Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems

Heechul Yun⁺, Rodolfo Pellizzoni*, Prathap Kumar Valsan⁺

⁺University of Kansas
*University of Waterloo
High-Performance Multicores for Embedded Real-Time Systems

• Why?
  – Intelligence → more performance
  – Space, weight, power (SWaP), cost
Challenge: Shared Memory Hierarchy

- Hardware resources are contented among the cores
- Tasks can suffer significant memory interference delays
Memory Interference Delay

- Can be extremely high in the worst-case

![Diagram showing memory interference delay with normalized execution time for different architectures and configurations.](image-url)
Modeling Memory Interference

• Common (false) assumptions on COTS systems
  – A single resource
    • Reality: multiple parallel resources (banks)
  – A constant memory service cost
    • Reality: it varies depending on the DRAM bank state
  – Round-robin arbitration, in-order processing
    • Reality: FR-FCFS can re-order requests
  – Both read and write requests are treated equally
    • Reality: writes are buffered and processed opportunistically
  – One outstanding request per core
    • Reality: an out-of-order core can generate parallel reqs.

Addressed in [Kim’14]

Addressed in This Work

Our Approach

- Realistic memory interference model for COTS systems
  - Memory-level parallelism (MLP) in COTS architecture
  - Write-buffering and opportunistic batch processing in MC

- DRAM bank-partitioning
  - Reduce interference

- Delay analysis
  - Compute worst-case memory interference delay of the task under analysis
Outline

• Motivation

• Background
  – COTS multicore architecture
  – DRAM organization
  – Memory controller

• Our approach

• 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: Multiple banks
DRAM Organization

- Core1
- Core2
- Core3
- Core4

- Bank 1
- Bank 2
- Bank 3
- Bank 4

- LLC
- Memory Controller (MC)
- DRAM DIMM

**Mess**

- intra-bank conflicts
- Inter-bank conflicts
• Private banking
  – OS kernel allocates pages from dedicated banks for each core

Eliminate intra bank conflicts
Bank Access Cost

A DRAM Bank

READ (Bank 1, Row 3, Col 7)

- State dependent access cost
  - Row hit: fast
  - Row miss: slow
Memory Controller

Memory requests from cores

Read request buffer

Write request buffer

Bank 1 scheduler

Bank 2 scheduler

Bank N scheduler

Channel scheduler

DRAM chip

Writes are buffered and processed opportunistically
“Intelligent” Read/Write Switching

• Intuition
  – Writes are not in the critical path. So buffer and process them opportunistically

• Algorithm [Hansson’14]
  – If there are reads, process them unless the write buffer is almost full (high watermark)
  – If there’s no reads and there is enough buffered writes (low watermark), process the writes until reads arrive

[Hansson’14] Hansson et al., “Simulating DRAM controllers for future system architecture exploration,” ISPASS’14
FR-FCFS Scheduling [Rixner’00]

• Priority order
  1. Row hit request
  2. Older request

• Maximize memory throughput

Outline

• Motivation
• Background
• Our approach
  – System model
  – Delay analysis
• Evaluation
• Conclusion
System Model

• Task
  – Solo execution time $C$
  – Memory demand (#of LLC misses): $H$

• Core
  – Can generate multiple, but **bounded**, parallel requests
    • Upper-bounded by L1 cache’s MSHR size

• Cache (LLC)
  – Assume no cache-level interference
    • Core-private or partitioned LLC
    • No MSHR contention

• DRAM controller
  – Efficient FR-FCFS scheduler, open-page policy
  – Separate read and write request buffer
    • Watermark scheme on processing writes
Delay Analysis

• Goal
  – Compute the worst-case memory interference delay of a task under analysis

• Request driven analysis
  – Based on the task’s own memory demand: \( H \)
  – Compute worst-case per request delay: \( RD \)
  – Memory interference delay = \( RD \times H \)

• Job driven analysis
  – Based on the other tasks’ memory requests over time
  – See paper
Key Intuition #1

- The number of competing requests $N_{rq}$ is **bounded**
  - Because the number of per-core parallel requests is bounded.
  - Example
    - Cortex-A15’s per-core bound = 6
    - $N_{rq} = 6 \times 3$ (cores) = 18
Key Intuition #2

• DRAM sub-commands of the competing memory requests are *overlapped*
  – Much *less pessimistic* than [Kim’14], which simply sums up each sub-command’s maximum delay
  – See paper for the proof
Key Intuition #3

- The worst-case delay happens when
  - The read buffer has $Nrq$ requests
  - And the write request buffer just becomes full
    - Start a write batch
  - Then the read request under analysis arrives

$RD = \text{read batch delay} + \text{write batch delay}$
Outline

• Motivation
• Background
• Our approach
• Evaluation
• Conclusion
Evaluation Setup

• Gem5 simulator
  – 4 out-of-order cores (based on Cortex-A15)
    • L2 MSHR size is increased to eliminate MSHR contention
  – DRAM controller model [Hansson’14]
  – LPDDR2 @ 533Mhz

• Linux 3.14
  – Use PALLOC[Yun’14] to partition DRAM banks and LLC

• Workload
  – Subject: *Latency, SPEC2006*
  – Co-runners: *Bandwidth (write)*
Results with the *Latency* benchmark

- **Ours(ideal):** Read only delay analysis (ignore writes)
- **Ours(opt):** assume writes are balanced over multiple banks
- **Ours(worst):** all writes are targeting one bank & all row misses

Results with *SPEC2006* Benchmarks

- Main source of pessimism:
  - The pathological case of write (LLC write-backs) processing
Conclusion

• Memory interference delay on COTS multicore
  – Existing analysis methods rely on strong assumptions

• Our approach
  – A realistic model of COTS memory system
    • Parallel memory requests
    • Read prioritization and opportunistic write processing
  – Request and job-driven delay analysis methods
    • Pessimistic but still can be useful for low memory intensive tasks

• Future work
  – Reduce pessimism in the analysis
Thank You