Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms

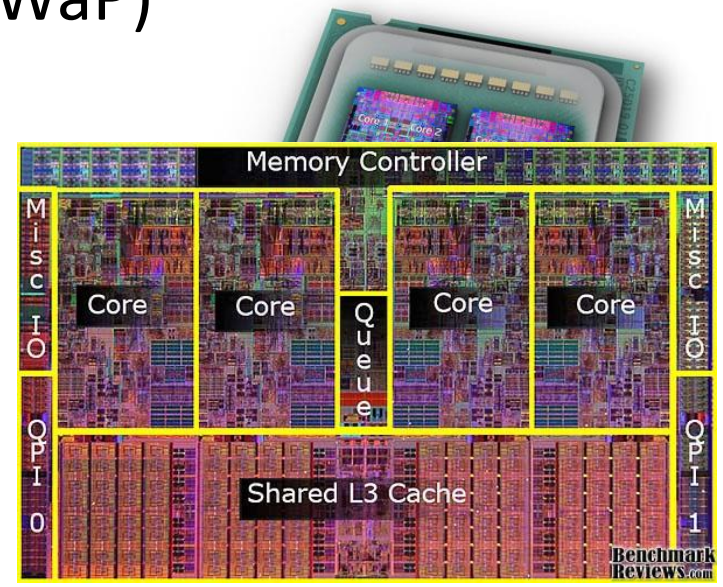
Heechul Yun, Prathap Kumar Valsan

University of Kansas

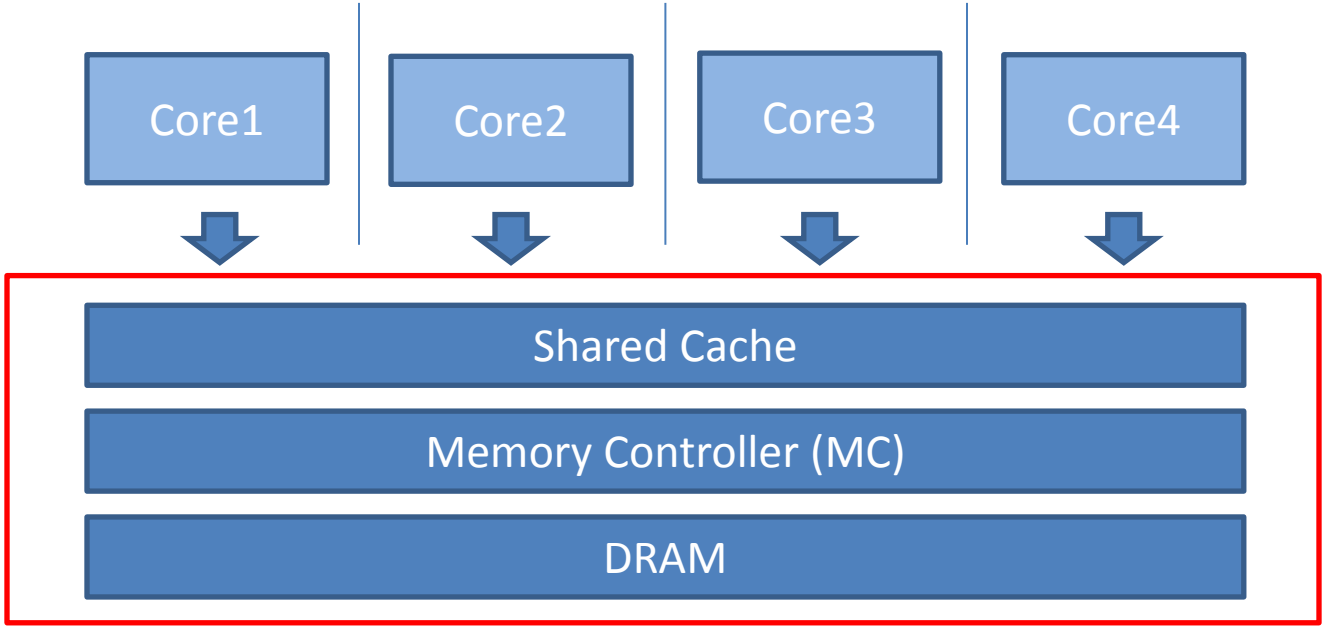
KU THE UNIVERSITY OF KANSAS

Multicore on Embedded Systems

- Benefits
  - Lots of sensor data to process
  - More performance, less cost
  - Save space, weight, power (SWaP)



Challenge: **Isolation**



- Hardware resources are contented among the cores
- Resulting in unpredictable performance
- We focus on the shared cache (LLC)

KU THE UNIVERSITY OF KANSAS

Cache Partitioning

- Page coloring
  - Software (OS) based partitioning technique
  - Partition cache-sets by manipulating physical addresses
- Way partitioning
  - Hardware support is required. (e.g., P4080)
  - Partition cache-ways
- **How effective for performance isolation?**

KU THE UNIVERSITY OF KANSAS

This Talk

- Evaluates the isolation effect of cache partitioning on three COTS multicore platforms

- We got some interesting results.
  - *W/o cache partitioning*, we observe up to **104X execution time increase** due to cache interference
  - *Even after partitioning the cache*, we observe **up to 14X exec. time increase** of a task running on a dedicated core accessing its dedicated cache partition

KU THE UNIVERSITY OF KANSAS

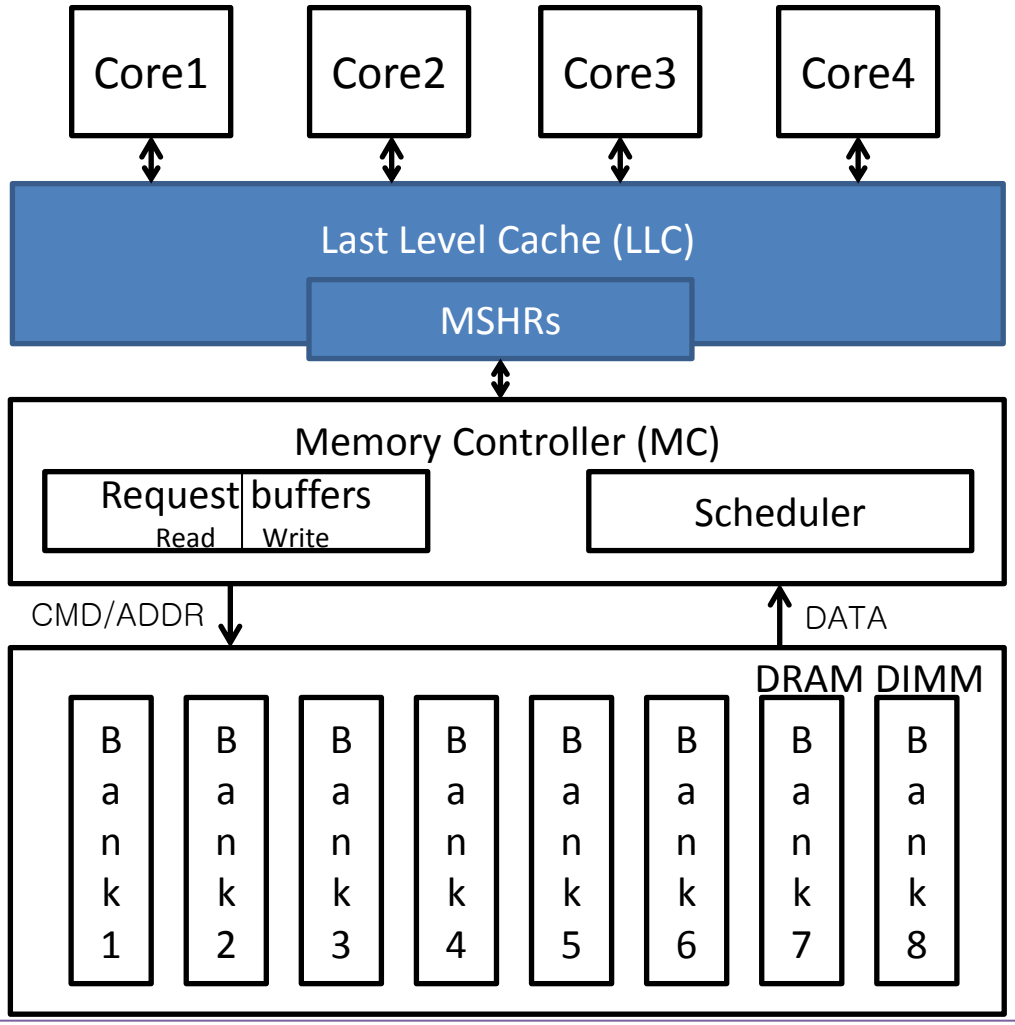
Outline

- Motivation
- **Background**
  - **COTS multicore architecture**
  - **Non-blocking cache**
- Evaluation
- Conclusion

Memory-Level Parallelism (MLP)

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

COTS Multicore Architecture



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

Non-blocking Caches[Kroft'81]

- Supporting multiple outstanding cache-misses
  - The core doesn't stall on a cache-miss
  - Use Miss-status-holding registers (MSHRs)
- How it works?
  - On a cache-miss, the request is stored on an empty MSHR and sent to the lower hierarchy
  - On receiving the data, the cache is updated and the MSHR entry is cleared

[Kroft'81] D. Kroft. "Lockup-free instruction fetch/prefetch cache organization," ISCA'81

Non-blocking Caches[Kroft'81]

- 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

[Kroft'81] D. Kroft. "Lockup-free instruction fetch/prefetch cache organization," ISCA'81

Outline

- Motivation
- Background
- **Evaluation**
  - **Used Platforms**
  - **Local/Global MLP**
  - **Experiment Results**
- Conclusion

Evaluation Platforms

	Cortex-A7	Cortex-A15	Nehalem
Core	4core @ 0.6GHz In-order	4core @ 1.6GHz Out-of-order	4core @ 2.8GHz Out-of-order
LLC	512KB	2MB	8MB
DRAM	2GB, 16banks	2GB. 16banks	4GB, 16banks

- COTS multicore platforms
  - Odroid-XU-E: 4x Cortex-A7 and 4x Cortex-A15
  - Dell desktop: Intel Xeon quad-core

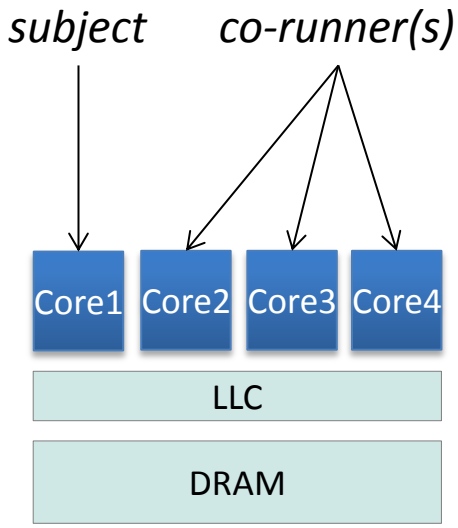
Identified MLP

	Cortex-A7	Cortex-A15	Nehalem
local MLP	1	6	10
global MLP	4	11	16

(See paper for our experimental identification methods)

- Local MLP
  - MLP of a core
- Global MLP
  - MLP of the shared memory hierarchy
  - min(#of LLC MSHRs, # of DRAM banks)

Cache Interference Experiments



- Measure the performance of the 'subject'
  - (1) alone, (2) with co-runners
  - (a) with and (b) w/o cache partitioning (equal partition)

Workload

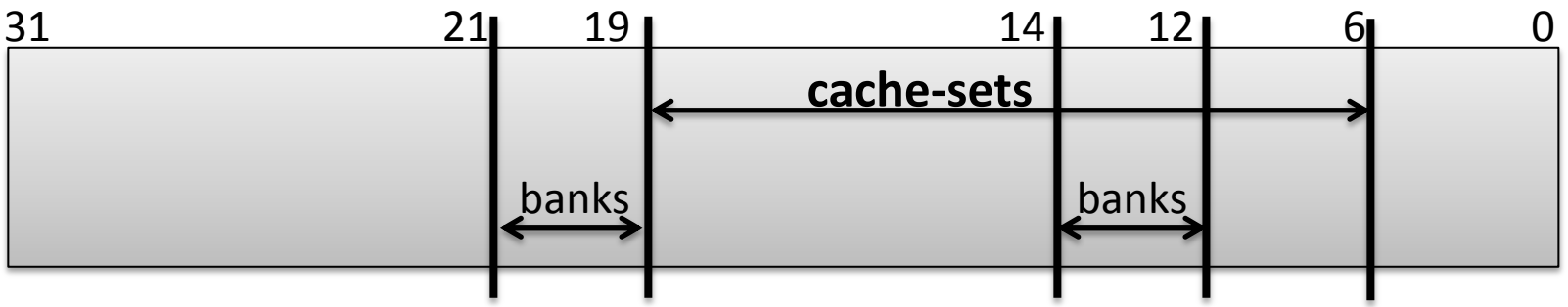
Experiment	Subject	Co-runner(s)
Exp. 1	Latency(LLC)	BwRead(DRAM)
Exp. 2	BwRead(LLC)	BwRead(DRAM)
Exp. 3	BwRead(LLC)	BwRead(LLC)
Exp. 4	Latency(LLC)	BwWrite(DRAM)
Exp. 5	BwRead(LLC)	BwWrite(DRAM)
Exp. 6	BwRead(LLC)	BwWrite(LLC)

Working-set size: (LLC) <= ¼ LLC, (DRAM) => 2X LLC

- Latency
  - A linked-list traversal, data dependency
- Bandwidth
  - An array reads or writes, no data dependency

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

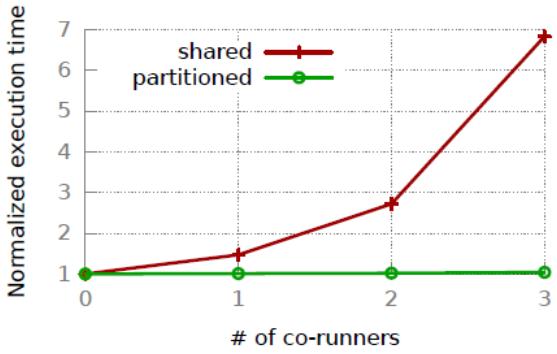
PALLOC Interface

- Example cache partitioning

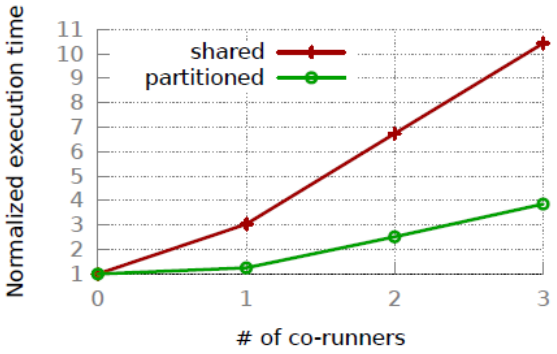
```
# echo 0x00003000 > /sys/kernel/debug/palloc/palloc_mask
  → bits: 12, 13
# cd /sys/fs/cgroup
# mkdir core0 core1 core2 core3
  ➔ create 4 cgroup partitions
# echo 0 > core0/palloc.bins
  ➔ allocate pages whose addr. bit 13 and bit 12 are both 0 (00)
# echo 1 > core1/palloc.bins
# echo 2 > core2/palloc.bins
# echo 3 > core3/palloc.bins
```

https://github.com/heechul/palloc

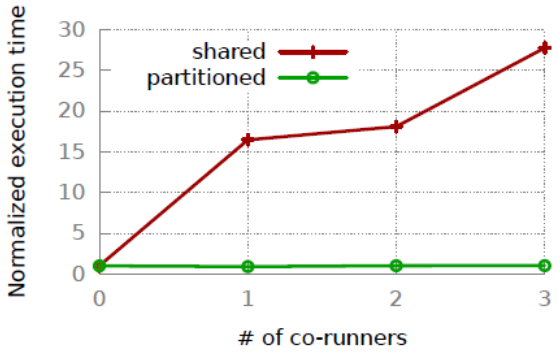
Exp1: Latency(LLC) vs. BwRead(DRAM)



(a) Cortex-A7



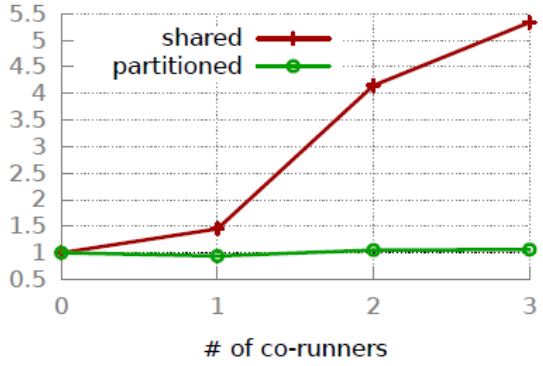
(b) Cortex-A15



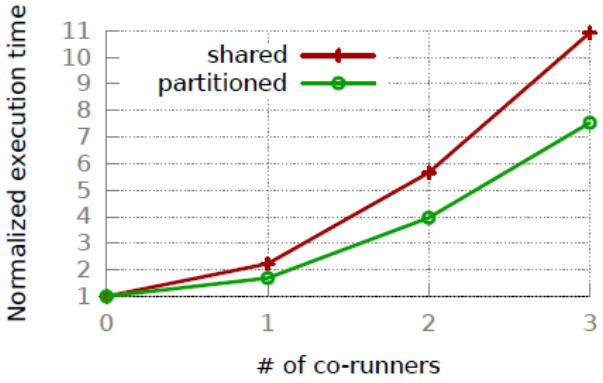
(c) Nehalem

- No interference on Cortex-A7 and Nehalem
- On Cortex-A15, Latency(LLC) suffers 3.8X slowdown even after partitioning the LLC

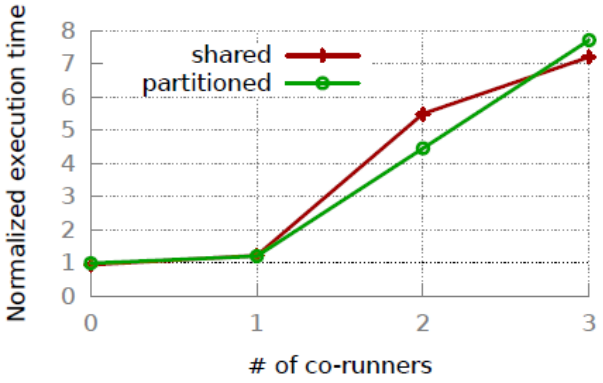
Exp2: BwRead(LLC) vs. BwRead(DRAM)



(a) Cortex-A7



(b) Cortex-A15



(c) Nehalem

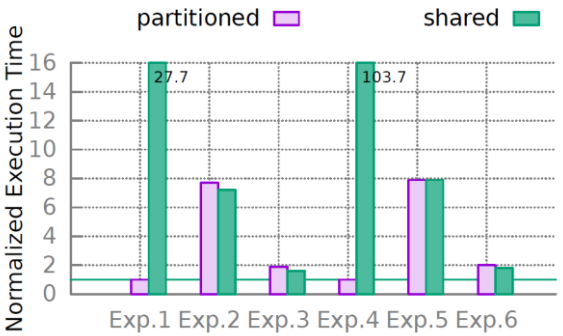
	Cortex-A7	Cortex-A15	Nehalem
local MLP	1	6	10
global MLP	4	11	16

- The **isolation benefit** of cache partitioning is almost completely **disappeared** on out-of-order cores
- Due to the shortage of MSHRs → **MSRH contention**

Summary of the Experiments

(a) Cortex-A7

(a) Cortex-A15



(a) Nehalem

- Write-intensive co-runners cause more delays
  - w/o cache partitioning: 104X slowdown
  - with cache partitioning: 14X slowdown
- MSHR contention is the main cause
  - Longer stay in the MSHR → more MSHR shortage

KU THE UNIVERSITY OF KANSAS

Conclusion

- We quantified the isolation effect of cache partitioning on three COTS multicore processors
- We found cache partitioning does not ensure predictable cache (hit) performance due to **MSHR contention**
- Future Work
  - More evaluation with macro benchmarks
  - Mechanism to avoid MSHR contention

KU THE UNIVERSITY OF KANSAS

Thank You