# Memory Access Control in Multiprocessor for Real-time Systems with Mixed Criticality

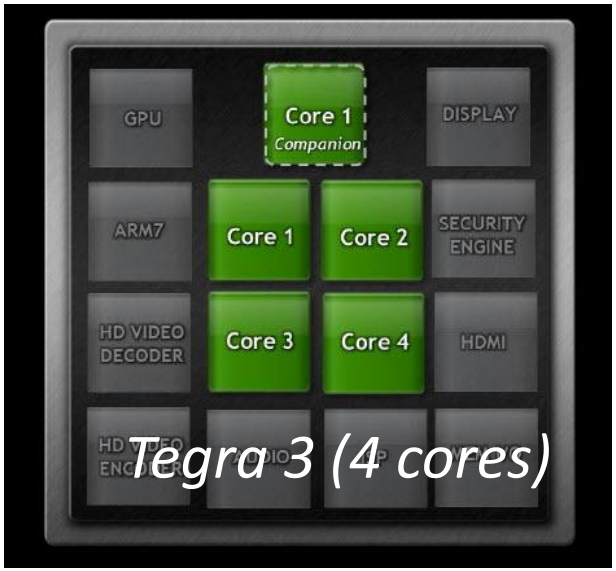
Heechul Yun[+], Gang Yao[+], Rodolfo Pellizzoni[*],
Marco Caccamo[+], Lui Sha[+]

University of Illinois at Urbana and Champaign[+]

Univerity of Waterloo[*]

# Multi-core Systems

- Mainstream in smartphone
  - Dual/quad-core smartphones
  - More performance with less power



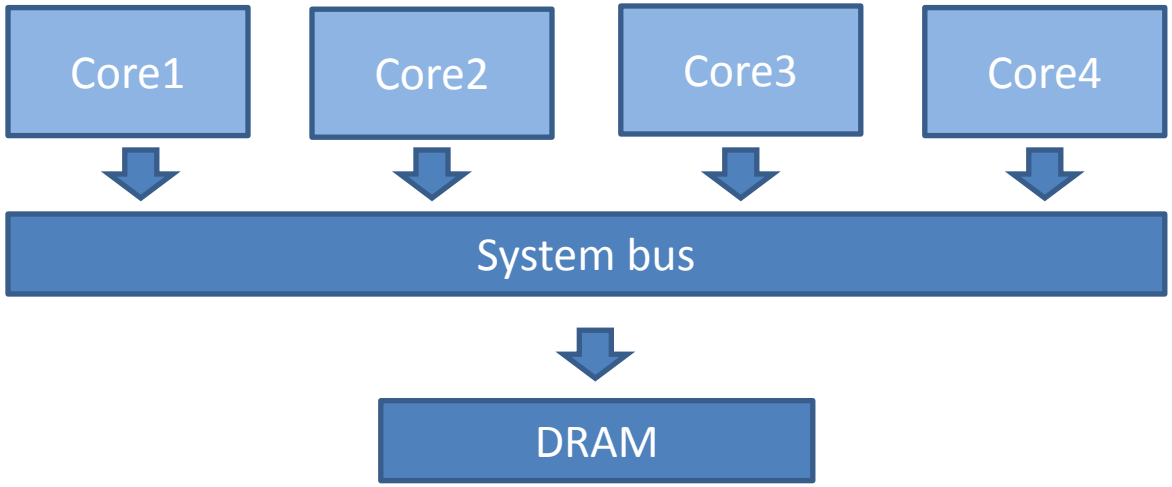
*Tegra 3 (4 cores)*

- Traditional embedded/real-time domains
  - Avionics companies are investigating [Nowotsch12]
    - 8 core P4080 processor from Freescale



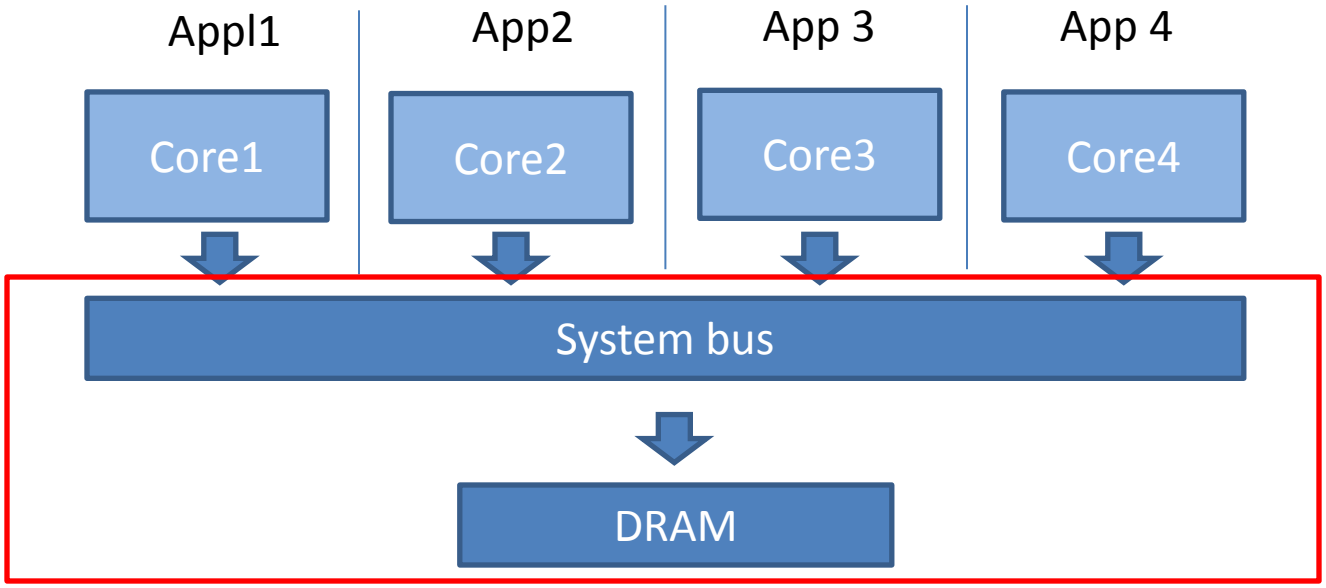
[Nowotsch12] "Leveraging Multi-Core Computing Architectures in Avionics", EDCC, 2012

# Challenge



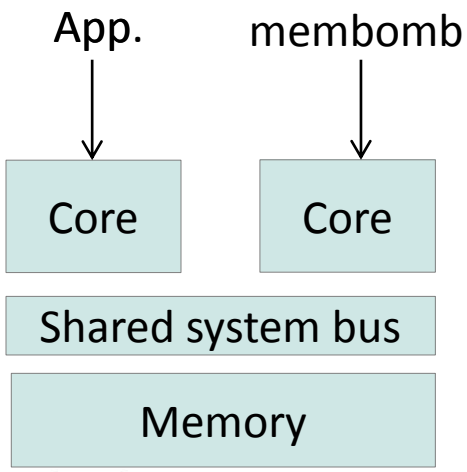

- Timing isolation is hard to achieve

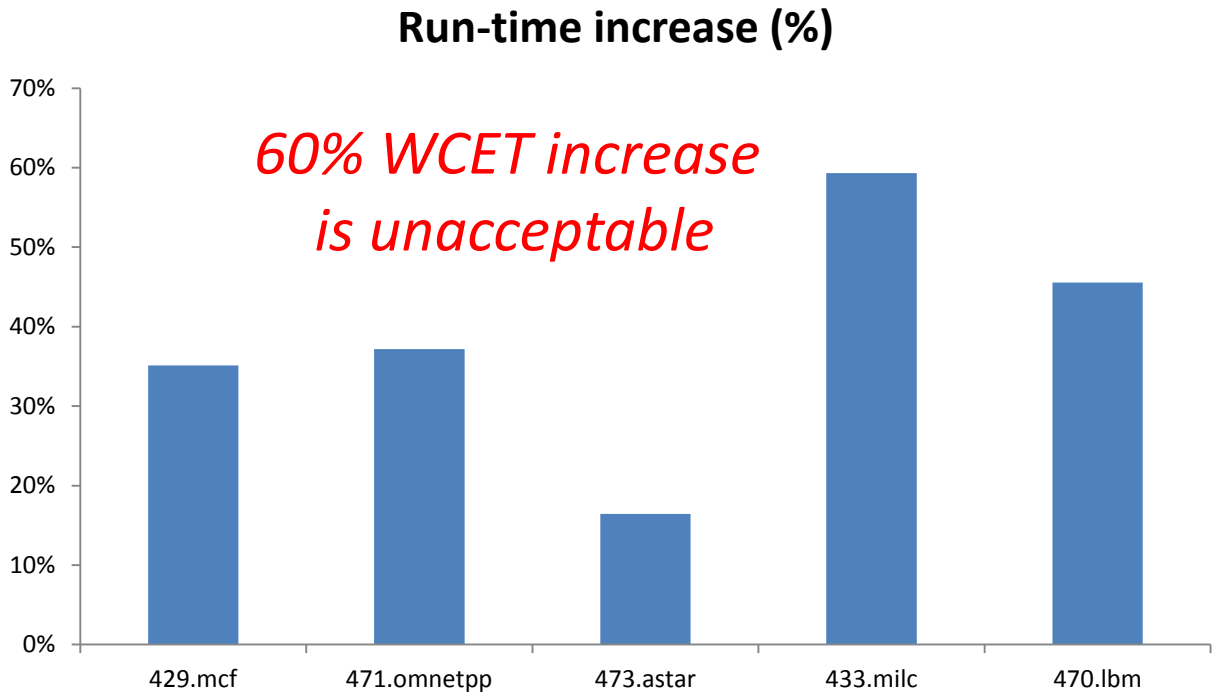
# Challenge




- Cores compete for shared HW resources
  - System bus, DRAM controller, shared cache, …

# Effect of Memory Contention




- Run-time increase due to contention
  - Five SPEC2006 benchmarks
  - Compared to solo execution

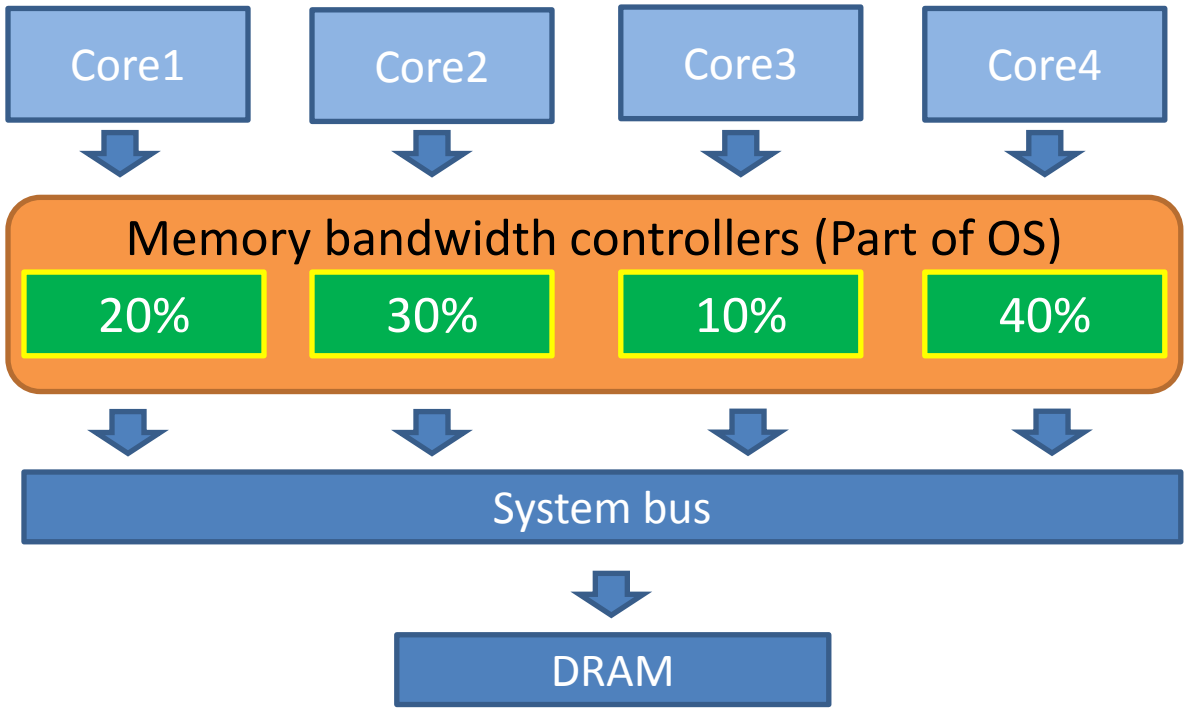
# Goal

- Mechanism to control memory contention
  - Software based controller for COTS multi-core processors
- Response time analysis accounting memory contention effect
  - Based on proposed software based controller

# Outline

- Motivation
- **Memory Access Control System**
- Response Time Analysis
- Evaluation
- Conclusion

# System Architecture




- Assign memory bandwidth to each core using per-core memory bandwidth controller

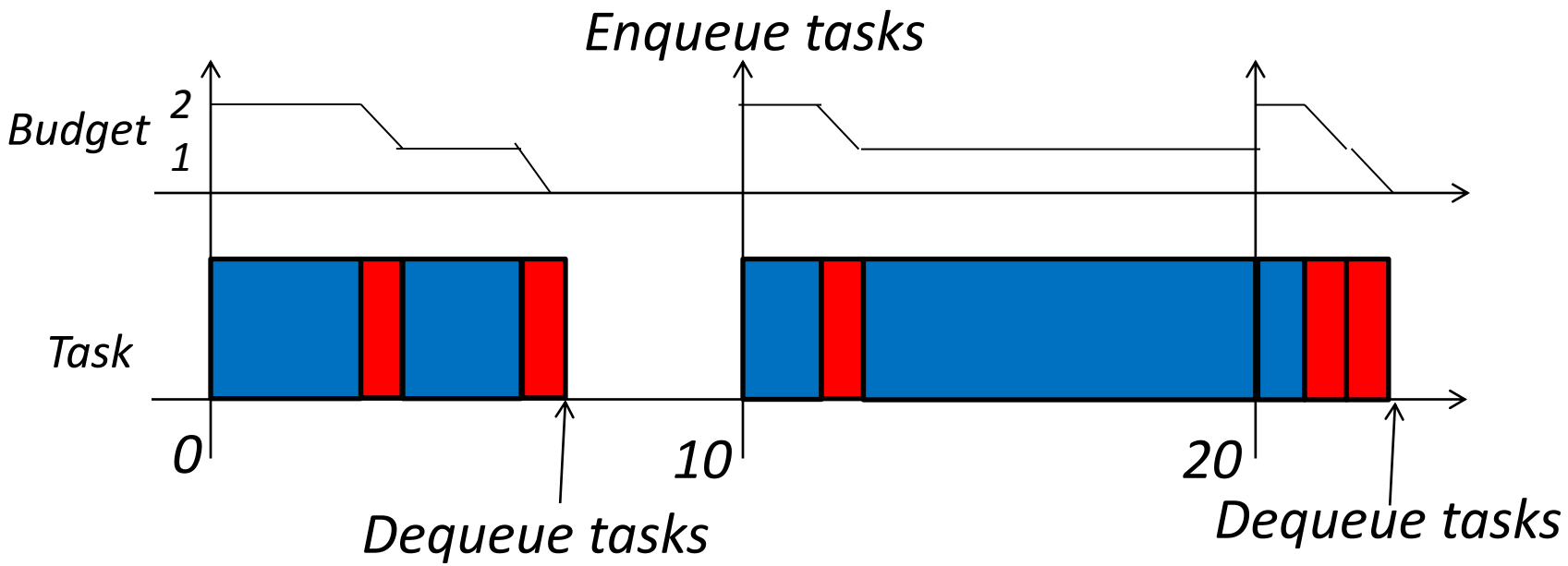
# Memory Bandwidth Controller

- Periodic server for memory resource

- Periodically **monitor** memory accesses of the core and **control** user specified bandwidth using OS scheduler
  - Monitoring can be efficiently done by using per-core hardware performance counter
  - Bandwidth = # memory accesses X avg. access time

# Memory Bandwidth Controller

- Period: 10 time unit, Budget: 2 memory accesses
  - memory access takes 1 time unit



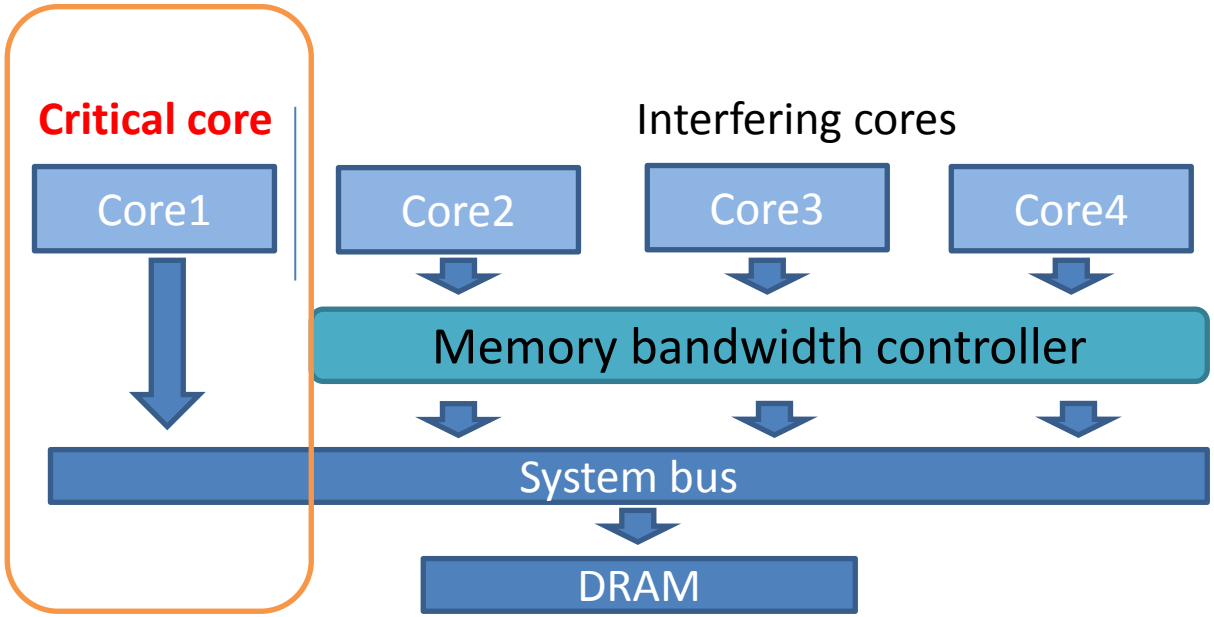

computation

memory fetch

# Outline

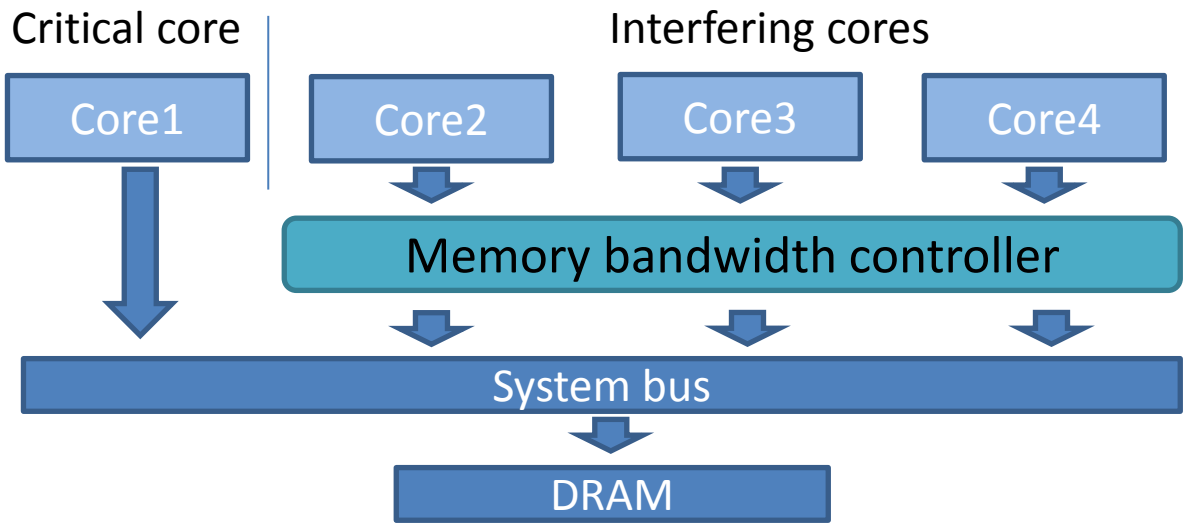
- Motivation
- Memory Bandwidth Control System
- **Response Time Analysis**
- Evaluation
- Conclusion

# System Model



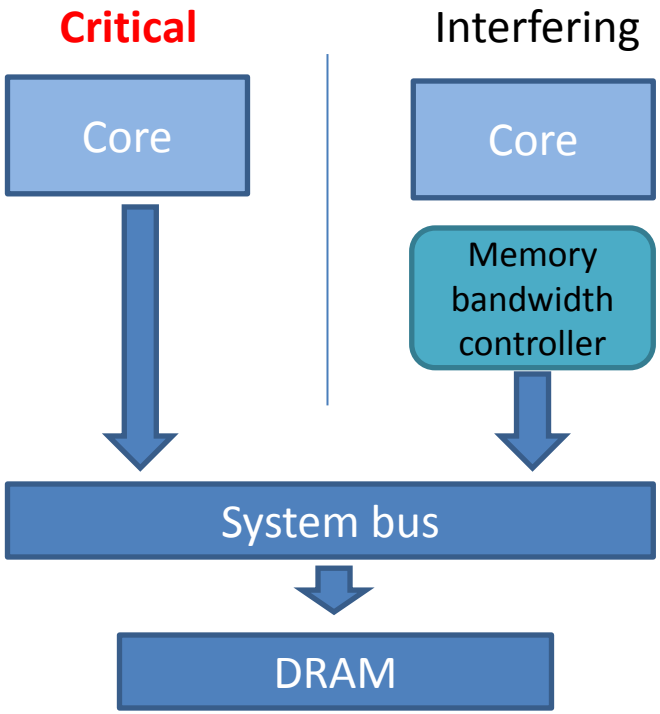

- Cores are partitioned based on criticality
- **Critical core** runs periodic real-time tasks with fixed priority scheduling algorithm
- **Interfering cores** run non-critical workload and regulated with proposed memory access controller

# Assumptions




- Private or partitioned last level cache (LLC)
- Round-robin bus arbitration policy
- Memory access latency is constant
- 1 LLC miss = 1 DRAM access

# Simple Case: One Interfering Core

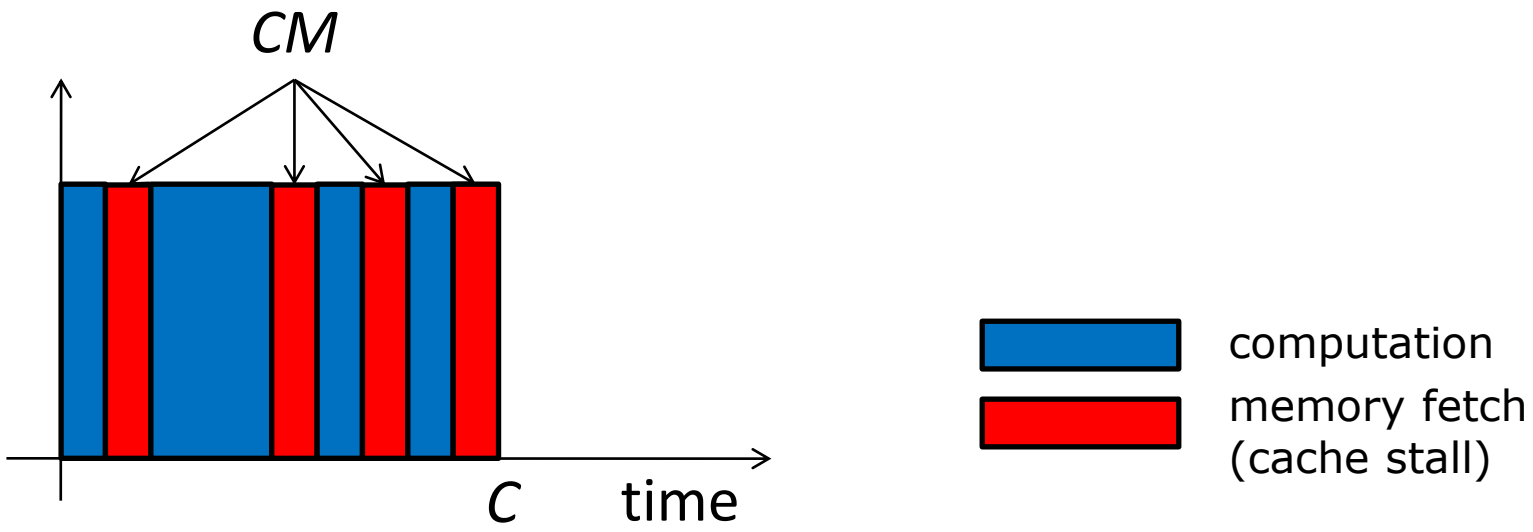



- Critical core - core under analysis
- Interfering core – generating memory interference

# Problem Formulation

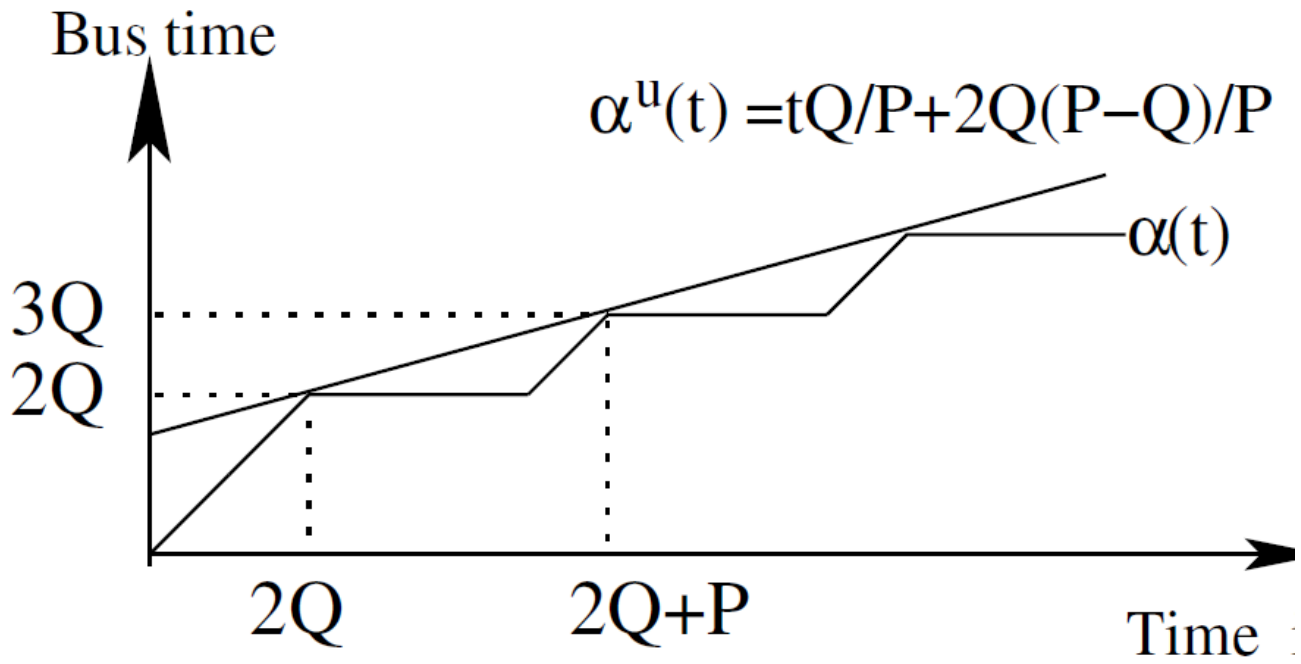
- For a given periodic real-time task set $T = \{\tau_1, \tau_1, \ldots, \tau_n\}$ on a critical core
- Problem:
  - Determine $T$ is schedulable on the critical core given memory access control budget *Q* and period *P* on the interfering core

# Task Model




- *C* : WCET of a task on isolated core (no interference)
- *CM*: number of last level cache misses (DRAM accesses)
- *L*: stall time of single cache miss

# Memory Interference Model




- *P* : memory access controller period
- *Q*: memory access time budget
- $\alpha^u(t)$: Linearized interfering memory traffic upper-bound

# Background [Pellizzoni07]

- Accounting Memory Interference
  - *Cache bound*: maximum interference time <= maximum number cache-accesses (CM) * L of the task under analysis
  - *Traffic bound*: maximum interference time <= maximum bus time requested by the interfering core

$$\widehat{C^{(k+1)}} = C + \min\{\underbrace{CM \cdot L}_{\text{Cache-bound}}, \underbrace{\alpha(\widehat{C^{(k)}})}_{\text{Traffic bound}}\}$$

$\hat{C}$: WCET account memory stall delay

$L$: stall time of single cache miss



[Pellizzoni07] "Toward the predictable integration of real-time cots based systems," RTSS, 2007

# Classic Response Time Analysis

$$R_i^{k+1} = C_i + \sum_{j<i} \left\lceil \frac{R_i^{\,k}}{T_j} \right\rceil * C_j$$

- Tasks are sorted in priority order
  - low index = high priority task
- $C_i$ : WCET of task *i* (in isolation w/o memory interference)
- $R_i$ : Response time of task *i*
- $T_j$ : Period of task *I*

# Extended Response Time Analysis

$$R_i^{k+1} = C_i + \sum_{j<i} \left\lceil \frac{{R_i}^k}{T_j} \right\rceil * C_j + \min\left(N\left(R^k\right) * L, \alpha^u\left(R^k\right)\right)$$

where $N(t) = \sum_{j \le i} \left\lceil \frac{t}{T_j} \right\rceil * CM_j$

$$\alpha^u(t) = t\frac{Q}{P} + 2\,\frac{Q(P-Q)}{P}$$

- $N(t)$: aggregated cache misses over time $t$
- $\alpha^u(t)$: interfering memory traffic over time $t$
- $P$: memory access control period
- $Q$: memory access time budget

- Proposed method achieves tighter response time than using $\hat{C}$

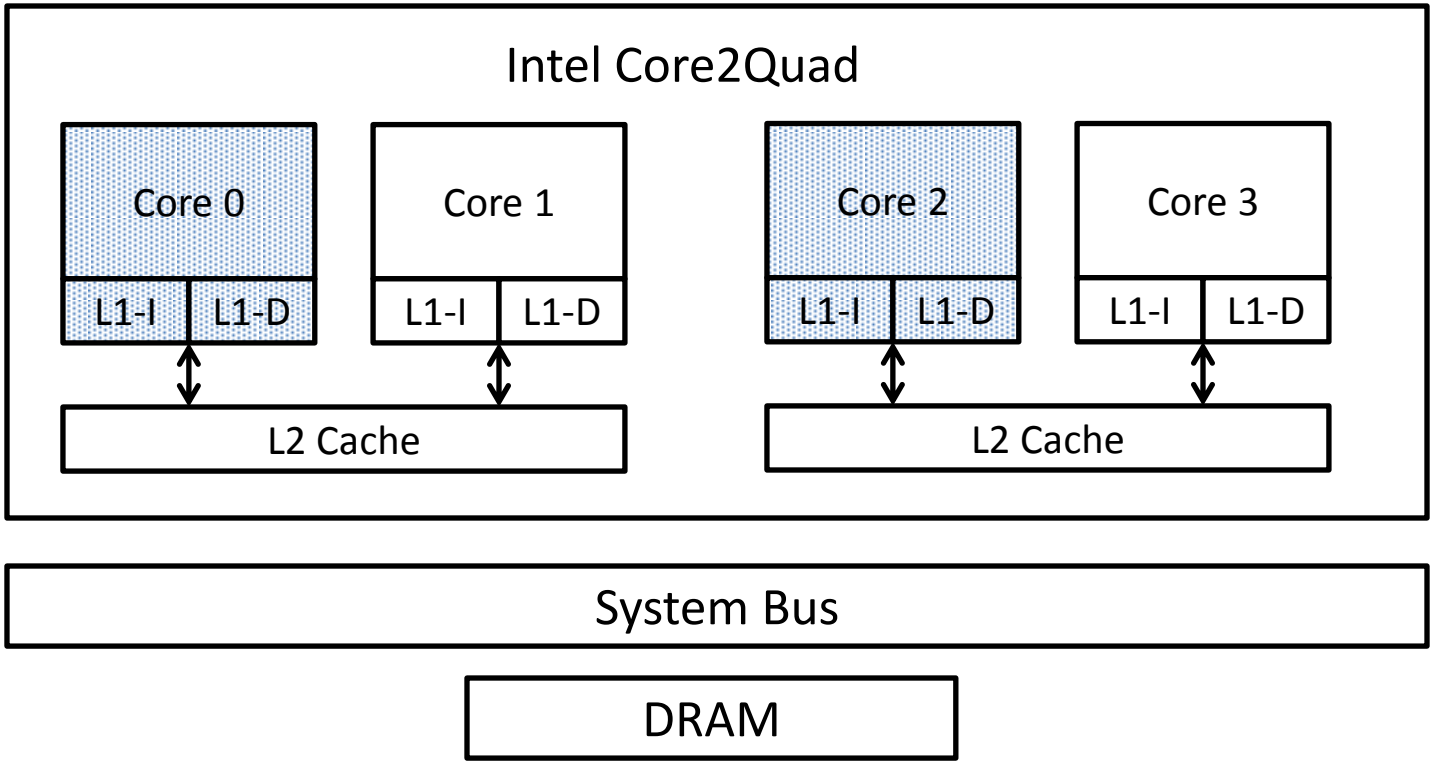
# Outline

- Motivation
- Memory Bandwidth Control System
- Response Time Analysis
- **Evaluation**
- Conclusion

# Linux Kernel Implementation

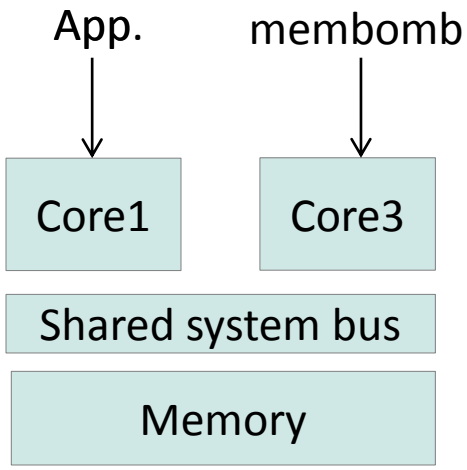
- Extending CPU bandwidth reservation feature of group scheduler
  - Specify core and bandwidth (memory budget, period)
    - mkdir /sys/fs/cgroup/core3; cd /sys/fs/cgroup/core3
    - echo 3 > cpuset.cpus ← core 3
    - **echo 10000 > cpu.cfs_period_us ← period**
    - **echo 500000 > cpu.cfs_quota_event ← cache-misses budget**
      - Added feature
  - Monitor memory usage at every scheduler tick and context switch

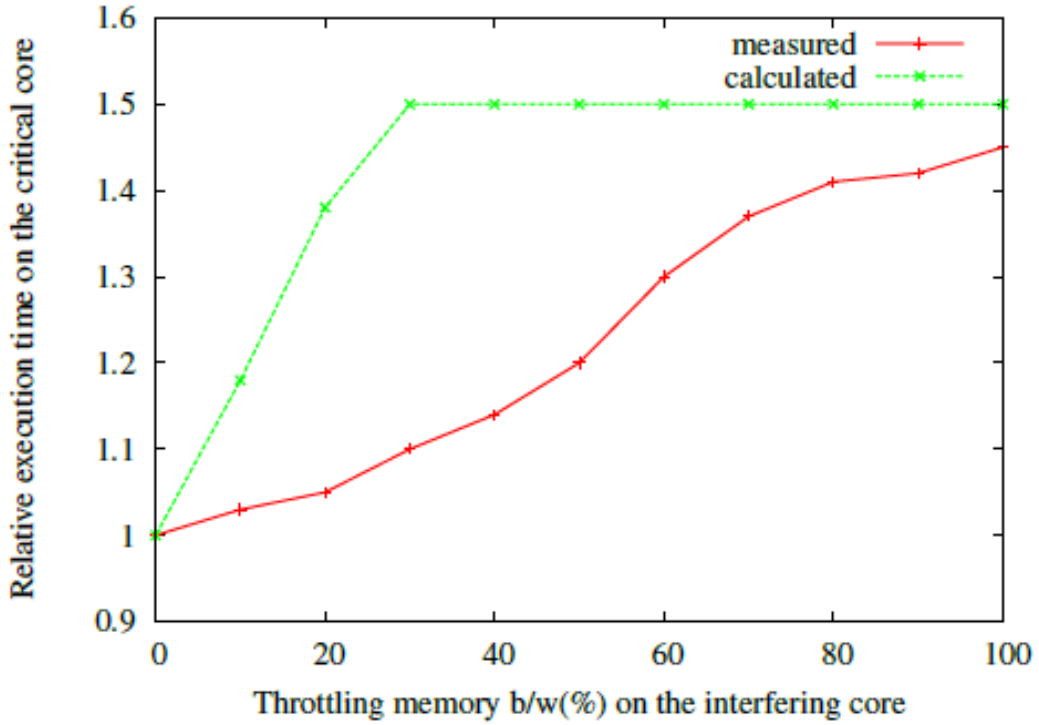
# Experimental Platform




- Core 0,2 were disabled to simulate a private LLC system
- Running a modified Linux 3.2 kernel
  - https://github.com/heechul/linux-sched-coreidle/tree/sched-3.2-throttle-v2

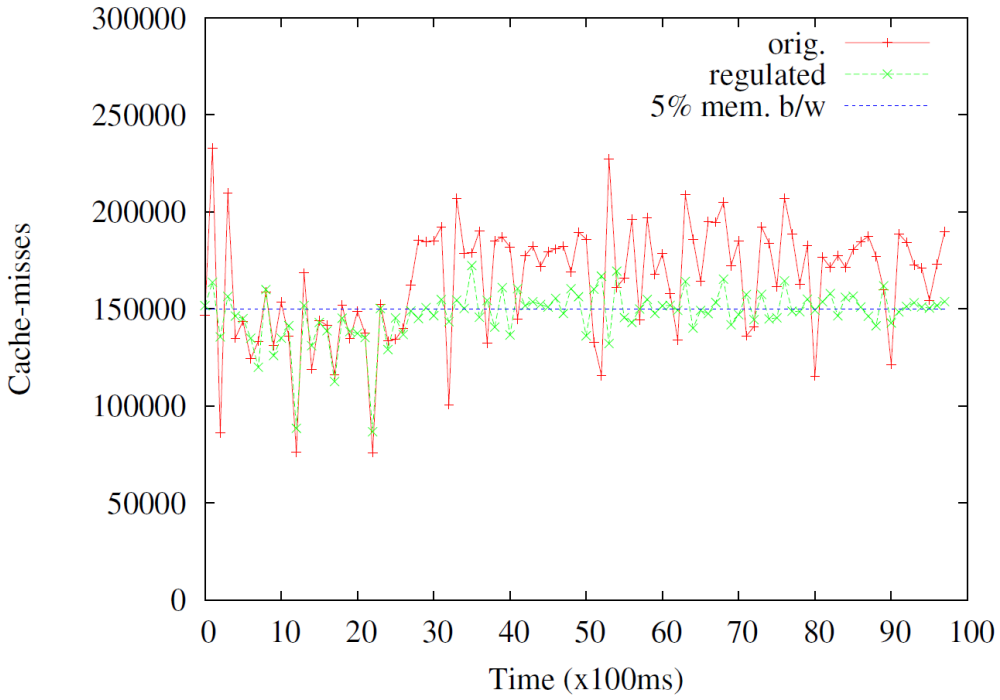
# Synthetic Task




- Core under analysis runs a synthetic task with 50% memory bandwidth
- Vary throttling budget of the interfering core from 0 to 100%
- Two findings: (1) we can control interference, (2) analysis provide an upper bound (albeit still pessimistic)

# H.264 Movie Playback




- Cache-miss counts sampled over every 100ms
- Some inaccuracy in regulation due to implementation limitation
  - Current version is improved accurate by using hardware overflow interrupt

# Conclusion

- Shared hardware resources in multi-core systems are big challenges for designing real-time systems
- We proposed and implemented a mechanism to provide memory bandwidth reservation capability on COTS multi-core processors
- We developed a response time analysis method using the proposed memory access control mechanism

# Thank you.