From Code to Weakly Hard Constraints: A Pragmatic End-to-End Toolchain for Timed C

Saranya Natarajan¹, Mitra Nasri², David Broman¹, Björn B. Brandenburg³, and Geoffrey Nelissen⁴

¹KTH Royal Institute of Technology, Sweden
²Delft University of Technology, Netherlands
³Max Planck Institute for Software Systems (MPI-SWS), Germany
⁴CISTER Research Centre, Polytechnic Institute Porto (ISEP-IPP), Portugal
Agenda

- Introduction
- Background and System Model
- Timing Analysis
- Schedulability Analysis
- Sensitivity Analysis
- Evaluation
- Limitation, Extension, and Conclusion
In real-time systems—such as autonomous aircraft, cars, and robots—the correctness of the system depends on both its logical and temporal correctness.
The implementation and validation of real-time system involve a number of distinct steps.

- Recurrent task
- Estimation of worst-case execution time (WCET)
- Schedulability analysis

In practice, each step is performed in isolation independently.

Theoretically, the soundness of each step relies on the correctness of previous steps.
Introduction

Uncertainties in release due to RTOS and underlying hardware.

Uncertainties in WCET due to assumptions in both static analysis and measurement-based approaches.

Uncertainties in whether the system is schedulable or not.

A binary analysis outcome (“schedulable” or “not schedulable”) is unsatisfying when the inputs have a high level of uncertainty.
Introduction

A end-to-end toolchain for Timed C

application to software

WCET estimation

schedulability analysis

sensitivity analysis
Overview of the end-to-end toolchain

**Timing analysis**
- Timed C Code
  - Instrumentation
  - Instrumented Code
- Job Set Generator
- Timing Traces
- Execute on Target Platform

**Schedulability Analysis**
- Job Set
- Schedulability Test

**Sensitivity Analysis**
Background

soft timing point (STP)

stp(expr1, expr2, n)

expr1 and expr2 are the lower and the upper bound relative to previous timing point

firm timing point (FTP)

ftp(expr1, expr2, n)

n is the resolution exponent of time value. The resolution is \(10^{-n}\) seconds

Timed C\(^1\)

Temporal constructs

concurrent construct

sdelay(expr, n)

fdelay(expr, n)

Timing points

temporal construct
Background

soft timing point

```c
1 task foo() {
2 stp(20, inf, ms);
3 while(1) {
4   work();
5   stp(60, 40, ms);
6 }
7 }
```

firm timing point

```c
1 task work() {
2   while(1) {
3     foo();
4     ftp(40, 20, ms);
5     bar();
6     stp(20, 10, ms);
7   }
8 }
```
In our analysis, we transform the instances of all the periodic GMF tasks to infinite set of jobs \( J \).

Each job refers to the execution of one frame.

A task \( \tau_i \) is defined by an offset \( O_i \) and finite sequence of \( N_i \) frames

\[ <F_{i,1}, F_{i,2}, \ldots, F_{i,N_i}> \]

A frame \( F_{i,j} \) is characterized by,

\( (C_{i,j}^{\text{max}}, C_{i,j}^{\text{min}}, A_{i,j}, D_{i,j}, Y_{i,j}) \)

A job \( J_{i,j} \) is characterized by,

\( (r_{i,j}^{\text{min}}, r_{i,j}^{\text{max}}, c_{i,j}^{\text{min}}, c_{i,j}^{\text{max}}, d_{i,j}, Y_{i,j}, \pi_{i,j}) \)

In Timed C, a frame refers to the code segment between two consecutive timing points.
The end-to-end toolchain uses a measurement-based timing analysis to estimate BCET, WCET, maximum jitter, and trigger precision.

**Timing Analysis**

<table>
<thead>
<tr>
<th>Stage #1</th>
<th>Stage #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrumentation</td>
<td>Generation of timing traces</td>
</tr>
<tr>
<td>Timed C code</td>
<td>Timing traces</td>
</tr>
<tr>
<td>KTC profiler</td>
<td>Execute Target platform</td>
</tr>
<tr>
<td>Instrumented Timed C code</td>
<td>C compiler</td>
</tr>
<tr>
<td></td>
<td>executable</td>
</tr>
</tbody>
</table>

**Step #1**
- Timed C code
- KTC profiler
- Instrumented Timed C code

**Step #2**
- Target-specific C code
- KTC compiler
- C compiler
- executable
Task `rts(int* a, int* b, int s)`

```c
// TP0
while(1){
    sense(&s);
    compute(b, a, s);
    critical{
        memcpy(a, b, 100);
    }
    fdelay(40, ms); // TP1
    actuate(a);
    sdelay(10, ms); // TP2
}
```

**Timed C construct** `critical` ensures that the execution of a code segment is not interrupted by `fdelay`.

**Trigger precision** is the extra time taken to escape out of the critical section.

The KTC profiler inserts 3 different instructions to measure **absolute arrival**, **start time**, and **finish time**.

`TP0-TP1`, `TP1-TP2`, and `TP2-TP1` are the code fragments.

**arrival** and **start time** instruction are inserted at the beginning of code fragment. **Finish time** instruction is inserted at the end of the code fragment.
The traces can be generated as a continuous log or a summary log.

The release jitter is platform specific parameter that cannot be computed from the timing trace.
Schedulability Analysis

The end-to-end toolchain integrates a customized and extended schedulability analysis tool (NPA) for non-preemptive job sets proposed by Nasri and Brandenburg[2].

NPA assesses the schedulability of a given finite set of jobs by exploring the space of all possible schedules of the jobs that the job level fixed priority scheduler can generate.

NPA can be applied to analyze recurrent workloads if it is possible to determine a finite observation window.

NPA relies on the notion of a schedule-abstraction graph to effectively search the extremely large space of possible schedules.
Schedulability Analysis

(a) Schedulability Analysis

(b) Table for Schedulability Analysis

(c) Graph for Schedulability Analysis

\[ O_{1,1} = 0 \]
\[ O_{1,2} = 8 \]
\[ O_{2,1} = 0 \]
\[ O^\text{max} = 8 \]
\[ H + O^\text{max} = 18 \]
\[ 0W = 27 \]
Schedulability Analysis

extension to NPA

- precedence constraints among frames of the same task.
- support for code abortion at the boundary of FTP.
- a safe finite observation window observation window for soft real-time GMF tasks
Sensitivity Analysis

Analyzes the sensitivity of the system w.r.t. each frame’s WCET under **weakly hard timing constraints**

**A weakly hard real-time task** $\tau_i$ **is feasible if any** $k_i$ **consecutive jobs of** $\tau_i$ **exhibits at most** $m_i$ **deadline misses**

Calculates the WCET margin for the strongest still-satisfied $(M,K)$ specification, where

$M = m_1 + \ldots + m_N$ and $K = k_1 + \ldots + k_N$

Given a window length $k_i$ for each task $\tau_i$, the toolchain calculates the largest scaling factor by which WCET of all tasks can be scaled such that sum of their individual deadline does not exceed $M$
### Sensitivity Analysis: Example

A GMF task set \( \{ \tau_1, \tau_2 \} \), where \( \tau_1 \) has two frames and \( \tau_2 \) has one frame.

| \( \tau_1 \) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | \( m_1 = 0 \) |
| \( \tau_2 \) | 0 | 0 | 0 | 0 | \( m_2 = 0 \) | \( M = 0 \) | \( \Delta = 1.24 \) |
| \( \tau_1 \) | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | \( m_1 = 1 \) |
| \( \tau_2 \) | 0 | 0 | 0 | 0 | \( m_2 = 0 \) | \( M = 1 \) | \( \Delta = 1.71 \) |
| \( \tau_1 \) | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | \( m_1 = 1 \) |
| \( \tau_2 \) | 1 | 0 | 0 | 0 | \( m_2 = 1 \) | \( M = 2 \) | \( \Delta = 1.98 \) |
| \( \tau_1 \) | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | \( m_1 = 2 \) |
| \( \tau_2 \) | 1 | 0 | 0 | 0 | \( m_2 = 1 \) | \( M = 3 \) | \( \Delta = 2.81 \) |

A1,2 = 10 ms and A2,1 = 30 ms

A2,1 = 50 ms

\( k_1 = k_2 = 2 \)

H = 200 ms

5 instances of \( \tau_1 \) (each with 2 frames) and 4 instances of \( \tau_2 \).
A user-specified utilization cap, a user-provided limit of interest $l_i$ for each $τ_i$, and by computing an upper limit based on the slack of original task set.
Sensitivity Analysis Overview

\[ \Delta_0^{\text{min}} = 0 \]

\[ \Delta_4^{\text{min}} = 3.3 \]

\[ \Delta_1^{\text{max}} = 1.65 \]

\[ \Delta_0^{\text{max}} = 8.3 \]

\[ \Delta_0^{\text{max}} - \Delta_1^{\text{min}} = 0.41 < \varepsilon \quad (\varepsilon = 0.5) \]

the algorithm terminates for \( M = 1 \)
Task sets were generated randomly following the period distribution of an automotive benchmark.

The experiment was carried out for 4 to 20 tasks in increments of 4. The number of frames in each task ranged from 1 to 4.

Timing traces were obtained by executing on a Raspberry Pi 2 Model B running a Debian based operating system (OS).

The end-to-end toolchain was run within a Docker environment on an Intel Xeon Gold 6148 CPU.
Evaluation

(a) Runtime

(b) Success ratio

(c) Calls to schedulability analysis
The task graph are based on the Paparazzi project.

Timing traces were obtained by executing on a ChipKIT Max32 board running FreeRTOS.

The end-to-end toolchain was run within a Docker environment on an MacBook Pro with two 3.1 GHz cores and 16 GB memory.

It is a synthetic case study inspired by an autonomous unmanned aerial vehicle (UAV).
Evaluation

This brief case study illustrates how the information provided by the toolchain can be used to optimize the temporal robustness of a system.

<table>
<thead>
<tr>
<th>Misses</th>
<th>WCET Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.55456852792</td>
</tr>
<tr>
<td>1</td>
<td>1.8654822335</td>
</tr>
<tr>
<td>2</td>
<td>2.02093908629</td>
</tr>
<tr>
<td>3</td>
<td>2.17639593909</td>
</tr>
<tr>
<td>5</td>
<td>2.25412436548</td>
</tr>
<tr>
<td>6</td>
<td>2.33185279188</td>
</tr>
<tr>
<td>7</td>
<td>2.40958121827</td>
</tr>
</tbody>
</table>
A legacy program written in C does not have to be translated into Timed C.

Though a measurement-based approach to timing analysis is used, this is not a conceptual limitation of the toolchain.

Though the schedulability analysis limits our toolchain to non-preemptive scheduling and regular job, for small microcontroller platforms, this is an ideal choice.

Sensitivity analysis is currently the primary scalability bottleneck.