Multicore Timing Verification: Integrating static timing and scheduling analysis

Sebastian Altmeyer

UNIVERSITY OF LUXEMBOURG

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

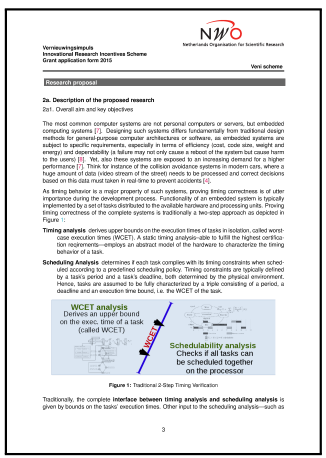

A Generic and Compositional Framework for Multicore Response Time Analysis

Sebastian Altmeyer, University of Luxembourg / University of Amsterdam
Robert I. Davis, University of York / INRIA, Paris-Rocquencourt
Leandro Indrusiak, University of York
Claire Maiza, Grenoble INP Verimag
Vincent Nelis, CISTER, ISEP, Porto
Jan Reineke, Saarland University

ABSTRACT

I. INTRODUCTION

The time is now: Timing Verification for Safety-Critical Multi-Cores
NWO Veni Research Project


NWO
Netherlands Organisation for Scientific Research

Vernieuwingsimpuls
Innovational Research Incentives Scheme
Grant application form 2015

Veni scheme

Research proposal

2a. Description of the proposed research

2a1. Overall aim and key objectives

Timing analysis derives upper bounds on the execution times of tasks in isolation, called worst-case execution times (WCET). A static timing analysis–able to fulfill the highest certification requirements—employs an abstract model of the hardware to characterize the timing behavior of a task.

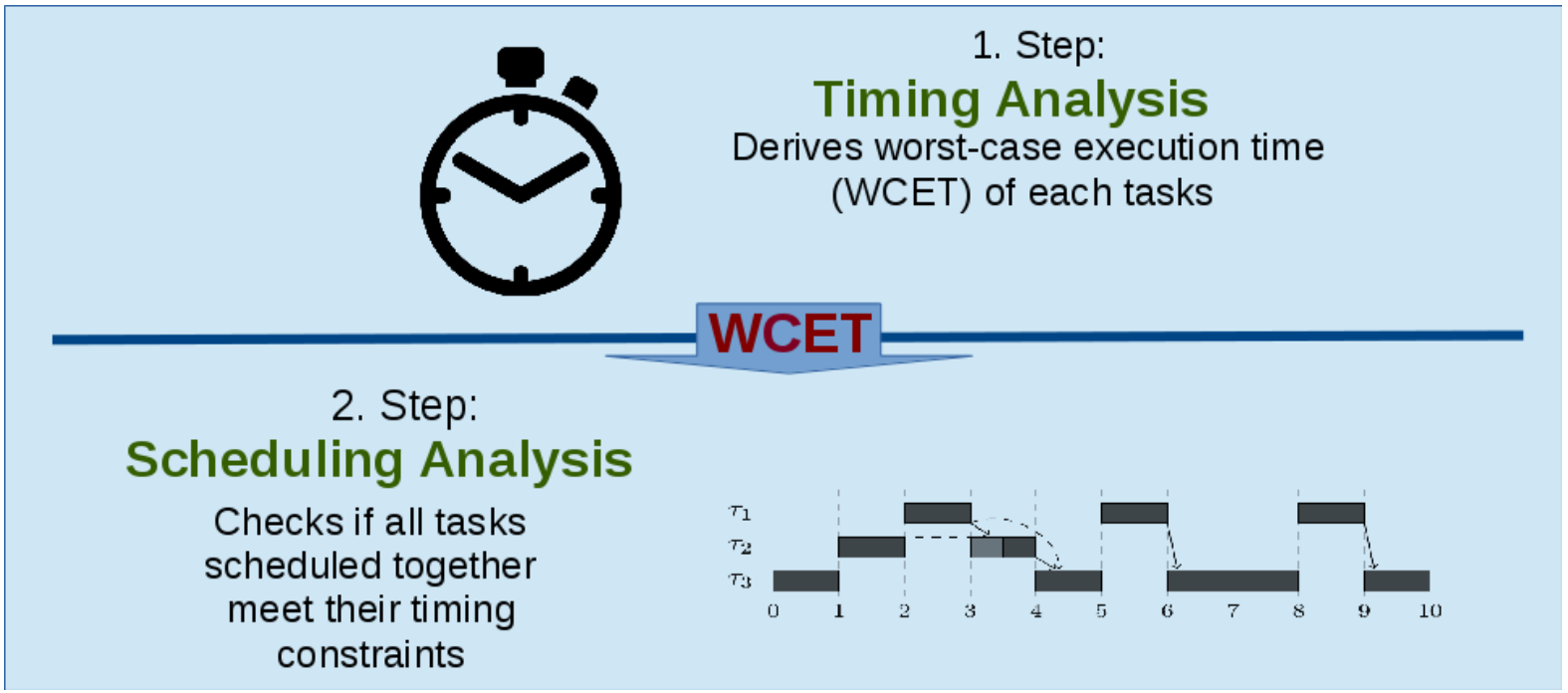
Scheduling Analysis determines if each task complies with its timing constraints when scheduled according to a predefined scheduling policy. Timing constraints are typically defined by a task's period and a task's deadline, both determined by the physical environment. Hence, tasks are assumed to be fully characterized by a triple consisting of a period, a deadline and an execution time bound, i.e. the WCET of the task.

Figure 1: Traditional 2-Step Timing Verification

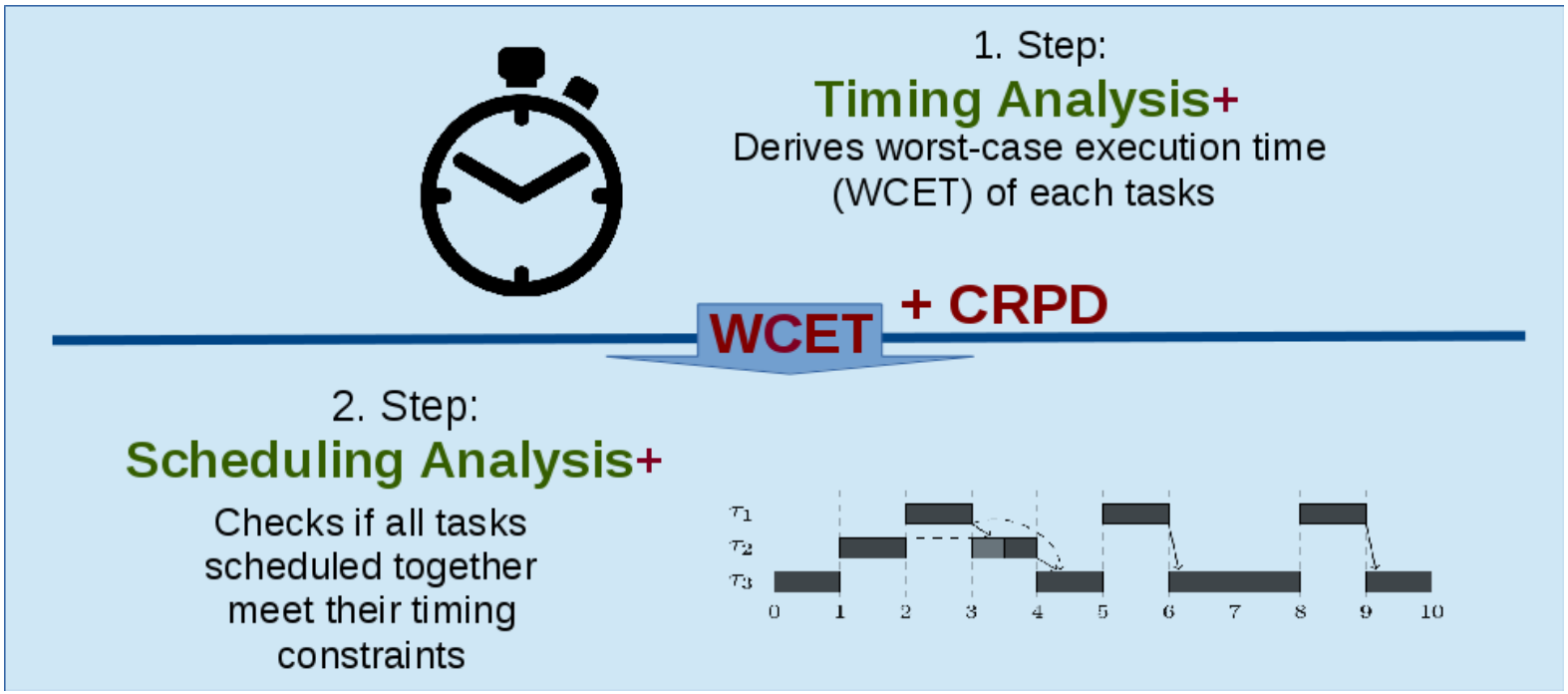
Traditionally, the complete **interface between timing analysis and scheduling analysis** is given by bounds on the tasks' execution times. Other input to the scheduling analysis–such as

3

Research Topic: Guaranteeing Timing Correctness



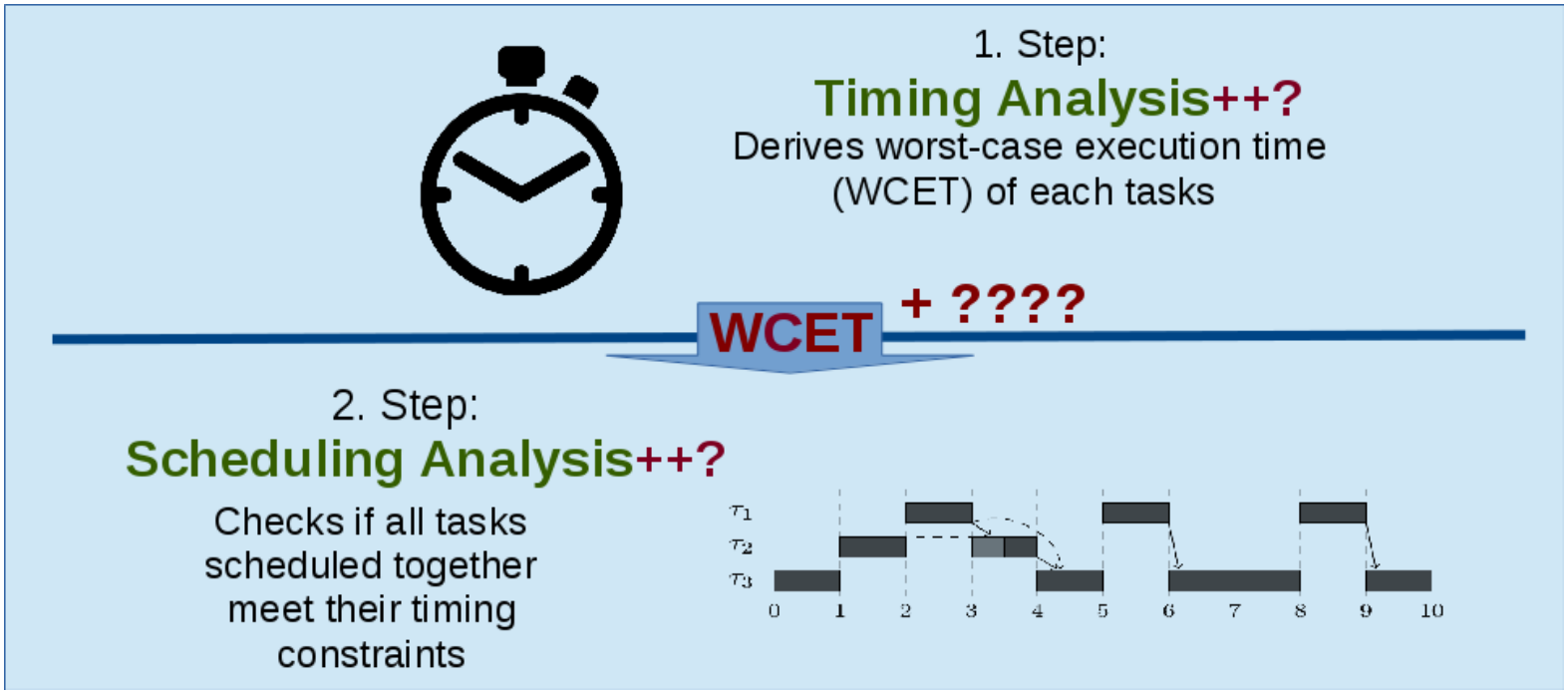
Timing Verification for pre-emptive Systems



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

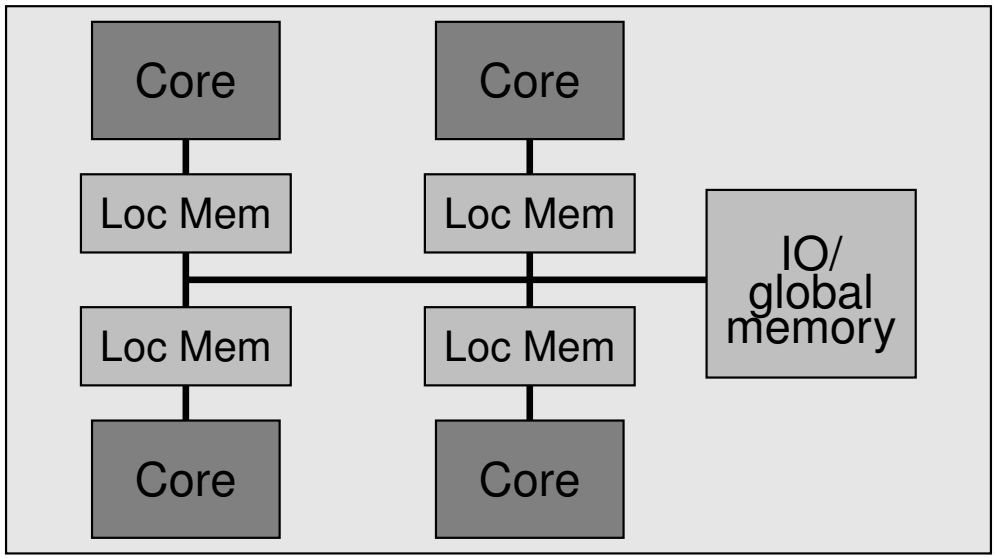
⇒ **repair it** by extending the interface

Timing Verification for Multicore Systems?



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

$\Rightarrow$ **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?

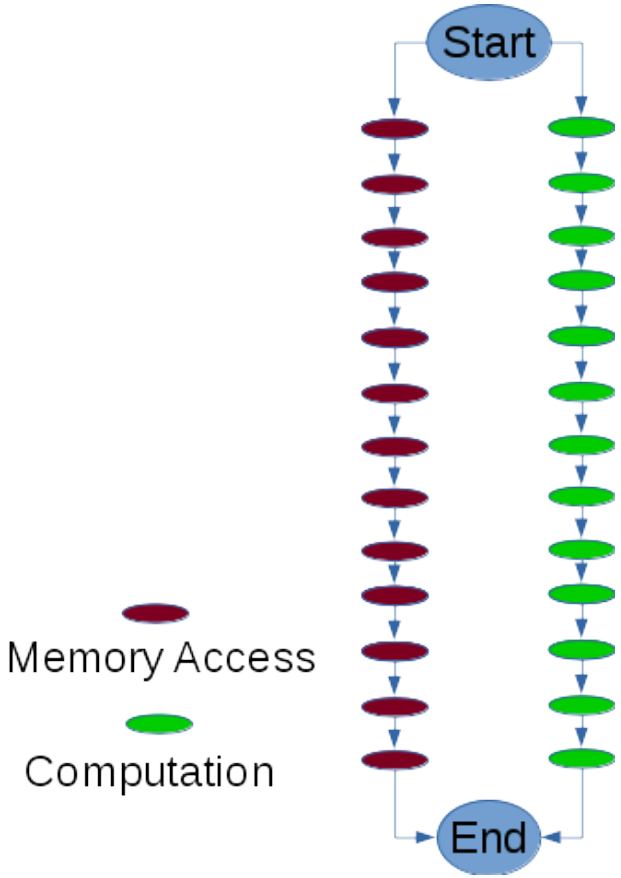
Yes, but it will be

- incomplete, or
- pessimistic

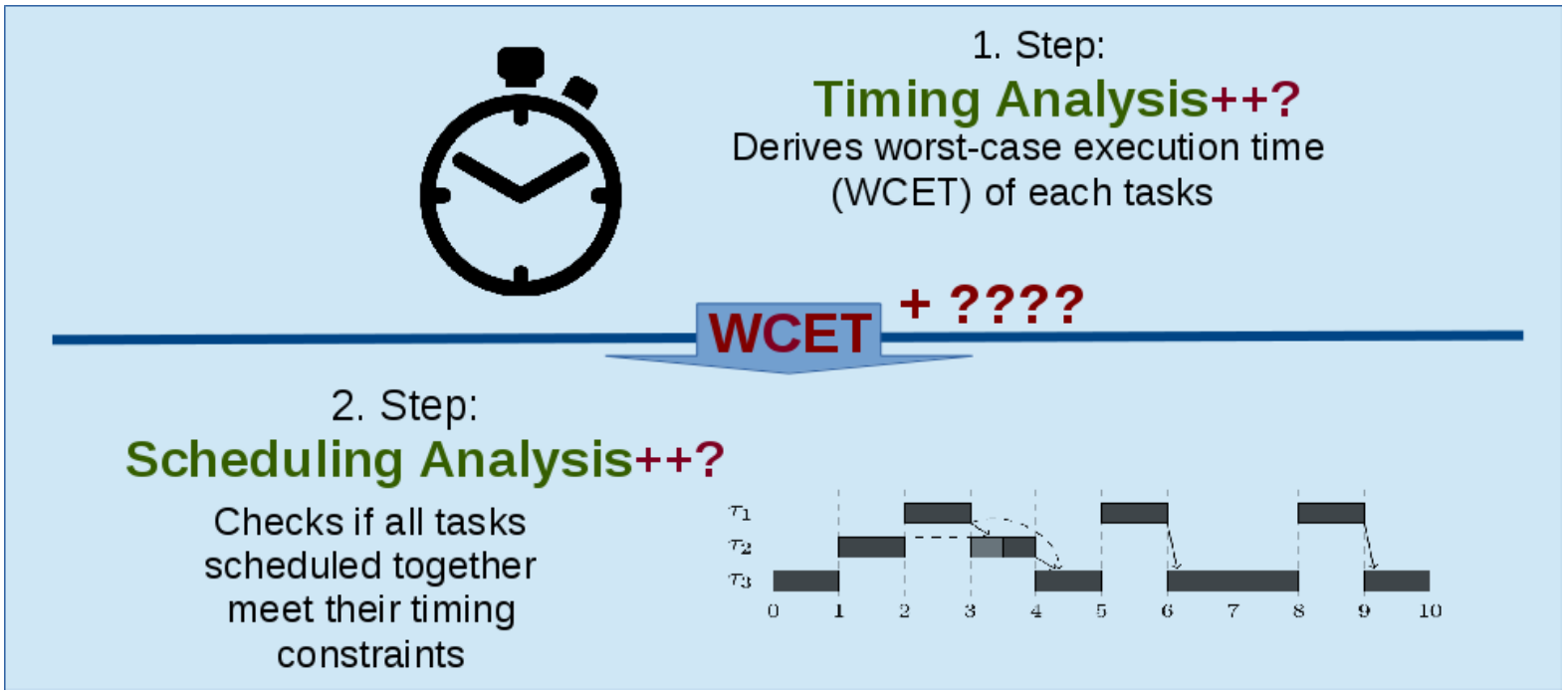
Consequence 2: ambiguous worst-case Path

Which path is worse with respect to

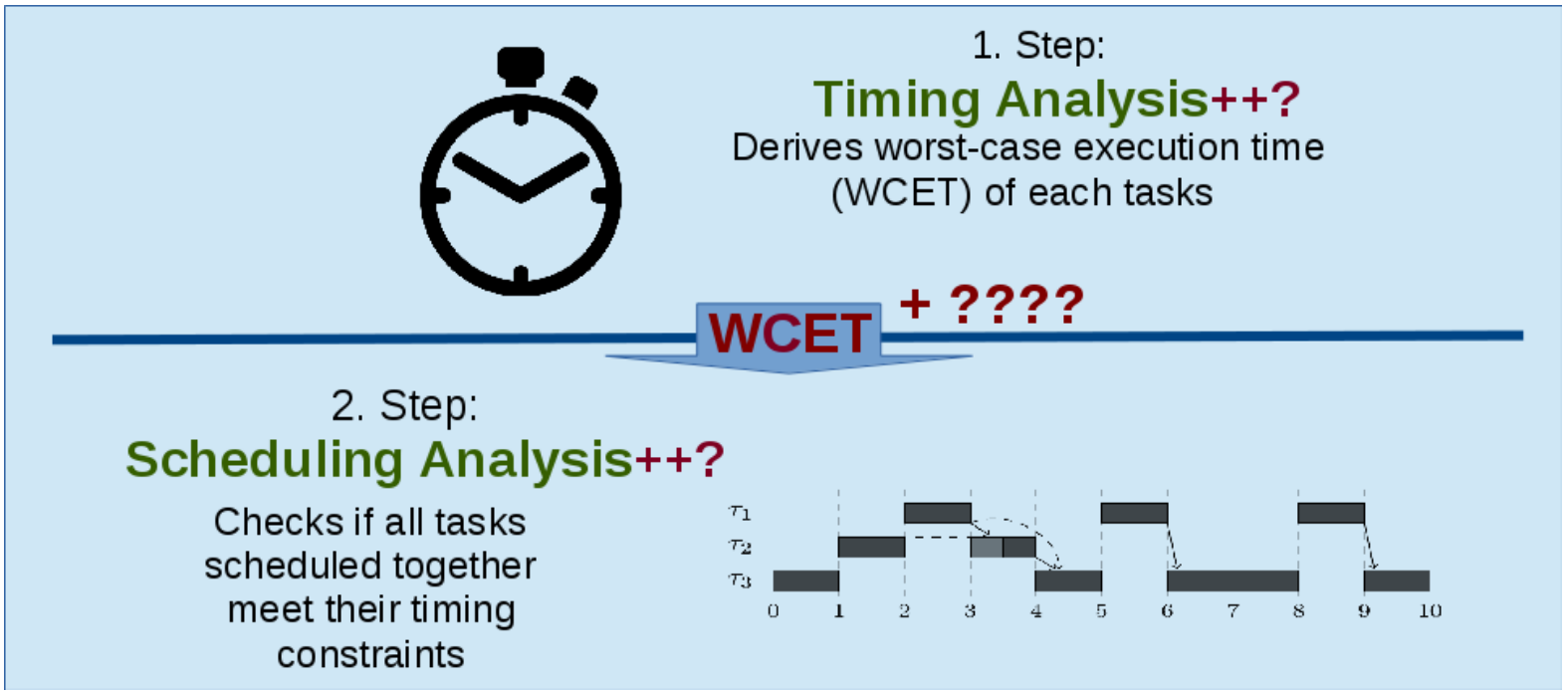
- a task scheduled on the same core?
- a task scheduled on another core?
- the pre-emption costs?
- ...



Timing Verification for Multicore Systems?



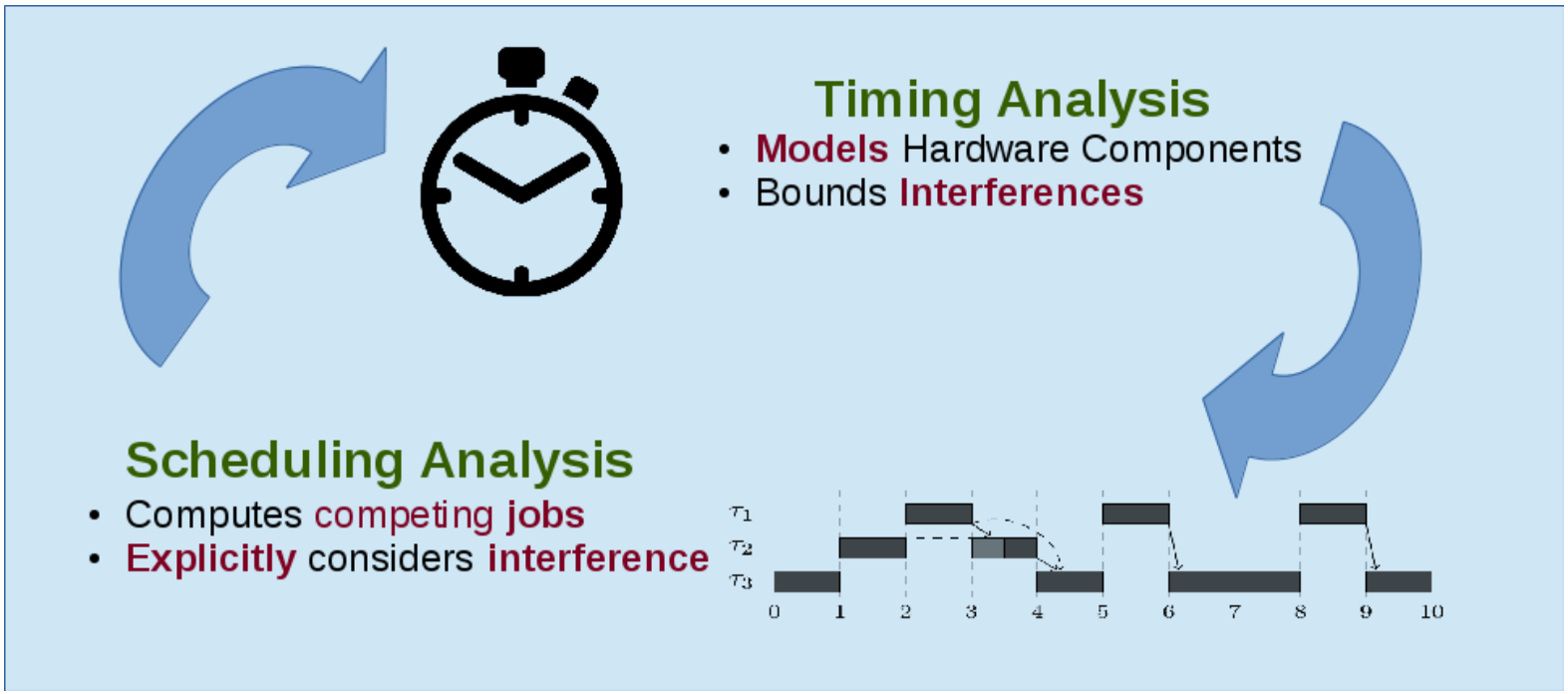
Timing Verification for Multicore Systems?



- **broken in case of multicores**:

$\Rightarrow$ **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 $\{\tau_1, \ldots, \tau_n\}$, per task

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

Task Model

n sporadic tasks $\{\tau_1, \ldots, \tau_n\}$, 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 $\{\tau_1, \ldots, \tau_n\}$, 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

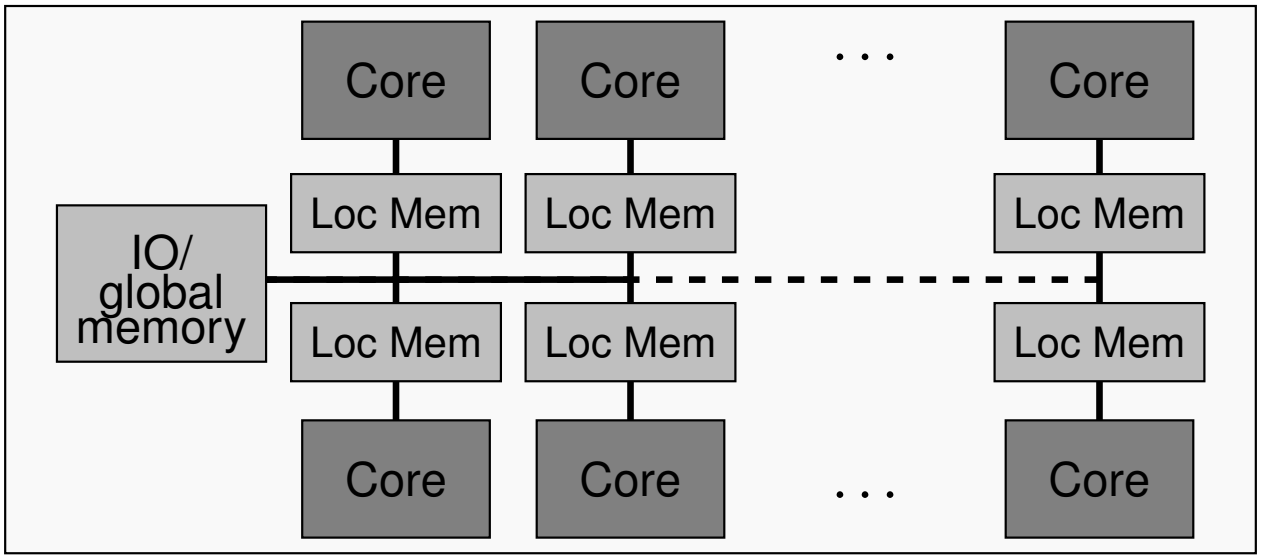
Instruction:

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

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

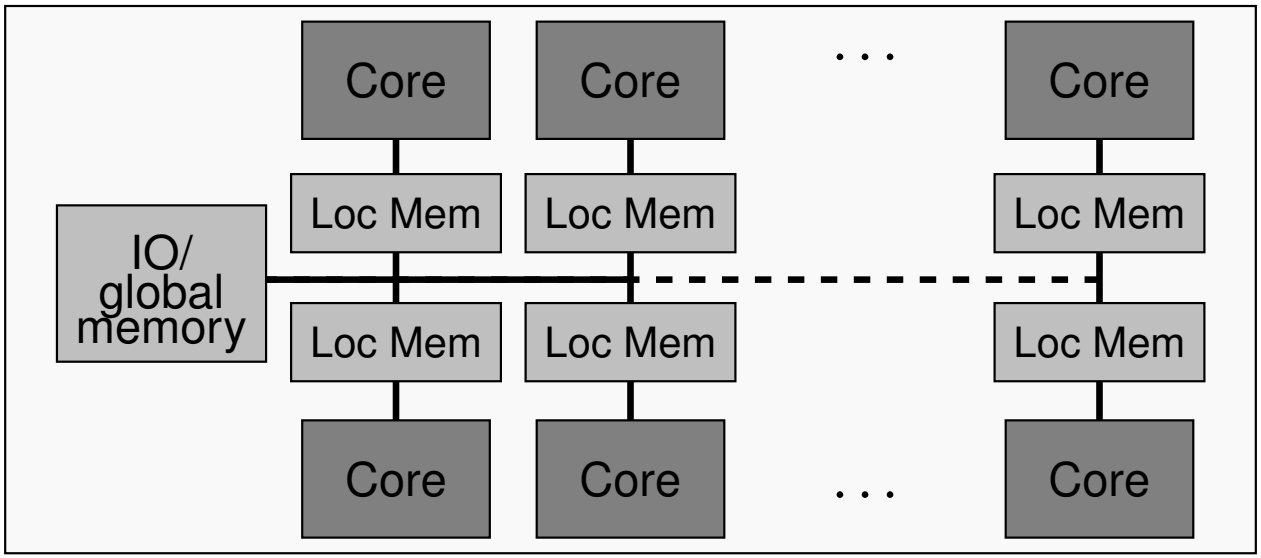
(simple, yet expressive task model)

Processor Model

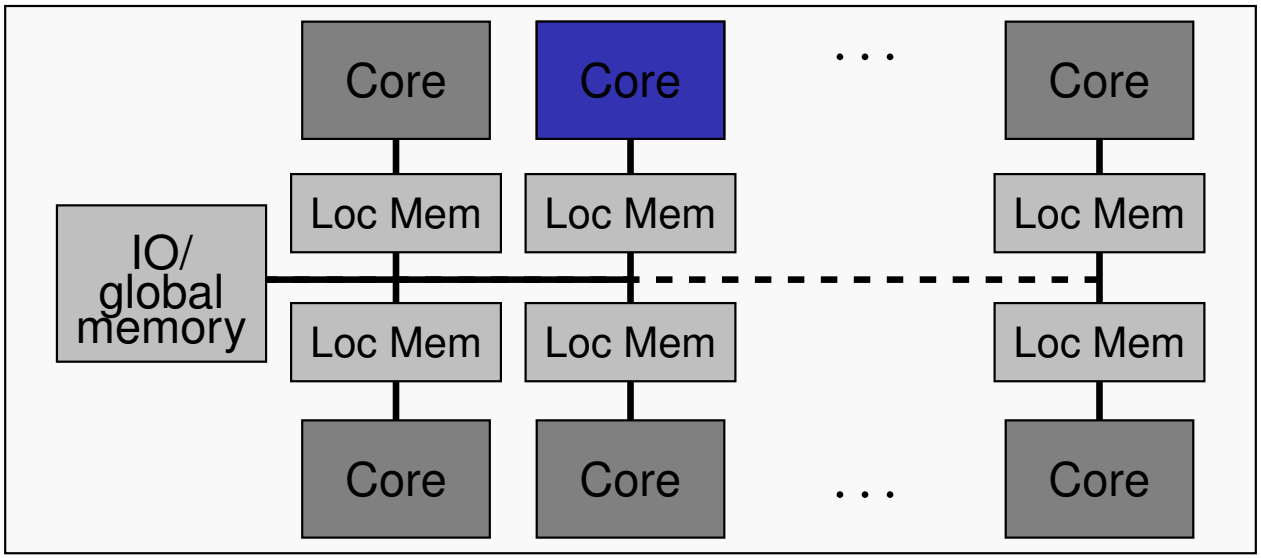


- ℓ identical cores $\{P_1, \ldots, P_\ell\}$,
- fixed-priority pre-emptive scheduling, partitioned tasks
- one shared bus
- local memories
- a global memory (DRAM)

Impact of the Multicore Components

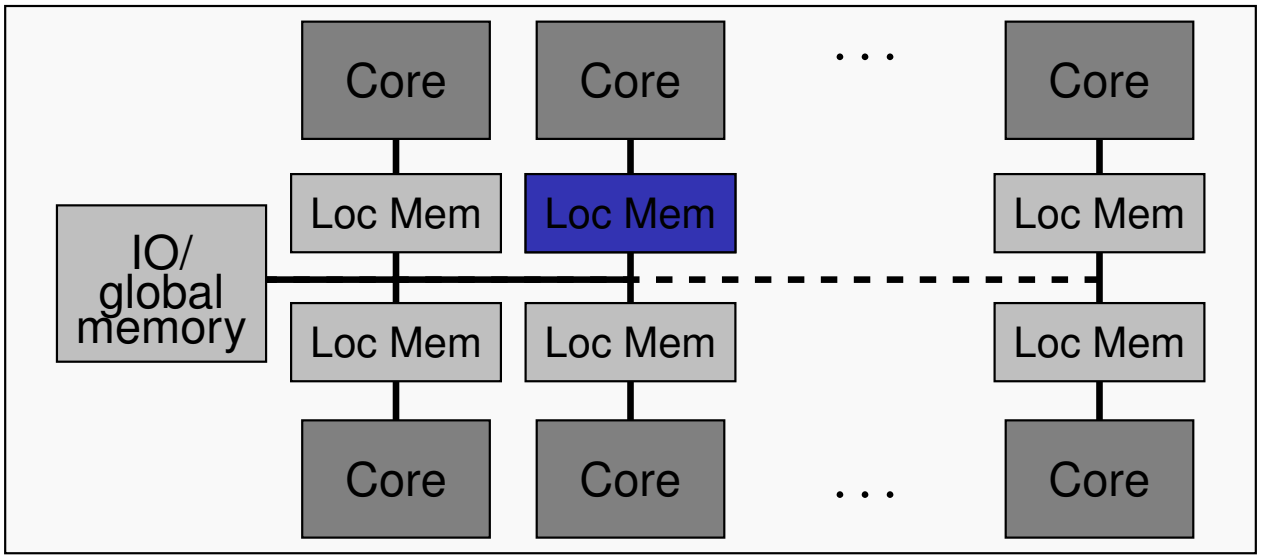


Impact of the Multicore Components



Core How long does it take to execute a trace?

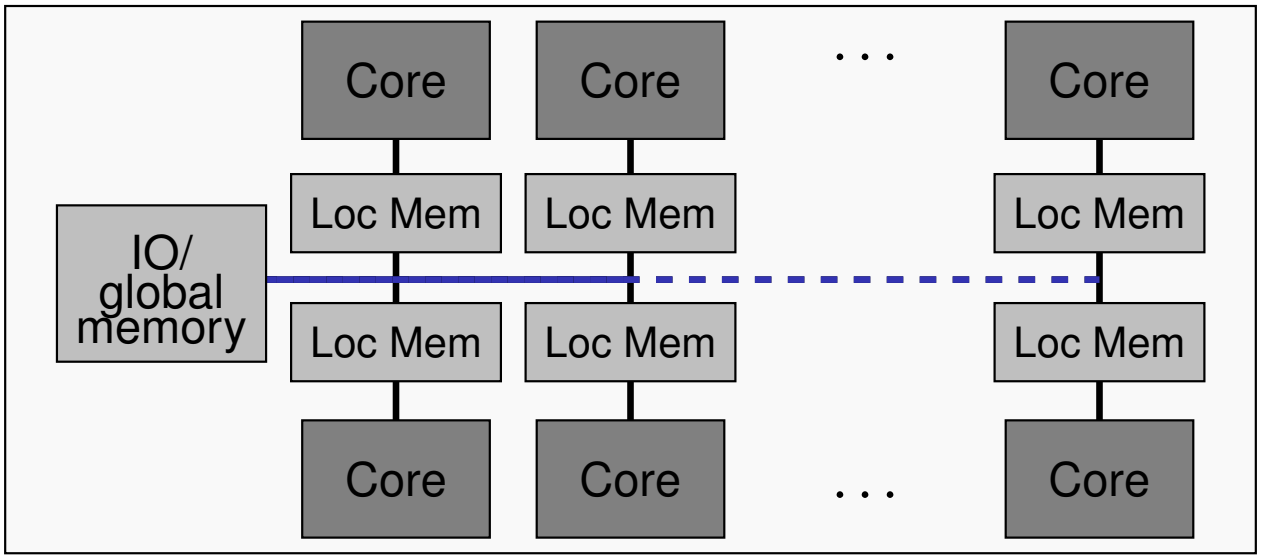
Impact of the Multicore Components



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

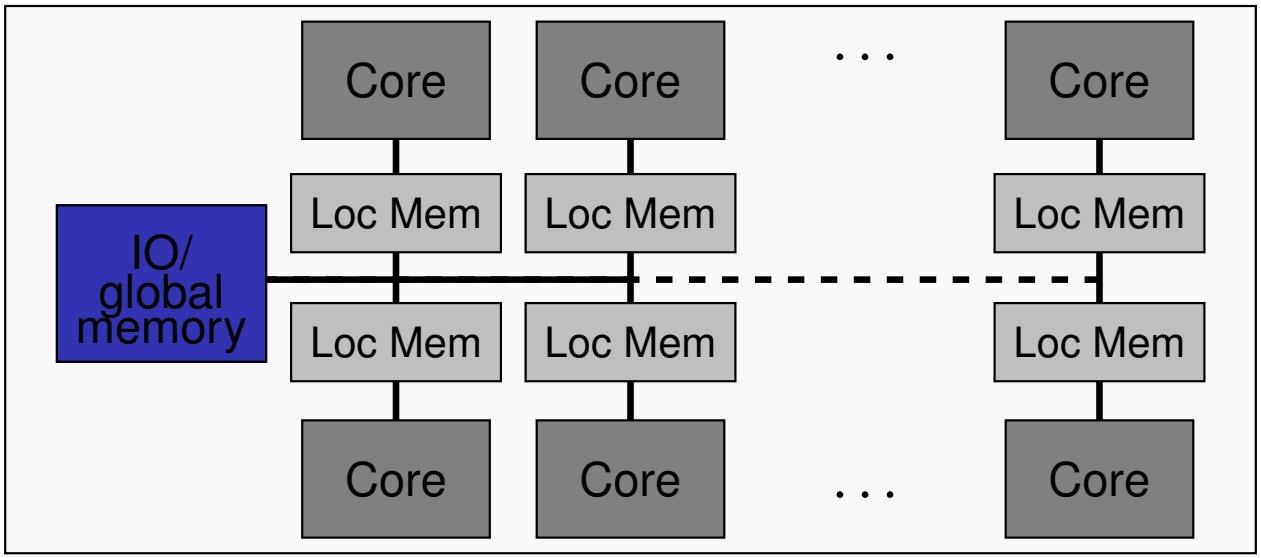


Core How long does it take to execute a trace?

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?

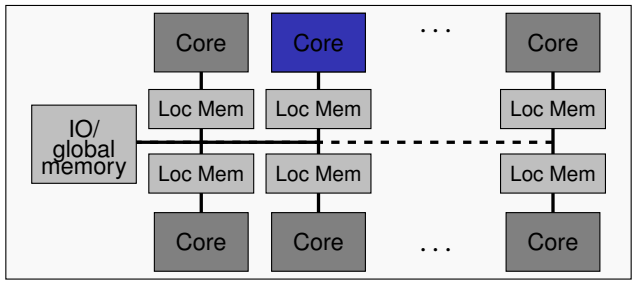
Local Memory How many memory requests go to the bus?

Bus How many competing accesses can occur?

Global Memory How many DRAM refreshes can occur?

Core: Processor Demand

$$\text{PROC} : \mathbb{O} \to \mathbb{N}$$



How long does it take to execute a trace?

Processor Demand of trace o:

$$\sum_{(\_,\Delta,\_)\in o} \Delta$$

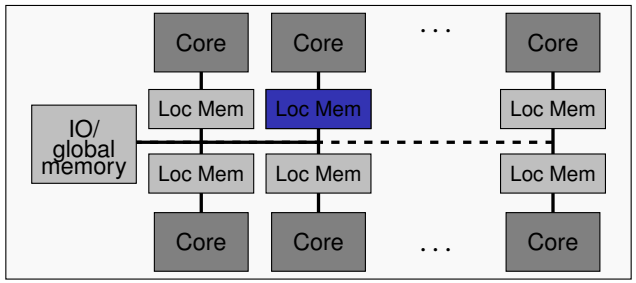
Processor demand of task τ_i:

$$\text{PD}_i = \max_{o\in O_i} \sum_{(\_,\Delta,\_)\in o} \Delta$$

(Δ: execution time of instruction without memory delays)

Local Memory: Memory Function/Demand

$$\mathrm{MEM}\colon \mathbb{O} \to \mathbb{N} \times 2^{2^{\mathbb{N}}} \times 2^{\mathbb{N}}$$



How many memory requests go to the bus?

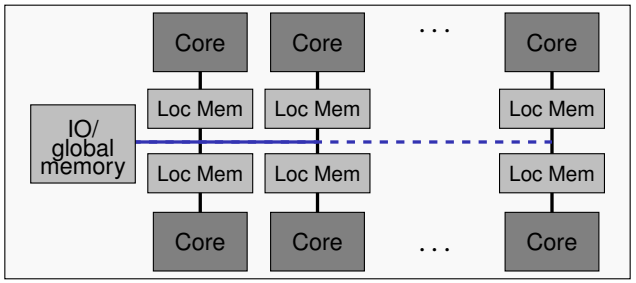
$$\mathrm{MEM}(o) = (\mathrm{MD}_o, \overline{\mathrm{UCB}}_o, \mathrm{ECB}_o)$$

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

Memory demand of task τ_i: $\mathrm{MD}_i = \max_{o \in O_i} \left\{ \mathrm{MD} \middle| \mathrm{MEM}(o) = (\mathrm{MD}, \_, \_) \right\}$

Bus Function

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



How many competing accesses can occur?

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

#bus accesses that delay task τ_i on processor P_x during time interval t

---

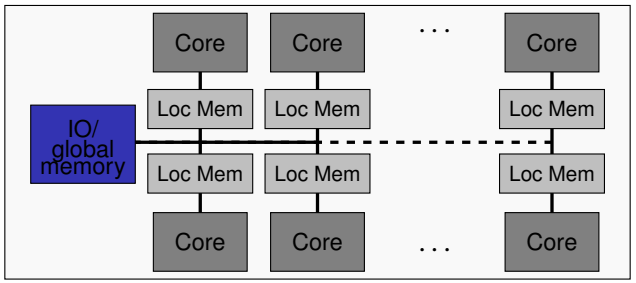
$S_i^x(t)$ #accesses due to τ_i and all higher priority tasks on P_x during t

$A_j^y(t)$ #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

$$\text{DRAM}: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

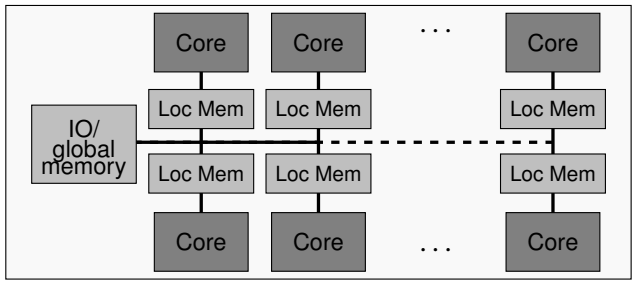


How many DRAM refreshes can occur?

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

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

Which components can we model:



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

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

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

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

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

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

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 τ_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 τ_i executing on processor P_x

---

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

From Component Model to Interferences

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

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

---

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

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

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 τ_i executing on processor P_x

---

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

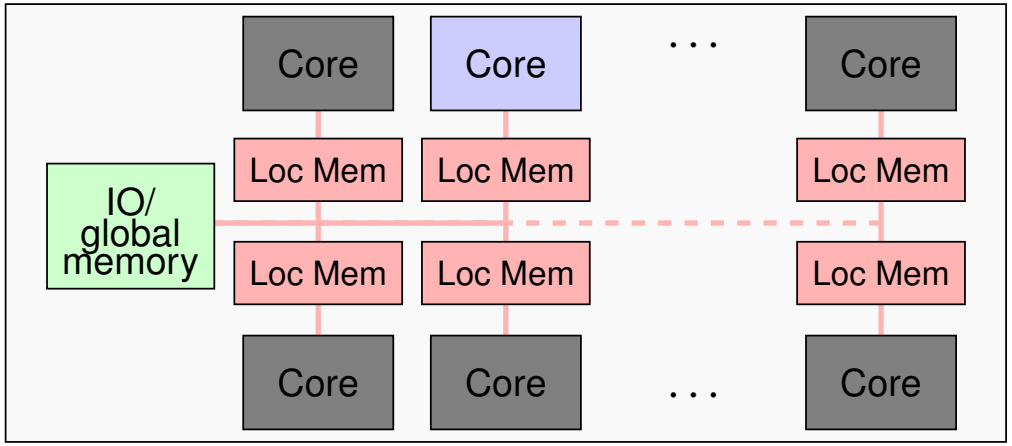
$$I^{\text{BUS}}(i, x, t) = \text{BUS}(i, x, t) \cdot d_{\text{main}}$$

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

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

where d_{refresh} is the refresh latency.

Multicore Response Time Analysis



$$R_i = \mathrm{PD}_i + I^{\mathrm{PROC}}(i, x, R_i) + I^{\mathrm{BUS}}(i, x, R_i) + I^{\mathrm{DRAM}}(i, x, R_i)$$

(solved via fixed-point iteration)

Task set feasible, if:

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

Use cases of the Framework

Timing Verification Tool

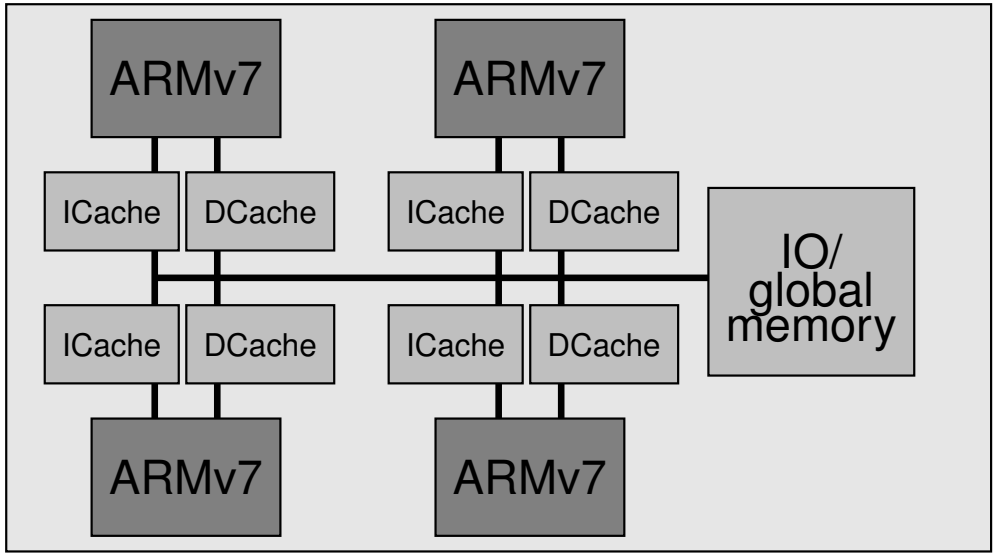
- for multicore systems including local/global memory and bus
- extensible to other sources of interference
- generic, i.e., can be adapted to different architectures
- supports scenario-based analysis

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

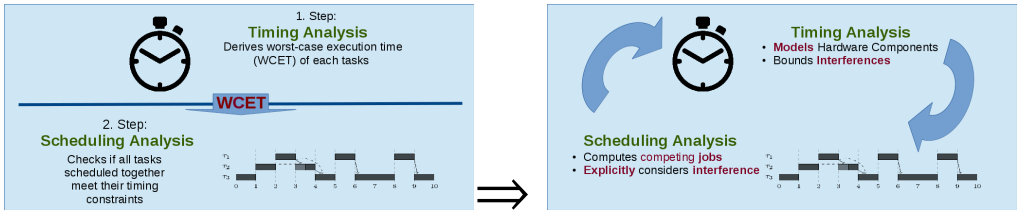
- System based on the ARM Cortex A5:



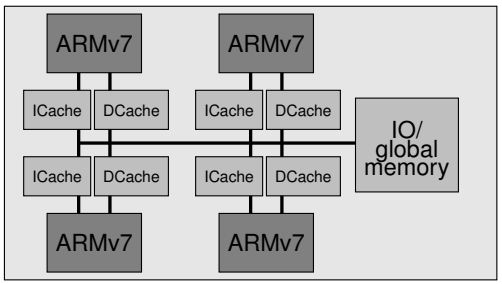
- 4 cores, separate instruction and data caches, FP/FIFO/TDMA bus, and distributed DRAM controller.
- Compared different configurations for a large number of randomly generated task sets

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:

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

where k is the number of instructions on trace o

Data Memory:

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

Memory Function - Scratchpad

$$\mathrm{MEM}\colon \mathbb{O} \to \mathbb{N} \times 2^{2^{\mathbb{N}}} \times 2^{\mathbb{N}}$$

Scratchpad memory function $\mathrm{SPM}\colon \mathbb{M} \to \{\mathit{true}, \mathit{false}\}$
Instruction Memory:

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

Data Memory:

$$\mathrm{MEM}^{\mathrm{da}}_{\mathrm{sp}}(o) = \left(\left| \left\{ m^{\mathrm{da}} \middle| \Big((\_, \_, r(m^{\mathrm{da}})) \in o \wedge \neg \mathrm{SPM}(m^{\mathrm{da}}) \Big) \right. \right.$$

$$\left. \left. \vee\, (\_, \_, w(m^{\mathrm{da}})) \in o \right\} \right|, \emptyset, \emptyset \Bigg)$$

Memory Function - LRU Cache

$$\mathrm{MEM}\colon \mathbb{O} \to \mathbb{N} \times 2^{2^{\mathbb{N}}} \times 2^{\mathbb{N}}$$

Cache function $\mathrm{Hit}\colon \mathbb{I} \times \mathbb{M} \to \{\mathit{true}, \mathit{false}\}$
Instruction Memory:

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

Data Memory:

$$\mathrm{MEM}^{\mathrm{da}}_{\mathrm{ca}}(o) = \left(\left| \left\{ m^{\mathrm{da}} | \left(\iota_i = (\_, \_, r(m^{\mathrm{da}})\right) \in o \wedge \neg \mathrm{Hit}(m^{\mathrm{da}}, \iota_i) \right) \right.\right.$$

$$\left.\left. \vee\, (\_, \_, w(m^{\mathrm{da}})) \in o \right\} \right|, \overline{\mathrm{UCB}}^{\mathrm{da}}_o, \mathrm{ECB}^{\mathrm{da}}_o \Bigg)$$

Bus Function - FIFO Bus

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

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

(equal number of slots v per processor)

$S_i^x(t)$ #bus accesses due to τ_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

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

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

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

$S_i^x(t)$ #bus accesses due to τ_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

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

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

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

- $S_i^x(t)$ #bus accesses due to τ_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

$$\text{DRAM}\colon \mathbb{N} \times \mathbb{P} \times \mathbb{N} \to \mathbb{N}$$

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

$\#$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_{refresh} is the refresh interval, $\#$rows the number of DRAM rows.