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
Motivation and Context

Multicore Response Time Analysis

Evaluation

Conclusions
Multicore Timing Verification: Traditional Approach

Implicit assumptions:

- Tasks can be analyzed independently
- WCETs are context independent
Problems with context-independent WCETs

Non-pre-emptive uniprocessor:

![Diagram showing τ1, τ2, τ3 with τ1 and τ3 green and τ2 blue, indicating τ1 + τ3 works well.]

Pre-emptive uniprocessor:

![Diagram showing the same structure as above, but with τ1 and τ3 being blue and τ2 green, indicating τ1 + τ3 works relatively well.]

Multicore:

- Core 1: τ1
- Core 2: τ2
- Core 3: τ3
- ...
Problems with context-independent WCETs

![Diagram showing multiple cores and memory access]

Core 1: $\tau_1$
Core 2: $\tau_2$
Core 3: $\tau_3$
...

Memory Access
Problems with context-independent WCETs

What is the context-independent worst-case delay?
Problems with context-independent WCETs

What is the context-independent worst-case delay?
Problems with context-independent WCETs

What is the context-independent worst-case delay?
Problems with context-independent WCETs

→ Highly inflated execution time bounds
  (multicore may perform worse than single cores)
Multicore Timing Verification: Isolation

Isolate tasks from each other, remove interference
Multicore Timing Verification: Isolation

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

Isolate tasks from each other, remove interference

Core 1: \( \tau_1 \)

Core 2:

Core 3:

...
Multicore Timing Verification: Isolation

Isolate tasks from each other, remove interference

Core 1: \( \tau_1 \)

Core 2:

Core 3:

...
Multicore Timing Verification: Isolation

Isolate tasks from each other, remove interference

Core 1: $\tau_1$
Core 2: $\tau_2$
Core 3: $\tau_3$
...

Pays for interference, even if there is none
Multicore Timing Verification: Fully Integrated Approach

- One, all-combining analysis
- Analyze exact interleavings
Multicore Timing Verification: Fully Integrated Approach

- One, all-combining analysis
- Analyze exact interleavings

Core 1: \( \tau_1 \)
Core 2: \( \tau_2 \)
Core 3: \( \tau_3 \)
...

Memory Access

Promises best precision
Multicore Timing Verification: Fully Integrated Approach

- One, all-combining analysis
- Analyze exact interleavings

Promises best precision
Multicore Timing Verification: Fully Integrated Approach

- One, all-combining analysis
- Analyze exact interleavings

Promises best precision

Memory Access  Jitter
Multicore Timing Verification: Fully Integrated Approach

- One, all-combining analysis
- Analyze exact interleavings

Promises best precision, but very high complexity. Too high?
Multicore Timing Verification: Comparisons

- Guaranteed performance
- Complexity

X Isolation
X Fully Integrated
X Traditional Timing Verification
Multicore Timing Verification: Comparisons

- Guaranteed performance
- Isolation
- Traditional Timing Verification
- Interference Analysis

- Fully Integrated
Interference Analysis

Decompose

\( \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \) and re-assemble \( \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \) over the response time.
Interference Analysis

Decompose

\[ \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \]

and re-assemble

\[ \tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \]
Interference Analysis

Decompose

\[ \tau_1 \tau_2 \tau_3 \Rightarrow \tau_1 \tau_2 \tau_3 \]

and re-assemble

over the response time:

release \hspace{1cm} \text{deadline}
Motivation and Context

Multicore Response Time Analysis

Evaluation

Conclusions
Analysis Framework

Multicore architecture with shared components:

```
Core
Loc Mem
Core
Loc Mem
Core
Loc Mem
...  ...
Core
Loc Mem
Core
Loc Mem
...  ...
Core
Loc Mem

IO/global memory
```

What is the impact of each component on a task's response time:

$$R_i = \text{Delay on the core} + \text{Delay on the bus/local memory} + \text{Delay on the global memory}$$
Analysis Framework

Multicore architecture with shared components:

What is the impact of each component on a task’s response time:

\[ R_i = \text{Delay on the core} + \text{Delay on the bus/local memory} + \text{Delay on the global memory} \]
Targeted 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  How long does it take to execute a task?

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

How long does it take to execute a task?

Provides:

- processor demand PD of a task
  i.e., execution time without any interference, memory delays, etc.
Local Memory: Memory Demand

\[ \text{MEM}(o) = (\text{MD}, \text{UCB}, \text{ECB}) \]

How many memory requests go to the bus?

Provides:
- memory demand MD, i.e., \# bus accesses
- metrics for the pre-emption costs (UCB, ECB)
Bus: Competing Accesses

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

How many competing accesses can occur?

---

Provides:

- \#bus accesses that delay task $\tau_i$ on processor $P_x$ during time $t$
Bus: Competing Accesses

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

How many competing accesses can occur?

Provides:
- **#bus accesses** that delay task \(\tau_i\) on processor \(P_x\) during time \(t\)

Uses

\[
S(t) \quad \#\text{competing accesses on same core}
\]

\[
A(t) \quad \#\text{competing accesses on all other cores}
\]

Derived using output of the memory function: MD, UCBs and ECBs
How many DRAM refreshes can occur?

Provides:
- \#DRAM refreshes during time $t$ with up to $m$ memory accesses
Which components can we model so far?

**Core**: any timing-compositional core

**Local Mem.**: Scratchpads, LRU/DM caches, partitioned caches, uncached systems (all for instruction and data)

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

**DRAM**: burst refreshes, distributed refreshes

and any combination thereof.
From Component Model to Interferences

$I^C(i, x, R_i)$

Interference/Delay of component $C$ during the response time $R_i$ of task $\tau_i$ executing on processor $P_x$
From Component Model to Interferences

\[ I^C(i, x, R_i) \]

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

\[ I^{\text{PROC}}(i, x, t) = \sum_{j \in \Gamma_x \land j \in h_p(i)} \left\lceil \frac{t}{T_j} \right\rceil P_{D_j} \]
From Component Model to Interferences

\[ I^C(i, x, R_i) \]

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

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

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

where \( d_{\text{main}} \) is the bus access latency to the global memory.
Interference/Delay of component $C$ during the response time $R_i$ of task $\tau_i$ executing on processor $P_x$

$$I^C(i, x, R_i)$$

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

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

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:

\[ \forall i : R_i \leq D_i \]
Motivation and Context

Multicore Response Time Analysis

Evaluation

Conclusions
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
Randomly generated task sets

Task set parameters

- 32 tasks in total, with 8 tasks per core, uniform core utilization
- each task was randomly assigned a task from Mälardalen benchmark suite (see table)
- implicit deadlines
- priorities in deadline monotonic order.

<table>
<thead>
<tr>
<th>Name</th>
<th># Instr. (PD)</th>
<th>Read/Write</th>
<th>MD</th>
<th>UCB</th>
<th>ECB</th>
</tr>
</thead>
<tbody>
<tr>
<td>adpcm_enc</td>
<td>628795</td>
<td>124168</td>
<td>38729</td>
<td>155</td>
<td>346</td>
</tr>
<tr>
<td>bsort100</td>
<td>272715</td>
<td>1305613</td>
<td>25464</td>
<td>31</td>
<td>135</td>
</tr>
<tr>
<td>compress</td>
<td>8793</td>
<td>3358</td>
<td>993</td>
<td>74</td>
<td>174</td>
</tr>
<tr>
<td>fdct</td>
<td>5923</td>
<td>3098</td>
<td>1088</td>
<td>67</td>
<td>193</td>
</tr>
<tr>
<td>lms</td>
<td>3023813</td>
<td>373874</td>
<td>120821</td>
<td>150</td>
<td>276</td>
</tr>
<tr>
<td>nsichneu</td>
<td>8648</td>
<td>4841</td>
<td>1582</td>
<td>397</td>
<td>589</td>
</tr>
</tbody>
</table>

...
Results: Core Utilization

1000 task sets per (core) utilization

Schedulable Tasksets
Core Utilization
reference config - perfect bus
reference config - FP bus
reference config - RR bus
reference config - TDMA bus
full-isolation architecture
reference config - PP bus
reference config - FIFO bus
uncached architecture
Results: Core Utilization

1000 task sets per (core) utilization

without local caches: worst performance
Results: Core Utilization

1000 task sets per (core) utilization

full isolation (TDMA bus + cache partitioning)
Results: Core Utilization

1000 task sets per (core) utilization

- reference config - perfect bus
- reference config - FP bus
- reference config - RR bus
- reference config - TDMA bus
- full-isolation architecture
- reference config - PP bus
- reference config - FIFO bus
- uncached architecture

round-robin/TDMA bus
Results: Core Utilization

1000 task sets per (core) utilization

Schedulable Tasksets vs Core Utilization for different reference configurations and architectures:
- Perfect bus
- Fixed-priority (FP) bus
- Round-robin (RR) bus
- Time-division multiplexing (TDMA) bus
- Full-isolation architecture
- Predictive/predictive (PP) bus
- First-in, first-out (FIFO) bus
- Uncached architecture

Fixed-Priority Bus: work-conserving, best performance
Results: Core Utilization

1000 task sets per (core) utilization

perfect bus: theoretical upper bound on the performance
Results: Bus Utilization

schedable task sets vs. bus utilization

reference config - perfect bus
reference config - FP bus
reference config - RR bus
reference config - TDMA bus
full-isolation architecture
reference config - PP bus
reference config - FIFO bus
uncached architecture

better results: bus/global memory is the bottleneck
Motivation and Context

Multicore Response Time Analysis

Evaluation

Conclusions
Conclusions

Multicore Response Time Analysis framework
▶ based on interference modelling
▶ directly aiming at response time
▶ parametric in the hardware configuration
▶ extensible to other sources of interference
▶ but ignores overlapping

Proof-Of-Concept Implementation
▶ based on ARM Cortex A5
▶ temporal isolation not needed
▶ promising results for work-conserving bus policies
Questions?