MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms

Apr 9, 2012

Heechul Yun[+], Gang Yao[+], Rodolfo Pellizzoni[*],
Marco Caccamo[+], Lui Sha[+]

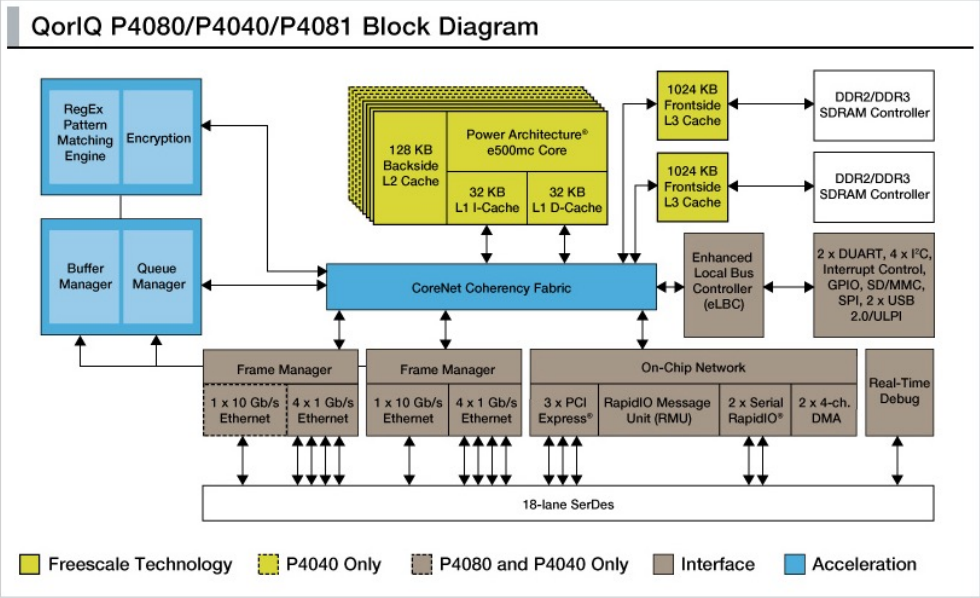
[+]University of Illinois at Urbana-Champaign
[*]University of Waterloo

Real-Time Applications

- Resource intensive real-time applications
  - Multimedia processing(*), real-time data analytic(**), object tracking
- Requirements
  - Need more performance and cost less ➔ Commercial Off-The Shelf (COTS)
  - Performance guarantee (i.e., temporal predictability and isolation)

(*) ARM, *QoS for High-Performance and Power-Efficient HD Multimedia*, 2010
(**) Intel, *The Growing Importance of Big Data and Real-Time Analytics*, 2012

Modern System-on-Chip (SoC)

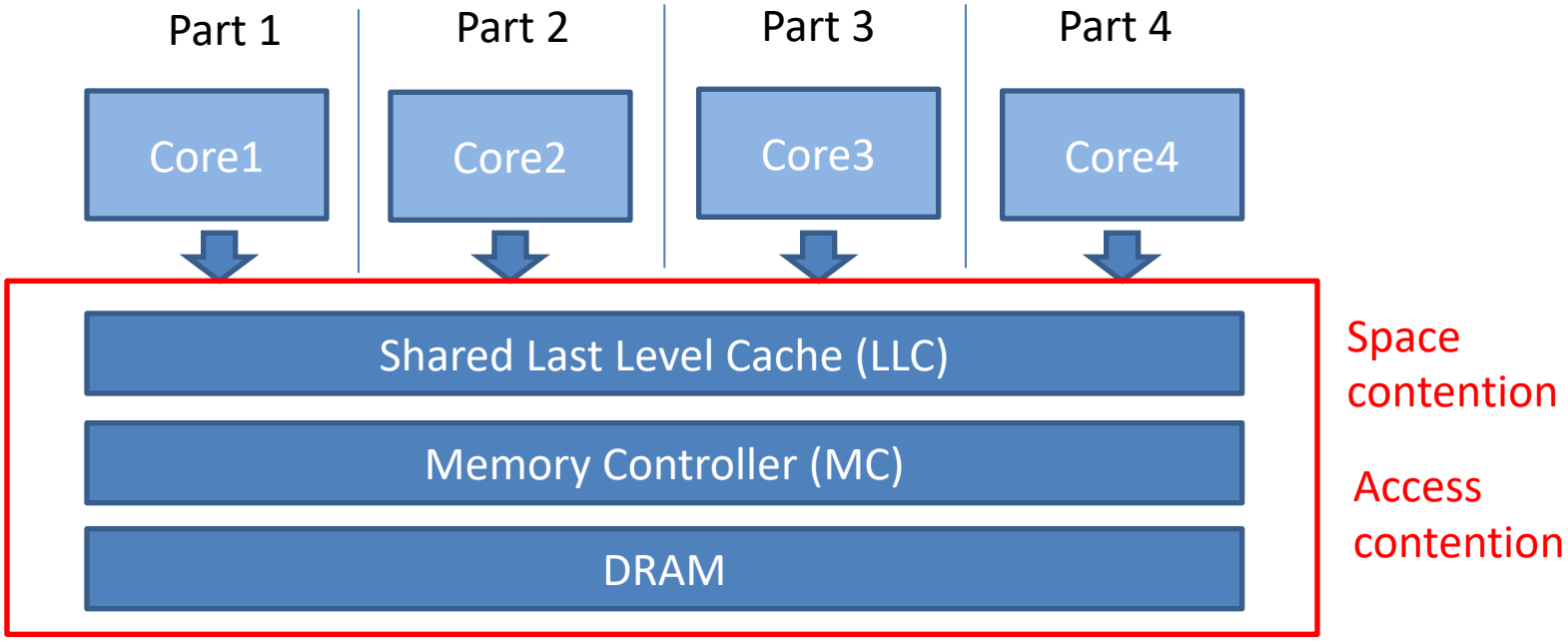


- More cores
  - Freescale P4080 has 8 cores
- More sharing
  - Shared memory hierarchy (LLC, MC, DRAM)
  - Shared I/O channels

⇨ *More performance*
Less cost

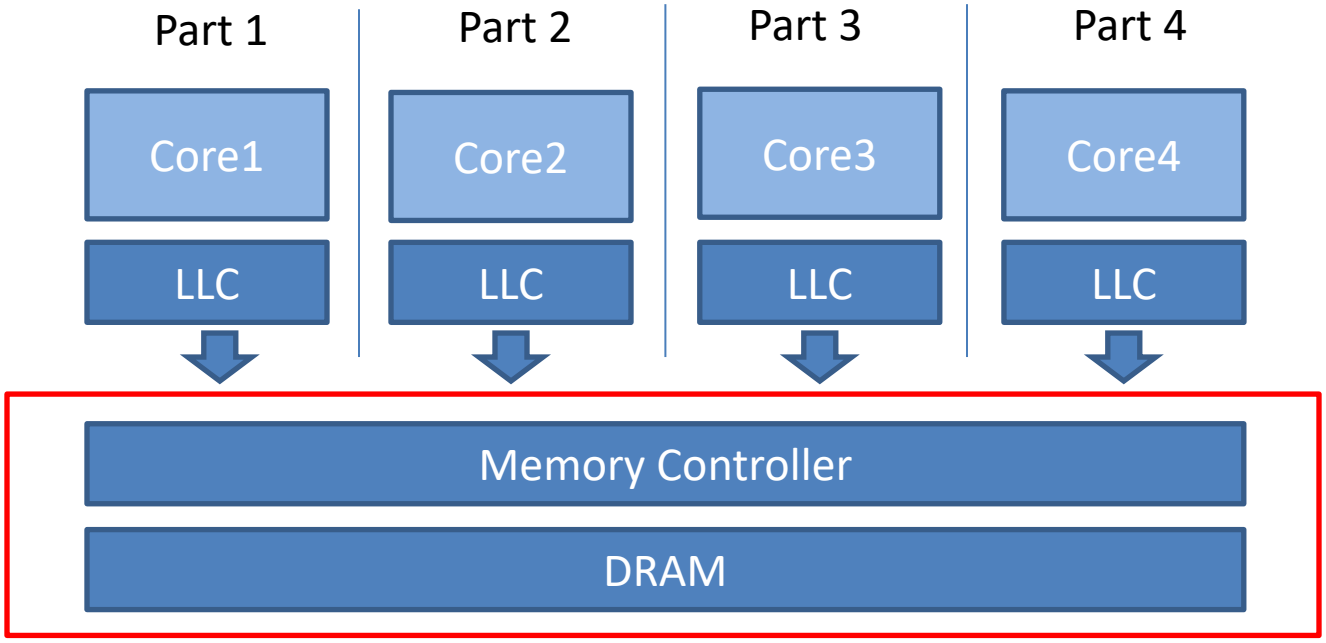
But, isolation?

Problem: Shared Memory Hierarchy



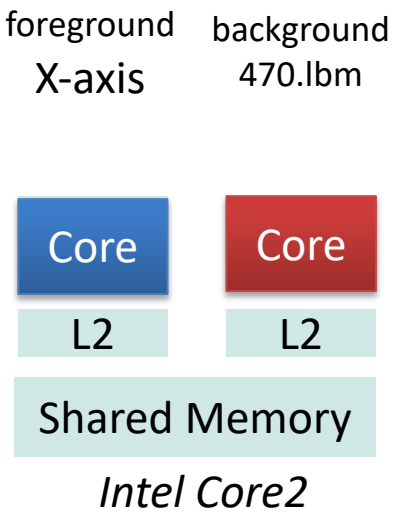
- Shared hardware resources
- OS has little control

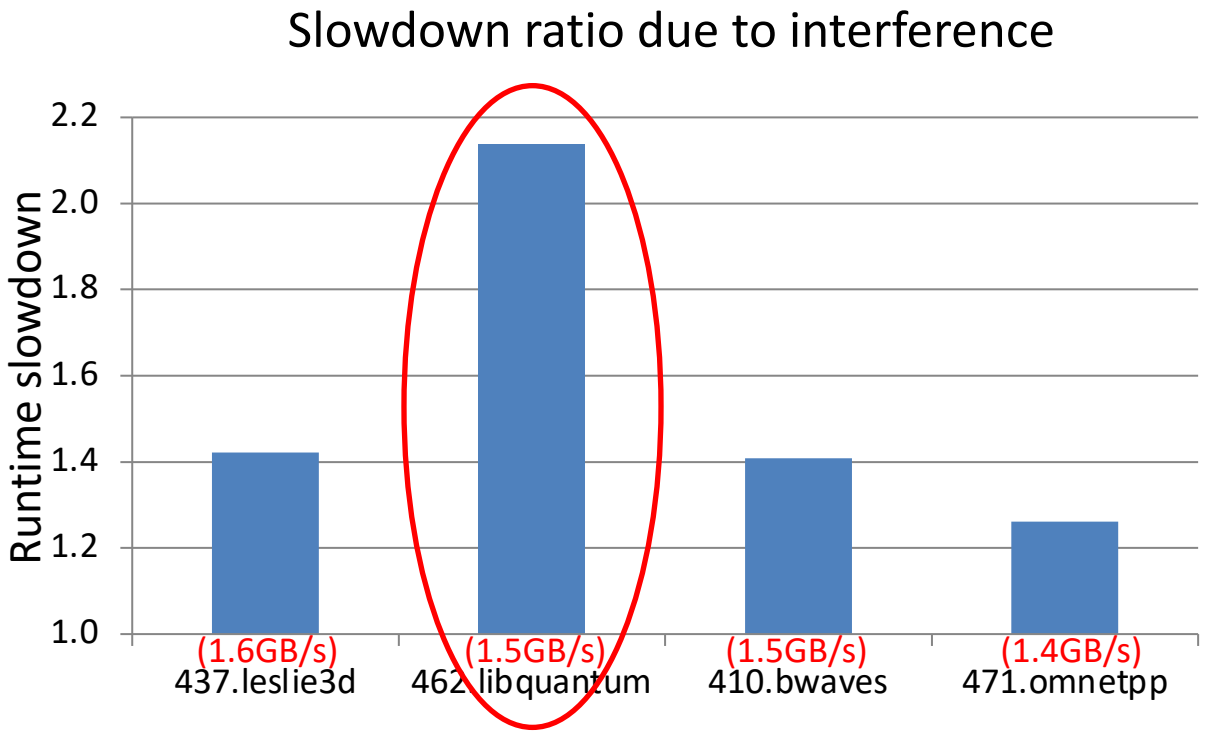
Memory Performance Isolation



- Q. How to guarantee worst-case performance?
  - Need to guarantee memory performance

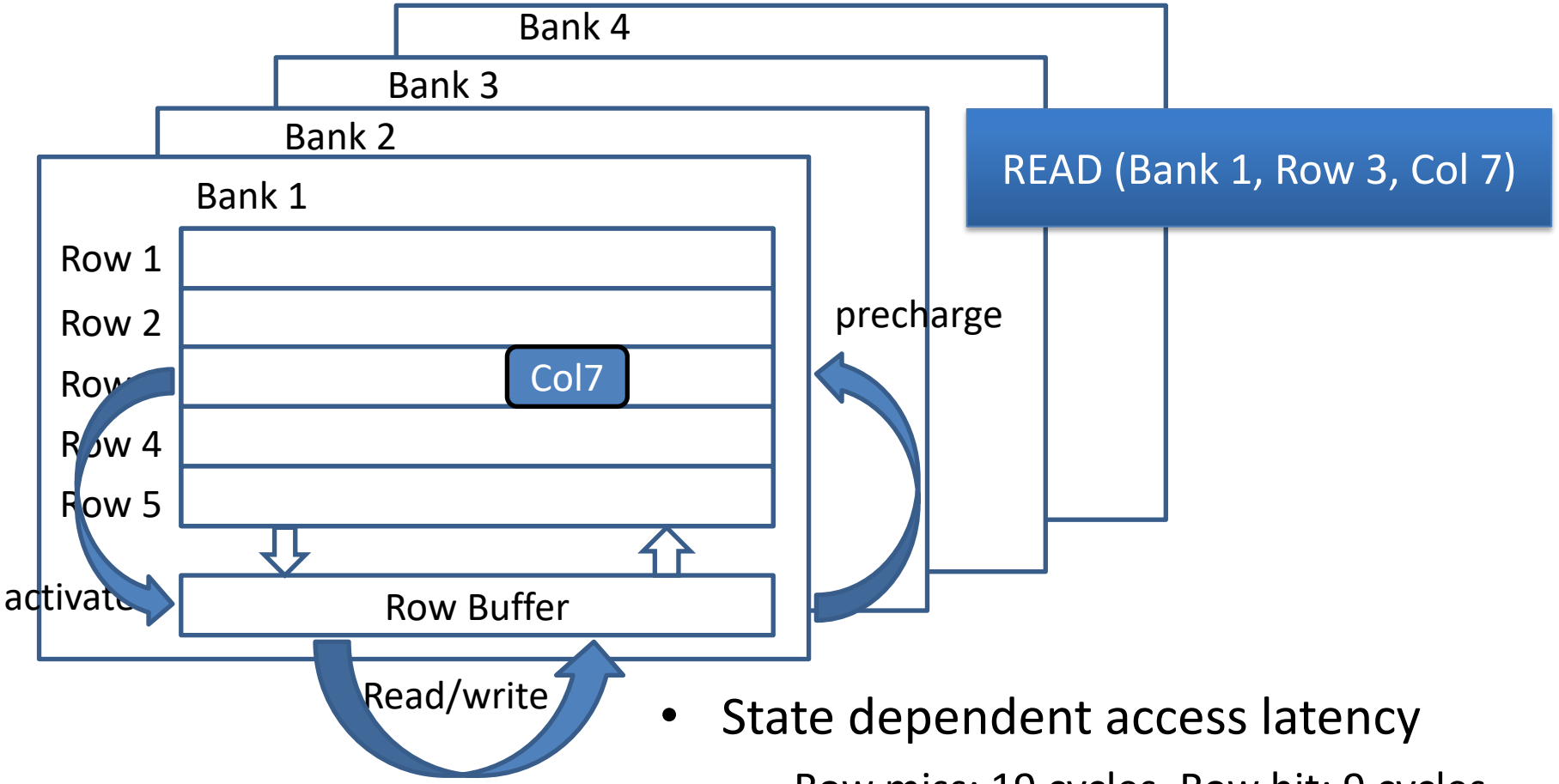
Inter-Core Memory Interference





- Significant slowdown (Up to 2x on 2 cores)
- Slowdown is not proportional to memory bandwidth usage

Background: DRAM Chip

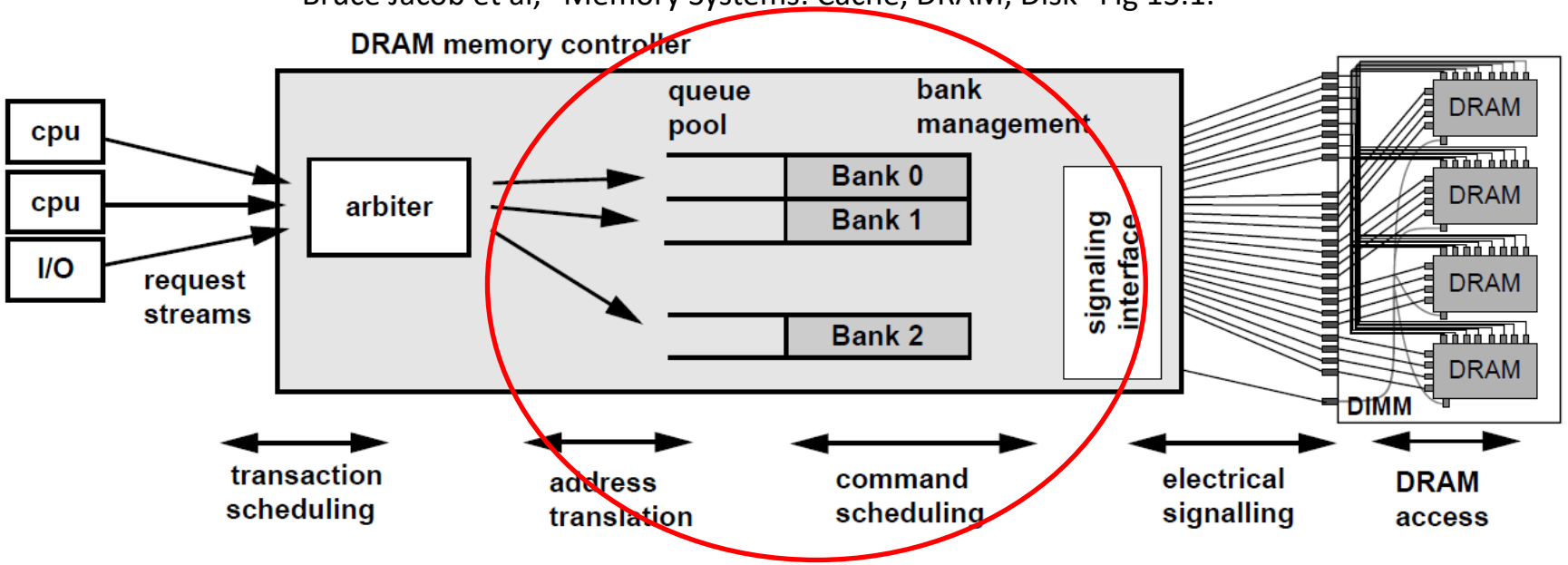


- State dependent access latency
  - Row miss: 19 cycles, Row hit: 9 cycles

(*) PC6400-DDR2 with 5-5-5 (RAS-CAS-CL latency setting)

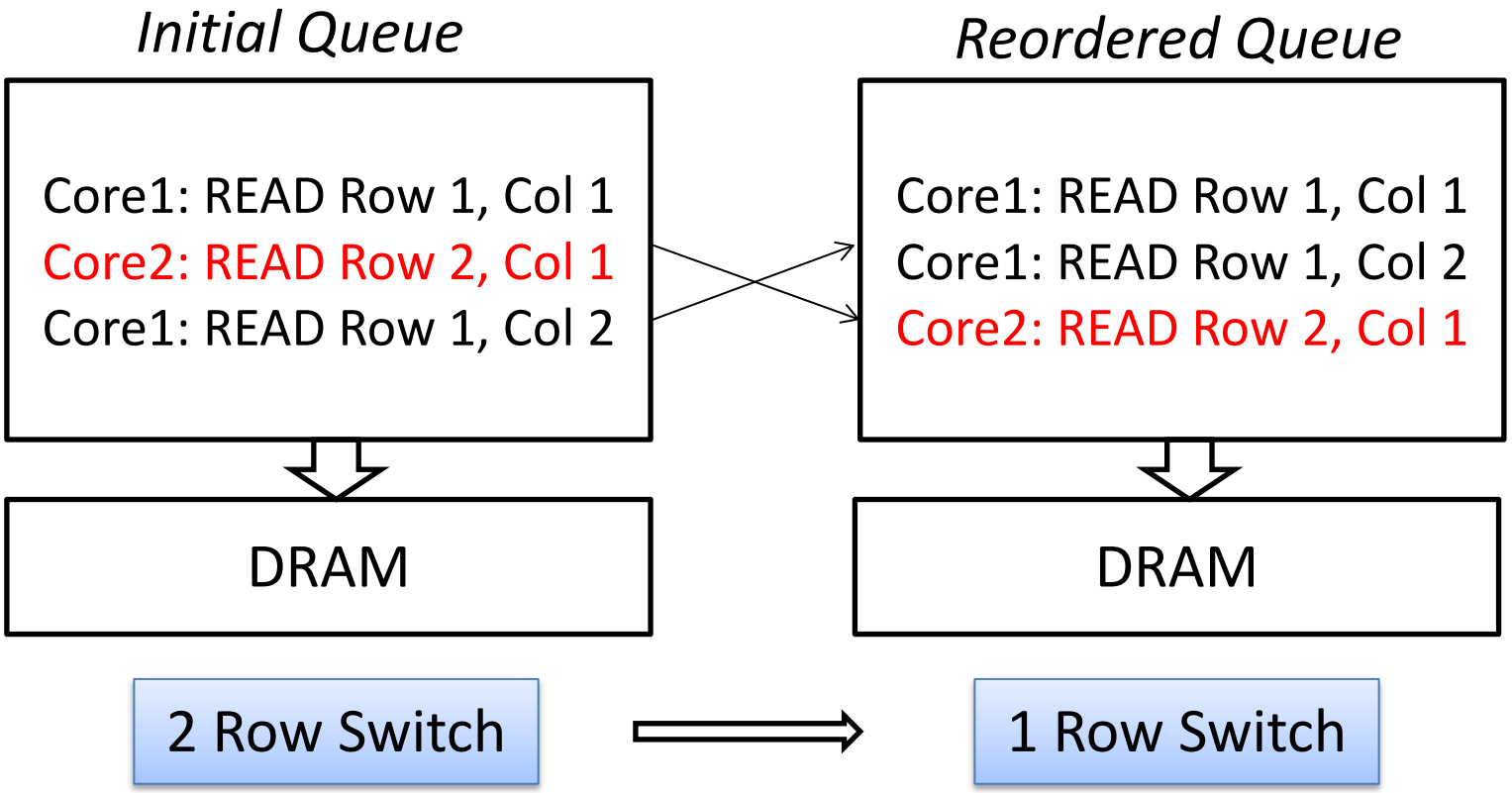
Background: Memory Controller(MC)

Bruce Jacob et al, "Memory Systems: Cache, DRAM, Disk" Fig 13.1.



- Request queue
  - Buffer read/write requests from CPU cores
  - Unpredictable queuing delay due to reordering

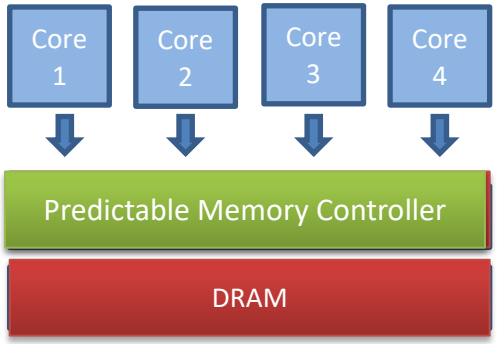
Background: MC Queue Re-ordering



- Improve row hit ratio and throughput
- Unpredictable queuing delay

Challenges for Real-Time Systems

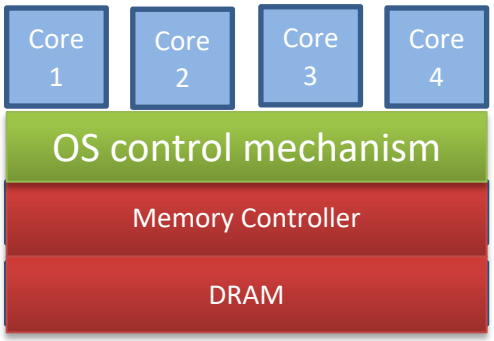
- Memory controller(MC) level queuing delay
  - Main source of interference
  - Unpredictable (re-ordering)
- DRAM state dependent latency
  - DRAM row open/close state



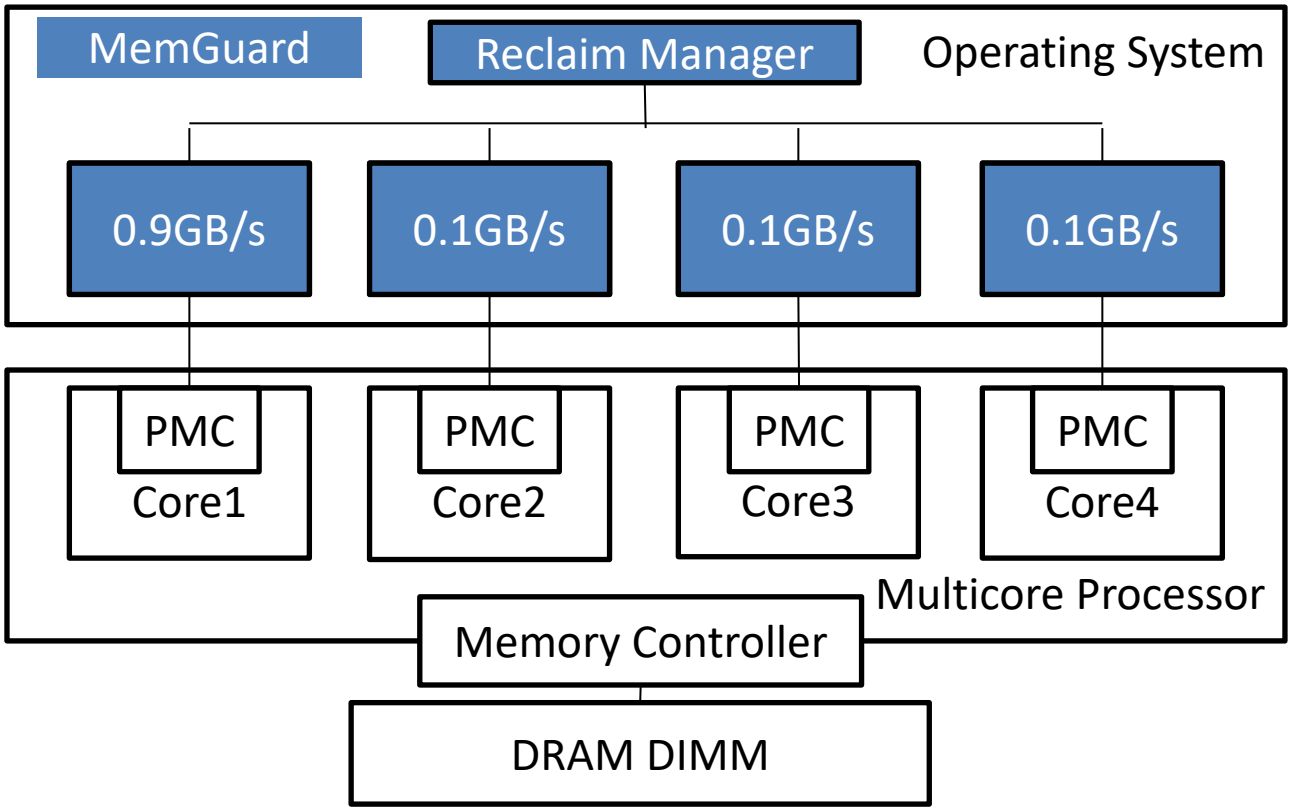
- State of Art
  - Predictable DRAM controller h/w: [Akesson'07] [Paolieri'09] [Reineke'11] → not in COTS

Our Approach

- OS level approach
  - Works on commodity h/w
  - Guarantees performance of each core
  - Maximizes memory performance if possible



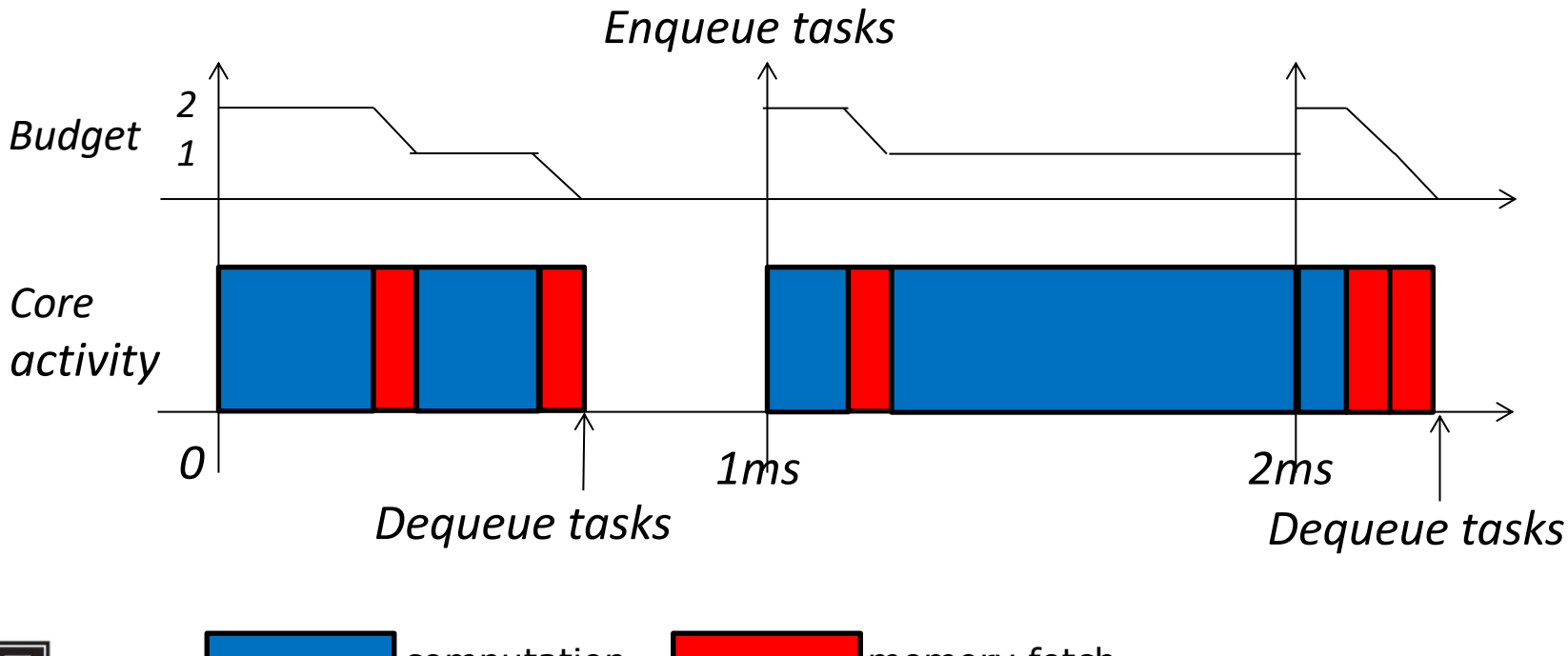
MemGuard



- Memory bandwidth reservation and reclaiming

Memory Bandwidth Reservation

- Idea
  - OS **monitor** and **enforce** each core's memory bandwidth usage



Memory Bandwidth Reservation

- Key Insight
  - B/W regulators control memory request rates
  - **(Cores)request rate $\leq$ (DRAM) service rate → (Memory controller) minimal queuing delay**

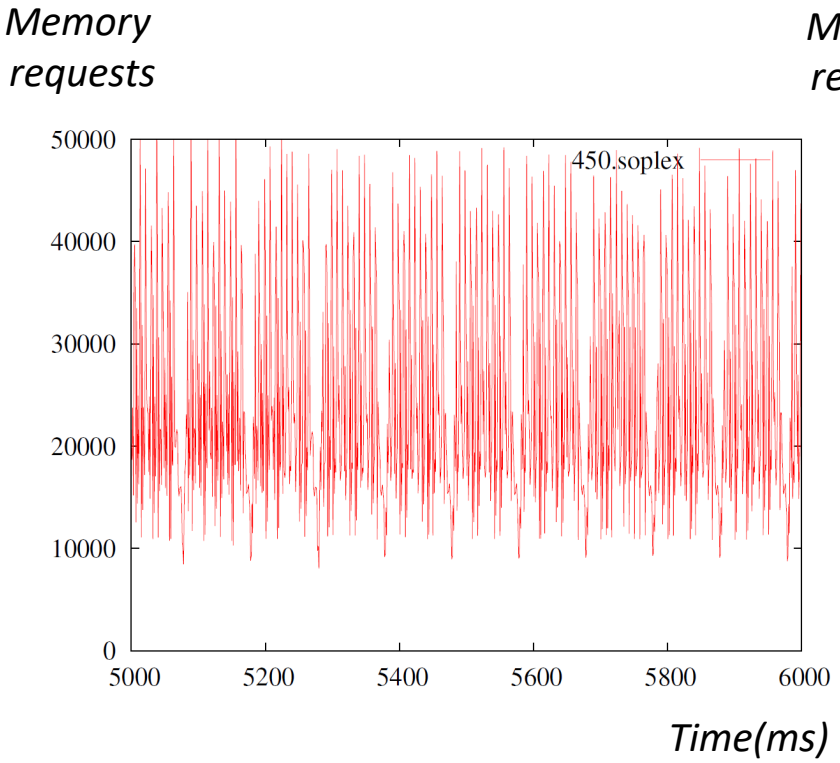
- System-wide reservation rule
  - up to the **guaranteed bandwidth r_{min}**
  - $\sum_{i=0}^{m} B_i \leq r_{min}$
    - m: #of cores
    - B_i: Core i's b/w reservation

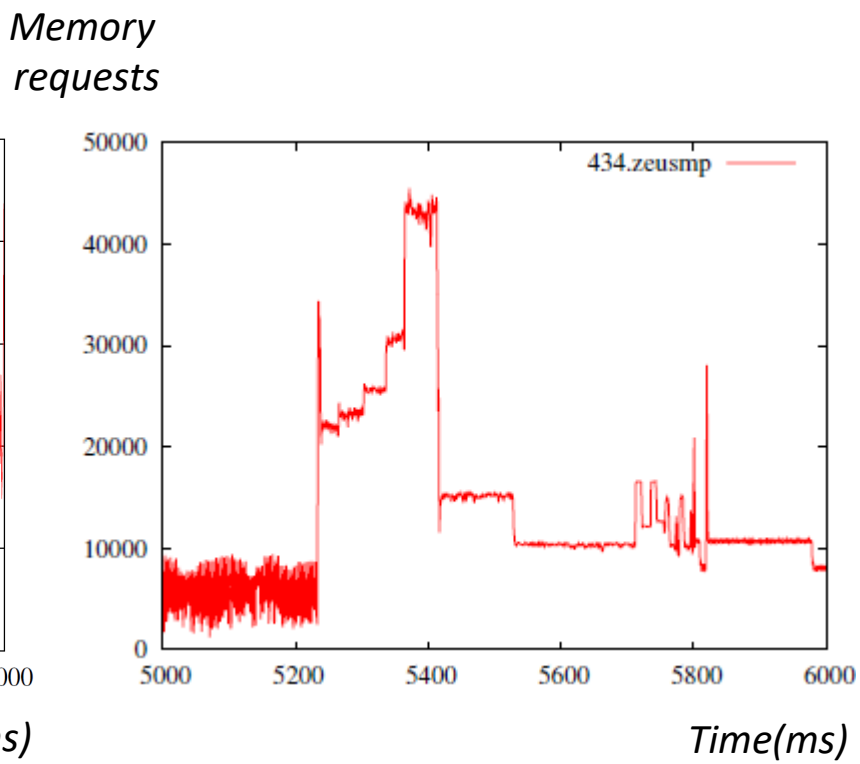
Guaranteed Bandwidth: r_{min}

- Worst-case DRAM performance (service rate)
  - All memory requests go to the same bank (no bank-level parallelism) and cause row miss
- Example (PC6400-DDR2*)
  - Peak B/W: 6.4GB/s
    - 64bytes I/O = 10ns, hide command latency by interleaving
  - Calculated guaranteed B/W: 1.3GB/s
    - PRE + ACT + RD + I/O (8x8bytes) = 47.5ns
  - Measured guaranteed B/W: **1.2GB/s**
- Performance Isolation
  - Sum of memory b/w reservation $\leq$ guaranteed b/w

(*) PC6400-DDR2 with 5-5-5 (RAS-CAS-CL latency setting)

Memory Access Pattern



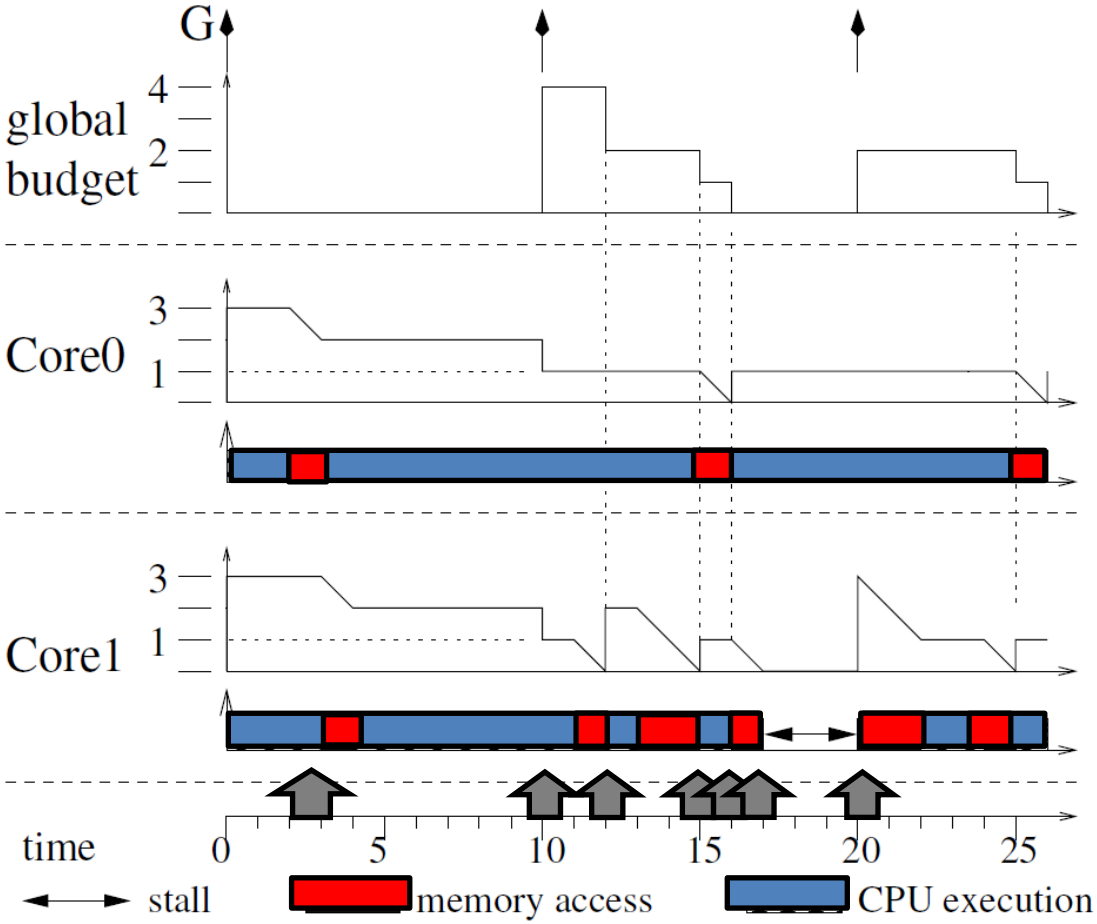


- Memory access patterns vary over time
- Static reservation is inefficient

Memory Bandwidth Reclaiming

- Key objective
  - Redistribute excess bandwidth to demanding cores
  - Improve memory b/w utilization
- Predictive bandwidth donation and reclaiming
  - Donate unneeded budget predictively
  - Reclaim on-demand basis

Reclaim Example



Time 0
- Initial budget 3 for both cores

Time 3,4
- Decrement budgets

Time 10 (period 1)
- Predictive donation (total 4)

Time 12,15
- Core 1: reclaim

Time 16
- Core 0: reclaim

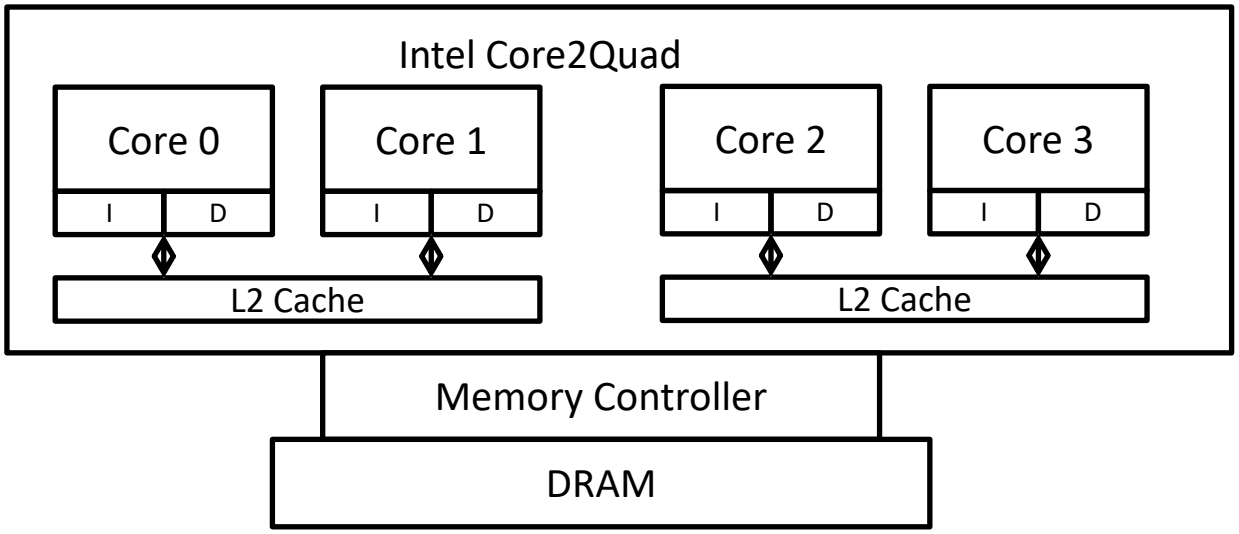
Time 17
- Core 1: no remaining budget; dequeue tasks

Time 20 (period 2)
- Core 0 donates 2
- Core 1 do not donate

Best-effort Bandwidth Sharing

- Key objective
  - Utilize **best-effort bandwidth** whenever possible
- Best-effort bandwidth
  - **After** all cores use their budgets (i.e., delivering guaranteed bandwidth), **before** the next period begins
- Sharing policy
  - Maximize throughput
  - Broadcast all cores to continue to execute

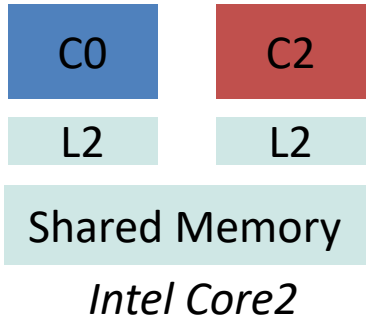
Evaluation Platform



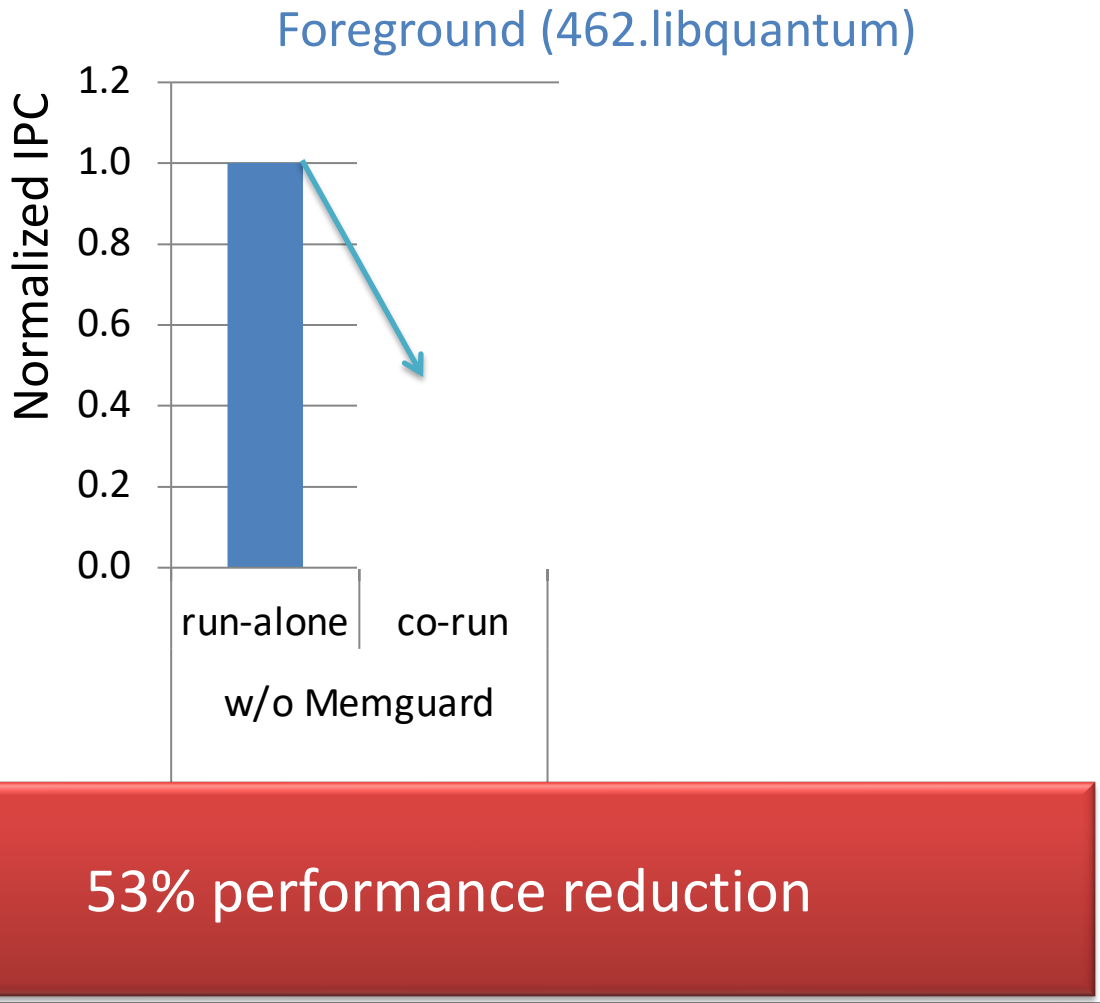
- Intel Core2Quad 8400, 4MB L2 cache, PC6400 DDR2 DRAM
  - Prefetchers were turned off for evaluation
  - Power PC based P4080 (8core) and ARM based Exynos4412(4core) also has been ported
- Modified Linux kernel 3.6.0 + MemGuard kernel module
  - https://github.com/heechul/memguard/wiki/MemGuard
- Used the entire 29 benchmarks from SPEC2006 and synthetic benchmarks

Evaluation Results

462.libquantum (foreground) memory hogs (background)



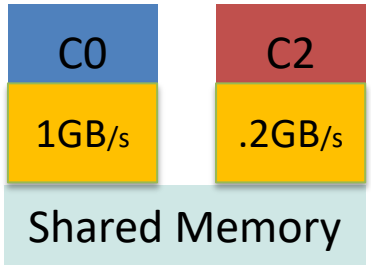
Intel Core2



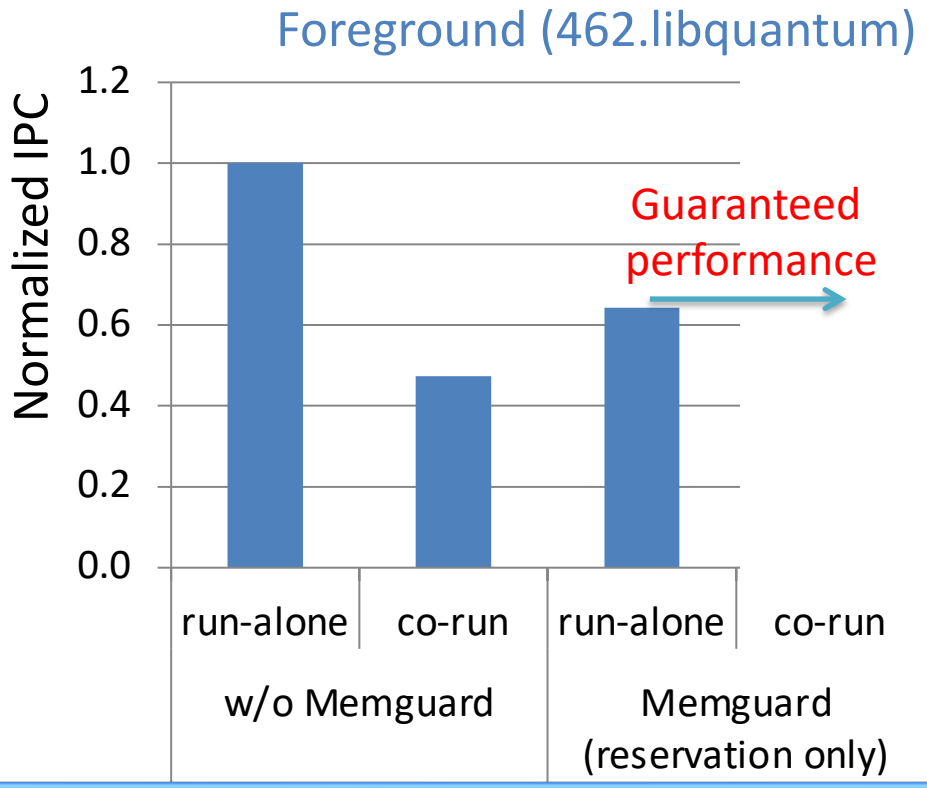
53% performance reduction

Evaluation Results

462.Libquantum (foreground) memory hogs (background)



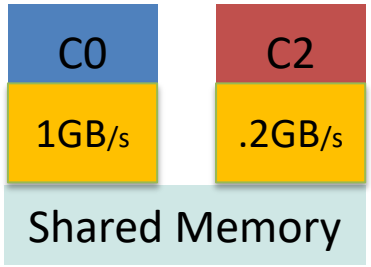
Intel Core2



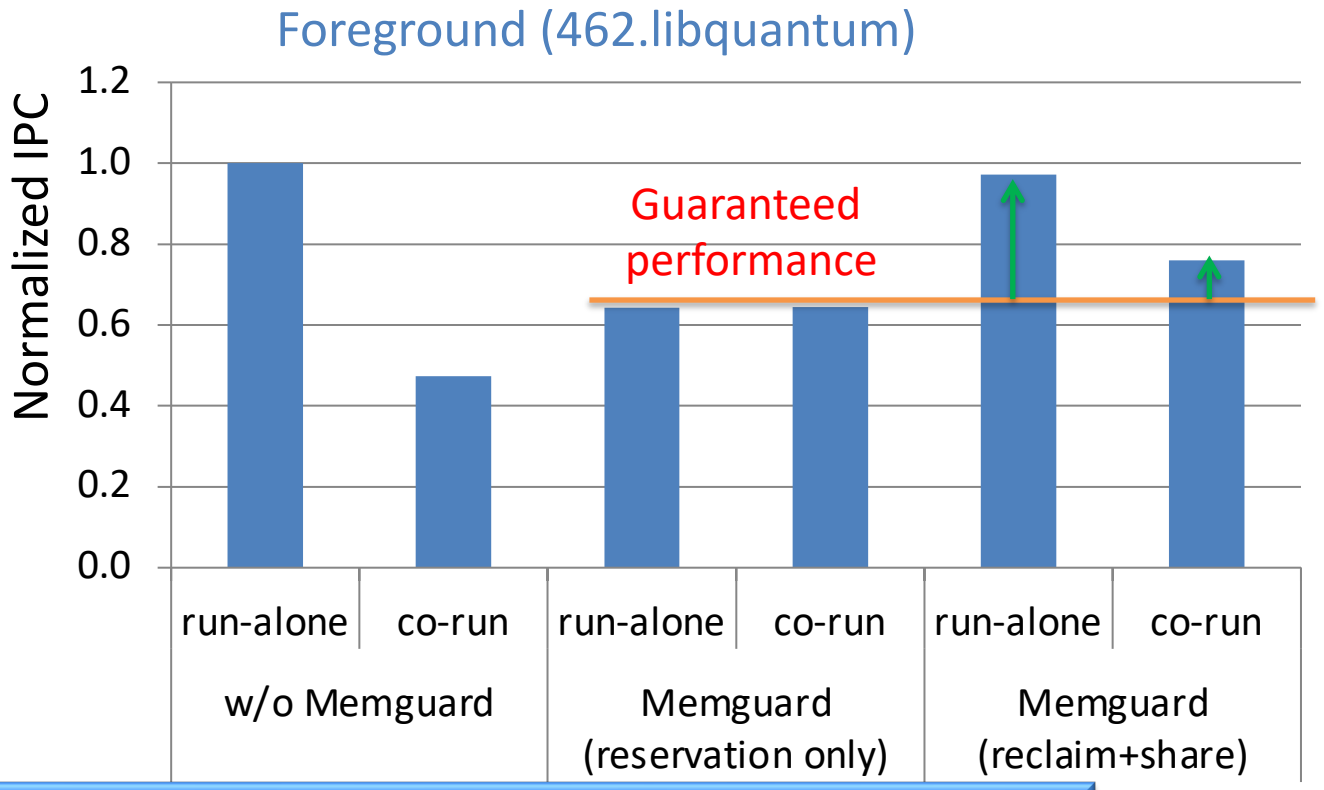
Reservation provides performance isolation

Evaluation Results

462.Libquantum (foreground) memory hogs (background)

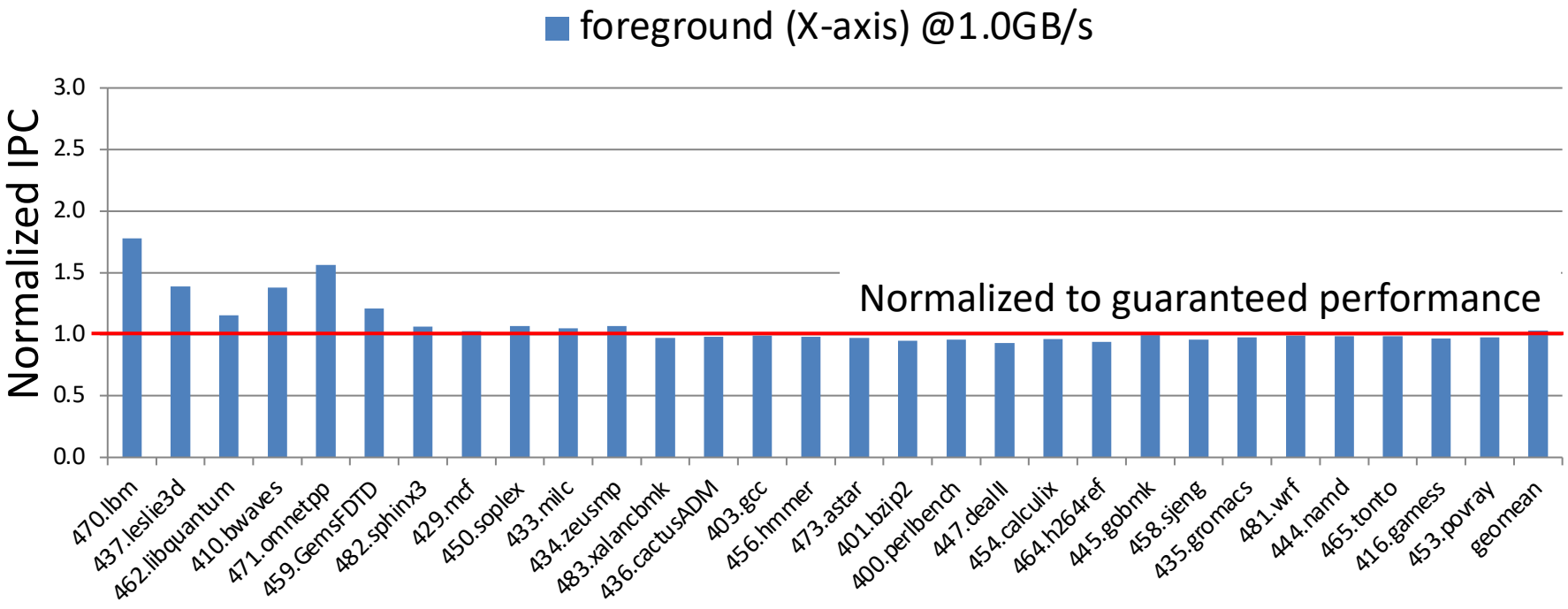


Intel Core2

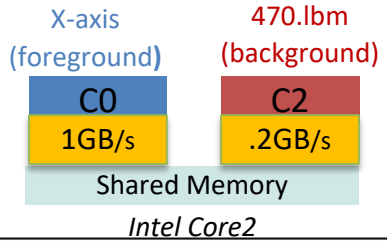


Reclaiming and Sharing maximize performance

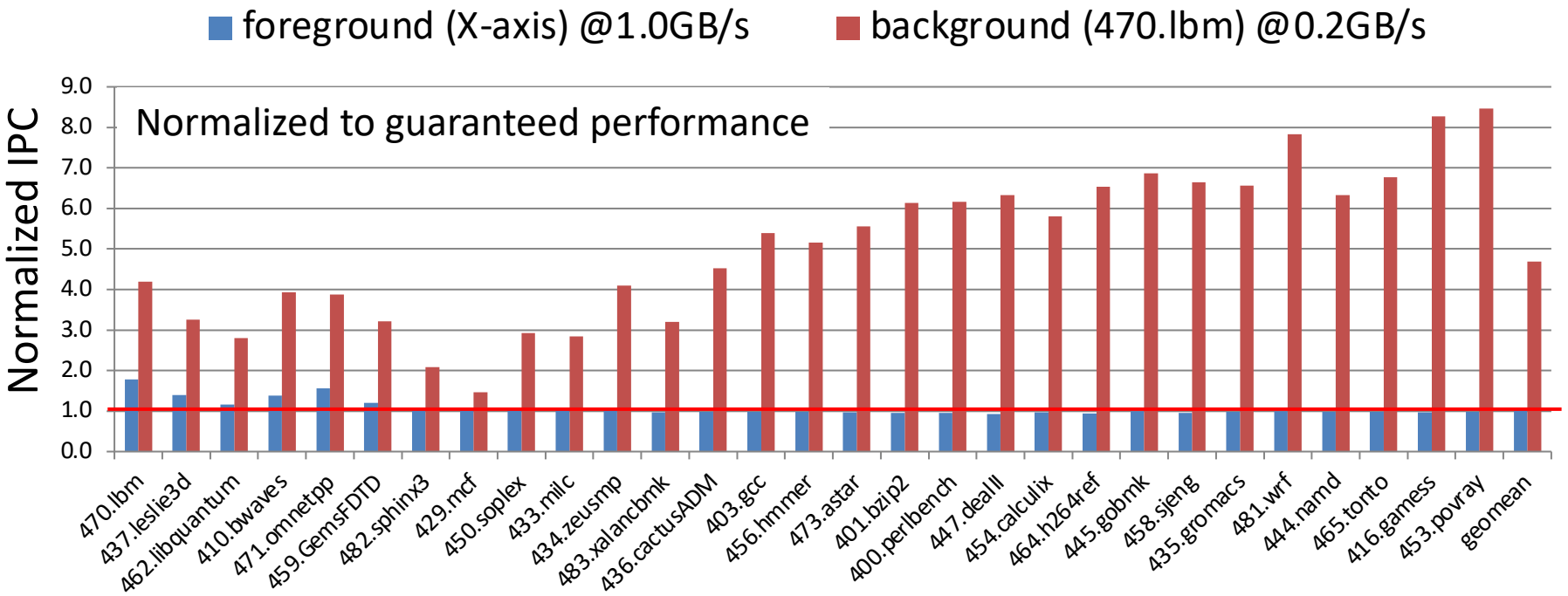
SPEC2006 Results



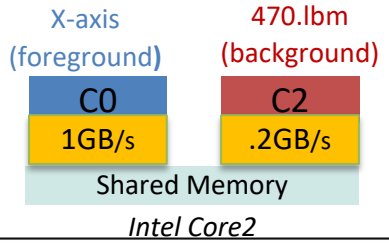
- Guarantee(soft) performance of foreground (x-axis)
  - W.r.t. 1.0GB/s memory b/w reservation



SPEC2006 Results



- Improve overall throughput
  - background: 368%, foreground(X-axis): 6%

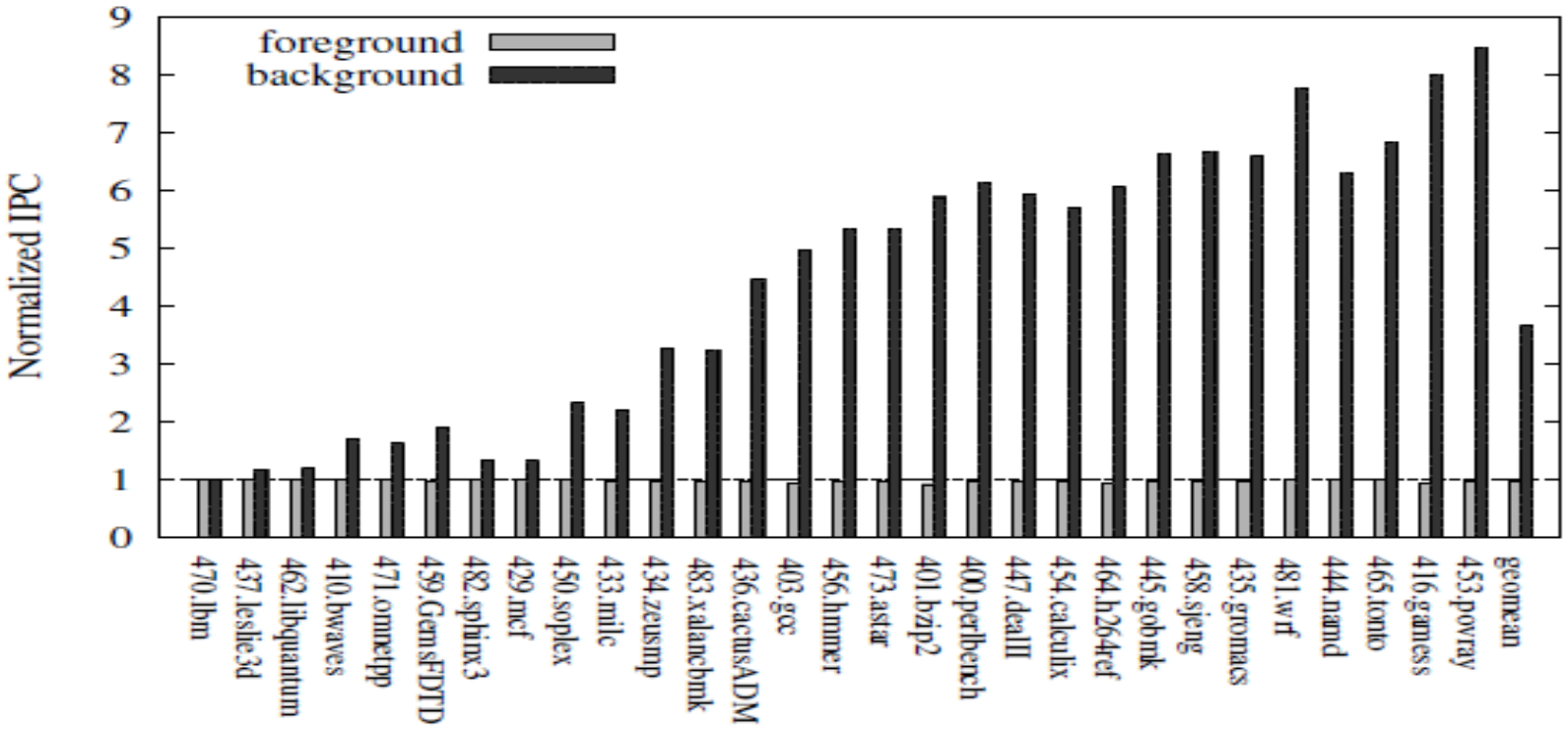


Conclusion

- Inter-Core Memory Interference
  - Big challenge for multi-core based real-time systems
  - Sources: queuing delay in MC, state dependent latency in DRAM
- MemGuard
  - OS mechanism providing efficient per-core memory performance isolation on COTS H/W
  - Memory bandwidth reservation and reclaiming support
  - https://github.com/heechul/memguard/wiki/MemGuard

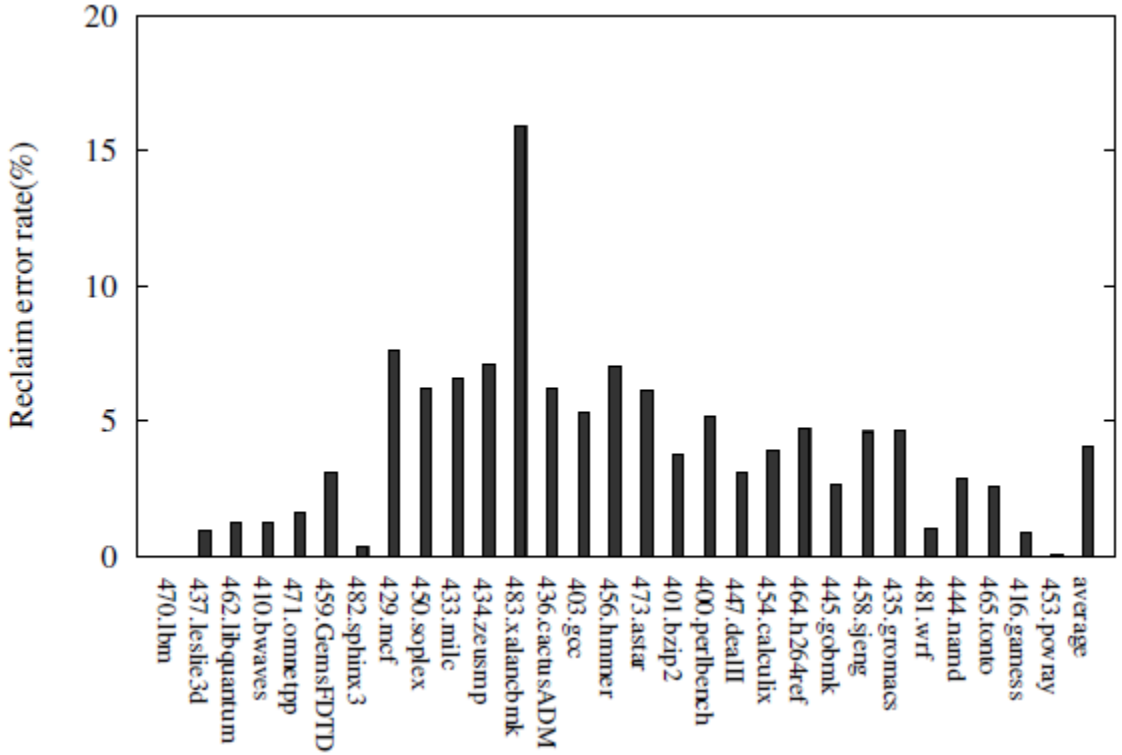
Thank you.

Effect of Reclaim

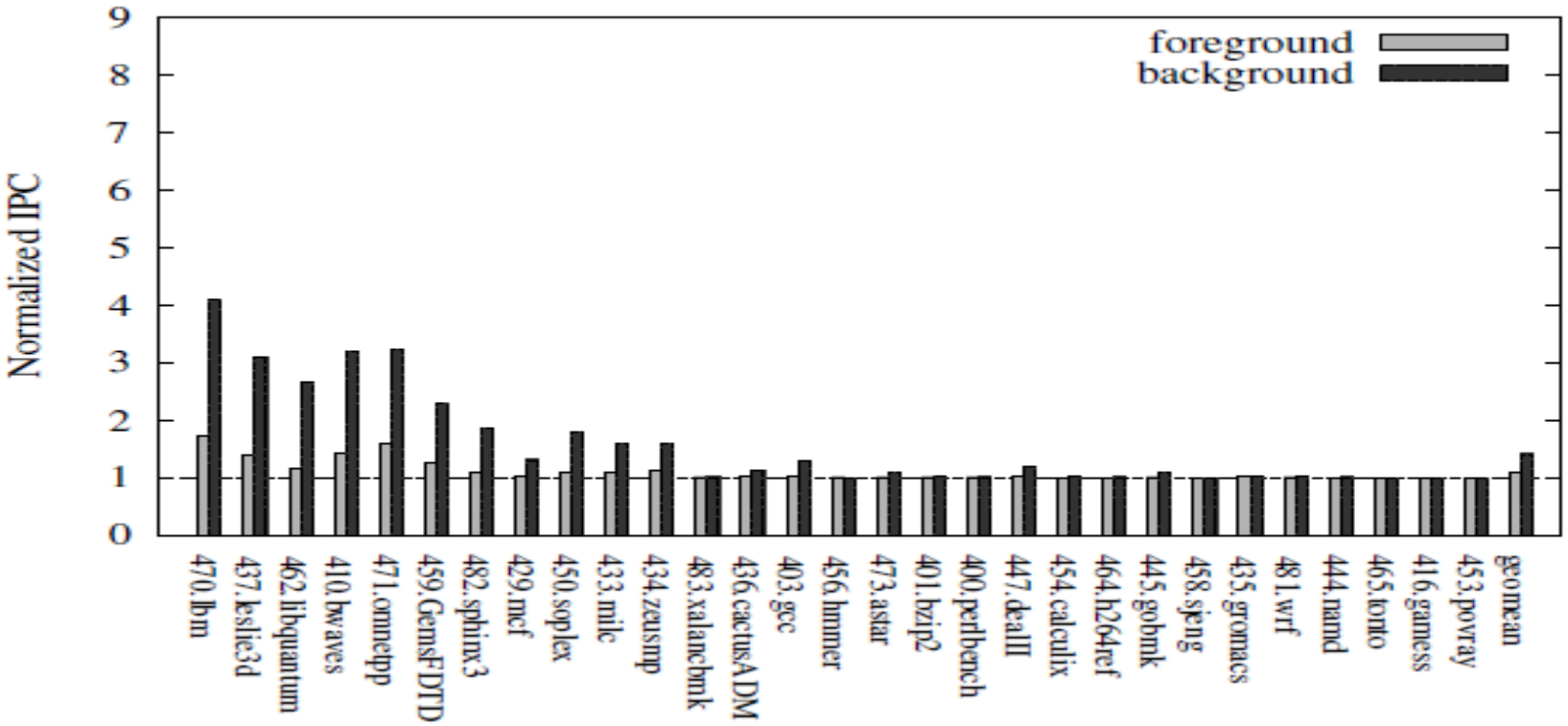


- IPC improvement of background (lbm@0.2GB/s) is 3.8x
- IPC reduction of foreground (SPEC@1.0GB/s) is 3%

Reclaim Underrun Error

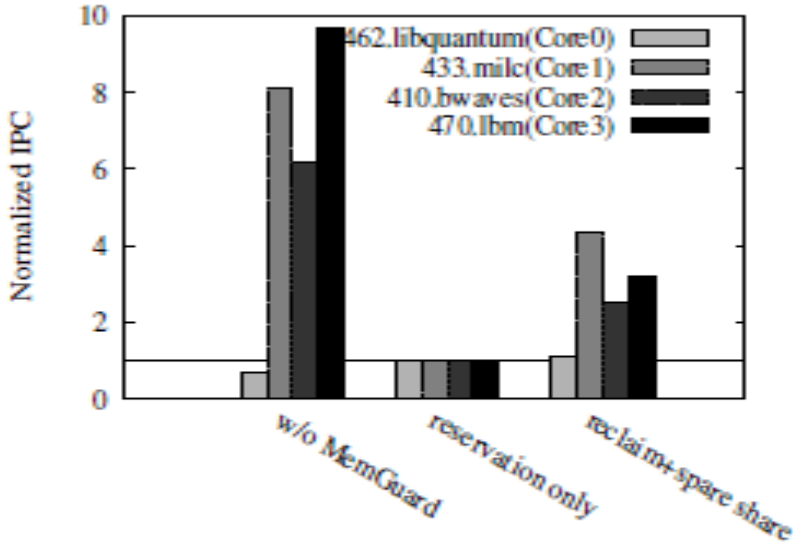


Effect of Spare Sharing

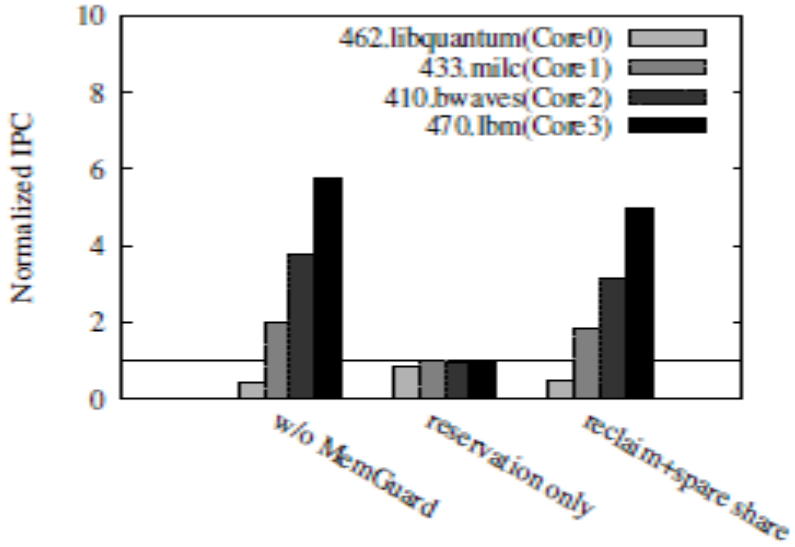


- IPC of background (lbm@0.2GB/s) improves 40%
- IPC of foreground (SPEC@1.0GB/s) also improves 9%

Isolation and Throughput Effect of r_{min}



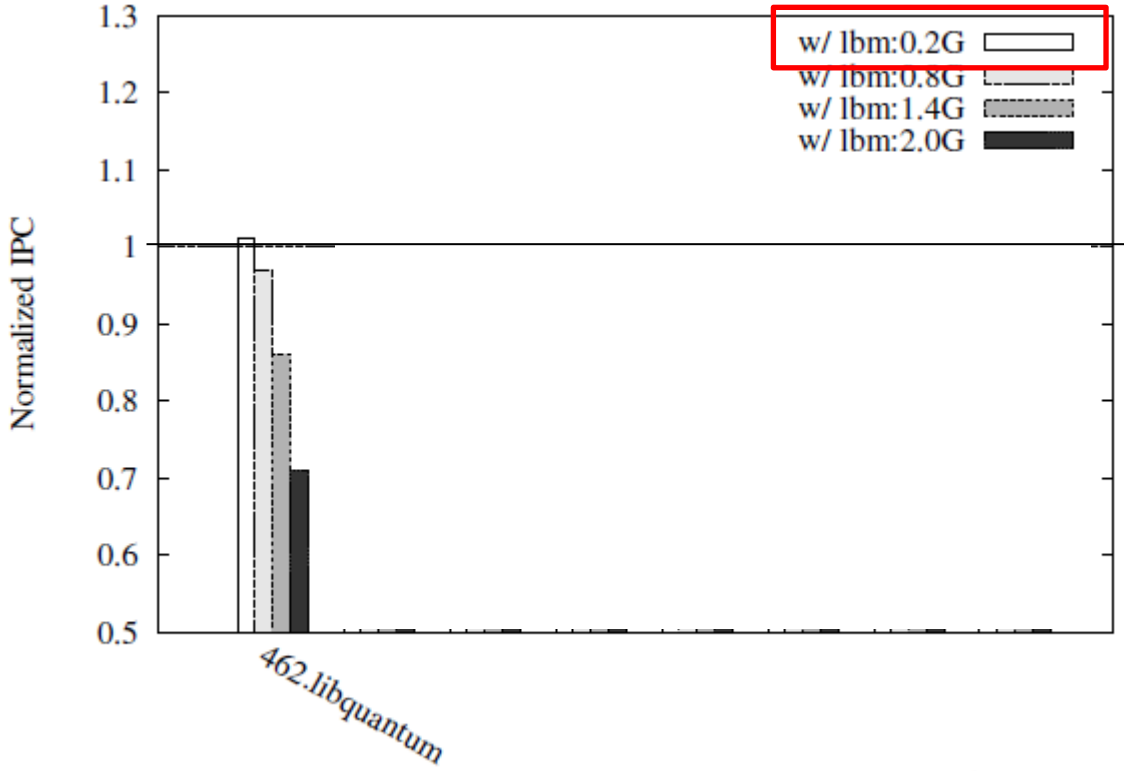
(a) r_{min}=1.2GB/s



(b) r_{min}=2.4GB/s

- 4 core configuration

Isolation Effect of Reservation



Isolation

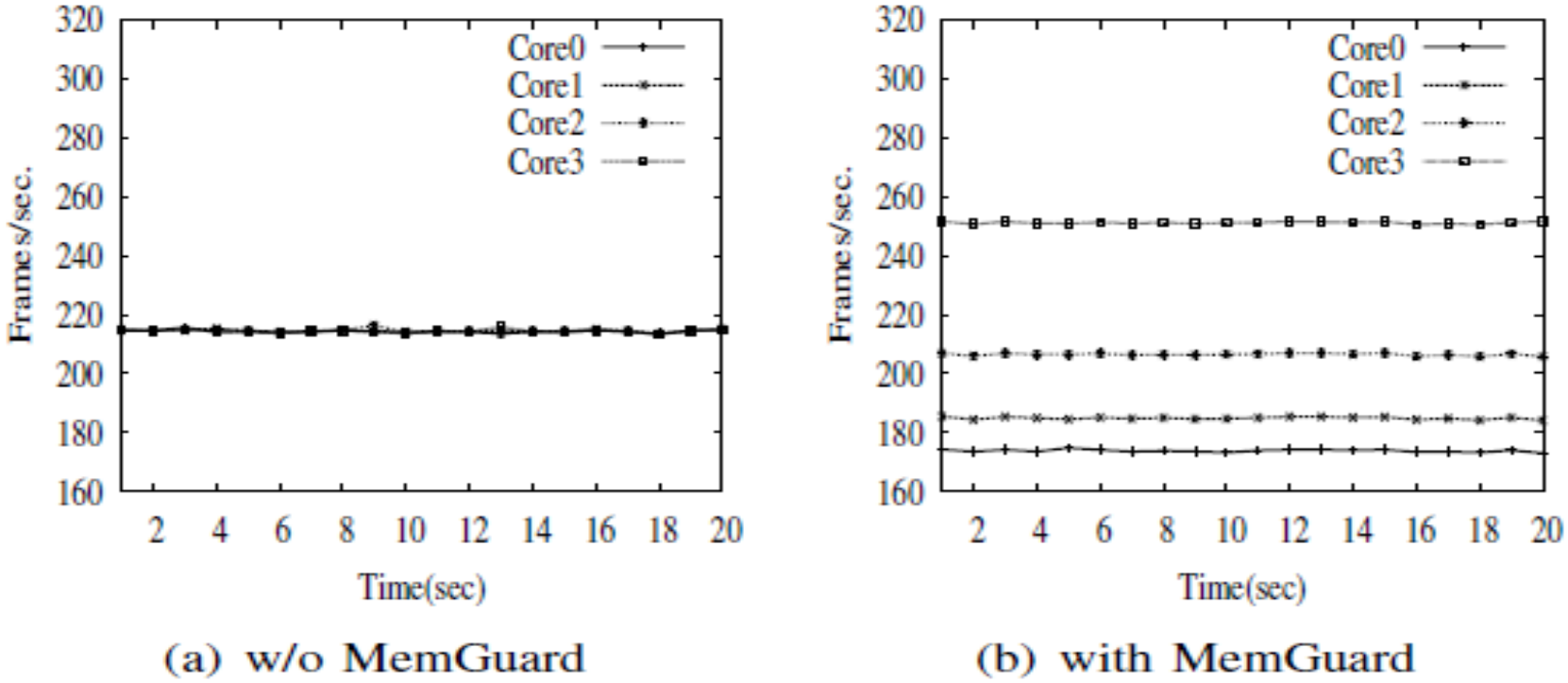
Core 2: 0.2 – 2.0 GB/s for lbm

Solo IPC@1.0GB/s

Core 0: 1.0 GB/s for X-axis

- Sum b/w reservation < r_{min} (1.2GB/s)→ Isolation
  - 1.0GB/s(X-axis) + 0.2GB/s(lbm) = r_{min}

Effect of MemGuard



- Soft real-time application on each core.
- Provides differentiated memory bandwidth
  - weight for each core=1:2:4:8 for the guaranteed b/w, spare bandwidth sharing is enabled

Hard/Soft Reservation on MemGuard

- Hard reservation (w/o reclaiming)
  - Can guarantee memory bandwidth B_i regardless of other cores at each period
  - Wasted if not used
- Soft reservation (w/ reclaiming)
  - Misprediction can caused missed b/w guarantee at each period
  - Error rate is small---less than 5%.
- Selectively applicable on per-core basis