0

Ranking: T0 > T1 > T2 > T3

	T0	T1	T2	T3
Stall times				

AVG: 3.5 bank access latencies

Putting It Together: PAR-BS Scheduling Policy

- PAR-BS Scheduling Policy

  (1) Marked requests first

  Batching

  (2) Row-hit requests first

  (3) Higher-rank thread first (shortest stall-time first)

  (4) Oldest first

  Parallelism-aware within-batch scheduling

- Three properties:
  - Exploits row-buffer locality **and** intra-thread bank parallelism
  - Work-conserving
    - Services unmarked requests to banks without marked requests
  - Marking-Cap is important
    - Too small cap: destroys row-buffer locality
    - Too large cap: penalizes memory non-intensive threads
- Many more trade-offs analyzed in the paper

Hardware Cost

- <1.5KB storage cost for
  - 8-core system with 128-entry memory request buffer

- No complex operations (e.g., divisions)

- Not on the critical path
  - Scheduler makes a decision only every DRAM cycle

Outline

- Background and Goal
- Motivation
    - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
    - Batching
    - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

System Software Support

- OS conveys each thread's priority level to the controller
  - Levels 1, 2, 3, … (highest to lowest priority)
- Controller enforces priorities in two ways
  - Mark requests from a thread with priority X only every Xth batch
  - Within a batch, higher-priority threads' requests are scheduled first

- Purely opportunistic service
  - Special very low priority level L
  - Requests from such threads never marked

- Quantitative analysis in paper

Outline

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

Evaluation Methodology

- 4-, 8-, 16-core systems
  - x86 processor model based on Intel Pentium M
  - 4 GHz processor, 128-entry instruction window
  - 512 Kbyte per core private L2 caches, 32 L2 miss buffers

- Detailed DRAM model based on Micron DDR2-800
  - 128-entry memory request buffer
  - 8 banks, 2Kbyte row buffer
  - 40ns (160 cycles) row-hit round-trip latency
  - 80ns (320 cycles) row-conflict round-trip latency

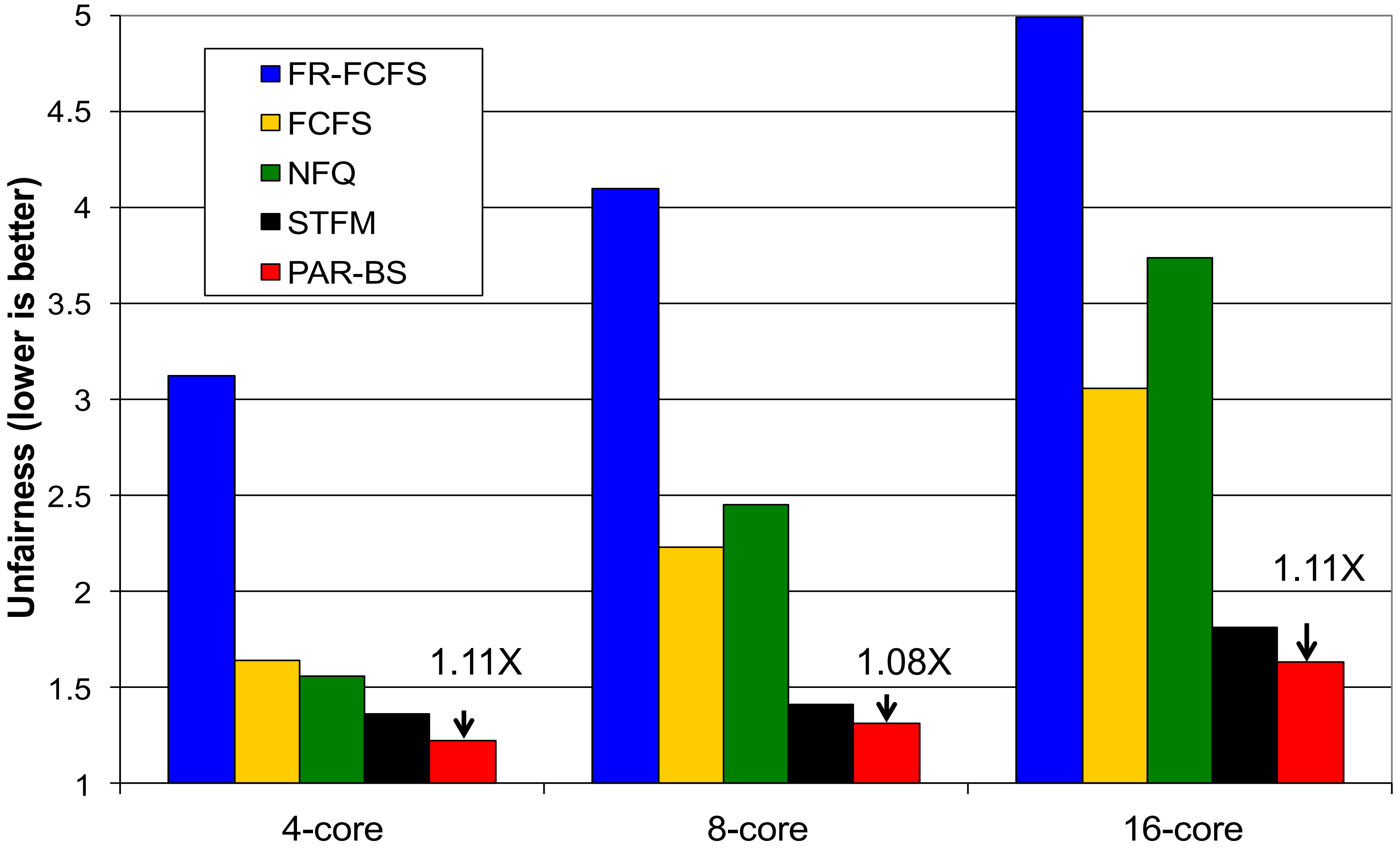
- Benchmarks
  - Multiprogrammed SPEC CPU2006 and Windows Desktop applications
  - 100, 16, 12 program combinations for 4-, 8-, 16-core experiments

Comparison with Other DRAM Controllers

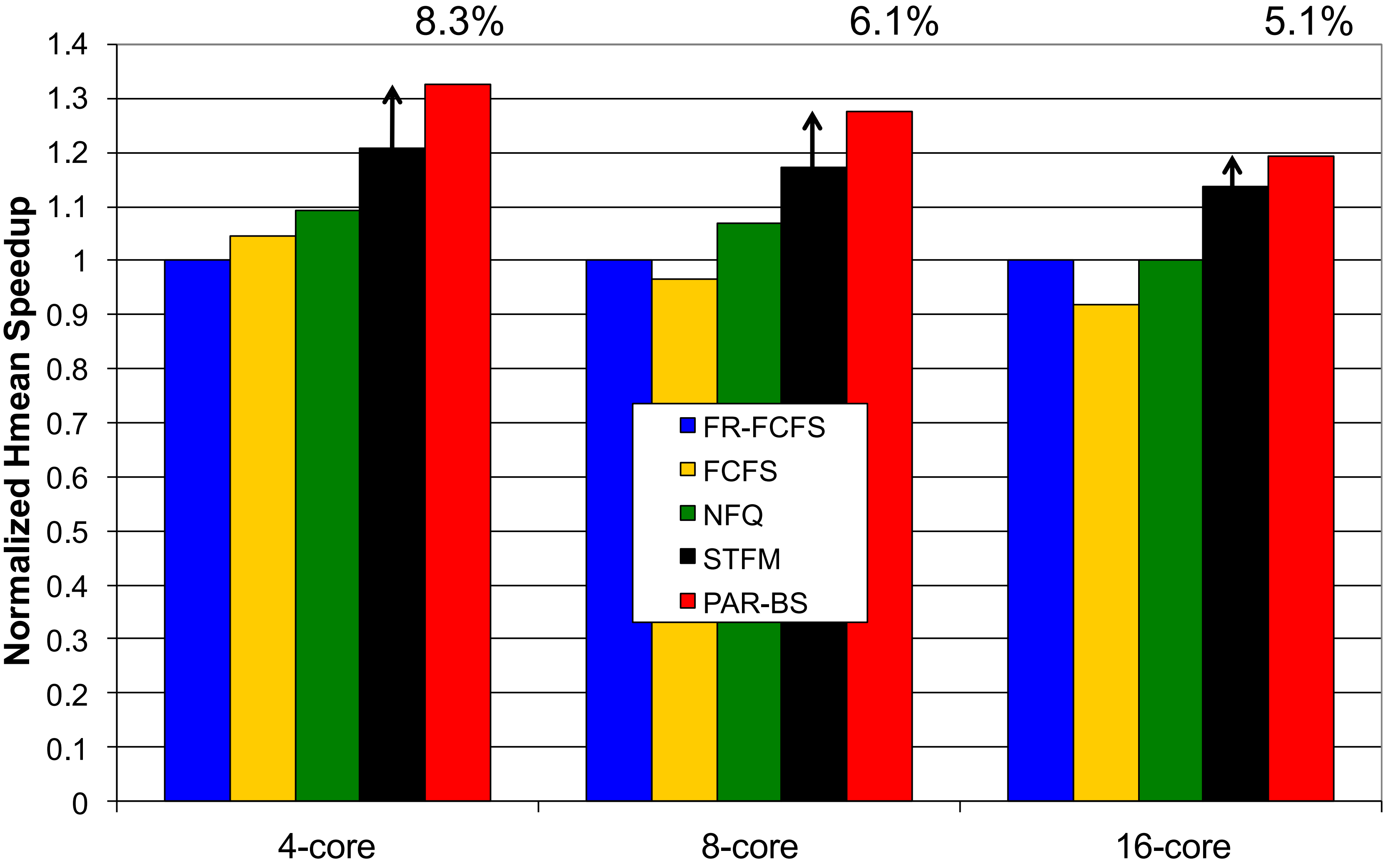
- Baseline FR-FCFS [Zuravleff and Robinson, US Patent 1997; Rixner et al., ISCA 2000]
  - Prioritizes row-hit requests, older requests
  - Unfairly penalizes threads with low row-buffer locality, memory non-intensive threads
- FCFS [Intel Pentium 4 chipsets]
  - Oldest-first; low DRAM throughput
  - Unfairly penalizes memory non-intensive threads
- Network Fair Queueing (NFQ) [Nesbit et al., MICRO 2006]
  - Equally partitions DRAM bandwidth among threads
  - Does not consider inherent (baseline) DRAM performance of each thread
  - Unfairly penalizes threads with high bandwidth utilization [MICRO 2007]
  - Unfairly prioritizes threads with bursty access patterns [MICRO 2007]
- Stall-Time Fair Memory Scheduler (STFM) [Mutlu & Moscibroda, MICRO 2007]
  - Estimates and balances thread slowdowns relative to when run alone
  - Unfairly treats threads with inaccurate slowdown estimates
  - Requires multiple (approximate) arithmetic operations

Unfairness on 4-, 8-, 16-core Systems

Unfairness = MAX Memory Slowdown / MIN Memory Slowdown [MICRO 2007]



System Performance (Hmean-speedup)



Outline

- Background and Goal
- Motivation
  - Destruction of Intra-thread DRAM Bank Parallelism
- Parallelism-Aware Batch Scheduling
  - Batching
  - Within-batch Scheduling
- System Software Support
- Evaluation
- Summary

Summary

- Inter-thread interference can destroy each thread's DRAM bank parallelism
  - Serializes a thread's requests → reduces system throughput
  - Makes techniques that exploit memory-level parallelism less effective
  - Existing DRAM controllers unaware of intra-thread bank parallelism
- A new approach to fair and high-performance DRAM scheduling
  - Batching: Eliminates starvation, allows fair sharing of the DRAM system
  - Parallelism-aware thread ranking: Preserves each thread's bank parallelism
  - Flexible and configurable: Supports system-level thread priorities → QoS policies
- PAR-BS provides better fairness and system performance than previous DRAM schedulers

Thank you. Questions?

Parallelism-Aware Batch Scheduling

Enhancing both Performance and Fairness of Shared DRAM Systems

Onur Mutlu and Thomas Moscibroda

Computer Architecture Group

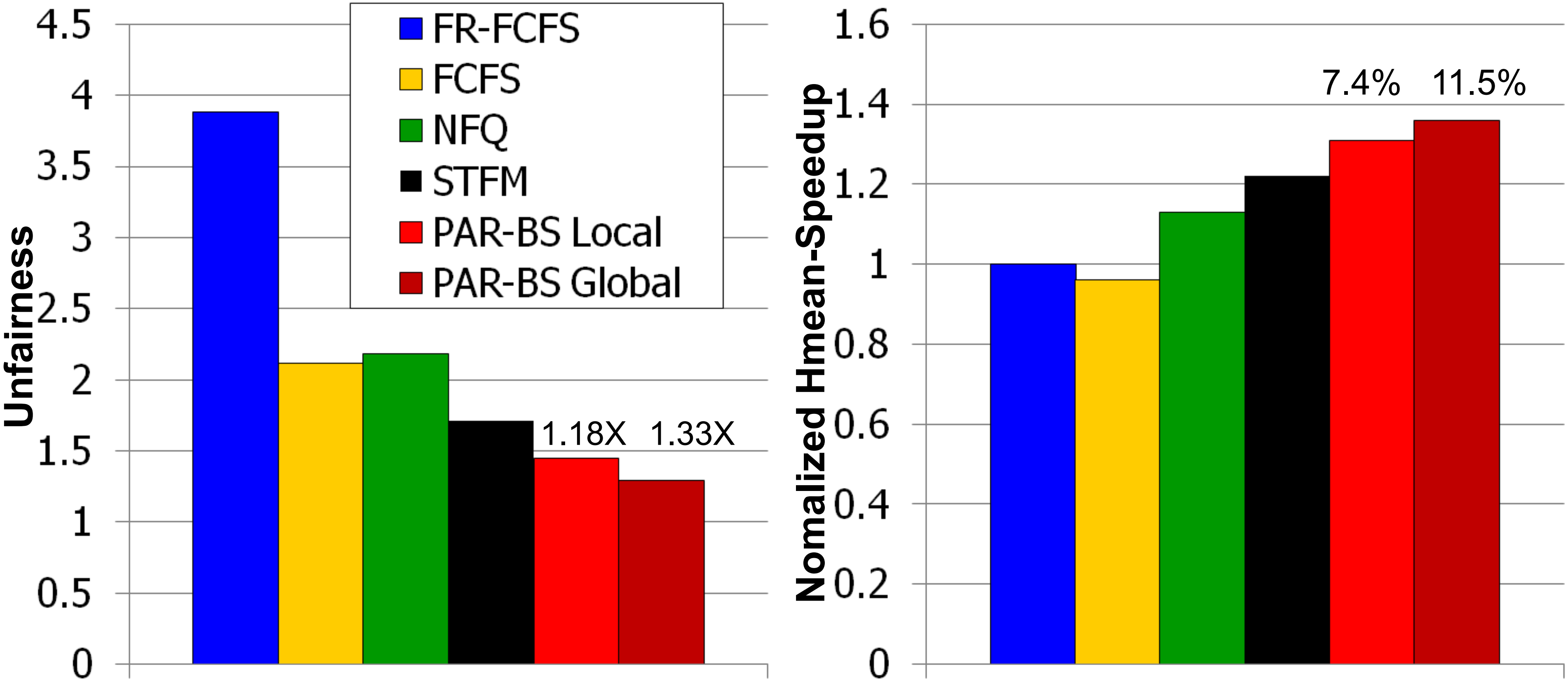
Microsoft Research

Backup Slides

Multiple Memory Controllers (I)

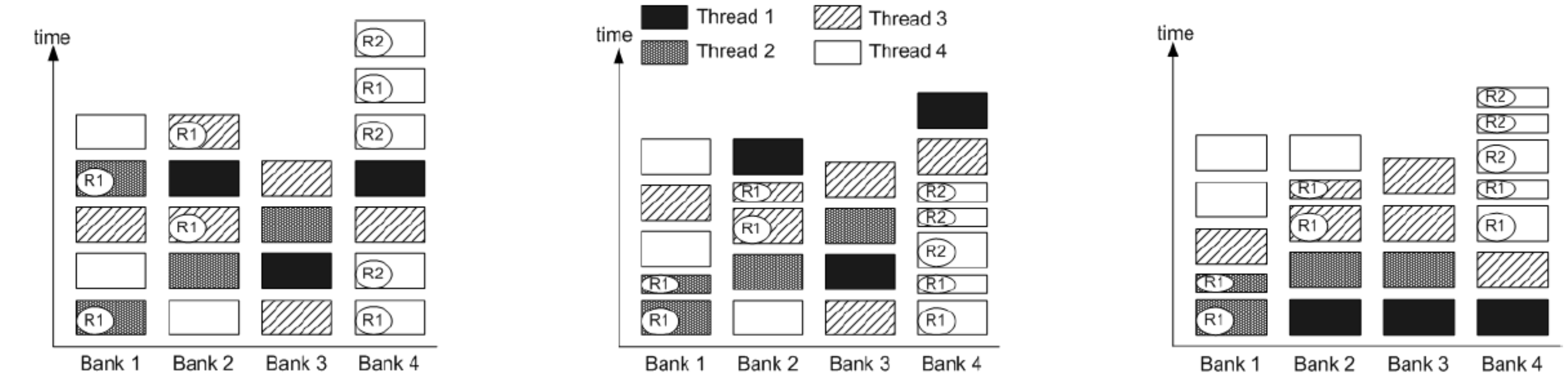
- Local ranking: Each controller uses PAR-BS independently
  - Computes its own ranking based on its local requests
- Global ranking: Meta controller that computes a global ranking across all controllers based on global information
  - Only needs to track bookkeeping info about each thread's requests to the banks in each controller
- The difference between the ranking computed by each scheme depends on the balance of the distribution of requests to each controller
  - Balanced → Local and global rankings are similar

Multiple Memory Controllers (II)



16-core system, 4 memory controllers

Example with Row Hits



(a) Arrival order (and FCFS schedule)

(b) FR-FCFS schedule

(c) PAR-BS schedule

	Stall time		Stall time		Stall time
Thread 1	4	Thread 1	5.5	Thread 1	1
Thread 2	4	Thread 2	3	Thread 2	2
Thread 3	5	Thread 3	4.5	Thread 3	4
Thread 4	7	Thread 4	4.5	Thread 4	5.5
AVG	5	AVG	4.375	AVG	3.125

End of Backup Slides

Now Your Turn to Analyze…

- Background, Problem & Goal
- Novelty
- Key Approach and Ideas
- Mechanisms (in some detail)
- Key Results: Methodology and Evaluation
- Summary
- Strengths
- Weaknesses
- Thoughts and Ideas
- Takeaways
- Open Discussion

PAR-BS Pros and Cons

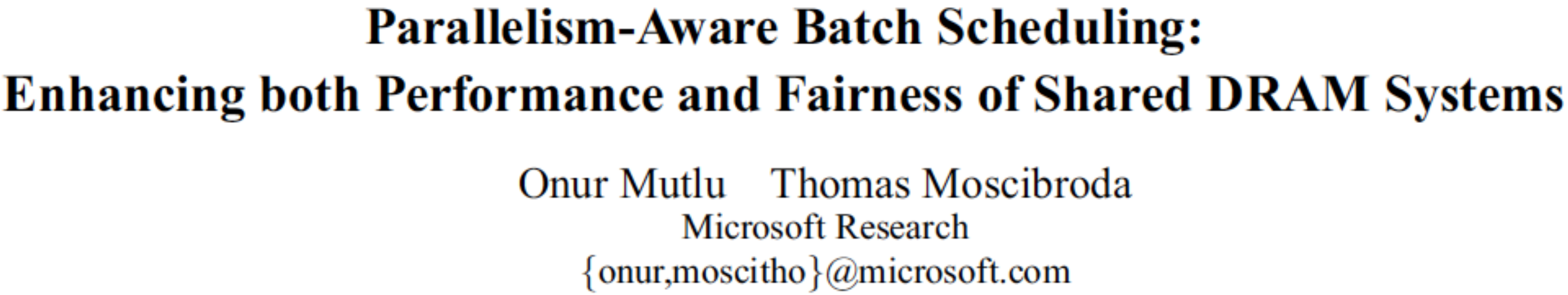
- Upsides:
  - First scheduler to address bank parallelism destruction across multiple threads
  - Simple mechanism (vs. STFM)
  - Batching provides fairness
  - Ranking enables parallelism awareness

- Downsides:
  - Does not always prioritize the latency-sensitive applications
  - Deadline guarantees?
  - Complexity?

- Some ideas implemented in real SoC memory controllers

More on PAR-BS

- Onur Mutlu and Thomas Moscibroda, **"Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems"** *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 63-74, Beijing, China, June 2008. [Summary] [Slides (ppt)]
One of the 12 computer architecture papers of 2008 selected as Top Picks by IEEE Micro.

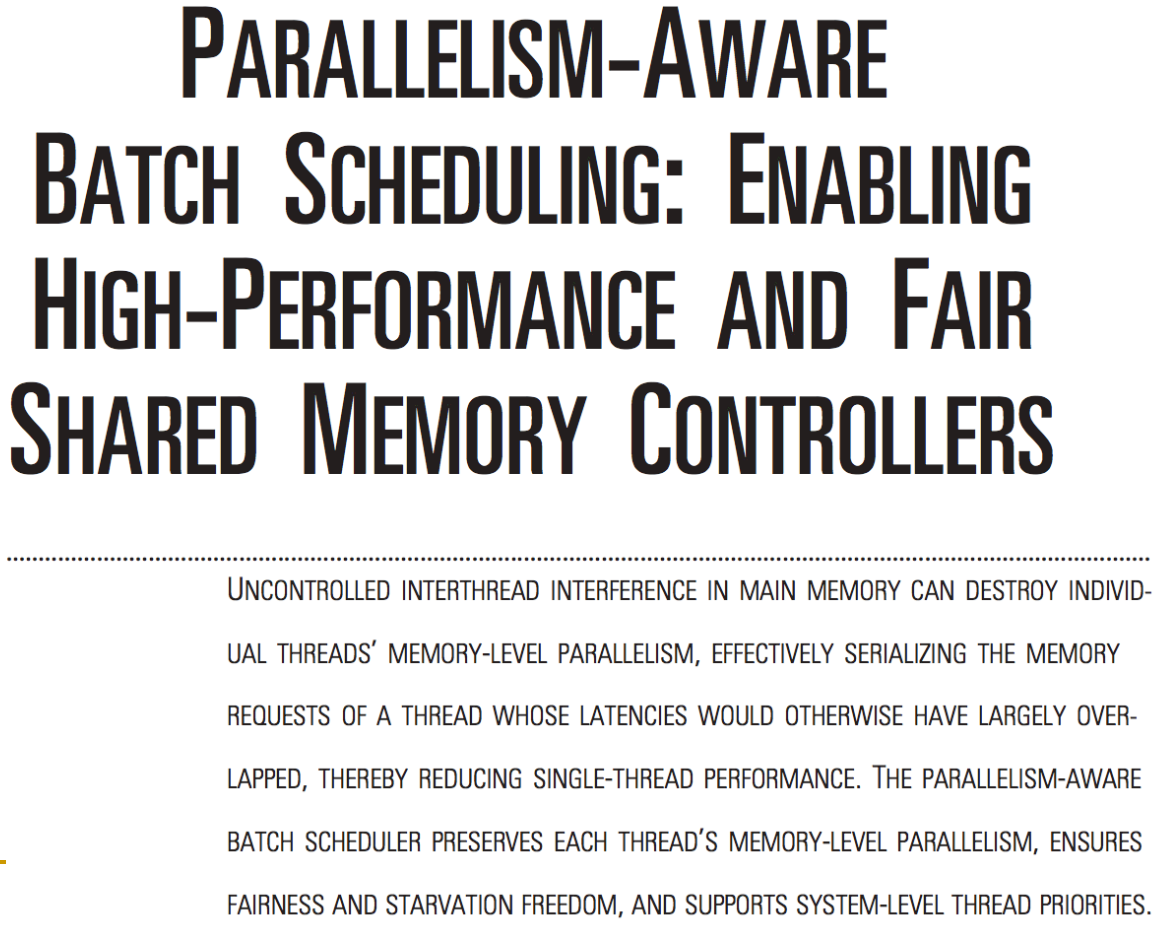


Parallelism-Aware Batch Scheduling:
Enhancing both Performance and Fairness of Shared DRAM Systems

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

More on PAR-BS

- Onur Mutlu and Thomas Moscibroda, **"Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Memory Controllers"** *IEEE Micro, Special Issue: Micro's Top Picks from 2008 Computer Architecture Conferences (**MICRO TOP PICKS**)*, Vol. 29, No. 1, pages 22-32, January/February 2009.



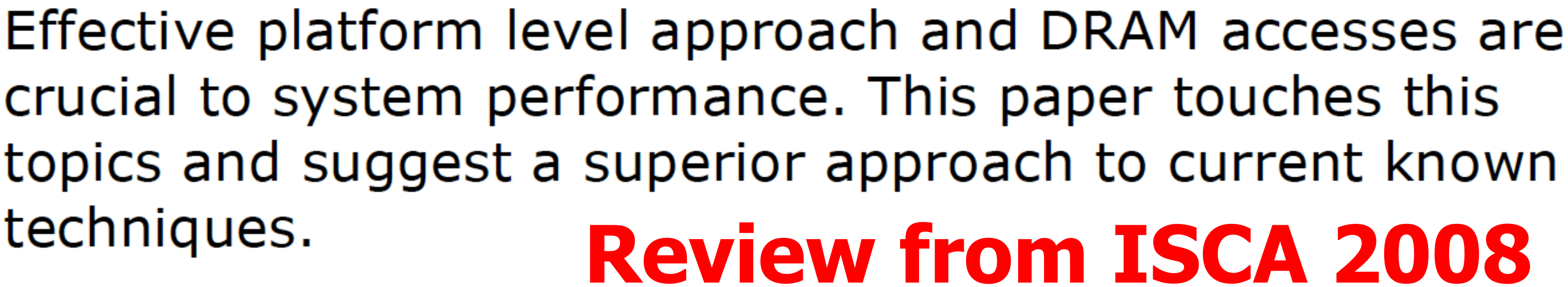
PARALLELISM-AWARE BATCH SCHEDULING: ENABLING HIGH-PERFORMANCE AND FAIR SHARED MEMORY CONTROLLERS

UNCONTROLLED INTERTHREAD INTERFERENCE IN MAIN MEMORY CAN DESTROY INDIVIDUAL THREADS' MEMORY-LEVEL PARALLELISM, EFFECTIVELY SERIALIZING THE MEMORY REQUESTS OF A THREAD WHOSE LATENCIES WOULD OTHERWISE HAVE LARGELY OVERLAPPED, THEREBY REDUCING SINGLE-THREAD PERFORMANCE. THE PARALLELISM-AWARE BATCH SCHEDULER PRESERVES EACH THREAD'S MEMORY-LEVEL PARALLELISM, ENSURES FAIRNESS AND STARVATION FREEDOM, AND SUPPORTS SYSTEM-LEVEL THREAD PRIORITIES.

Some History

On PAR-BS

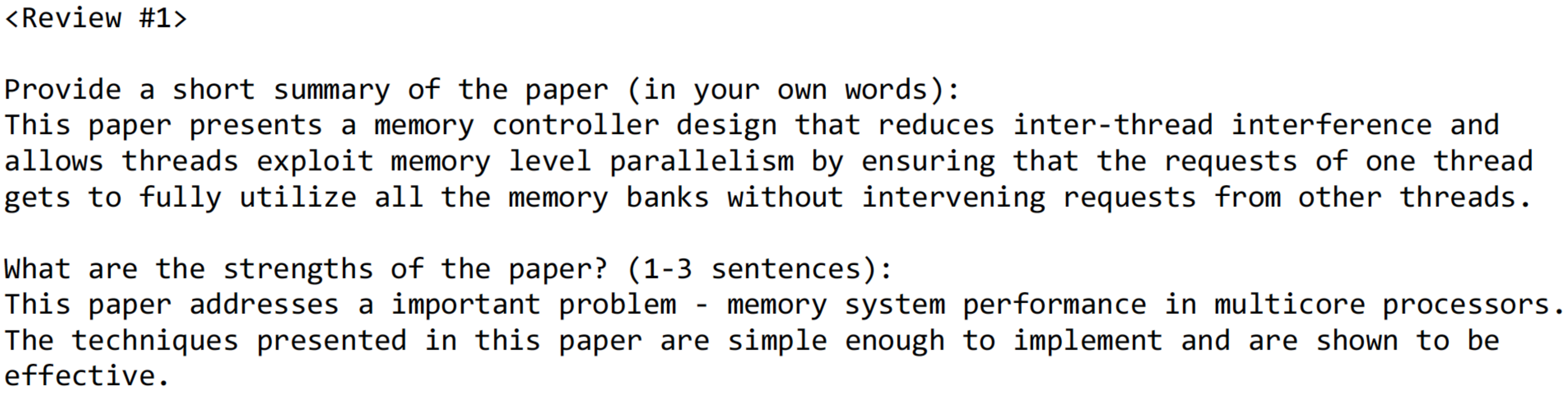
- Variants implemented in Samsung SoC memory controllers



Effective platform level approach and DRAM accesses are crucial to system performance. This paper touches this topics and suggest a superior approach to current known techniques.

Review from ISCA 2008

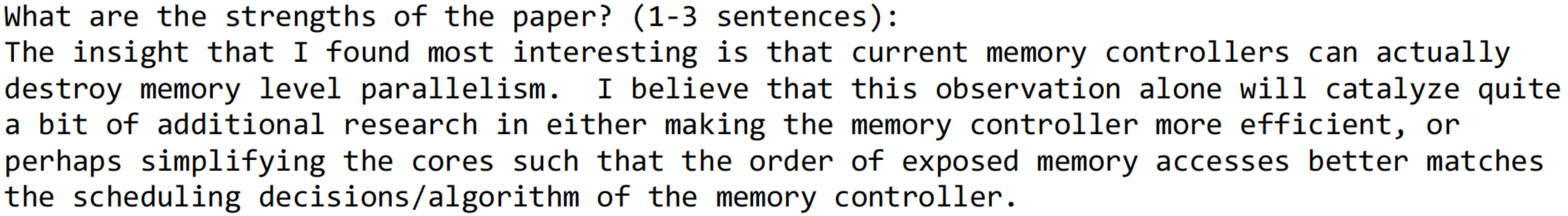
Top Picks 2009 Review #1



<Review #1>

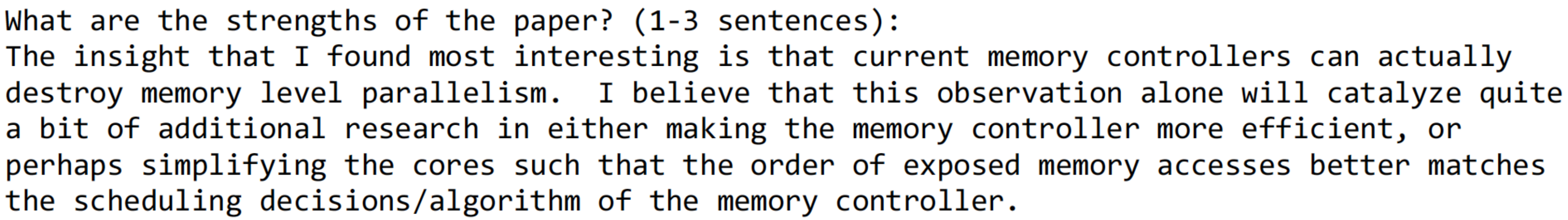
Provide a short summary of the paper (in your own words):
This paper presents a memory controller design that reduces inter-thread interference and allows threads exploit memory level parallelism by ensuring that the requests of one thread gets to fully utilize all the memory banks without intervening requests from other threads.

What are the strengths of the paper? (1-3 sentences):
This paper addresses a important problem - memory system performance in multicore processors. The techniques presented in this paper are simple enough to implement and are shown to be effective.



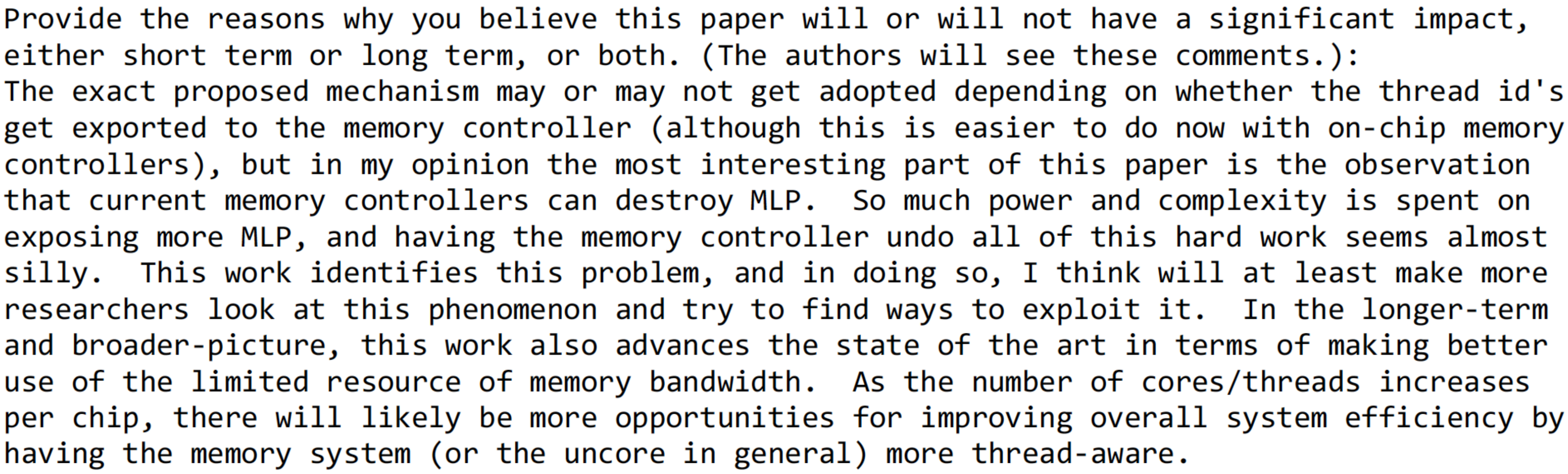
What are the strengths of the paper? (1-3 sentences):
The insight that I found most interesting is that current memory controllers can actually destroy memory level parallelism. I believe that this observation alone will catalyze quite a bit of additional research in either making the memory controller more efficient, or perhaps simplifying the cores such that the order of exposed memory accesses better matches the scheduling decisions/algorithm of the memory controller.

Top Picks 2009 Review #3



What are the strengths of the paper? (1-3 sentences):

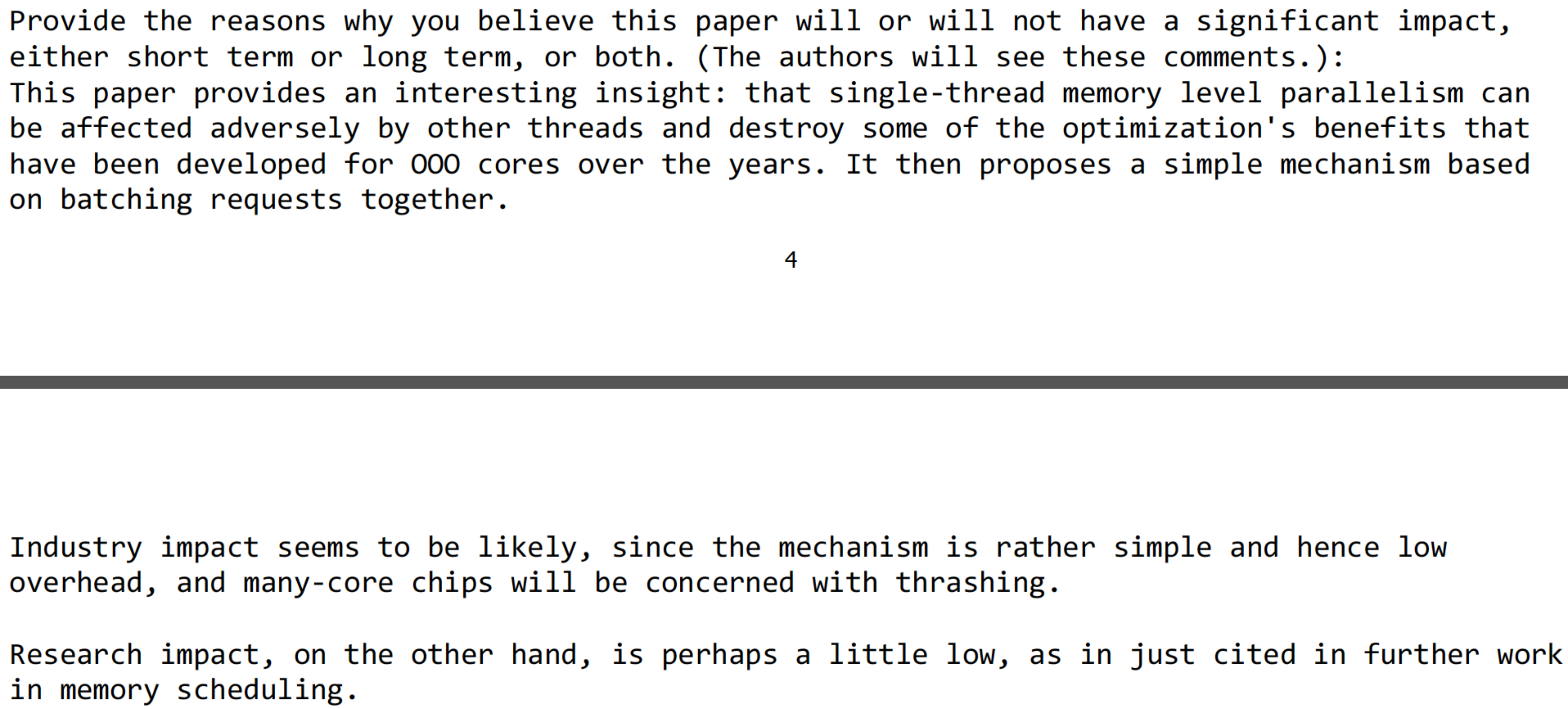
The insight that I found most interesting is that current memory controllers can actually destroy memory level parallelism. I believe that this observation alone will catalyze quite a bit of additional research in either making the memory controller more efficient, or perhaps simplifying the cores such that the order of exposed memory accesses better matches the scheduling decisions/algorithm of the memory controller.



Provide the reasons why you believe this paper will or will not have a significant impact, either short term or long term, or both. (The authors will see these comments.):

The exact proposed mechanism may or may not get adopted depending on whether the thread id's get exported to the memory controller (although this is easier to do now with on-chip memory controllers), but in my opinion the most interesting part of this paper is the observation that current memory controllers can destroy MLP. So much power and complexity is spent on exposing more MLP, and having the memory controller undo all of this hard work seems almost silly. This work identifies this problem, and in doing so, I think will at least make more researchers look at this phenomenon and try to find ways to exploit it. In the longer-term and broader-picture, this work also advances the state of the art in terms of making better use of the limited resource of memory bandwidth. As the number of cores/threads increases per chip, there will likely be more opportunities for improving overall system efficiency by having the memory system (or the uncore in general) more thread-aware.

Top Picks 2009 Review #4



Provide the reasons why you believe this paper will or will not have a significant impact, either short term or long term, or both. (The authors will see these comments.):
This paper provides an interesting insight: that single-thread memory level parallelism can be affected adversely by other threads and destroy some of the optimization's benefits that have been developed for OOO cores over the years. It then proposes a simple mechanism based on batching requests together.

4

Industry impact seems to be likely, since the mechanism is rather simple and hence low overhead, and many-core chips will be concerned with thrashing.

Research impact, on the other hand, is perhaps a little low, as in just cited in further work in memory scheduling.

Seminar in Computer Architecture

Meeting 4: PAR-BS Memory Scheduler

Prof. Onur Mutlu

ETH Zürich
Fall 2020
8 October 2020