Memory Servers for Multicore Systems

Rodolfo Pellizzoni and Heechul Yun
Outline

• Delay analysis and compositionality
• Solution: Memory Server
• Core-level analysis
• System-level analysis
• Evaluation
• Conclusions
Outline

• Delay analysis and compositionality
• Solution: Memory Server
• Core-level analysis
• System-level analysis
• Evaluation
• Conclusions
Resource Contention

We need a **resource delay analysis** to bound such effects

---

![Diagram showing resource contention]

**Core**

**Core**

**Core**

**Shared Physical Resource**

**INTERFERENCE!!!**
Analysis Taxonomy

- **Request-driven bound:**
  - Compute maximum per-Request Delay $RD$
  - $H$ : # requests task under analysis
  - Delay = $RD \times H$
  - Computed delay can be very pessimistic!

- **Job-driven bound:**
  - $\overline{H}$ : bound on # requests by other cores
  - Delay = $f(H, \overline{H})$
  - Non compositional – requires knowledge of tasks running on all cores!
Example: DRAM Write Buffering

- Reads are prioritized over writes; writes are buffered and transmitted in non-preemptive batches (ARM A15 – 18 req.)
- If a write batch begins right before a read under analysis arrives, the request is delayed by 18 write requests…
COTS Arbiters are Unfair

• The main issue: unfair arbitration
  – $M$ cores
  – A request can be delayed by $\gg M - 1$ other requests

• Many other examples…
  – First-Ready FCFS DRAM arbitration
  – Non-blocking caches
  – Wormhole NoC arbitration
How bad can it be?

Even with just 4 cores, measured execution time is ~8x larger

Provable bound is much higher due to write buffering
Summary: The Problem

- Compositional analysis is highly desirable
- Develop each application (core) independently
- But bounds are bad due to request-driven analysis
Summary: The Problem

- Compositional analysis is highly desirable
- Develop each application (core) independently
- But bounds are bad due to request-driven analysis
Summary: The Problem

- Or forget about compositionality to use job-driven bounds...
- ... but then any change to other applications forces me to redo timing analysis
Summary: The Problem

- Or forget about compositionality to use job-driven bounds...
- ... but then any change to other applications forces me to redo timing analysis

Let me change a compiler flag here…

Shared Physical Resource
Summary: The Problem

-compositionality to use job-driven bounds, but then any change to other applications forces me to redo timing analysis.

OMG!!!

Let me change a compiler flag here...
Summary: The Problem

- Or forget about compositionality to use job-driven bounds...
- ...but then any change to other applications forces me to redo timing analysis
- Let me change a compiler flag here…

OMG!!!

We need a way to keep compositionality but use job-driven bounds!
Outline

- Delay analysis and compositionality
- **Solution: Memory Server**
- Core-level analysis
- System-level analysis
- Evaluation
- Conclusions
Memory Server

How do we solve the problem?

1. Add a server with budget $Q_p$ to each core $p$

2. Server constraints the core to make no more than $Q_p$ accesses to the shared resource every $P$ time units

3. Now we know the maximum resource accesses by other cores, so we can use job-driven analysis!
Server Implementation

- Reserve per-core memory bandwidth via the OS scheduler
  - Use h/w PMC to monitor memory request rate

Heechul Yun et al: MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-core Platforms, RTAS’13
Memory vs Hierarchical Server

- Isn’t this similar to a hierarchical server? Yes

- Multiple applications on the same core
Memory vs Hierarchical Server

- Isn’t this similar to a hierarchical server? Yes

- Step 1: perform **local** analysis to derive sched. interface
Memory vs Hierarchical Server

• Isn’t this similar to a hierarchical server? Yes

• Step 2: perform **system-level** analysis to check sched.
  - Ex: $\sum Q_p / P \leq U_b$
Memory vs Hierarchical Server

- For memory server:
  - Multiple application on different cores
Memory vs Hierarchical Server

• For memory server:

  - Step 1: perform **core-level** analysis to derive interface
Memory vs Hierarchical Server

• For memory server:

```
Sched. Interface
```

Step 2: perform **system-level** analysis to check sched.
Interfaces and Compositionality

• In both cases, the interface represents a **contract**
  – Each application’s developer guarantees that he will respect the contract
  – Check schedulability based on the contracts for the other applications, not the tasks inside

I want to add a new task to my application...

Do what you want, just respect the contract
Memory vs Hierarchical Server

- Key difference: interfaces
- For fixed $P$, the schedulability of an application in a hierarchical server depends only on the assigned budget $Q_p$
Memory vs Hierarchical Server

- Key difference: interfaces
- ... but the delay (hence schedulability) for memory server also depends on the total budget $QB_p$ assigned to other cores!

Key intuition: there is no global resource utilization bound – it depends on the tasks within the application.
Detailed Contribution

• Core-level analysis computes 2D feasibility region for a core
  – Based on state-of-the-art DRAM delay analysis; but any request-driven / job-driven analysis could be used

• System-level analysis determines feasibility
  – Is there a budget assignment that satisfies all interfaces?

• Both analyses have reasonable complexity
Outline

• Delay analysis and compositionality
• Solution: Memory Server
• Core-level analysis
• System-level analysis
• Evaluation
• Conclusions
Core-level Analysis

• Fixed priority scheduling

Bini & Buttazzo: *Schedulability analysis of periodic fixed priority systems, TC’04*

For each task...

• Compute set of time instants
• Iff for any time instant $\bar{t}$: \(\text{computation demand}([0, \bar{t}]) \leq \bar{t}\), task is schedulable
Core-level Analysis

- Fixed priority scheduling
  Bini & Buttazzo: *Schedulability analysis of periodic fixed priority systems, TC’04*

- For each task...

- With resource interference:
  \[
  \text{computation demand}([0, \overline{t})) + \text{resource delay}([0, \overline{t})) \leq \overline{t}
  \]
Regulation: tasks in the busy interval are delayed because they exhaust the budget

\[ \text{resource delay}(\overline{0, t}) \leq \overline{t} - \text{computation demand}(\overline{0, t}) \]

Regulation delay decreases with larger budget values \( Q_p \)
Resource Delay: Contention

• Contention: tasks in the busy interval are delayed due to requests of other cores

\[
\text{resource delay}(\bar{t}) \leq \bar{t} - \text{computation demand}(\bar{t})
\]

• Contention delay decreases with smaller budget values \( QB_p \)
Memory Delay: One Time Instant

- Key obs.: resource delay is the max of regulation, contention
  - Independent of analysis; true bc of how regulation works
- Task is schedulable at $\bar{t}$ if it meets both regulation and contention constraints
Interface: One Task

- Task is schedulable if it meets constraints at any time instant $\bar{t}$
Interface: One Task

- Task is schedulable if it meets constraints at any time instant $\bar{t}$.
Interface: One Core

- Task set is schedulable if all tasks are schedulable

Region for $\tau_2$

Region for $\tau_1$
Interface: One Core

- Task set is schedulable if all tasks are schedulable

Region for the core
Outline

- Delay analysis and compositionality
- Solution: Memory Server
- Core-level analysis
- System-level analysis
- Evaluation
- Conclusions
System-level analysis

- Start with smallest budget values $Q_p$
- Determine budget of other cores $QB_p$
- If not feasible, move to next budget value

Interface for Core#1

Interface for Core#2
System-level analysis

- Start with smallest budget values $Q_p$
- Determine budget of other cores $Q B_p$
- If not feasible, move to next budget value
System-level analysis

- Start with smallest budget values $Q_p$
- Determine budget of other cores $QB_p$
- If not feasible, move to next budget value

Interface for Core#1

Interface for Core#2

OK!
Result Summary

• System-level analysis is optimal (based on computed core-level interfaces)
  – If there is a feasible budget assignment, it will find it

• Computation complexity
  – $M$ cores, $N$ tasks per core, $K$ time instances per task
  – Core-level analysis is $O(N \cdot K \cdot \log(N \cdot K))$
  – System-level analysis is $O(M^2 \cdot N \cdot K)$

• Note: $K$ is $O(N \cdot \max(\text{deadline})/\min(\text{period}))$, but generally much smaller
Outline

- Delay analysis and compositionality
- Solution: Memory Server
- Core-level analysis
- System-level analysis
- Evaluation
- Conclusions
Evaluation

- Randomly generated task sets
- Only shared resource is DRAM
- FR-FCFS arbitration, private banks, write buffering

Heechul Yun et al: Parallelism-Aware Memory Interference Delay Analysis for COTS Multicore Systems, ECRTS’15

- ARM A15, LPDDR2@533Mhz, $P = 1\text{ms}$
- Random task bandwidth in $[0, B_{\text{max}}]$:
  - $B_{\text{max}}$: fraction of maximum memory bandwidth
Analyses

- **non-comp**
  - Non-compositional: all tasks on all cores are known
  - No regulation, but can use job-driven analysis

- **rw-reg**

- **read-reg**
  - Memory server. PMCs measure either read+write requests, or read only

- **legacy**
  - All servers have the same fixed budget

- **unreg**
  - No regulation, compositional, request-driven only
$M = 4, \ N = 10, \ B_{\text{max}} = 1\%$

This is the price of compositionality with our approach.
\( M = 4, \; N = 10, \; B_{\text{max}} = 1\% \)

This is the price of compositionality without regulation.
$M = 4, \ N = 10, \ B_{\text{max}} = 1\%$

This is the improvement over previous regulation approach

Renato Mancuso et al: *WCET(m) Estimation in Multi-Core Systems using Single Core Equivalence, ECRTS’15*
$M = 4, N = 10, B_{\text{max}} \in [0\%, 10\%]$
Outline

• Delay analysis and compositionality
• Solution: Memory Server
• Core-level analysis
• System-level analysis
• Evaluation
• Conclusions
Conclusions

• Resource regulation leads to a “better” compositionality by establishing **contracts** (interface) on resource usage.

• The approach works if the cost of regulation is less than the improvement over request-driven analysis.

• Downside: interface is more complex compared to hierarchical scheduling.
  – Note: this paper computes a tight interface – it might be easier to use a (linearized?) subset of feasibility region.

• Future work:
  – Multiple applications per core
  – Multiple resources
  – More complex resources (NoC)
Thank You!

Questions?