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

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 task.

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

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?
- ...

Start

Memory Access

Computation

End
Timing Verification for Multicore Systems?

1. Step:
   **Timing Analysis++?**
   Derives worst-case execution time (WCET) of each task.

2. Step:
   **Scheduling Analysis++?**
   Checks if all tasks scheduled together meet their timing constraints.

破壊しがちな場合のマルチコアでは再考が必要です。
Timing Verification for Multicore Systems?

1. Step:
Timing Analysis++?
Derives worst-case execution time (WCET) of each tasks

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

What we are not doing

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

\[ n \text{ sporadic tasks } \{\tau_1, \ldots, \tau_n\}, \text{ per task} \]

- **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 \( G_i \)
- set of **traces** \( O_i \), ordered lists of instructions

**Instruction:**

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

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

(simple, yet expressive task model)
Processor Model

- $\ell$ 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

- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem
- Core
- Loc Mem

- IO/global memory

- How long does it take to execute a trace?
- How many memory requests go to the bus?
- How many competing accesses can occur?
- How many DRAM refreshes can occur?
Impact of the Multicore Components

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 $o$:

$$\sum_{(\Delta_{i,j}) \in o} \Delta$$

Processor demand of task $\tau_i$:

$$PD_i = \max_{o \in O_i} \sum_{(\Delta_{i,j}) \in o} \Delta$$

($\Delta$: execution time of instruction without memory delays)
Local Memory: Memory Function/Demand

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

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

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

**BUS:** \( N \times P \times N \rightarrow N \)

How many competing accesses can occur?

**BUS**(\( i, x, t \))

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

\( S^x_i(t) \) \#accesses due to \( \tau_i \) and all higher priority tasks on \( P_x \) during \( t \)

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

\( S^x_i(t) \) and \( A^y_j(t) \) 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**: $\mathcal{O} \rightarrow \mathbb{N}$ any timing-composable core

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

\[ I^{\text{Comp}}(i, x, R_i) \]

Interference/Delay of component \( \text{Comp} \) on the response time \( R_i \)
of task \( \tau_i \) executing on processor \( P_x \)
$I^{\text{Comp}}(i, x, R_i)$

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

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

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

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

where \( d_{\text{main}} \) is the bus access latency to the global memory.
From Component Model to Interferences

\[ I^{\text{Comp}}(i, x, R_i) \]

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

\[ I^{\text{PROC}}(i, x, t) = \sum_{j \in \Gamma_x \land j \in \text{hp}(i)} \left\lceil \frac{t}{T_j} \right\rceil \cdot PD_j \]

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

where \( d_{\text{main}} \) 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 \( d_{\text{refresh}} \) 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

\[
\text{MEM}: \emptyset \rightarrow \mathbb{N} \times 2^{2^\mathbb{N}} \times 2^\mathbb{N}
\]

Instruction Memory:

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

where \( k \) is the number of instructions on trace \( o \)

Data Memory:

\[
\text{MEM}_{\text{da}}^\text{nc}(o) = \left( \left\{ \ell_i | \ell_i \in o \land \ell_i = (-, -, r/w[m_{\text{da}}]) \right\}, \emptyset, \emptyset \right)
\]
Memory Function - Scratchpad

\[ \text{MEM}: \emptyset \rightarrow \mathbb{N} \times 2^{2\mathbb{N}} \times 2^{\mathbb{N}} \]

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

Instruction Memory:

\[
\text{MEM}_{\text{sp}}^{\text{in}}(o) = \left( \| \{ m^{\text{in}} | (m^{\text{in}}, \_ , \_) \in o \land \neg \text{SPM}(m^{\text{in}}) \} \| , \emptyset , \emptyset \right)
\]

Data Memory:

\[
\text{MEM}_{\text{sp}}^{\text{da}}(o) = \left( \| \{ m^{\text{da}} | (\_, \_, r(m^{\text{da}})) \in o \land \neg \text{SPM}(m^{\text{da}}) \} \| , \emptyset , \emptyset \right)
\]

\[
\lor (\_, \_, w(m^{\text{da}})) \in o \left\| , \emptyset , \emptyset \right)\)
Memory Function - LRU Cache

\[ \text{MEM: } \emptyset \rightarrow \mathbb{N} \times 2^2 \times 2^N \]

Cache function Hit: \( I \times M \rightarrow \{true, false\} \)

Instruction Memory:

\[
\text{MEM}^{\text{in}}_{\text{ca}}(o) = \left( \{ m^{\text{in}}_{i} = (m^{\text{in}}, _, _) \in o \land \neg \text{Hit}(m^{\text{in}}, i) \} \right) \cup \text{UCB}^{\text{in}}_{o}, \text{ECB}^{\text{in}}_{o}
\]

Data Memory:

\[
\text{MEM}^{\text{da}}_{\text{ca}}(o) = \left( \{ m^{\text{da}}(i) = ( _, _, r(m^{\text{da}})) \in o \land \neg \text{Hit}(m^{\text{da}}, i) \} \right)
\]

\[
\lor ( _, _, w(m^{\text{da}})) \in o \} \cup \text{UCB}^{\text{da}}_{o}, \text{ECB}^{\text{da}}_{o}
\]
Bus Function - FIFO Bus

**BUS**: \( \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N} \)

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

(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

**BUS**: \( \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{N} \)

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

\( \min \left( S_i^x(t), \sum_{y \neq x} L_i^y(t) \right) \) 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.