

# Multicore Timing Verification: Integrating static timing and scheduling analysis

Sebastian Altmeyer



OVSTR Meeting Paris  
October 15th, 2015

# Content of this Talk

## A Generic and Compositional Framework for Multicore Response Time Analysis

Sebastian Altmeyer, Robert I. Davis, Leandro Indrusiak, Claire Maiza, Vincent Nelis, Jan Reineke

RTNS 2015



## The time is now: Timing Verification for Safety-Critical Multi-Cores

NWO Veni Research Project

# Research Topic: Guaranteeing Timing Correctness



1. Step:

## Timing Analysis

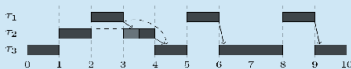
Derives worst-case execution time (WCET) of each tasks

**WCET**

2. Step:

## Scheduling Analysis

Checks if all tasks scheduled together meet their timing constraints



# Timing Verification for pre-emptive Systems



1. Step:

## Timing Analysis+

Derives worst-case execution time (WCET) of each tasks

**WCET + CRPD**

2. Step:

## Scheduling Analysis+

Checks if all tasks scheduled together meet their timing constraints



**Insufficient** in case of **pre-emption**:

⇒ **repair it** by extending the interface

# Timing Verification for Multicore Systems?



1. Step:

## Timing Analysis++?

Derives worst-case execution time (WCET) of each tasks

**WCET** + ?????

2. Step:

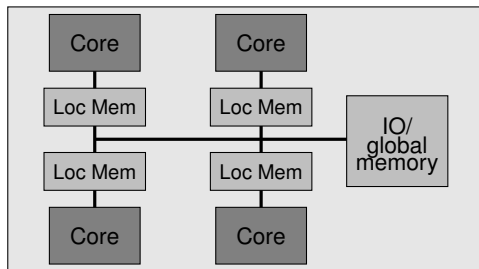
## Scheduling Analysis++?

Checks if all tasks scheduled together meet their timing constraints



**How to repair it for multicores?**

# Timing Verification for Multicore Systems - Problem



## Truly parallel task execution

- ▶ **interferences** on processor, local memory, bus, global memory
- ▶ strong **interdependencies** between tasks

⇒ **a task can't be analyzed in isolation** anymore.

# Consequence 1: ambiguous Notion of WCET

## **What is a task's WCET on a multicore?**

Is it the time when executed

- ▶ in isolation?
- ▶ with worst-case contention?
- ▶ with actual co-running tasks?

## **Can we assign a unique WCET per task?**

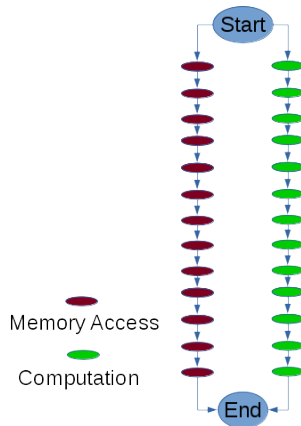
Yes, but it will be

- ▶ incomplete, or
- ▶ pessimistic

## Consequence 2: ambiguous worst-case Path

**Which path is worse** with respect to

- ▶ a task scheduled on the same core?
- ▶ a task scheduled on another core?
- ▶ the pre-emption costs?
- ▶ ...





# Timing Verification for Multicore Systems?



1. Step:

## Timing Analysis++?

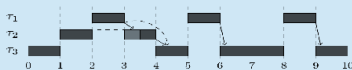
Derives worst-case execution time (WCET) of each tasks

**WCET** + ????

2. Step:

## Scheduling Analysis++?

Checks if all tasks scheduled together meet their timing constraints



# Timing Verification for Multicore Systems?



1. Step:

## Timing Analysis++?

Derives worst-case execution time (WCET) of each tasks

**WCET** + ?????

2. Step:

## Scheduling Analysis++?

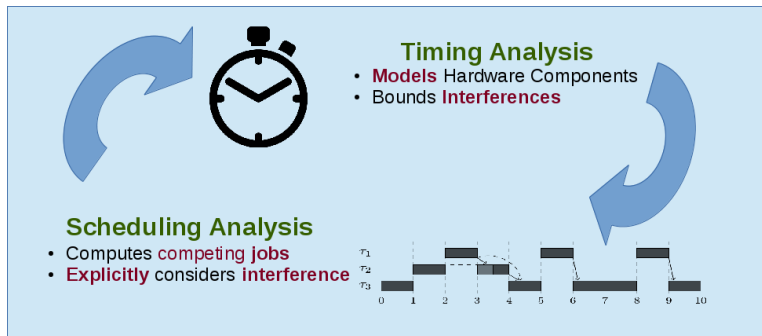
Checks if all tasks scheduled together meet their timing constraints



▶ **broken in case of multicores:**

⇒ **re-think it** from scratch

# Timing Verification for Multicore Systems!



We need to

- ▶ change traditional timing verification process
- ▶ use a different task model
- ▶ omit notion of WCET
- ▶ but reuse existing techniques!

# A Generic and Compositional Framework for Multicore Response Time Analysis

1. different task model (no single WCET, but traces)
2. abstract representation of the multicore components
3. interference computation
4. interference-aware response time computation

# A Generic and Compositional Framework for Multicore Response Time Analysis

1. different task model (no single WCET, but traces)
2. abstract representation of the multicore components
3. interference computation
4. interference-aware response time computation

## What we are not doing

- ▶ a fully integrated analysis
- ▶ an analysis based on isolation
- ▶ a partial solution

# Task Model

$n$  sporadic tasks  $\{\tau_1, \dots, \tau_n\}$ , per task

- ▶ **period**  $T_i$
- ▶ **deadline**  $D_i$  (with  $D_i \leq T_i$ )
- ▶ **execution time** bound  $C_i$

# Task Model

$n$  sporadic tasks  $\{\tau_1, \dots, \tau_n\}$ , per task

- ▶ **period**  $T_i$
- ▶ **deadline**  $D_i$  (with  $D_i \leq T_i$ )
- ▶ **execution time** bound  $C_i$
- ▶ set of **traces**  $O_i$ , ordered lists of instructions

# Task Model

$n$  sporadic tasks  $\{\tau_1, \dots, \tau_n\}$ , per task

- ▶ **period**  $T_i$
- ▶ **deadline**  $D_i$  (with  $D_i \leq T_i$ )
- ▶ **execution time** bound  $C_i$
- ▶ set of **traces**  $O_i$ , ordered lists of instructions

**Instruction:**

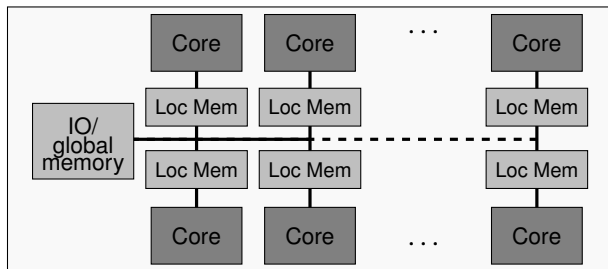
$$\iota = (m^{\text{in}}, \Delta, it)$$

- ▶ instruction's **memory address**  $m^{\text{in}}$
- ▶ **execution time**  $\Delta$  without memory delays
- ▶ **instruction type**  $it$  (read/write/execute)

(simple, yet expressive task model)

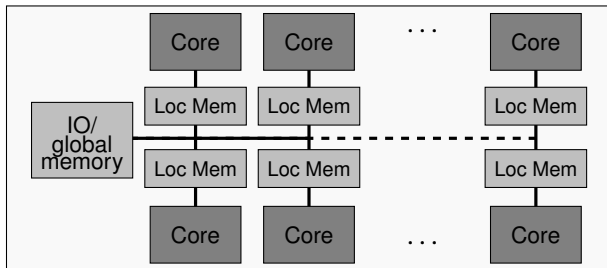


# Processor Model

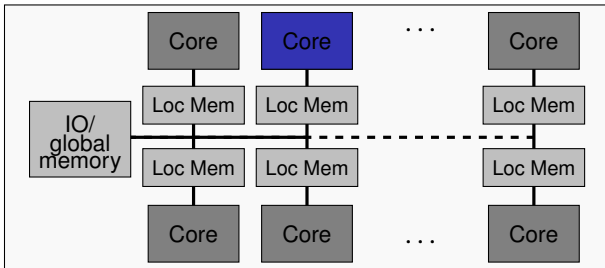


- ▶  $\ell$  identical cores  $\{P_1, \dots, P_\ell\}$ ,
- ▶ fixed-priority pre-emptive scheduling, partitioned tasks
- ▶ one shared bus
- ▶ local memories
- ▶ a global memory (DRAM)

# Impact of the Multicore Components

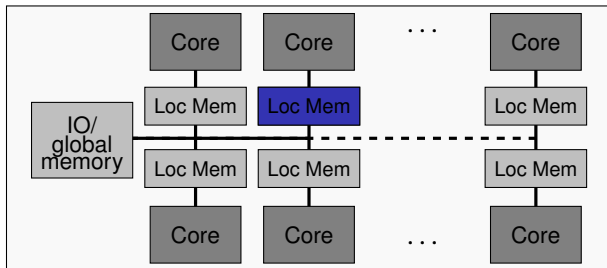


# Impact of the Multicore Components



**Core** How long does it take to execute a trace?

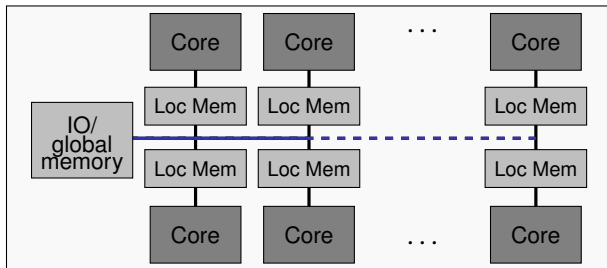
# Impact of the Multicore Components



**Core** How long does it take to execute a trace?

**Local Memory** How many memory requests go to the bus?

# Impact of the Multicore Components

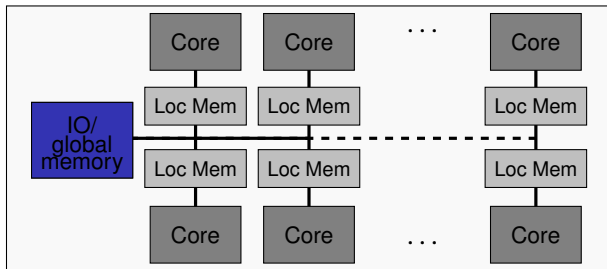


**Core** How long does it take to execute a trace?

**Local Memory** How many memory requests go to the bus?

**Bus** How many competing accesses can occur?

# Impact of the Multicore Components



**Core** How long does it take to execute a trace?

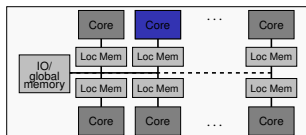
**Local Memory** How many memory requests go to the bus?

**Bus** How many competing accesses can occur?

**Global Memory** How many DRAM refreshes can occur?

# Core: Processor Demand

PROC:  $\mathbb{O} \rightarrow \mathbb{N}$



How long does it take to execute a trace?

---

Processor Demand of trace  $\sigma$ :

$$\sum_{(-, \Delta, -) \in \sigma} \Delta$$

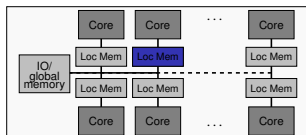
Processor demand of task  $\tau_i$ :

$$PD_i = \max_{\sigma \in \mathcal{O}_i} \sum_{(-, \Delta, -) \in \sigma} \Delta$$

( $\Delta$ : execution time of instruction without memory delays)

# Local Memory: Memory Function/Demand

$$\text{MEM}: \mathbb{O} \rightarrow \mathbb{N} \times 2^{2^N} \times 2^N$$



How many memory requests go to the bus?

---

$$\text{MEM}(o) = (\text{MD}_o, \overline{\text{UCB}}_o, \text{ECB}_o)$$

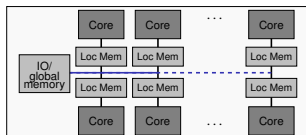
- ▶ # bus accesses (memory demand MD)
- ▶  $\overline{\text{UCB}}_o$  pre-emption costs when  $o$  is pre-empted
- ▶  $\text{ECB}_o$  pre-emption costs when trace  $o$  pre-empts another task

$$\text{Memory demand of task } \tau_i: \text{MD}_i = \max_{o \in O_i} \left\{ \text{MD} \mid \text{MEM}(o) = (\text{MD}, -, -) \right\}$$



# Bus Function

$$\text{BUS: } \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N}$$



How many competing accesses can occur?

---

$$\text{BUS}(i, x, t)$$

#bus accesses that delay task  $\tau_i$  on processor  $P_x$  during time interval  $t$

---

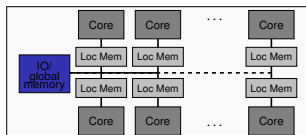
$S_i^x(t)$  #accesses due to  $\tau_i$  and all higher priority tasks on  $P_x$  during  $t$

$A_j^y(t)$  #accesses due to all tasks of priority  $j$  or higher on  $P_y \neq P_x$  during  $t$

$S_i^x(t)$  and  $A_j^y(t)$  use memory demand  $MD_i$ , UCBs and ECBs from memory function

# DRAM Function

DRAM:  $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$



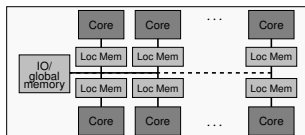
How many DRAM refreshes can occur?

---

$\text{DRAM}(t, m)$

#DRAM refreshes during time  $t$  with up to  $m$  memory accesses

# Which components can we model:



PROC:  $\mathbb{O} \rightarrow \mathbb{N}$  any timing-composable core

MEM:  $\mathbb{O} \rightarrow \mathbb{N} \times 2^{2^{\mathbb{N}}} \times 2^{\mathbb{N}}$

Scratchpads, LRU/DM Caches, partitioned Caches,  
uncached Systems,

BUS:  $\mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N}$

Fixed-Priority Bus, TDMA, Round-Robin, Processor  
Priority

DRAM:  $\mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N}$

Burst Refreshes, Distributed Refreshes

and any combination thereof.

# From Component Model to Interferences

$$I^{Comp}(i, x, R_i)$$

Interference/Delay of component *Comp* on the response time  $R_i$  of task  $\tau_i$  executing on processor  $P_x$

---

## From Component Model to Interferences

$$I^{Comp}(i, x, R_i)$$

Interference/Delay of component *Comp* on the response time  $R_i$  of task  $\tau_j$  executing on processor  $P_x$

---

$$I^{PROC}(i, x, t) = \sum_{j \in \Gamma_x \wedge j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil PD_j$$

---

## From Component Model to Interferences

$$I^{Comp}(i, x, R_i)$$

Interference/Delay of component *Comp* on the response time  $R_i$  of task  $\tau_i$  executing on processor  $P_x$

---

$$I^{PROC}(i, x, t) = \sum_{j \in \Gamma_x \wedge j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil PD_j$$

---

$$I^{BUS}(i, x, t) = BUS(i, x, t) \cdot d_{main}$$

where  $d_{main}$  is the bus access latency to the global memory.

---

## From Component Model to Interferences

$$I^{Comp}(i, x, R_i)$$

Interference/Delay of component *Comp* on the response time  $R_i$  of task  $\tau_i$  executing on processor  $P_x$

---

$$I^{PROC}(i, x, t) = \sum_{j \in \Gamma_x \wedge j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil PD_j$$

---

$$I^{BUS}(i, x, t) = BUS(i, x, t) \cdot d_{main}$$

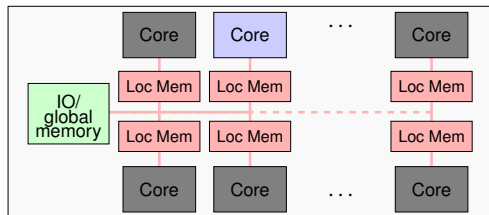
where  $d_{main}$  is the bus access latency to the global memory.

---

$$I^{DRAM}(i, x, t) = DRAM(t, BUS((i, x, t))) \cdot d_{refresh}$$

where  $d_{refresh}$  is the refresh latency.

# Multicore Response Time Analysis



$$R_i = PD_i + I^{\text{PROC}}(i, x, R_i) + I^{\text{BUS}}(i, x, R_i) + I^{\text{DRAM}}(i, x, R_i)$$

(solved via fixed-point iteration)

Task set feasible, if:

$$\forall i : R_i \leq D_i$$



# Use cases of the Framework

## Timing Verification Tool

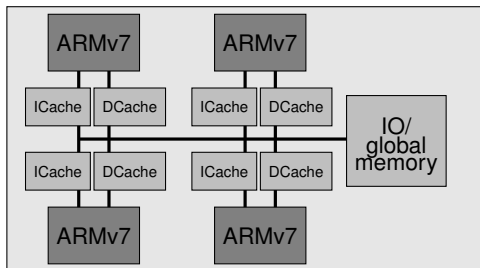
- ▶ for multicore systems including local/global memory and bus
- ▶ extensible to other sources of interference
- ▶ generic, i.e., can be adapted to different architectures
- ▶ supports scenario-based analysis

## Design-Space Exploration

- ▶ to evaluate design choices
- ▶ to determine fitness of components for hard real-time systems  
(Is task isolation really necessary?)

# Proof-of-Concept Instantiation

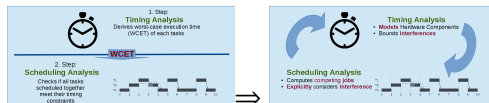
- ▶ System based on the ARM Cortex A5:



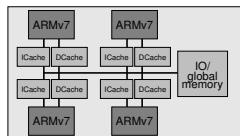
- ▶ 4 cores, separate instruction and data caches, FP/FIFO/TDMA bus, and distributed DRAM controller.
- ▶ Compared different configurations for a large number of randomly generated task sets

# Conclusions

- ▶ Integrated timing verification process



- ▶ Generic timing analysis for Multicores



- ▶ Still plenty of open issues ...



# Memory Function - no local cache

$$\text{MEM}: \mathbb{O} \rightarrow \mathbb{N} \times 2^{2^{\mathbb{N}}} \times 2^{\mathbb{N}}$$

---

Instruction Memory:

$$\text{MEM}_{\text{nc}}^{\text{in}}(o) = (k, \emptyset, \emptyset)$$

where  $k$  is the number of instructions on trace  $o$

Data Memory:

$$\text{MEM}_{\text{nc}}^{\text{da}}(o) = \left( \left\{ \iota_i \mid \iota_i \in o \wedge \iota_i = (-, -, r/w[m^{\text{da}}]) \right\}, \emptyset, \emptyset \right)$$

# Memory Function - Scratchpad

$$\text{MEM}: \mathbb{O} \rightarrow \mathbb{N} \times 2^{2^{\mathbb{N}}} \times 2^{\mathbb{N}}$$

---

Scratchpad memory function SPM:  $\mathbb{M} \rightarrow \{\text{true}, \text{false}\}$

Instruction Memory:

$$\text{MEM}_{\text{sp}}^{\text{in}}(o) = \left( \left| \left\{ m^{\text{in}} \mid (m^{\text{in}}, -, -) \in o \wedge \neg \text{SPM}(m^{\text{in}}) \right\} \right|, \emptyset, \emptyset \right)$$

Data Memory:

$$\text{MEM}_{\text{sp}}^{\text{da}}(o) = \left( \left| \left\{ m^{\text{da}} \mid ((-, -, r(m^{\text{da}})) \in o \wedge \neg \text{SPM}(m^{\text{da}})) \right. \right. \right. \\ \left. \left. \left. \vee (-, -, w(m^{\text{da}})) \in o \right\} \right|, \emptyset, \emptyset \right)$$

# Memory Function - LRU Cache

$$\text{MEM}: \mathbb{O} \rightarrow \mathbb{N} \times 2^{2^{\mathbb{N}}} \times 2^{\mathbb{N}}$$

---

Cache function Hit:  $\mathbb{I} \times \mathbb{M} \rightarrow \{\text{true}, \text{false}\}$

Instruction Memory:

$$\text{MEM}_{\text{ca}}^{\text{in}}(o) = \left( \left| \left\{ m^{\text{in}} | l_i = (m^{\text{in}}, \rightarrow, -) \in o \wedge \neg \text{Hit}(m^{\text{in}}, l_i) \right\} \right|, \overline{\text{UCB}}_o^{\text{in}}, \text{ECB}_o^{\text{in}} \right)$$

Data Memory:

$$\text{MEM}_{\text{ca}}^{\text{da}}(o) = \left( \left| \left\{ m^{\text{da}} | (l_i = (\rightarrow, \rightarrow, r(m^{\text{da}})) \in o \wedge \neg \text{Hit}(m^{\text{da}}, l_i) \right. \right. \right. \\ \left. \left. \left. \vee (\rightarrow, \rightarrow, w(m^{\text{da}})) \in o \right\} \right|, \overline{\text{UCB}}_o^{\text{da}}, \text{ECB}_o^{\text{da}} \right)$$

## Bus Function - FIFO Bus

$$\text{BUS}: \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N}$$

---

$$\text{BUS}(i, x, t) = S_i^x(t) + \sum_{\forall y \neq x} A_n^y(t) + 1$$

(equal number of slots  $v$  per processor)

$S_i^x(t)$  #bus accesses due to  $\tau_i$  and all higher priority tasks on  $P_x$  during  $t$

$A_j^y(t)$  #bus accesses due to all tasks of priority  $j$  or higher on  $P_y \neq P_x$  during  $t$



## Bus Function - TDMA Bus

$$\text{BUS}: \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N}$$

---

$$\text{BUS}(i, x, t) = S_i^x(t) + ((\ell - 1) \cdot v) \cdot S_i^x(t) + 1$$

( $v$  adjacent slots per core, cycle of length  $\ell \cdot v$ )

$S_i^x(t)$  #bus accesses due to  $\tau_i$  and all higher priority tasks on  $P_x$  during  $t$

$A_j^y(t)$  #bus accesses due to all tasks of priority  $j$  or higher on  $P_y \neq P_x$  during  $t$

## Bus Function - Fixed Priority Bus

$$\text{BUS}: \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N}$$

---

$$\text{BUS}(i, x, t) = S_i^x(t) + \sum_{\forall y \neq x} A_i^y(t) + \min\left(S_i^x(t), \sum_{y \neq x} L_i^y(t)\right) + 1$$

$\min\left(S_i^x(t), \sum_{y \neq x} L_i^y(t)\right)$  bounds the blocking due lower priority tasks

$S_i^x(t)$  #bus accesses due to  $\tau_i$  and all higher priority tasks on  $P_x$  during  $t$

$A_j^y(t)$  #bus accesses due to all tasks of priority  $j$  or higher on  $P_y \neq P_x$  during  $t$

# DRAM Function

$$\text{DRAM}: \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N}$$

---

$$\text{DRAM}(t, m)$$

#DRAM refreshes during time  $t$  with up to  $m$  memory accesses

---

Burst refreshes:

$$\text{DRAM}_{\text{burst}}(t, m) = \left\lceil \frac{t}{T_{\text{refresh}}} \right\rceil \cdot \# \text{rows}$$

Distributed refreshes:

$$\text{DRAM}_{\text{dist}}(t, m) = \min \left( m, \left\lceil \frac{t \cdot \# \text{rows}}{T_{\text{refresh}}} \right\rceil \right)$$

where  $T_{\text{refresh}}$  is the refresh interval,  $\# \text{rows}$  the number of DRAM rows.