Integrated Timing Analysis of Application and Operating Systems Code

Lee Kee CHONG
Clement BALLABRIGA  Van-Thuan PHAM
Sudipta CHATTOPADHYAY  Abhik ROYCHOWDHURY
Example of a real-time system

- Hazardous environment - in need of an autonomous robot

- Uneven, tight terrain - self balancing on two wheels

Example:
A robot used in Fukushima disaster cleanup
<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
</table>
| balance | Needs to keep upright all the time.  
*Must consistently run at 50Hz* |
| navigation | Auto navigation and collision detection.  
*Must consistently run at 20Hz* |
| remote  | Receives remote commands.  
*Must finish processing within 100 ms* |

Requires precise timing, high reliability

Real-time Operating System (RTOS)
Goal: We want to verify that all real-time constraints are met!

- Worst Case Response Time (WCRT) of an application running on top of an RTOS

- Need to consider timing effects of RTOS:
  - System call
  - Interrupt
Application timing analysis

Heptane [ECRTS ‘01]
SWEET [WCET ‘05]
Otawa [ERTS ’06]
Chronos [SCP ‘07]

Real-time operating system (RTOS) timing analysis

RTEMS [ECRTS ‘01]
OSE [RT-TOOLS ‘02]
µC/OS-II [CSE ’09]
seL4 [RTSS ’11]

Timing analysis of application + RTOS?
Can we analyze app and RTOS in isolation and combine the result?

Yes...

Add fixed delay to account for all mode/context switches

But... gross overestimation may occur

Due to loss of application context
Application layer

{ m₃, m₂, m₁, m₀ }

(4-way FIFO cache)

inserted last

System call

m₄, m₅

inserted earliest

OS layer

{ ?, ?, ?, ?, ? }

Application layer

Must assume all possible cache states!

Analysis of RTOS in isolation

memory blocks read
\{ m_3, m_2, m_1, m_0 \}

Application layer

\{ m_5, m_4, m_3, m_2 \}

Only one feasible cache state

Takes into account application context

System call

m_4, m_5
We propose...

- A timing analysis framework for app + OS
- Consider application context when analyzing OS
- Take into account timing effects of all interferences from OS
Assumptions

- Fixed priority preemptive scheduling
- Each interrupt is serviced by an interrupt service routine (ISR)
- A task/ISR can be periodic or sporadic
- For a sporadic task/ISR, assume release/arrival at its minimum inter-arrival time
Key idea

Capture *changes* to underlying *micro-architectural states* at application layer.
Key idea

Capture timing effects compositionally

- Main task
- ISR
- Preemtping task
- System call
- System call

WCRT computation

{WCET, micro-arch summary}

{WCET}

{WCET, micro-arch summary}
Assume static predictor, fixed penalty

Pipeline flushed at entry/exit of system call and ISR

Cache damage: Bounds the number of cache conflicts per cache set

In this work, we concentrate on the effect of caches.
WCRT computation

CRPD analysis

Main task

{WCET}

ISR

{WCET, cache damage}

Preempting task

Preemption Delay

Cache Related

Main task

{WCET, cache damage}

System call

{WCET, cache damage}

System call

{WCET, cache damage}
Our experiment uses FIFO caches for instruction and data

**Problem 1**: Existing CRPD analyses are not safe for FIFO caches

Not bounded by ECB (Evicting cache blocks) and/or UCB (Useful cache blocks)

**Problem 2**: Also not suitable for data caches

**Data reference access pattern**
Our approach

Optimizes “allocation” of preemptions to maximize CRPD

- Preempting task/ISR
- Preempted task
- Cache damage
- Persistence information
- CRPD analysis
A memory block $m$ is said to be *persistent* if it can never be evicted from cache.

Cache damage due to preemption can *disrupt* persistence of $m$.

Needed to bound the damage on preempted tasks.
char x, y, A[32], B[16];
void func() {
    for (int i = 0; i < 8; i++) {
        if (i<4) {
            x = A[i];
        }
        y = B[i];
    }
}

Loop context | Persistent blocks
-------------|------------------
Iteration 0 .. 3 | none
Iteration 4 .. 7 | m2

m0, m2 conflicts in a direct-mapped cache
Our approach

Optimizes “allocation” of preemptions that maximizes CRPD (by maximizing disruption to persistent blocks)
Our experiment uses FIFO caches for instruction and data

**Problem 1:** Existing CRPD analyses are not safe for FIFO caches

**Solution:** Bounds disruption to persistent blocks

**Problem 2:** Not suitable for data caches

**Solution:** Compute persistent blocks for each loop context
Experimental setup

- Implemented in Chronos [SCP ’07]
- \(\mu\text{C/OS-II}\) RTOS kernel
- ARM9 processor
  - Single core
  - In-order pipeline
  - Static branch predictor
  - FIFO cache (data and instruction)
Implementation

- Based on Bally2 robot controller
- Ported to uC/OS-II
- 3 main tasks: balance, navigation, remote
- 4 interrupt service routines

Results – WCET overestimation

<table>
<thead>
<tr>
<th></th>
<th>Estimated WCET</th>
<th>Observed WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>3.62</td>
<td>2.79</td>
</tr>
<tr>
<td>navigation</td>
<td>3.51</td>
<td>2.97</td>
</tr>
<tr>
<td>remote</td>
<td>2.55</td>
<td>2.15</td>
</tr>
<tr>
<td>tick_isr</td>
<td>2.49</td>
<td>2.49</td>
</tr>
<tr>
<td>gyro_isr</td>
<td>4.78</td>
<td>4.78</td>
</tr>
<tr>
<td>inclino_isr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>infrared_isr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RTSS 2013, Vancouver
Results – WCRT overestimation

Analyzing RTOS in isolation vs. Integrated analysis analysis:

- **Balance**: -38.7%
- **Navigation**: -26.8%
- **Remote**: -40.4%

RTSS 2013, Vancouver
Verification of real-time constraints

<table>
<thead>
<tr>
<th>Task</th>
<th>Real-time constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>Must consistently run at 50Hz</td>
</tr>
<tr>
<td>navigation</td>
<td>Must consistently run at 20Hz</td>
</tr>
<tr>
<td>remote</td>
<td>Must finish processing within 100ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Deadline (ms)</th>
<th>Observed WCRT (ms)</th>
<th>Estimated WCRT (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance</td>
<td>20</td>
<td>0.240</td>
<td>1.331</td>
</tr>
<tr>
<td>navigation</td>
<td>50</td>
<td>9.013</td>
<td>36.936</td>
</tr>
<tr>
<td>remote</td>
<td>100</td>
<td>0.245</td>
<td>1.478</td>
</tr>
</tbody>
</table>

All tasks meet their deadlines
Summary

- More *integrated* timing analysis of real-time application in the presence of supervisory software (RTOS)
- Capture timing effects of OS through *compositional* analysis
- Evaluated on *real hardware* with realistic robotic application scenario
- *Verification* of real-time constraints for each task in the evaluated robot controller

Our robot control infrastructure will be made available.