Timing Analysis of Concurrent Programs under Context Bounds

Jonathan Kotker, Dorsa Sadigh, Sanjit Seshia
University of California, Berkeley
Motivation: Cyber-Physical Systems

Cyber-Physical = Computation + Physical Processes
Quantitative analysis of programs is crucial:

*How long does it take?*

*How much energy does it consume?*

- **Safety-critical embedded systems**: Does the brake-by-wire software always actuate the brakes within 1 ms?

- **Energy-limited sensor nets**: How much energy must the sensor node harvest for RSA encryption?
Timing Analysis Problems

- Worst-case execution time (WCET) estimation
- Estimating distribution of execution times
- Threshold property: produce test cases that violates program deadline

All three problems can be solved if we could predict the execution time of arbitrary program paths.
Timing Analysis with Interrupts

Current code-level analysis techniques assume no interrupts, but practical embedded software is interrupt-driven.

NASA Toyota Unintended Acceleration Report
Lack of support in timing analysis tools for interrupt-driven code.
Why is timing analysis of interrupt-driven software a hard problem?

- Path Explosion: Unbounded number of interleavings of tasks and interrupt service routines (ISRs)
- Platform Modeling: Interrupts impact processor operation
Program with N tasks (main + ISRs)

Hardware Platform

Timing Analysis Tool

Execution time of arbitrary paths (WCET, distribution, threshold property)
Program with N tasks (main + ISRs)

Hardware Platform

Timing Analysis Tool

Execution time of arbitrary paths (WCET, distribution, threshold property)
Priority pre-emptive scheduling

- Tasks are ordered by priority
- If a higher-priority task interrupts a lower-priority task, the lower-priority task cannot later interrupt the higher-priority task
Assumptions in this work - 2

Lower-bound on interrupt inter-arrival time

There exists an $\alpha > 0$ such that $\alpha < \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \ldots$
Atomicity
Code should ideally be structured into atomic sections, perhaps by disabling and re-enabling interrupts*

* Our approach works with any atomicity model.
Contributions

- With these three assumptions, we compute a context bound and perform context-bounded analysis (Qadeer and Rehof, 2005).

- Number of interleaved paths can still be exponential in the context bound
  - Obtaining measurements can be tedious
  - Basis paths drastically reduce number of paths to be measured to be polynomial in size of sequential program

- Experiments on a real embedded platform show that WCET and execution times of arbitrary paths can be predicted accurately
Related Work

Context-bounded analysis

- Context-Bounded Model Checking of Concurrent Software
  Shaz Qadeer and Jakob Rehof (2005)
  - Introduces context-bounded analysis
  - Does not address timing analysis

- One Stack to Run Them All: Reducing Concurrent Analysis to Sequential Analysis under Priority Scheduling
  - Transforms a concurrent program with priority pre-emptive scheduling to a sequential program
  - Reduction applies for reachability only
Related Work

Timing Analysis

- Schedulability Analysis
  - Analyzes if a task can meet its deadline despite pre-emption
  - Treats tasks as primitive objects
  - Does not capture code correlation across tasks

  - Assembly-level
  - Threshold property, not WCET analysis
  - Assumes WCET is already given
Outline

- Approach
- Experimental Setup
- Hardware
- Results
- Summary and Future Work
ANALYSIS PHASE

- Compute context bound
- Generate final sequential program
- Run timing analysis tool (GAMETIME)

PROGRAM WITH n TASKS

TEST SUITE

MEASUREMENT AND PREDICTION PHASE

- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)
Approach

**ANALYSIS PHASE**

1. Compute context bound
2. Generate final sequential program
3. Run timing analysis tool (GAME TIME)

**MEASUREMENT AND PREDICTION PHASE**

**PROGRAM WITH n TASKS**

- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)

**TEST SUITE**

**PROGRAM WITH n TASKS**
Bound on total number of "context switches" between tasks

For a context bound of 1, the first task can be interrupted at most once, at either of the two interrupt points.
Finding a Context Bound

Lower bound on interrupt inter-arrival time: $\alpha$

Set $A = \alpha$, $CB = 1$

Compute sequential program

Compute $T_w$ (WCET)

$T_w < A$?

YES

Context bound = $CB$

NO

$CB++; A = CB \cdot \alpha$

Loop terminates if ISR services the interrupt in time less than $\alpha$
Approach

**ANALYSIS PHASE**
- Compute context bound
- Generate final sequential program
- Run timing analysis tool (GAME TIME)

**PROGRAM WITH \( n \) TASKS**

**TEST SUITE**

**MEASUREMENT AND PREDICTION PHASE**
- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)
ANALYSIS PHASE

1. Compute context bound
2. Generate final sequential program
3. Run timing analysis tool (GAME TIME)

PROGRAM WITH n TASKS

TEST SUITE

MEASUREMENT AND PREDICTION PHASE

1. Compile Program for Platform
2. Measure timing on Test Suite
3. Predict timing properties (worst-case, distribution)
Model occurrence of interrupt points as “function calls” and bound the number of these “function calls” (using a global counter).
Approach

**ANALYSIS PHASE**

1. Compute context bound
2. Generate final sequential program
3. Run timing analysis tool (GAME TIME)

**PROGRAM WITH n TASKS**

**TEST SUITE**

**MEASUREMENT AND PREDICTION PHASE**

1. Compile Program for Platform
2. Measure timing on Test Suite
3. Predict timing properties (worst-case, distribution)
Approach

ANALYSIS PHASE

- Compute context bound
- Generate final sequential program
- Run timing analysis tool (GAMETIME)

PROGRAM WITH n TASKS

TEST SUITE

MEASUREMENT AND PREDICTION PHASE

- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)
Basis Paths
Example: Modular Exponentiation

- Common operation in cryptography, used for public-key encryption and decryption.

- “What is $\text{base}^{\text{exponent}} \pmod{\text{prime}}$?”

- Exponentiation is performed using *square-and-multiply*, where the exponent is progressively divided by two, while the base is progressively squared.
modexp(base, exponent) {
    result = 1;

    if (((exponent & 1) == 1) { 
        result = (result * base) % p;
    }
    exponent >>= 1;
    base = (base * base) % p;

    if (((exponent & 1) == 1) { 
        result = (result * base) % p;
    }
    exponent >>= 1;
    base = (base * base) % p;

    return result;
}
Basis Paths

(a) CFG

(b) Basis paths
\[ x_1, x_2, x_3 \]

(c) Additional path \( x_4 \)

(d) Vector representations

\[ x_1 = (1, 1, 0, 0, 1, 1, 0, 0, 1) \]
\[ x_2 = (1, 0, 1, 1, 1, 1, 0, 0, 1) \]
\[ x_3 = (1, 1, 0, 0, 1, 0, 1, 1, 1) \]
\[ x_4 = (1, 0, 1, 1, 1, 0, 1, 1, 1) \]

\[ x_4 = x_2 + x_3 - x_1 \]
Theorem on Estimating Program Path Timing (Pictorial view)

\[ \mu_{\text{max}} \text{ bounds mean perturbation to basic block timing based on which path it lies on} \]

\[ \xi \text{ is } O(b \mu_{\text{max}}) \]
Approach

ANALYSIS PHASE

Compute context bound
Generate final sequential program
Run timing analysis tool (GAME TIME)

PROGRAM WITH n TASKS

TEST SUITE

MEASUREMENT AND PREDICTION PHASE

Compile Program for Platform
Measure timing on Test Suite
Predict timing properties (worst-case, distribution)
Approach

ANALYSIS PHASE

- Compute context bound
- Generate final sequential program
- Run timing analysis tool (GAME TIME)

PROGRAM WITH n TASKS

TEST SUITE

MEASUREMENT AND PREDICTION PHASE

- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)
Platform
Luminary Micro Interface to iRobot Create

- LM3S8962
- 32 Bit ARM Cortex M3
  - 5 stage pipeline
- UART interface to iRobot Create
- No cache
- No OS
Sensors
iRobot Create

- ADXL-322 accelerometer
- iRobot sensors
  - Buttons
  - Bumpers
  - Cliff sensors
- Use ISRs for accelerometer and sensor
**Approach**

**ANALYSIS PHASE**
- Compute context bound
- Generate final sequential program
- Run timing analysis tool (GAME TIME)

**PROGRAM WITH n TASKS**

**MEASUREMENT AND PREDICTION PHASE**
- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)

**TEST SUITE**


**Approach**

**ANALYSIS PHASE**

- Compute context bound
- Generate final sequential program
- Run timing analysis tool (**GAME TIME**)

**PROGRAM WITH n TASKS**

**TEST SUITE**

**MEASUREMENT AND PREDICTION PHASE**

- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)
Test Suite

- Test suite are test cases that drive the program along *basis paths* in sequential code.
- Each test case describes *initial values for variables* and the *points where an interrupt should happen*. 
**Challenge**

How to Force Interrupts

*Hardware Interrupt*

Can be modeled by setting a GPIO pin to high voltage, and wiring that high voltage to another GPIO pin.
Software Interrupt

- Can be modeled by embedding the ARM assembly instruction, SVC, in the code.
- Modify the interrupt vector table to include our interrupt handler.

MyHandler PROC
EXPORT MyHandler
SVC 11
ENDP

Vector Table in Startup.s

EXPORT __Vectors
__Vectors DCD __initial_sp ; Top of Stack
DCD reset ; Reset Handler
:
DCD MyHandler ; SVC Call Handler
Forcing Interrupts: Assumptions

We forced interrupts **through software**.

- Overhead for the SVC call will add to context switch overhead.
- Programs timed with SysTickTimer
  - Timer wraps around after 16,777,261 cycles

**Upper bound on program execution time**
Approach

ANALYSIS PHASE

Compute context bound
Generate final sequential program
Run timing analysis tool (GAMETIME)

PROGRAM WITH n TASKS

TEST SUITE

MEASUREMENT AND PREDICTION PHASE

Compile Program for Platform
Measure timing on Test Suite
Predict timing properties (worst-case, distribution)
ANALYSIS PHASE

- Compute context bound
- Generate final sequential program
- Run timing analysis tool (GAME TIME)

PROGRAM WITH n TASKS

TEST SUITE

MEASUREMENT AND PREDICTION PHASE

- Compile Program for Platform
- Measure timing on Test Suite
- Predict timing properties (worst-case, distribution)
Timing Prediction

- With measurements, assign weights to edges in control-flow graph of sequential code
- Use weights to predict runtimes for other arbitrary inputs and interleavings
## Characteristics of Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Lines of Code</th>
<th>Nodes in CFG</th>
<th>Edges in CFG</th>
<th>Total number of paths</th>
<th>Number of basis paths</th>
<th>Context Bound</th>
<th>Interrupt Inter-arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>modexp</td>
<td>60</td>
<td>60</td>
<td>70</td>
<td>500</td>
<td>12</td>
<td>1</td>
<td>1ms</td>
</tr>
<tr>
<td>iRobot-1</td>
<td>210</td>
<td>55</td>
<td>60</td>
<td>33</td>
<td>5</td>
<td>1</td>
<td>1ms</td>
</tr>
<tr>
<td>iRobot-2</td>
<td>230</td>
<td>141</td>
<td>160</td>
<td>3362</td>
<td>17</td>
<td>1</td>
<td>1ms</td>
</tr>
<tr>
<td>iRobot-3</td>
<td>230</td>
<td>97</td>
<td>108</td>
<td>1281</td>
<td>10</td>
<td>2</td>
<td>50μs</td>
</tr>
<tr>
<td>iRobot-4</td>
<td>280</td>
<td>213</td>
<td>244</td>
<td>33728</td>
<td>30</td>
<td>1</td>
<td>1ms</td>
</tr>
<tr>
<td>iRobot-5</td>
<td>250</td>
<td>179</td>
<td>206</td>
<td>65088</td>
<td>27</td>
<td>1</td>
<td>1ms</td>
</tr>
</tbody>
</table>
if(irobot_sensors_valid && irobot_sensors_buttons && ButtonAdvance) { ... }

// Bumps, cliffs, or wheel drops?
if(irobot_sensors_valid &&
   (irobot_sensors_bumps_drops & BumpEither) ||
   (irobot_sensors_bumps_drops & WheelDropAll) ||
   irobot_sensors_cliffL || ...) { ... }

// State machine
switch(state){
   case RESET: ....
   case BACKUP: ....
   case DRIVE: ....
}

// Interrupt points

Interrupt point 1

Interrupt point 2

Interrupt point 3
iRobot Code with Interrupt Points
(iRobot-1: Fewest states)
iRobot Code with Interrupt Points
(iRobot-2: One more state)
iRobot Code with Interrupt Points
(iRobot-3: Context bound of two)
iRobot Code with Interrupt Points (iRobot-4: More states)
iRobot Code with Interrupt Points
(iRobot-5: States that use the accelerometer)
Under a certain set of reasonable assumptions, GAME TIME can be used to predict times for interrupt-driven programs.

**Ongoing/Future work**
- Extend to other scheduling strategies.
- Expand evaluation to larger benchmarks with several ISRs.
- Analysis of energy consumption.
Thank you!