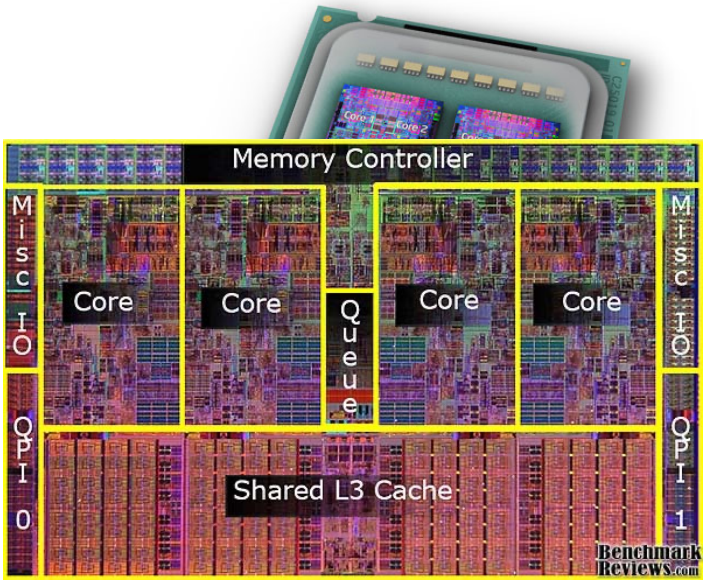
Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems

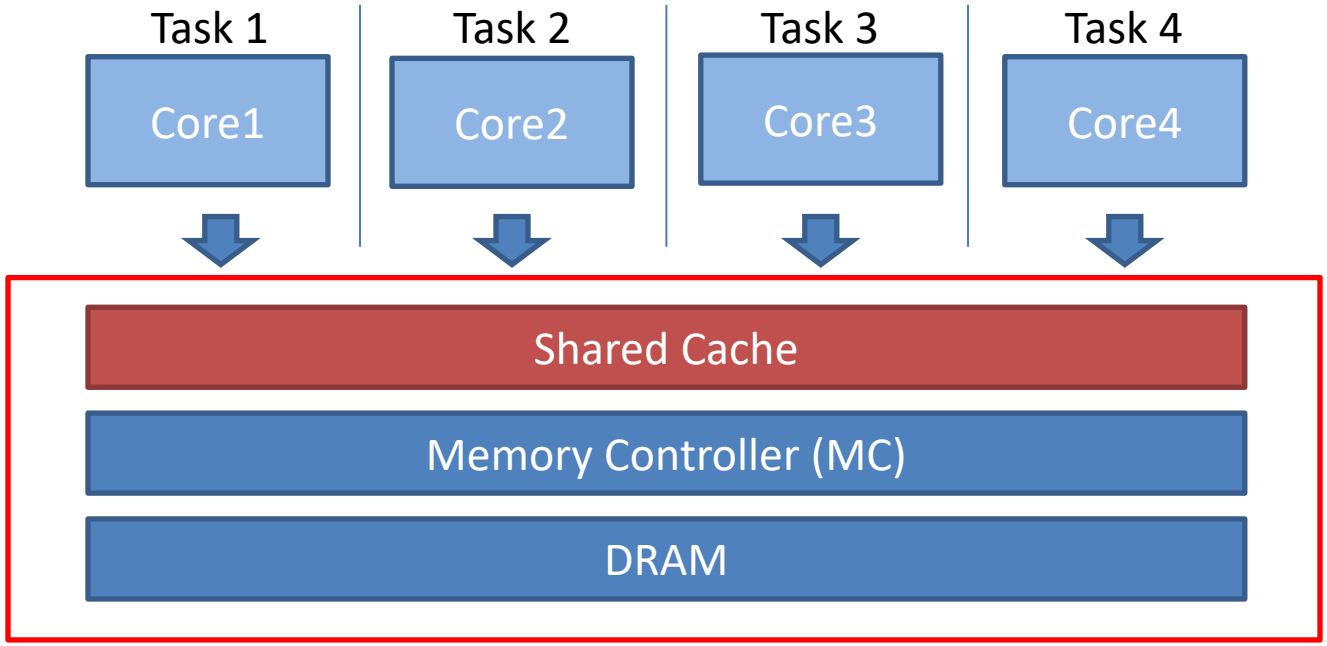
Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi
University of Kansas

High-Performance **Multicores** for **Real-Time** Systems

- Why?
  - **Intelligence** → more performance
  - Space, weight, power (SWaP), cost



Time Predictability Challenge



- Shared hardware resource contention can cause significant **interference delays**
- **Shared cache** is a major shared resource

Cache Partitioning

- Can be done in either software or hardware
  - Software: page coloring
  - Hardware: way partitioning
- Eliminate unwanted cache-line evictions

- Common assumption
  - cache partitioning → performance isolation
    - If working-set fits in the cache partition

- Not necessarily true on out-of-order cores using **non-blocking caches**

Outline

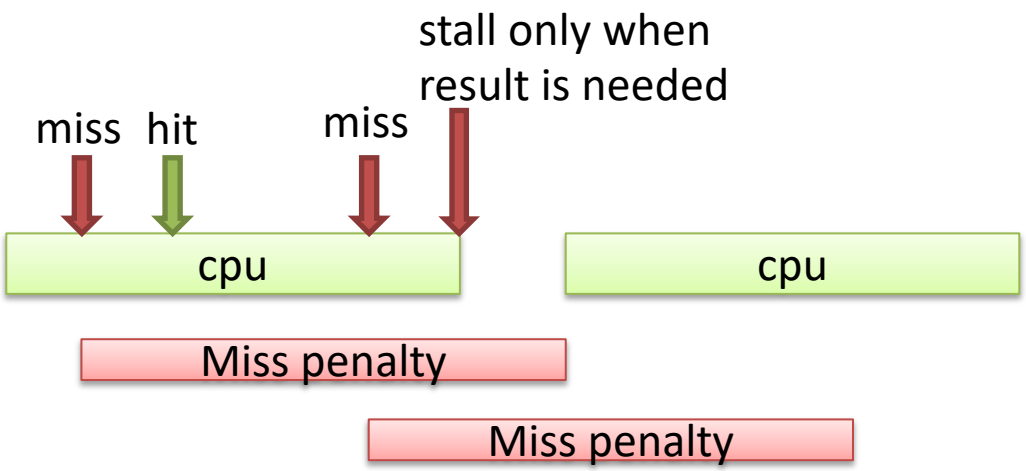
- *Evaluate* the isolation effect of cache partitioning
  - On four real COTS multicore architectures
  - Observed up to **21X slowdown** of a task running on a dedicated core accessing a dedicated cache partition
- *Understand* the source of contention
  - Using a cycle-accurate full system simulator
- *Propose* an OS/architecture collaborative solution
  - For better cache performance isolation

KU THE UNIVERSITY OF KANSAS

Memory-Level Parallelism (MLP)

- Broadly defined as the number of **concurrent memory requests** that a given architecture can handle at a time

Non-blocking Cache(*)



Multiple outstanding misses

- Can serve cache hits under multiple cache misses
  - Essential for an out-of-order core and any multicore
- Miss-Status-Holding Registers (MSHRs)
  - On a miss, allocate a MSHR entry to track the req.
  - On receiving the data, clear the MSHR entry

(*) D. Kroft. "Lockup-free instruction fetch/prefetch cache organization," ISCA'81

Non-blocking Cache

- #of cache MSHRs → memory-level parallelism (MLP) of the cache
- What happens if all MSHRs are occupied?
  - The cache is **locked up**
  - Subsequent accesses---including **cache hits**---to the cache **stall**
  - We will see the impact of this in later experiments

COTS Multicore Platforms

	Cortex-A7	Cortex-A9	Cortex-A15	Nehalem
Core	4core @ 1.4GHz In-order	4core @ 1.7GHz **Out-of-order**	4core @ 2.0GHz **Out-of-order**	4core @ 2.8GHz **Out-of-order**
LLC (shared)	512KB	1MB	2MB	8MB

- COTS multicore platforms
  - Odroid-XU4: 4x Cortex-A7 and 4x Cortex-A15
  - Odroid-U3: 4x Cortex-A9
  - Dell desktop: Intel Xeon quad-core (Nehalem)

Identified MLP

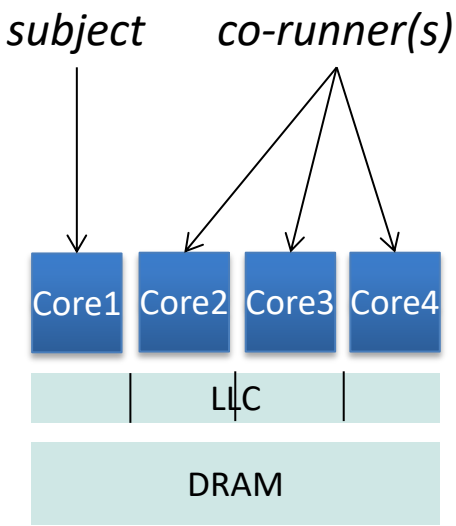
	Cortex-A7	Cortex-A9	Cortex-A15	Nehalem
Local MLP	1	4	6	10
Global MLP	4	4	11	16

(See paper for our experimental identification method)

- Local MLP
  - MLP of a core-private cache
- Global MLP
  - MLP of the shared cache (and DRAM)

KU THE UNIVERSITY OF KANSAS

Cache Interference Experiments



- Measure the performance of the 'subject'
  - (1) alone, (2) with co-runners
  - LLC is partitioned (equal partition) using PALLOC (*)
- Q: Does cache partitioning provide isolation?

(*) Heechul Yun, Renato Mancuso, Zheng-Pei Wu, Rodolfo Pellizzoni. "PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms." RTAS'14

IsolBench: Synthetic Workloads

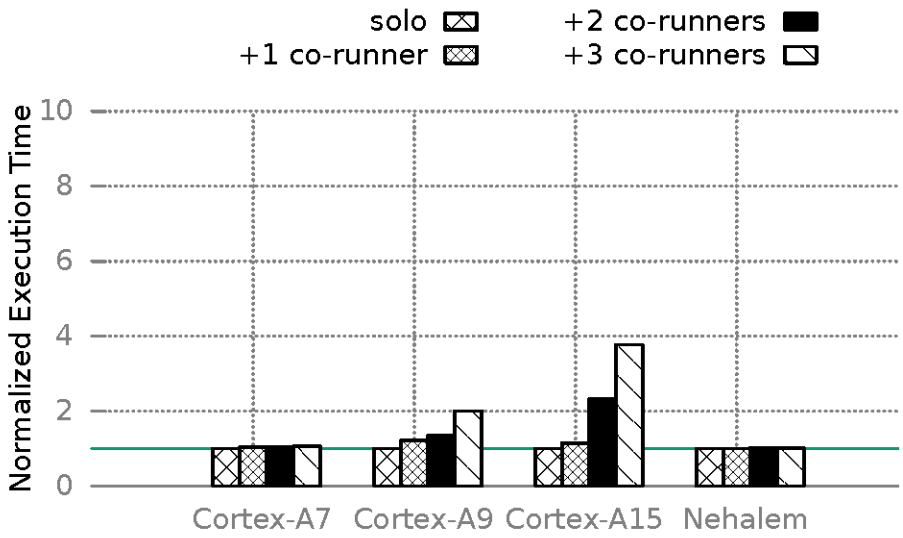
Experiment	Subject	Co-runner(s)
Exp. 1	Latency(LLC)	BwRead(DRAM)
Exp. 2	BwRead(LLC)	BwRead(DRAM)
Exp. 3	BwRead(LLC)	BwRead(LLC)
Exp. 4	Latency(LLC)	BwWrite(DRAM)
Exp. 5	BwRead(LLC)	BwWrite(DRAM)
Exp. 6	BwRead(LLC)	BwWrite(LLC)

Working-set size: (LLC) < ¼ LLC → cache-hits, (DRAM) > 2X LLC → cache misses

- Latency
  - A linked-list traversal, data dependency, one outstanding miss
- Bandwidth
  - An array reads or writes, no data dependency, multiple misses

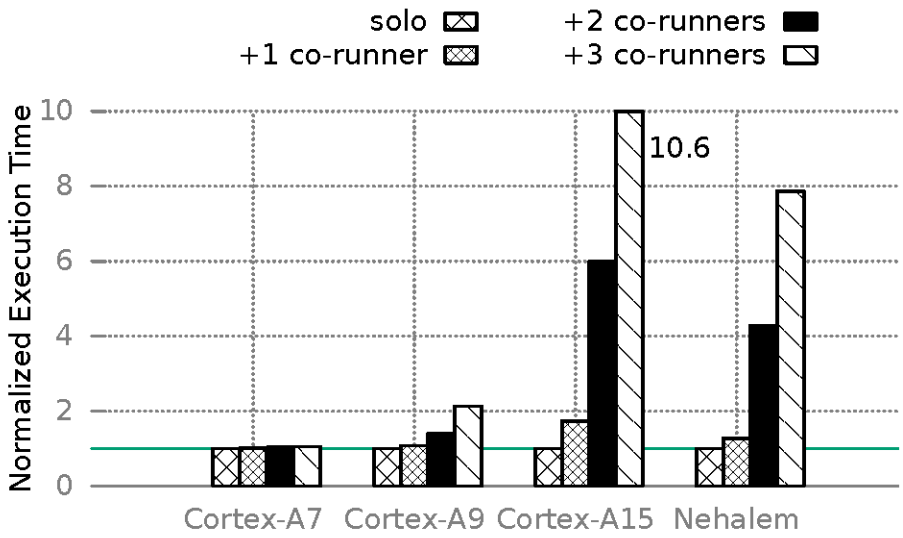
- **Subject** benchmarks: **LLC partition fitting**

Latency(LLC) vs. BwRead(DRAM)



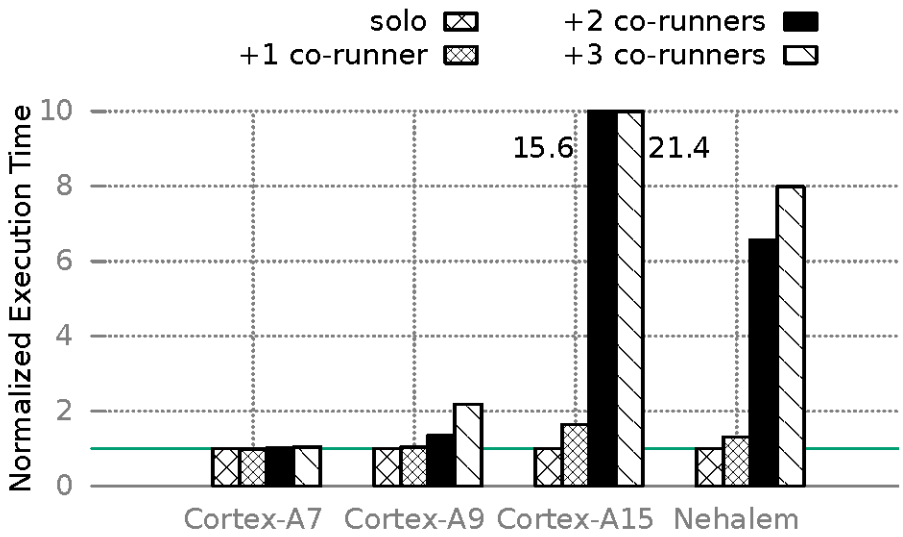
- No interference on Cortex-A7 and Nehalem
- On Cortex-A15, Latency(LLC) suffers 3.8X slowdown
  - despite partitioned LLC

BwRead(LLC) vs. BwRead(DRAM)



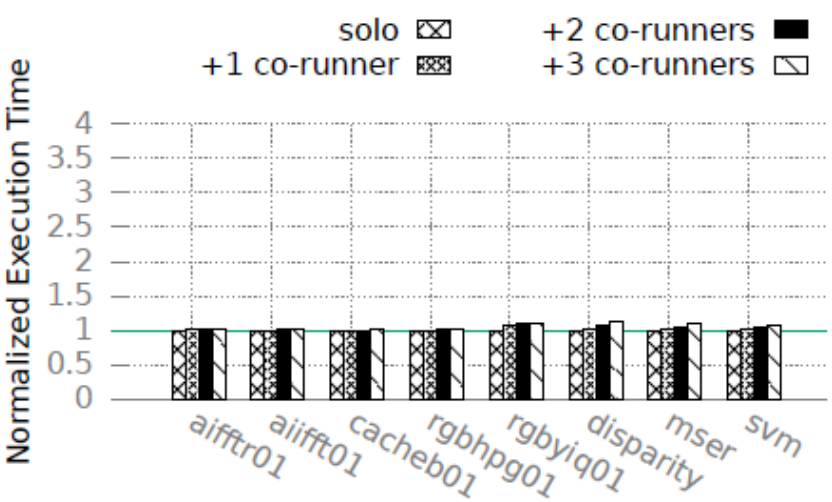
- Up to 10.6X slowdown on Cortex-A15
- Cache partitioning != performance isolation
  - On all tested out-of-order cores (A9, A15, Nehalem)

BwRead(LLC) vs. BwWrite(DRAM)

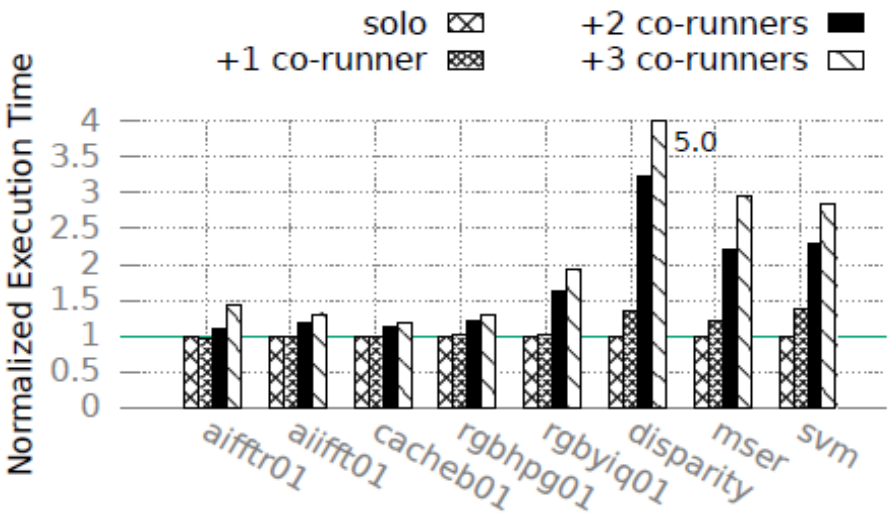


- Up to 21X slowdown on Cortex-A15
- Writes generally cause more slowdowns
  - Due to write-backs

EEMBC and SD-VBS



Cortex-A7 (in-order)



Cortex-A15 (out-of-order)

- X-axis: EEMBC, SD-VBS (cache partition fitting)
  - Co-runners: BwWrite(DRAM)
- Cache partitioning != performance isolation

MSHR Contention

	Cortex-A7	**Cortex-A9**	**Cortex-A15**	**Nehalem**
Local MLP	1	**4**	**6**	**10**
Global MLP	4	**4**	**11**	**16**

(See paper for our experimental identification method)

- Shortage of cache MSHRs → lock up the cache
- LLC MSHRs are shared resources
  - 4cores x L1 cache MSHRs > LLC MSHRs

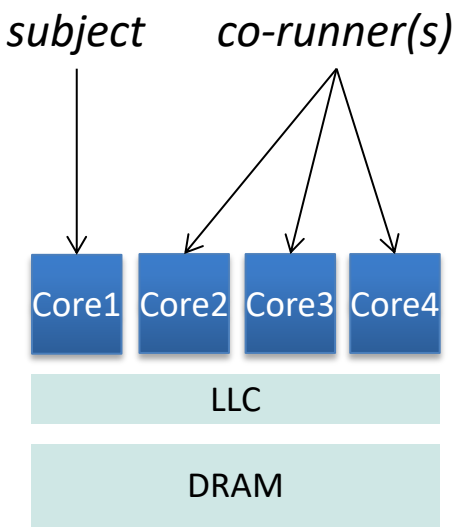
Outline

- Evaluate the isolation effect of cache partitioning
- ***Understand* the source of contention**
  - Effect of LLC MSHRs
  - Effect of L1 MSHRs
- Propose an OS/architecture collaborative solution

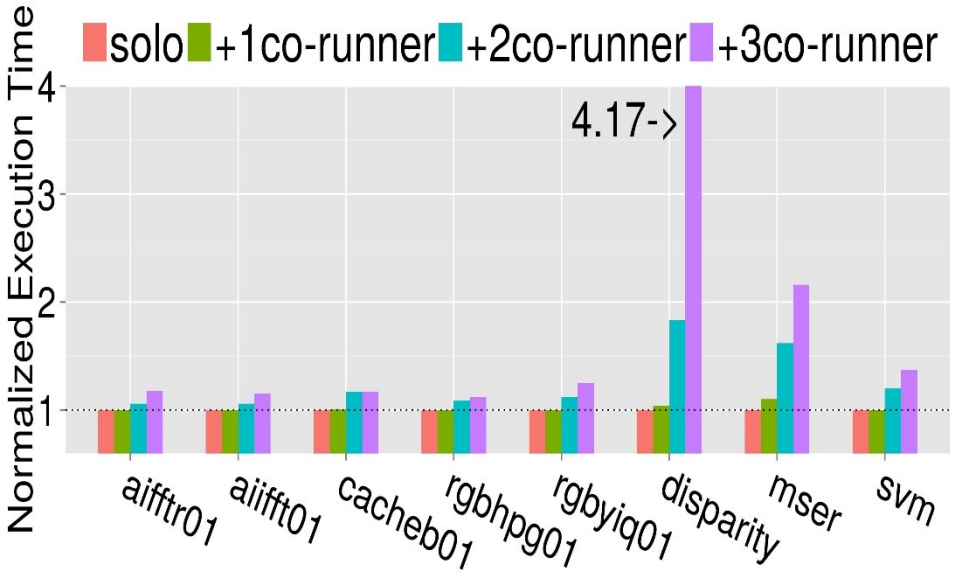
KU THE UNIVERSITY OF KANSAS

Simulation Setup

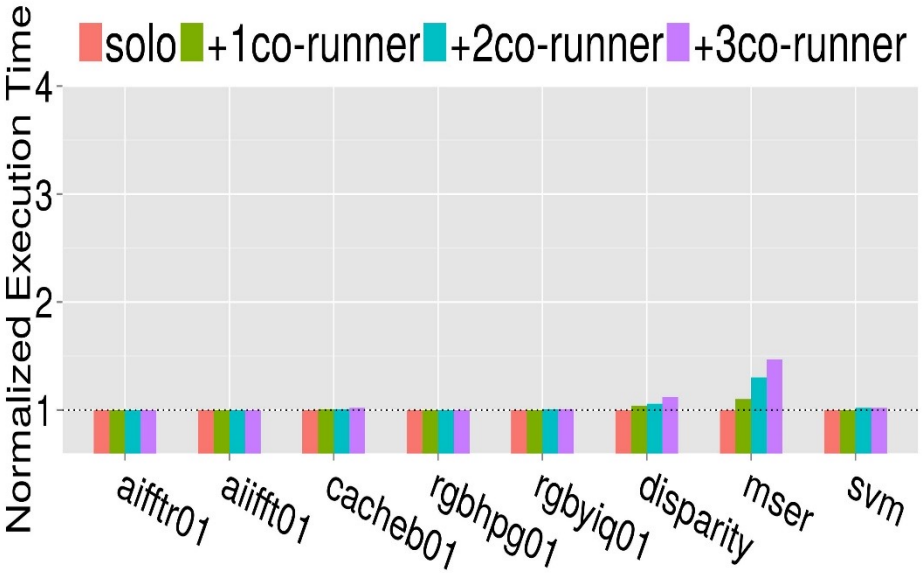
- Gem5 full-system simulator
  - 4 out-of-order cores (modeling Cortex-A15)
    - L1: 32K I&D, LLC (L2): 2048K
  - **Vary #of L1 and LLC MSHRs**
- Linux 3.14
  - Use PALLOC to partition LLC
- Workload
  - IsolBench, EEMBC, SD-VBS, SPEC2006



Effect of LLC (Shared) MSHRs



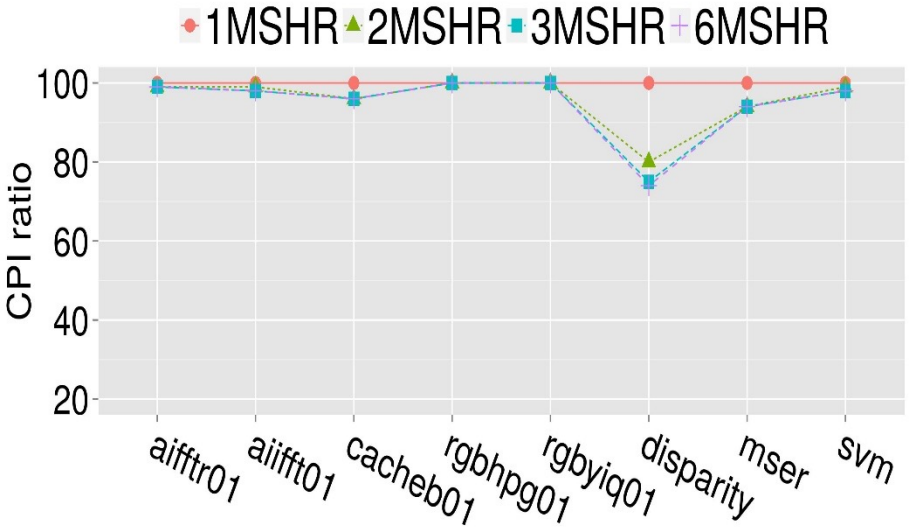
L1:6/LLC:**12** MSHRs



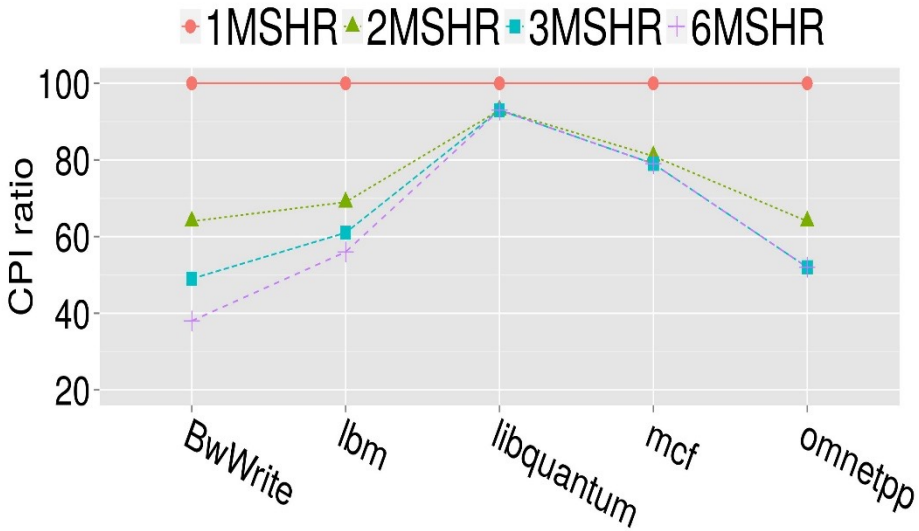
L1:6/LLC:**24** MSHRs

- Increasing LLC MSHRs eliminates the MSHR contention
- Issue: scalable?
  - MSHRs are highly associative (must be accessed in parallel)

Effect of L1 (Private) MSHRs



EEMBC, SD-VBS



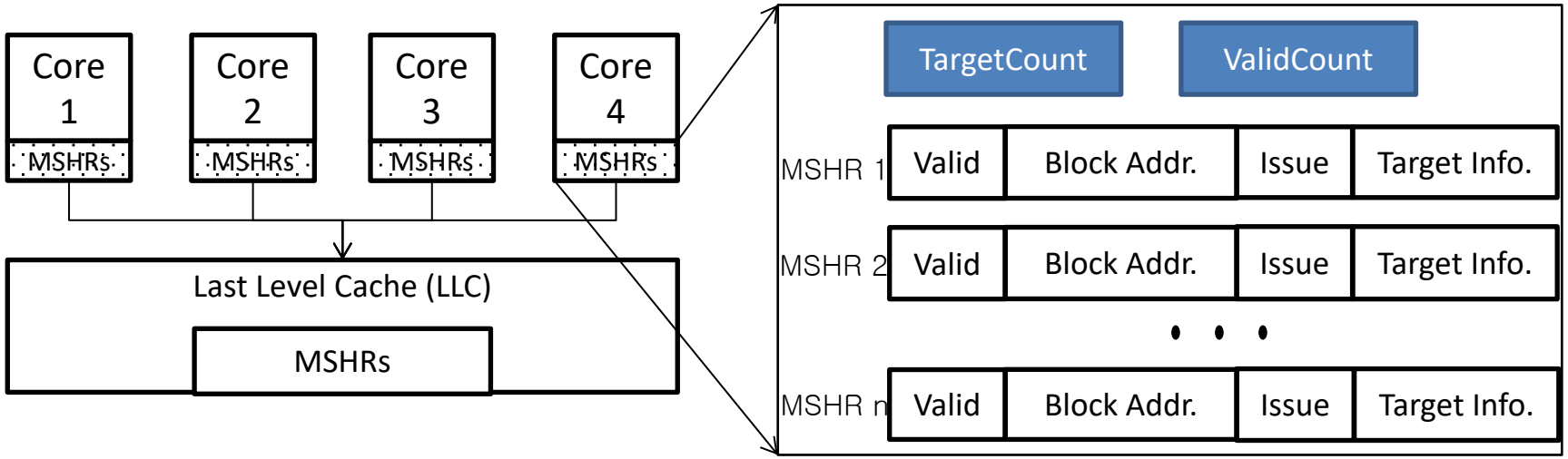
IsolBench,SPEC2006

- Reducing L1 MSHRs can eliminate contention
- Issue: single thread performance. How much?
  - It *depends.* L1 MSHRs are often underutilized
  - EEMBC: low, SD-VBS: medium, SPEC2006: high

Outline

- *Evaluate* the isolation effect of cache partitioning
- *Understand* the source of contention
- ***Propose* an OS/architecture collaborative solution**

OS Controlled MSHR Partitioning



- Add two registers in each core's L1 cache
  - *TargetCount*: max. MSHRs (set by OS)
  - *ValidCount*: used MSHRs (set by HW)
- OS can control each core's MLP
  - By adjusting the core's *TargetCount* register

OS Scheduler Design

- Partition LLC MSHRs by enforcing

$$\sum_{i=1}^{m} TargetCount_i \leq N_{mshr}^{LLC},$$

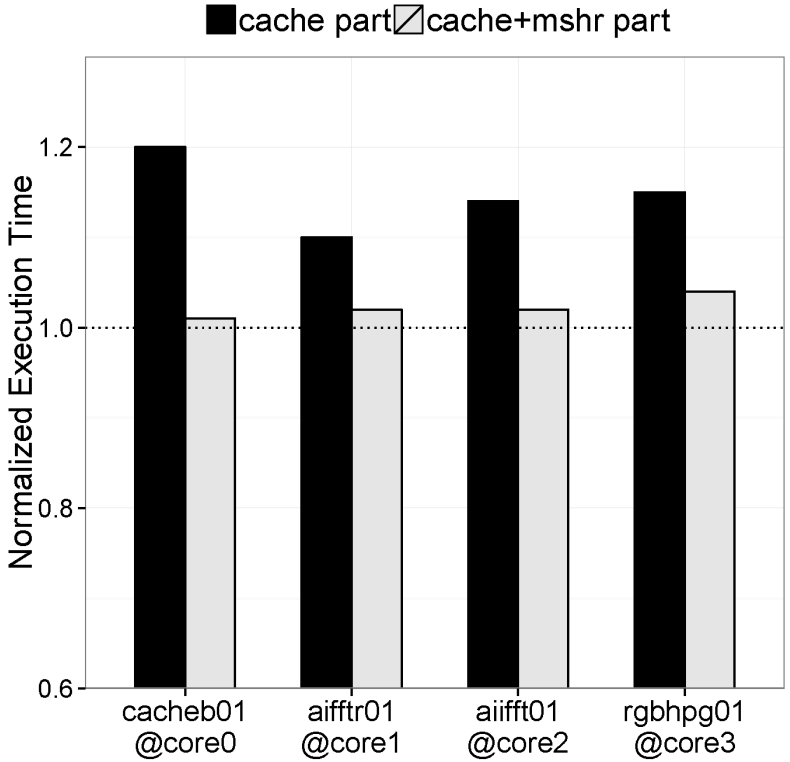
$$1 \leq TargetCount_i \leq N_{mshr}^{L1}$$

- When RT tasks are scheduled
  - RT tasks: reserve MSHRs
  - Non-RT tasks: share remaining MSHRs

- Implemented in Linux kernel
  - `prepare_task_switch()` at `kernel/sched/core.c`

Case Study

- 4 RT tasks (EEMBC)
  - One RT per core
  - Reserve 2 MSHRs
  - P: 20,30,40,60ms
  - C: ~8ms
- 4 NRT tasks
  - One NRT per core
  - Run on slack
- Up to 20% WCET reduction
  - Compared to cache partition only

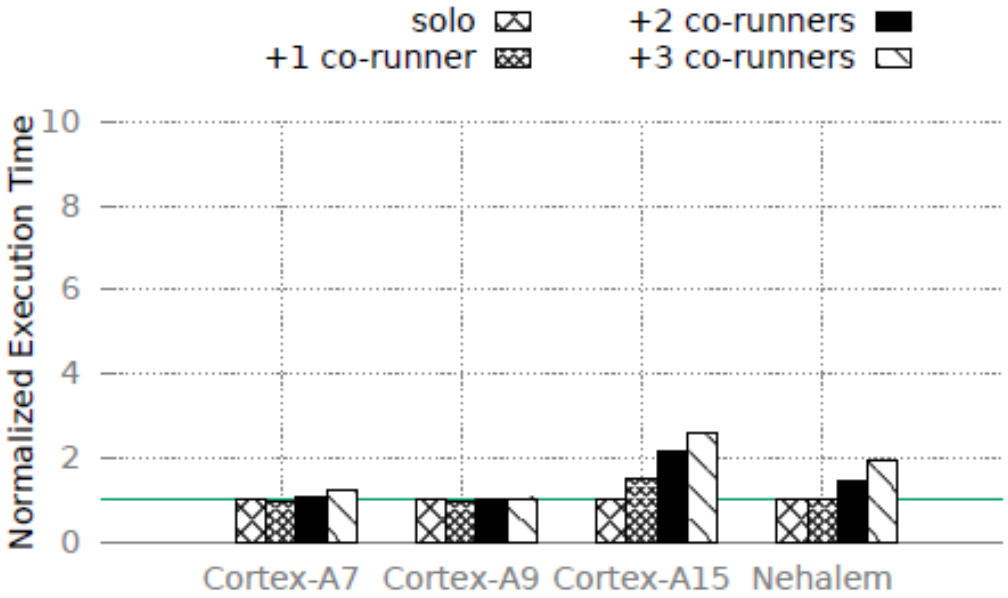


Conclusion

- Evaluated the effect of cache partitioning on four real COTS multicore architectures and a full system simulator
- Found cache partitioning does not ensure cache (hit) performance isolation due to **MSHR contention**
- Proposed low-cost architecture extension and OS scheduler support to eliminate MSHR contention
- Availability
  - https://github.com/CSL-KU/IsolBench
  - Synthetic benchmarks (*IsolBench*),test scripts, kernel patches

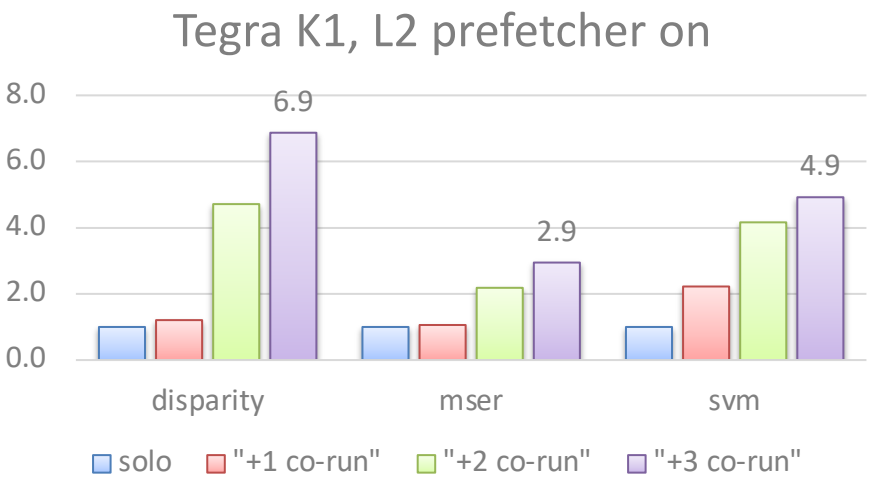
KU THE UNIVERSITY OF KANSAS

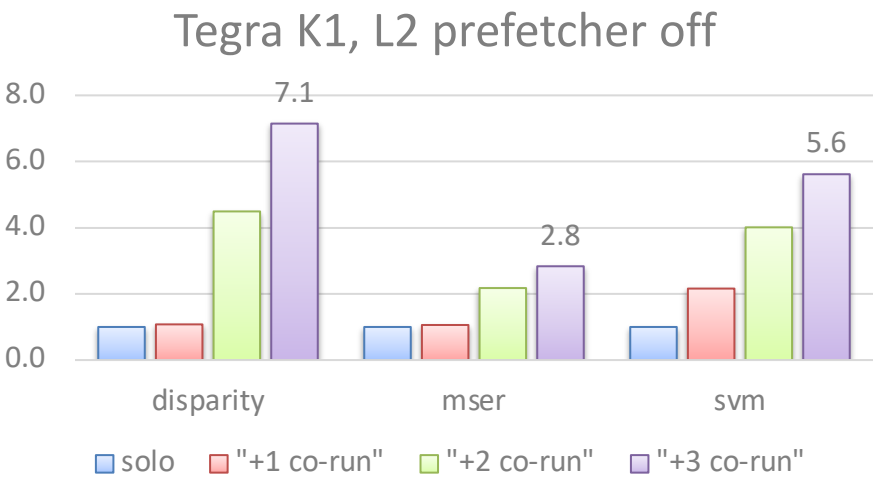
Exp3: BwRead(LLC) vs. BwRead(LLC)



- Cache bandwidth contention is **not** the main source
  - Higher cache-bandwidth co-runners cause **less** slowdowns

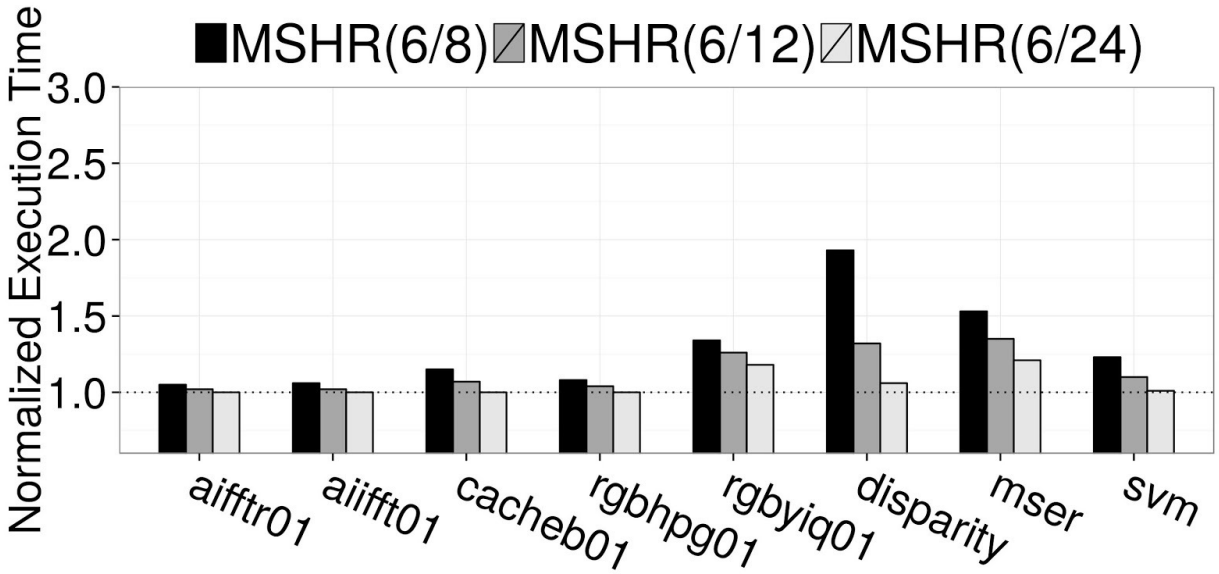
L2 Prefetcher





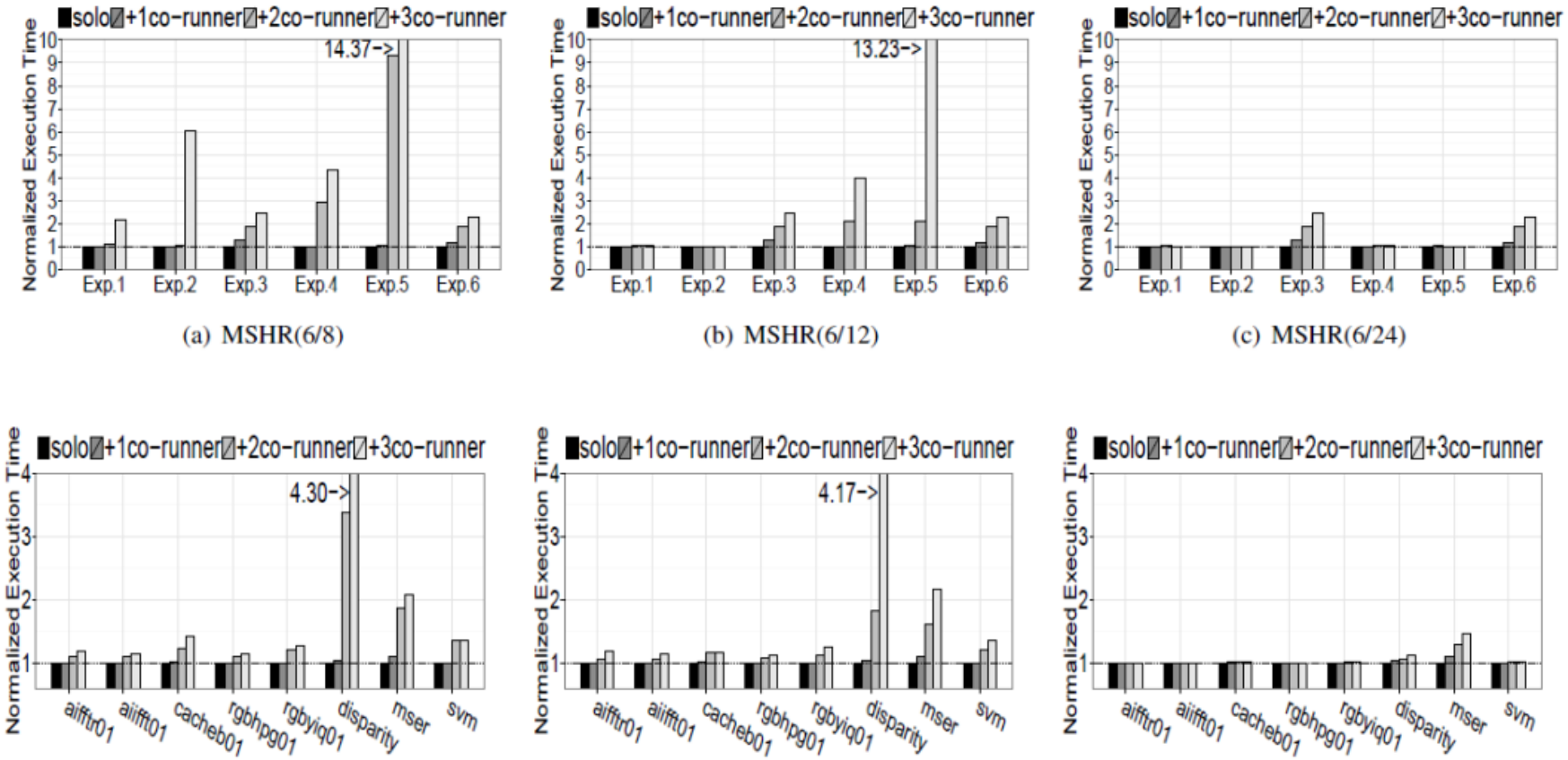
- Insignificant impact

Real Benchmarks as Co-runners



- 3x 470.lbm as co-runners
- MSHR contention is still a problem to some

Effect of LLC (Shared) MSHRs



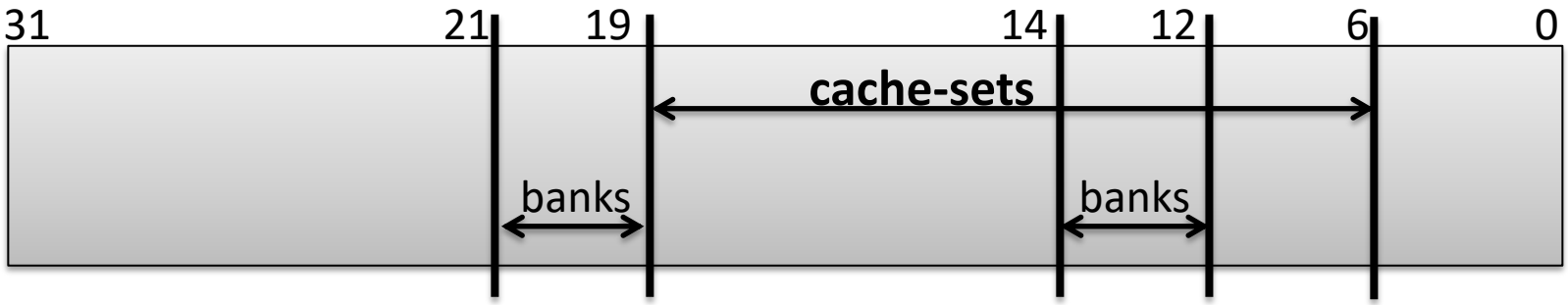
(a) MSHR(6/8)

(b) MSHR(6/12)

(c) MSHR(6/24)

PALLOC [Yun'14]

- Linux buddy allocator replacement
  - Support physical address-aware page allocation
    - Can partition the cache (and DRAM banks)
    - Support XOR addressing in Intel platforms



Intel Xeon 3530 physical address map

[Yun'14] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, Rodolfo Pellizzoni. "PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms." RTAS'14

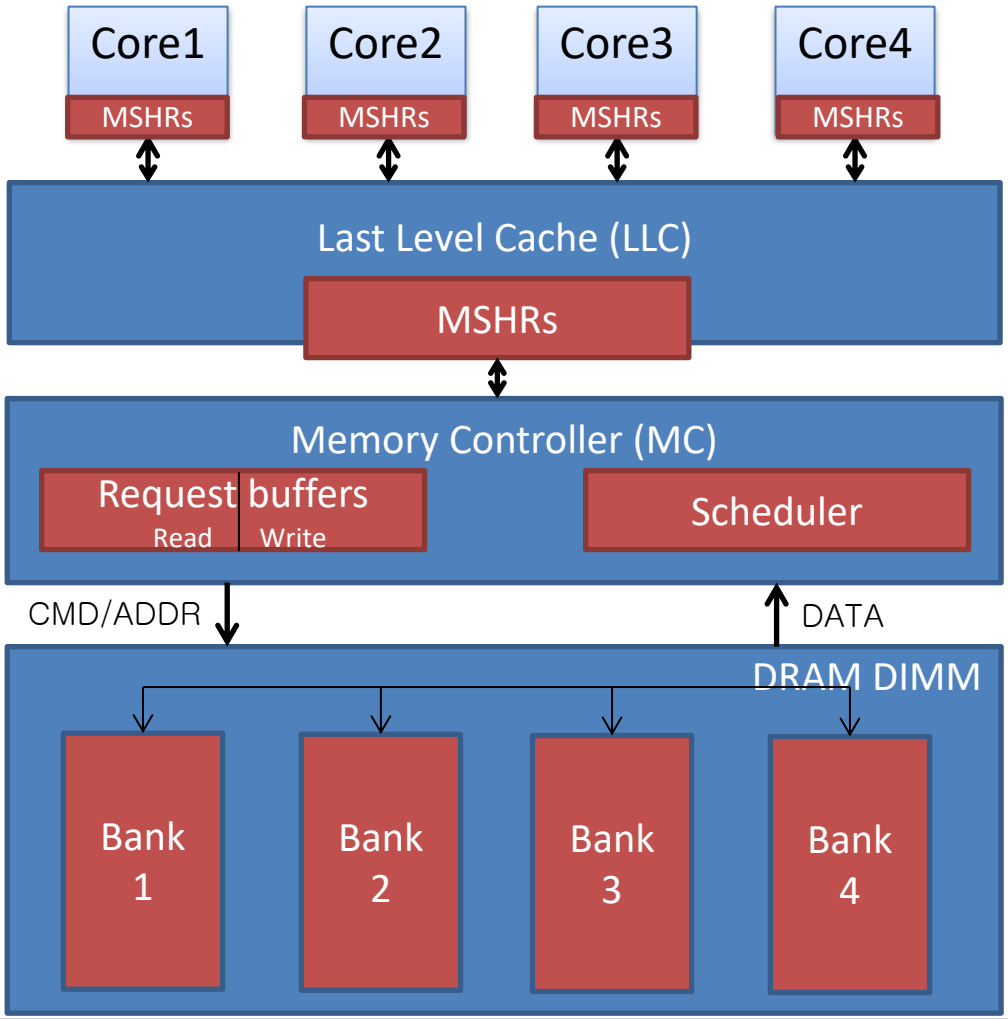
PALLOC Interface

- Example cache partitioning

```
# echo 0x00003000 > /sys/kernel/debug/palloc/palloc_mask
  → bits: 14, 15
# cd /sys/fs/cgroup
# mkdir core0 core1 core2 core3
  ➔ create 4 cgroup partitions
# echo 0 > core0/palloc.bins
  ➔ allocate pages whose addr. bit 14 and bit 15 are both 0 (00)
# echo 1 > core1/palloc.bins
# echo 2 > core2/palloc.bins
# echo 3 > core3/palloc.bins
```

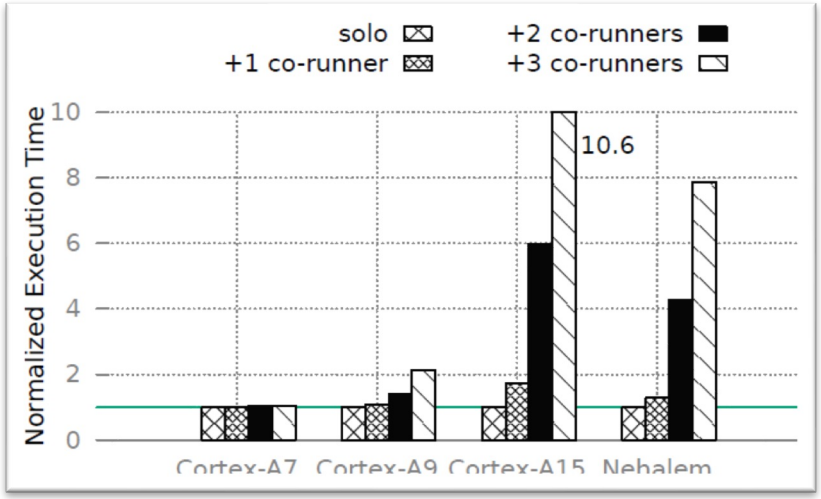
https://github.com/heechul/palloc

COTS Multicore Architecture

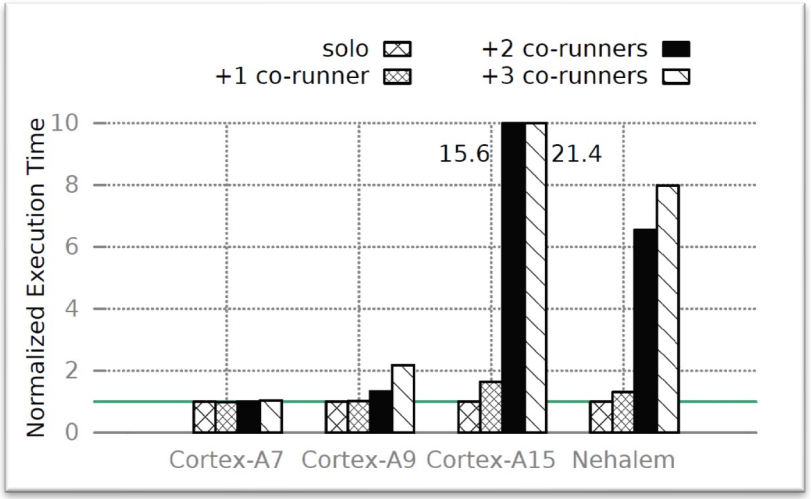


- Out-of-order core: Multiple memory requests
- Non-blocking caches: Multiple cache-misses
- MC and DRAM: Multiple banks

Effect of Writes



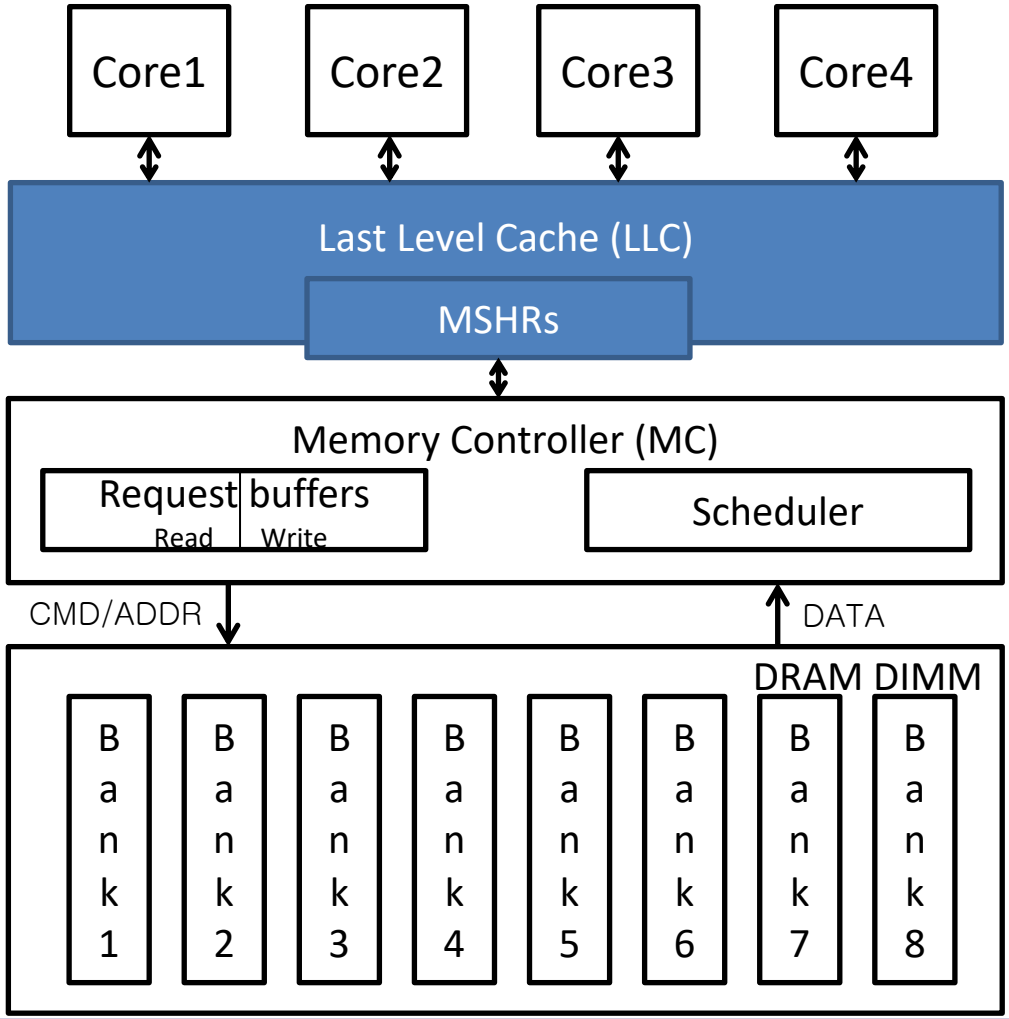
Exp2: BwRead(LLC) vs. **BwRead(DRAM)**



Exp5: BwRead(LLC) vs. **BwWrite(DRAM)**

- Write-intensive co-runners cause *more* slowdowns
  - BwRead(DRAM): up to 10.6X, BwWrite(DRAM): up to 21.4X
- Because writes stay longer in the MSHRs
  - Due to additional write-backs

COTS Multicore Architecture



- Out-of-order core: Multiple memory requests
- Non-blocking caches: Multiple cache-misses
- MC and DRAM: Bank-level parallelism

EEMBC, SD-VBS Workload

Benchmark	L1-MPKI	L2-MPKI	Description
EEMBC Automotive, Consumer [1]			
aifftr01	3.64	0.00	FFT (automotive)
aiifft01	3.99	0.00	Inverse FFT (automotive)
cacheb01	2.14	0.00	Cache buster (automotive)
rgbhpg01	1.59	0.00	Image filter (consumer)
rgbyiq01	3.81	0.01	Image filter (consumer)
SD-VBS: San Diego Vision Benchmark Suite [35]. (input: sqcif)			
disparity	56.92	0.13	Disparity map
mser	16.12	0.57	Maximally stable regions
svm	7.81	0.01	Support vector machines

- Subject
  - Subset of EEMBC, SD-VBS
  - **High L2 hit**, Low L2 miss
- Co-runners
  - BwWrite(DRAM): High L2 miss, write-intensive