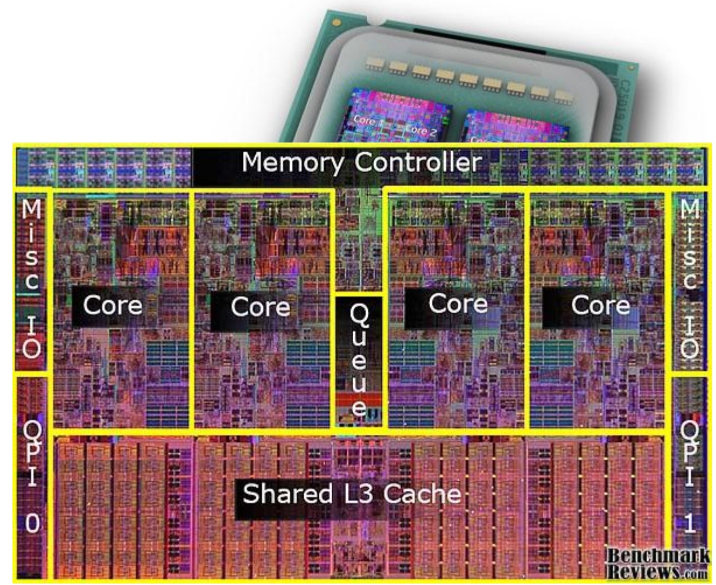
Cache Partitioning on Contemporary COTS Multicore Processors

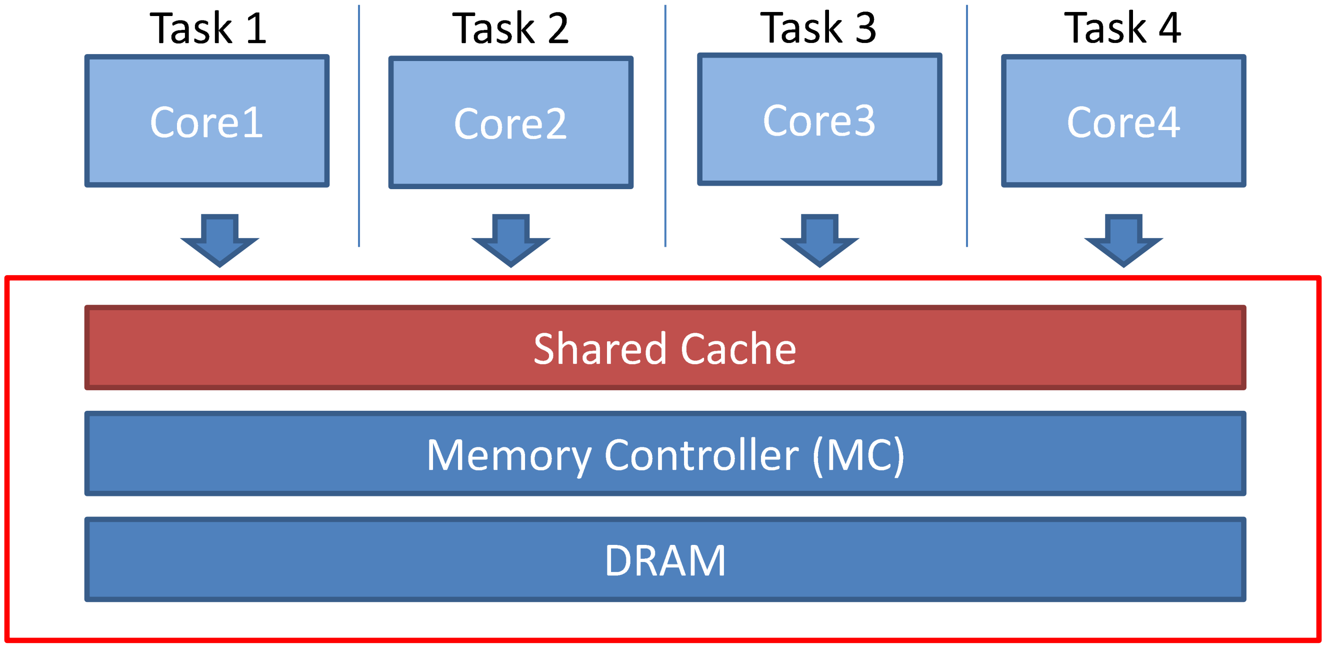
4/21/2017
Heechul Yun
Assistant Professor, University of Kansas

High-Performance Multicores for Intelligent Safety Critical Systems

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

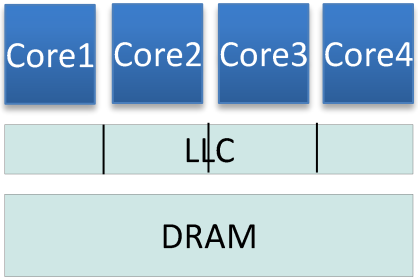


Time Predictability Challenge



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

Cache Partitioning



- Page coloring
  - Control cache-sets (OS)
- Way partitioning
  - Control cache ways (HW)

- Goal: Eliminate unwanted cache-line evictions

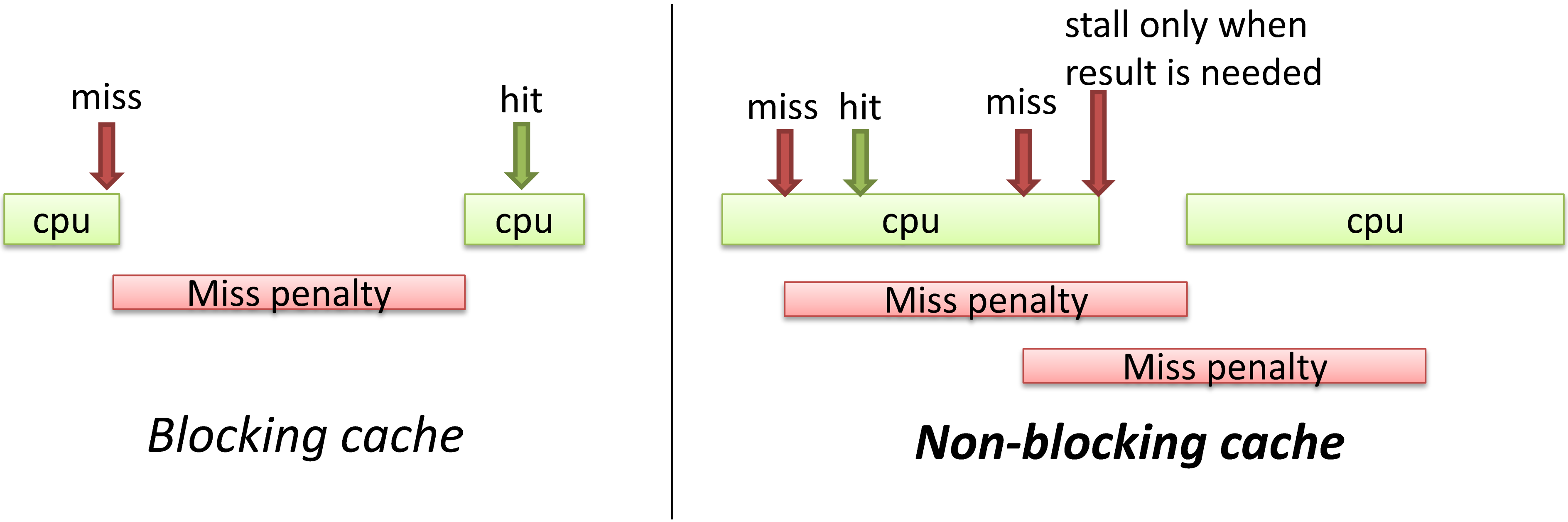
- Common assumption
  - Cache partitioning → performance isolation
    - If working-set fits in the cache partition

- Not necessarily true on “modern” caches

This Talk

- Isolation Performance of Cache Partitioning
  - Page coloring (4 ARM, 1 Intel)
  - Way partitioning (Intel CAT)
- Sources of Inter-core Cache Interferences
  - Miss Status Holding Registers (MSHRs)
  - Complex organization and mapping
- Recommendations
  - Multicore architecture for avionics/automotive

Non-blocking Cache

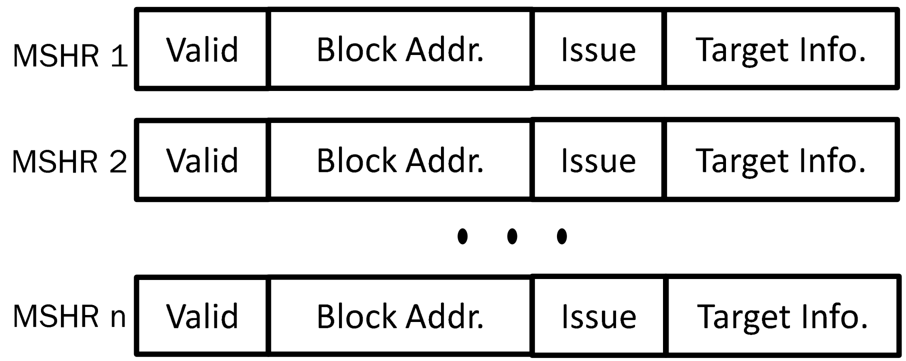


- Can serve cache hits under multiple cache misses
  - Essential for an out-of-order core and any multicore

(*) D. Kroft. “Lockup-free instruction fetch/prefetch cache organization,” ISCA’81

Miss Status Holding Registers

- Hardware structure
  - keeps track of outstanding misses



- Operation
  - On a miss, allocate a MSHR entry to track the req.
  - On receiving the data, clear the MSHR entry

- #of MSHRs
  - Memory-level parallelism (MLP) of the cache

Blocking of a Non-blocking Cache

- What happens if all MSHRs are occupied?
  - CPU's access to the cache is **blocked**
  - Until the pending misses are completed

- Blocked shared LLC
  - Can delay ALL cores, incl. cache-hit requests
  - A pending cache miss could take **100's of CPU cycles** to complete (access to DRAM is slow)
  - We will see the impact of this in later experiments

COTS Multicore Platforms

	ARM Cortex-A7	ARM Cortex-A9	ARM Cortex-A15^{O}	ARM Cortex-A15^{T}	Intel Nehalem
Core	4core @ 1.4GHz In-order	4core @ 1.7GHz **Out-of-order**	4core @ 2.0GHz **Out-of-order**	4core @ 2.0GHz **Out-of-order**	4core @ 2.8GHz **Out-of-order**
LLC (shared)/ Prefetcher	512KB Off	1MB Off	2MB On	2MB Off	8MB Off
Platform	Odroid-XU4	Odroid-U3	Odroid-XU4	Tegra TK1	Dell T3500

- COTS multicore platforms
  - Odroid-XU4: 4x Cortex-A7 and 4x Cortex-A15
  - Tegra TK1: 4x Cortex-A15
  - Odroid-U3: 4x Cortex-A9
  - Dell desktop: Intel Xeon quad-core (Nehalem)

KU THE UNIVERSITY OF KANSAS

Measuring Memory-Level Parallelism

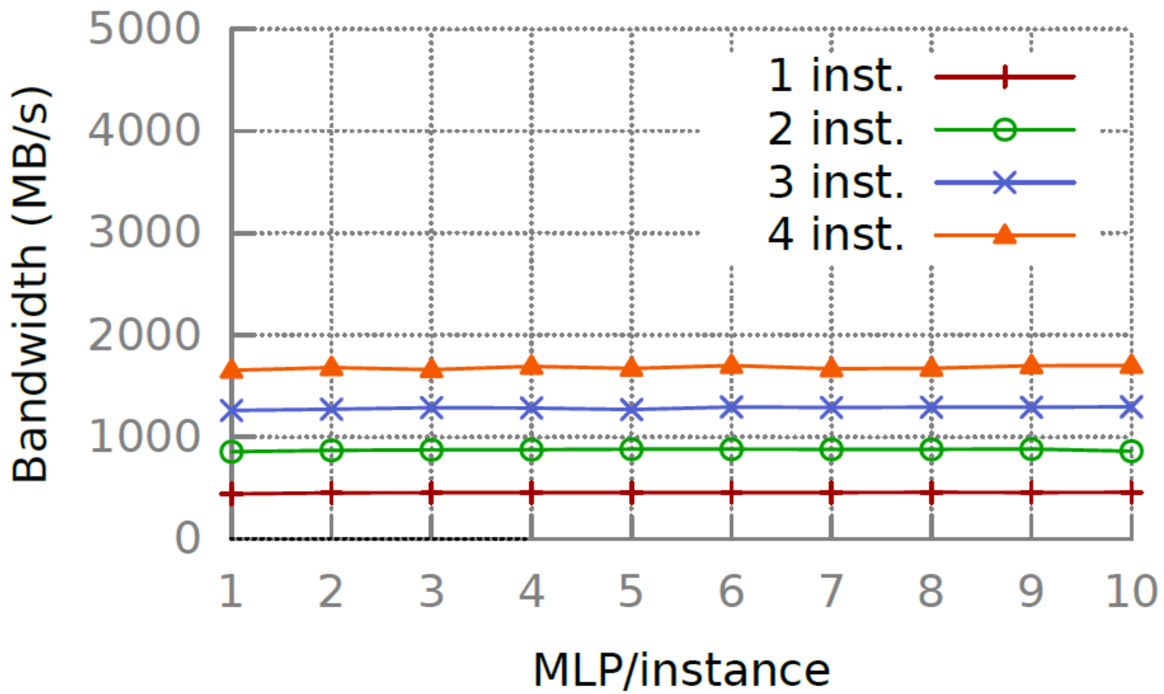
```
static int* list[MAX_MLP];
static int next[MAX_MLP];

long run(long iter, int mlp)
{
    long cnt = 0;
    for (long i = 0; i < iter; i++) {
        switch (mlp) {
        case MAX_MLP:
            .
            .
        case 2:
            next[1] = list[1][next[1]];
            /* fall-through */
        case 1:
            next[0] = list[0][next[0]];
        }
        cnt += mlp;
    }
    return cnt;
}
```

- Measuring # of MSHRs
- The benchmark (*)
  - Concurrent list traversal
  - #of lists = MLP

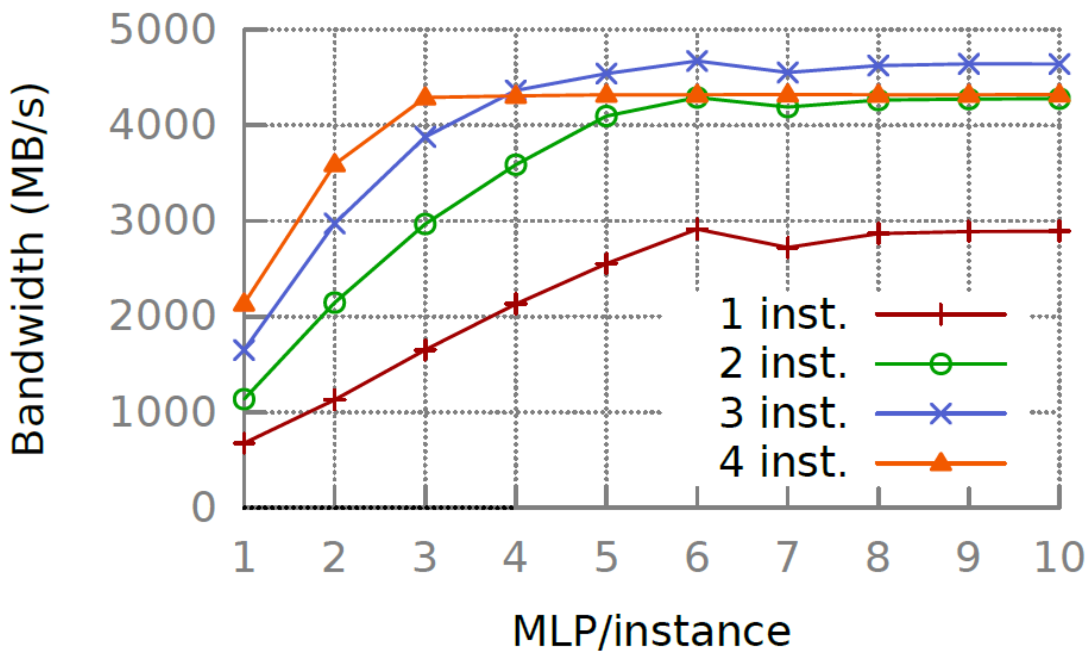
D. Eklov et al., “Bandwidth bandit: quantitative characterization of memory contention.” PACT’12

Cortex A7 (in-order)



- A single thread can generate one request at a time
  - Local MLP = 1
- 4 threads generate 4 requests at a time
  - Global MLP = 4

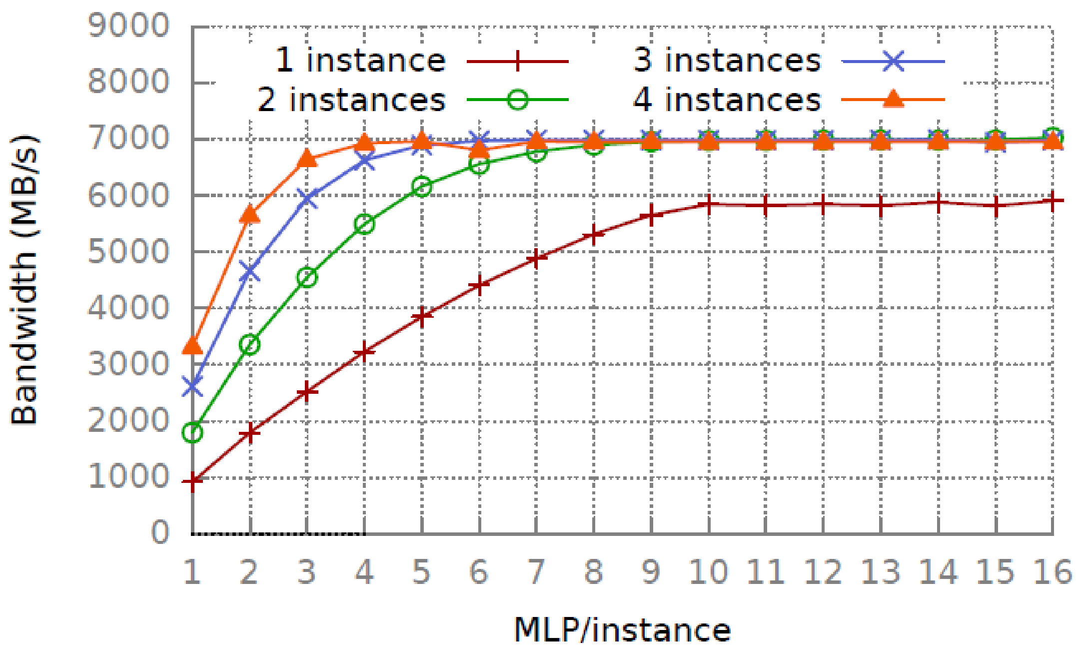
Cortex-A15^T (out-of-order)



- A single thread can generate up to 10 concurrent requests
  - Local MLP = 6
- 4 threads generate up to 11
  - Global MLP = 11 (*)

(*) ARM. Cortex-A15 Technical Reference Manual, Rev: r2p0, 2011

Intel Nehalem (out-of-order)



- A single thread can generate up to 10 concurrent requests
  - Local MLP = 10
- 4 threads generate up to 16 (4 x 4) concurrent requests
  - Global MLP = 16 (*)

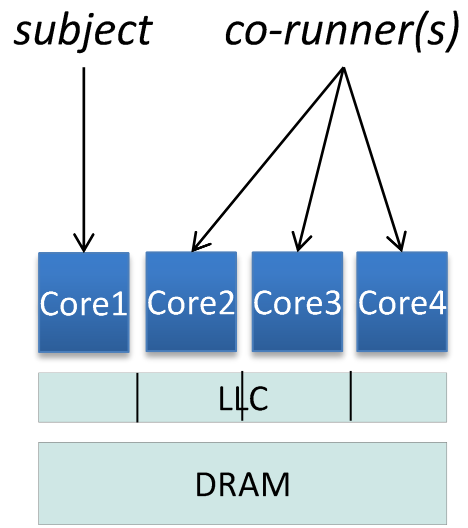
(*) However, this is due to #of DRAM banks, not the LLC. Nehalem's LLC support up to 32

Identified Memory Level Parallelism

	Cortex-A7	Cortex-A9	Cortex-A15^{O,T}	Nehalem
Local MLP	1	4	6	10
Global MLP	4	4	11	16

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

Cache Interference Experiments



- Measure the performance of the 'subject'
  - (1) alone, (2) with co-runners
  - Last-Level Cache (LLC) is evenly partitioned 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

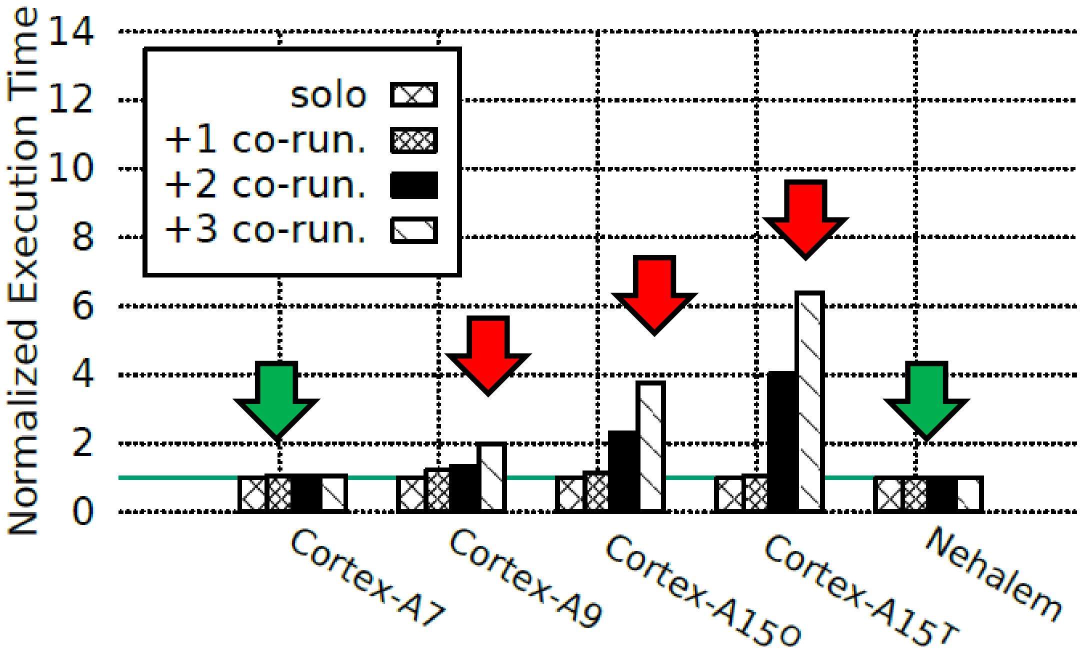
Experiment	Subject	Co-runner(s)
Exp. 1	Latency(LLC)	BwRead(DRAM)
Exp. 2	BwRead(LLC)	BwRead(DRAM)
Exp. 3	BwRead(LLC)	BwRead(LLC)
Exp. 4	Latency(LLC)	BwWrite(DRAM)
Exp. 5	BwRead(LLC)	BwWrite(DRAM)
Exp. 6	BwRead(LLC)	BwWrite(LLC)

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

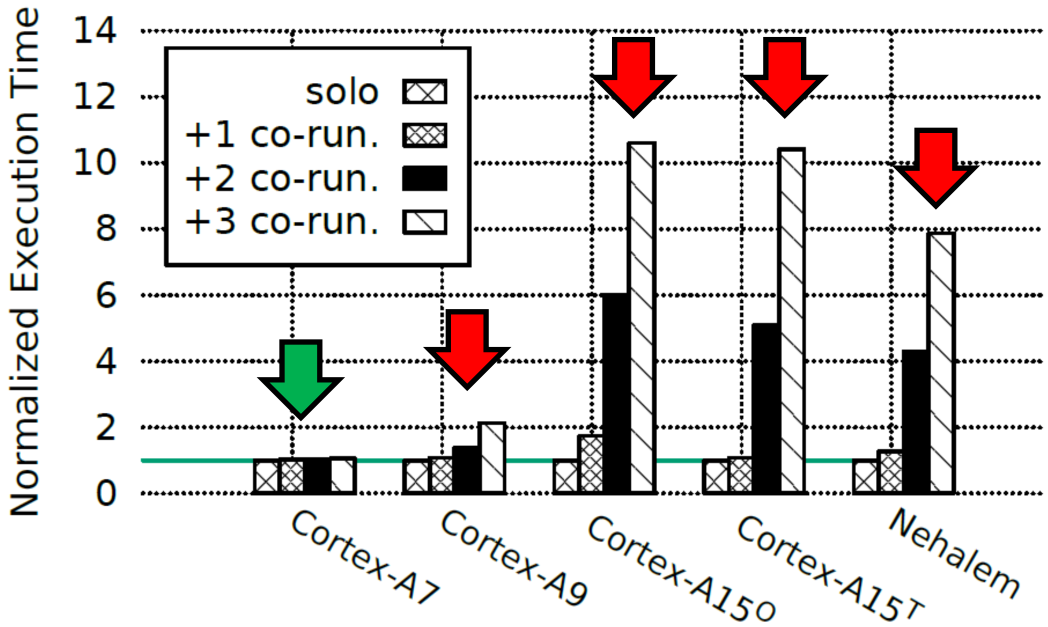
- **Subject** benchmarks: **LLC (Last-Level Cache) partition fitting**

Latency(LLC) vs. BwRead(DRAM)



- No interference on Cortex-A7 and Nehalem
- On Cortex-A15^T, Latency(LLC) suffers 6.4X slowdown
  - despite partitioned LLC

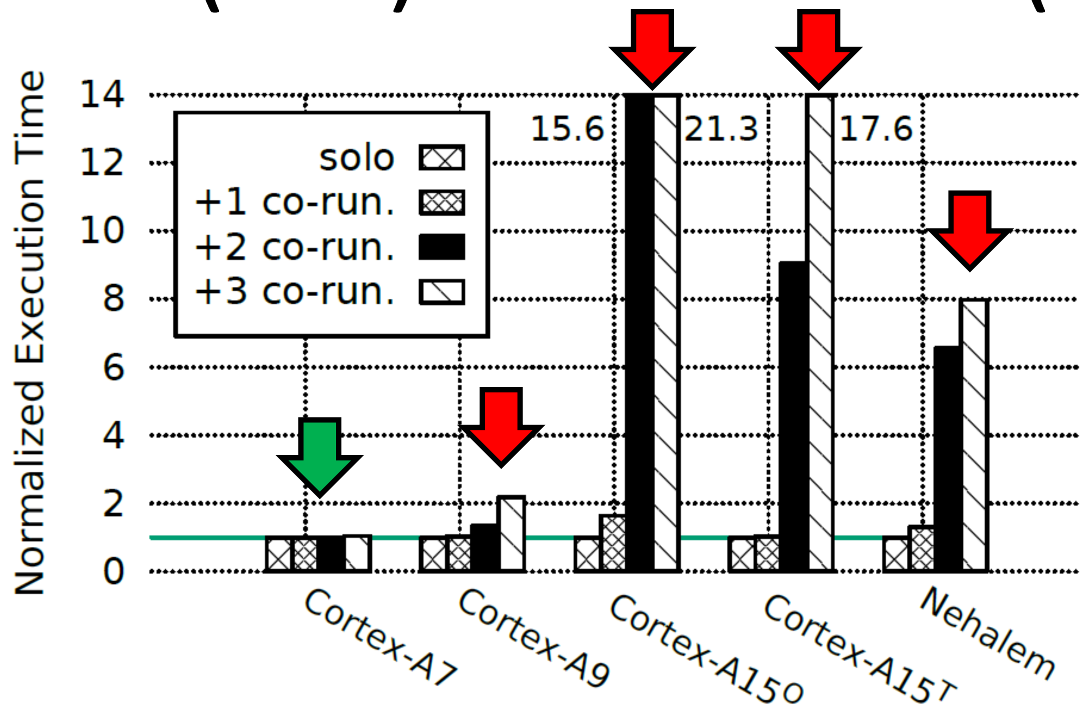
BwRead(LLC) vs. BwRead(DRAM)



- Up to 10.6X slowdown on Cortex-A15^{O} (8X in Nehalem)
- Cache partitioning != performance isolation
  - On all tested out-of-order cores (A9, A15, Nehalem)

KU THE UNIVERSITY OF KANSAS

BwRead(LLC) vs. BwWrite(DRAM)



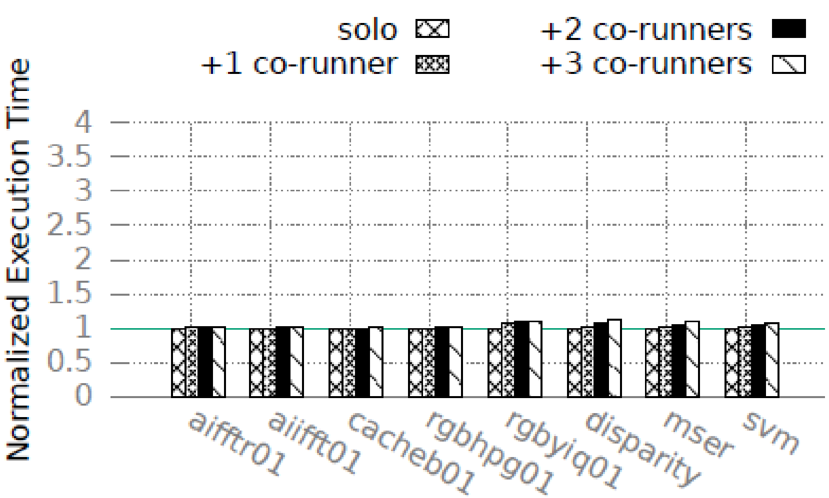
- Up to **21X slowdown** on Cortex-A15^{O} (8X in Nehalem)
- Writes generally cause more slowdowns
  - Due to write-backs

EEMBC, SD-VBS Workload

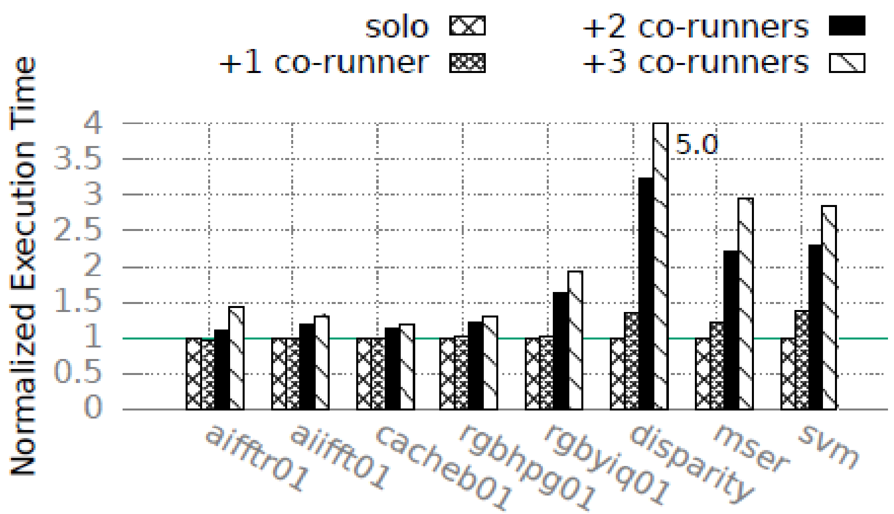
Benchmark	L1-MPKI	L2-MPKI	Description
EEMBC Automotive, Consumer [1]			
aifftr01	3.64	0.00	FFT (automotive)
aiifft01	3.99	0.00	Inverse FFT (automotive)
cacheb01	2.14	0.00	Cache buster (automotive)
rgbhpg01	1.59	0.00	Image filter (consumer)
rgbyiq01	3.81	0.01	Image filter (consumer)
SD-VBS: San Diego 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

EEMBC and SD-VBS



Cortex-A7 (in-order)



Cortex-A15[O] (out-of-order)

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

MSHR Contention

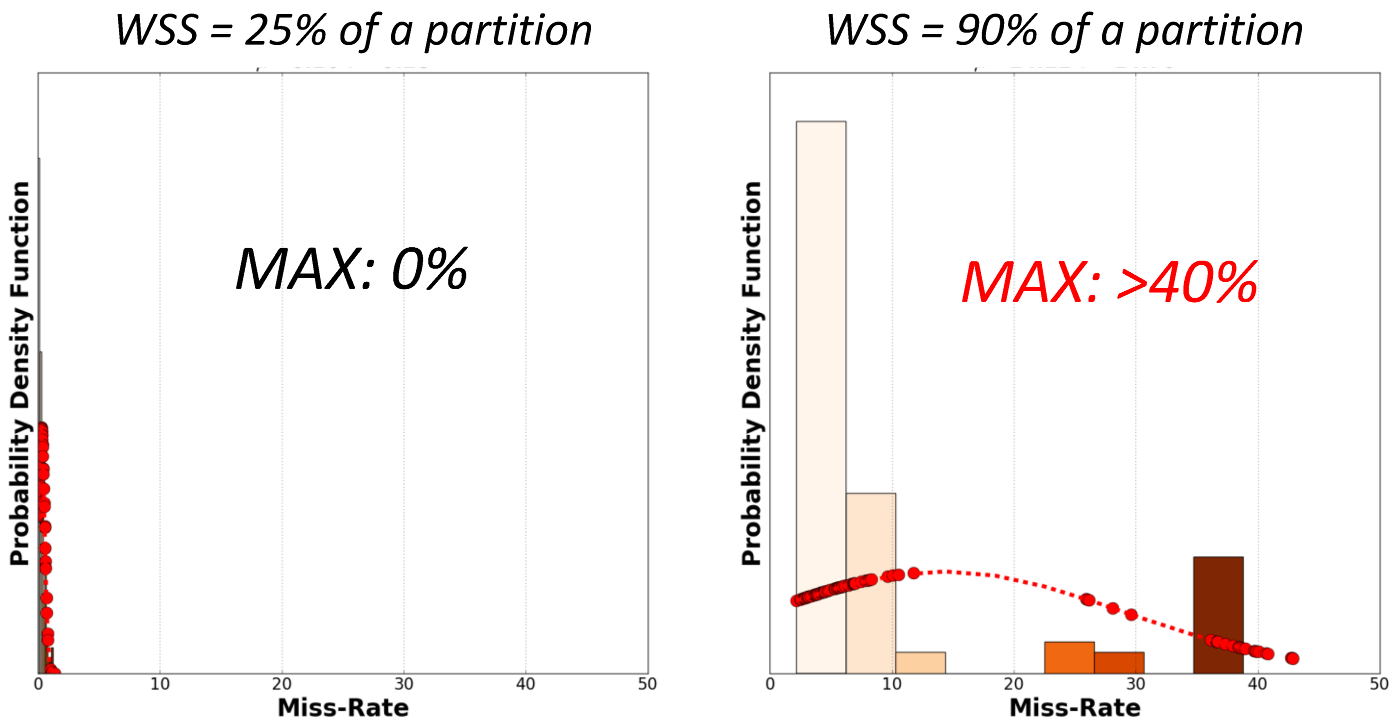
	Cortex-A7	**Cortex-A9**	**Cortex-A15[O,T]**	**Nehalem**
Local MLP	1	**4**	**6**	**10**
Global MLP	4	**4**	**11**	**16**

- Shortage of cache MSHRs lock up the cache
- LLC MSHRs are shared resources
  - 4cores x L1 cache MSHRs > LLC MSHRs
- Good news
  - Recent Intel processors seem immune to this problem
  - But …

Intel Cache Allocation Technology (CAT)

- CAT: Intel's cache management hardware support
  - Support **way-based partitioning**
  - Support flexible app-partition binding mechanism
- How effective is it in terms of isolation?
- We performed *preliminary evaluation* of CAT using an Intel Xeon E5-2658 v3 processor
  - Finding: even after partitioning, there seems to be significant *non-determinism*.

Run-to-Run Miss Rate Variation



- Experiment
  - Measure the LLC miss-rate of a **cache partition fitting** benchmark
- **Result**
  - **Non zero cache miss, high run-to-run variation (2 – 43%)**

Intel CAT: Challenges

- Complex cache organization and addressing
  - Use of multiple cache slices
  - Use of undisclosed hash function for mapping

   Difficult to remove conflict misses (page coloring)
- Effect of cache replacement algorithm
  - Partition migration requires flushing (*)
  - Set dueling? (**)
- Difficult to analyze and control the worst-case
  - Not ideal for real-time systems.

(*) Meng et al., "vCAT: Dynamic Cache Management using CAT Virtualization." RTAS17
(**) http://blog.stuffedcow.net/2013/01/ivb-cache-replacement/;

Summary

- Cache partitioning may not be as predictable and deterministic as we believed (wanted)

- We still can overserve inter/intra-core interference even after partitioning
  - MSHR contention
  - Conflict misses

Recommendations on Real-Time Friendly Multicore Architectures

- More visibility to software
  - Per-core/app monitoring of shared resources
    - DRAM access count/row hit-miss/latency,
    - LLC access/miss/occupancy/latency
    - Examples: Intel CMT (LLC occupancy), MBM (dram b/w)
  - Mapping functions
    - Cache slice/set mapping, DRAM row/bank/rank mapping
    - Examples: AMD memory controller

Recommendations on Real-Time Friendly Multicore Architectures

- More control by software
  - Control shared hardware resources
    - Examples: [Valsan16] (Cache MSHRs), Intel CAT (cache space)
  - Tag additional information
    - On instructions and memory
    - Examples: ARM TrustZone (secure memory)
    - Criticality, determinism, reliability (rowhammer) **new hardware/software interface is needed!**

Thank You

This presentation is based on the following publications:

- Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. "Addressing Isolation Challenges of Non-blocking Caches for Multicore Real-Time Systems." *Real-Time Systems* (In minor revision)
- Prathap Kumar Valsan, Heechul Yun, Farzad Farshchi. "Taming Non-blocking Caches to Improve Isolation in Multicore Real-Time Systems." *IEEE Intl. Conference on Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2016. ***Best Paper Award.***
- Heechul Yun, Prathap Kumar Valsan. Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms. *Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT)*, 2015

IsolBench

https://github.com/CSL-KU/IsolBench

Summary

- Evaluated the effect of cache partitioning
  - On modern COTS multicore architectures
  - Based on page coloring
  - Based on Intel CAT (way partitioning)
- Findings
  - Cache partitioning does **not** ensure cache (hit) performance isolation
  - **MSHR contention** and other issues
- **IsolBench**
  - Developed synthetic benchmarks, test scripts, kernel patches to evaluate multicore processors
  - https://github.com/CSL-KU/IsolBench

Discussion

- Why is MSHR contention important?
  - Timing attack
    - Malicious code can significantly inflate the execution times of critical tasks on *different cache partitions*
  - Memory intensive applications are increasing.
    - E.g., vision, artificial intelligence, machine learning.

- What are the other sources of contention to worry?
  - Cache, system bus bandwidth
  - DRAM bandwidth (*), banks mapping/allocation (**), controller scheduling algorithms (***)

(*) Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. MemGuard: Memory Bandwidth Reservation S ystem for Efficient Performance Isolation in Multi-core Platforms. *IEEE RTAS,* IEEE, 2013
(**) Heechul Yun, Renato Mancuso, Zheng-Pei Wu, Rodolfo Pellizzoni. "PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms." *IEEE RTAS,* 2014
(***) Prathap Kumar Valsan, Heechul Yun. MEDUSA: A Predictable and High-Performance DRAM Controller for Multicore b ased Embedded Systems *IEEE CPSNA*, 2015