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

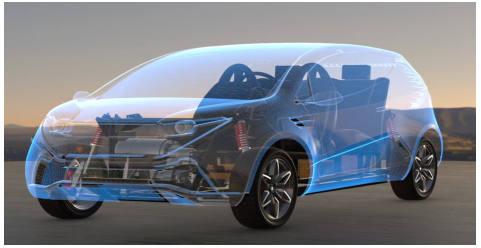
Saranya Natarajan[1], Mitra Nasri[2], David Broman[1], Björn B. Brandenburg[3], and Geoffrey Nelissen[4]

[1]KTH Royal Institute of Technology, Sweden
[2]Delft University of Technology, Netherlands
[3]Max Planck Institute for Software Systems (MPI-SWS), Germany
[4]CISTER 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

Real-Time System



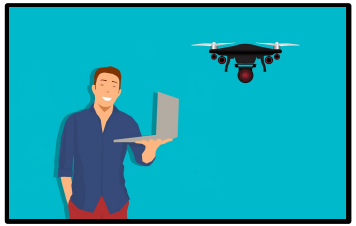
Autonomous Cars

Automation Lines in Industries

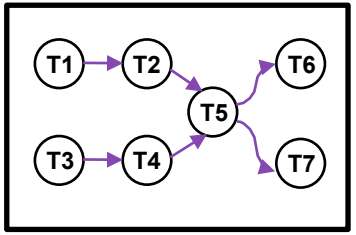
Auto Pilot System

In real-time systems—such as autonomous aircraft, cars, and robots—the correctness of the system depends on both its logical and temporal correctness

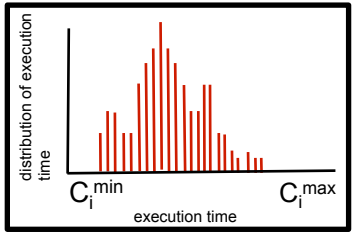
Introduction



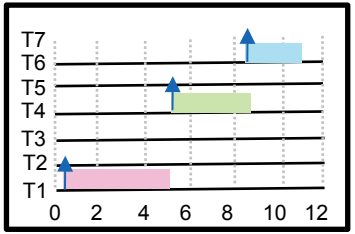
The implementation and validation of real-time system involve a number of distinct steps.



recurrent task



estimation of worst-case execution time (WCET)

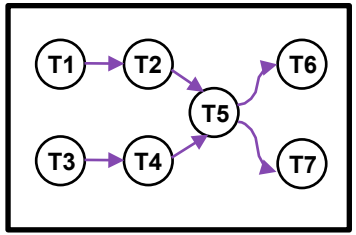


schedulability analysis

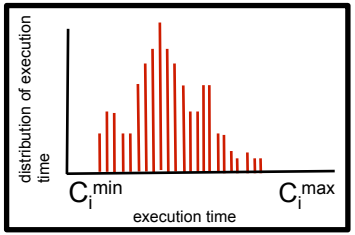
In practice, each step is performed in isolation independently.

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

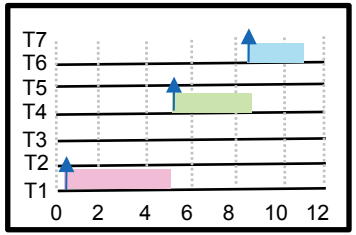
Introduction



application to software



WCET estimation



schedulability analysis

Uncertainties in release due to RTOS and underlying hardware.

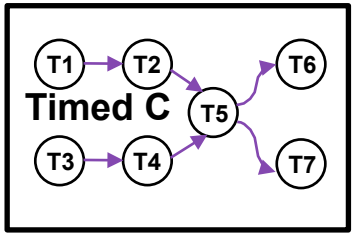
Uncertainties in WCET due to assumptions in both static analysis and measurement-based approaches.

Uncertainties in whether the system is schedulable or not

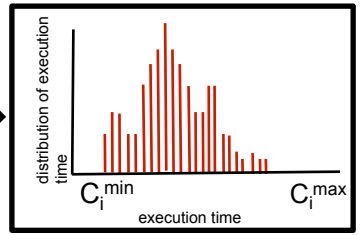
A binary analysis outcome ("schedulable" or "not schedulable") is unsatisfying when the inputs have a high level of uncertainty.

Introduction

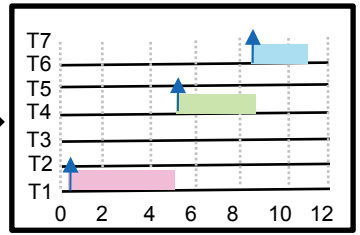
A end-to-end toolchain for Timed C



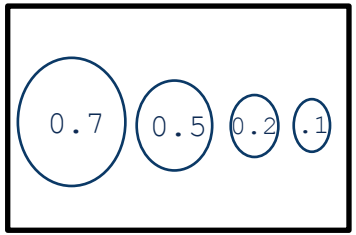
application to software



WCET estimation

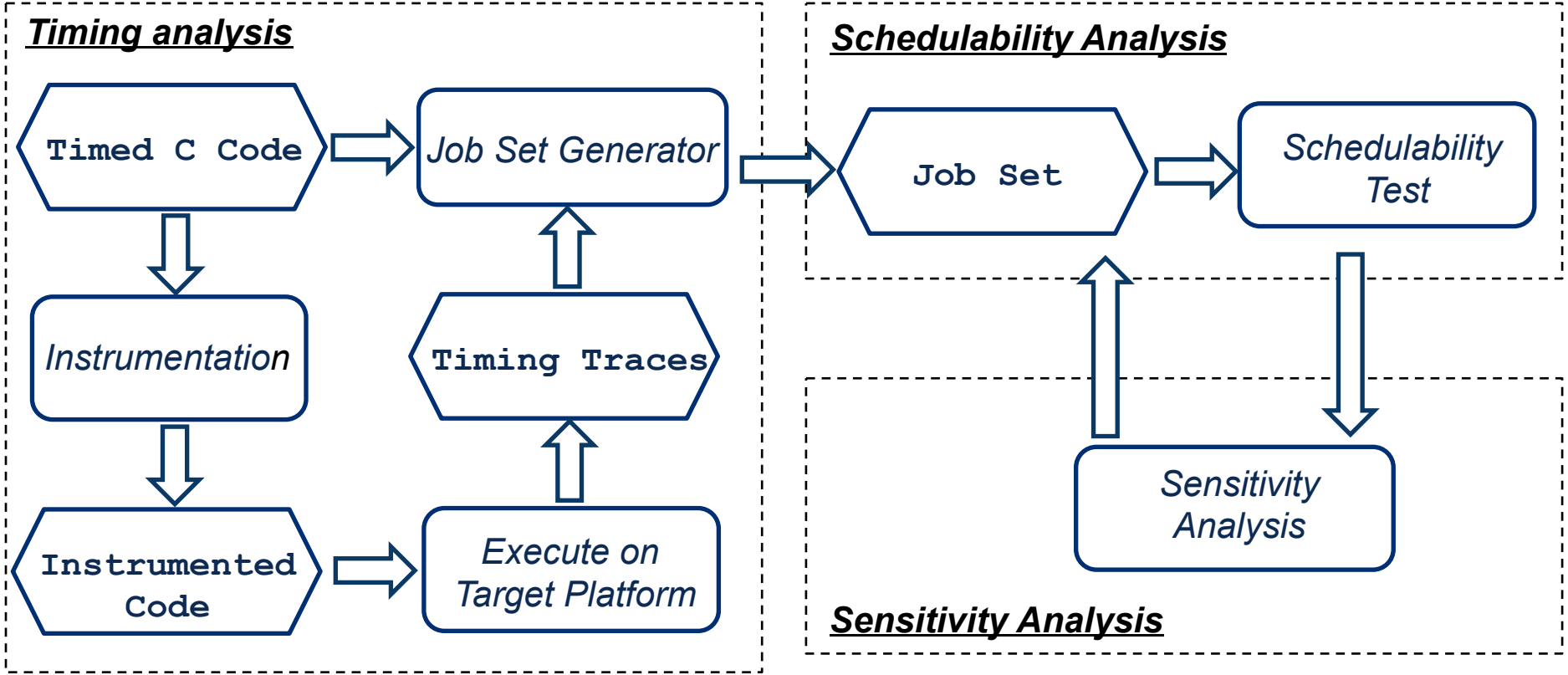


schedulability analysis

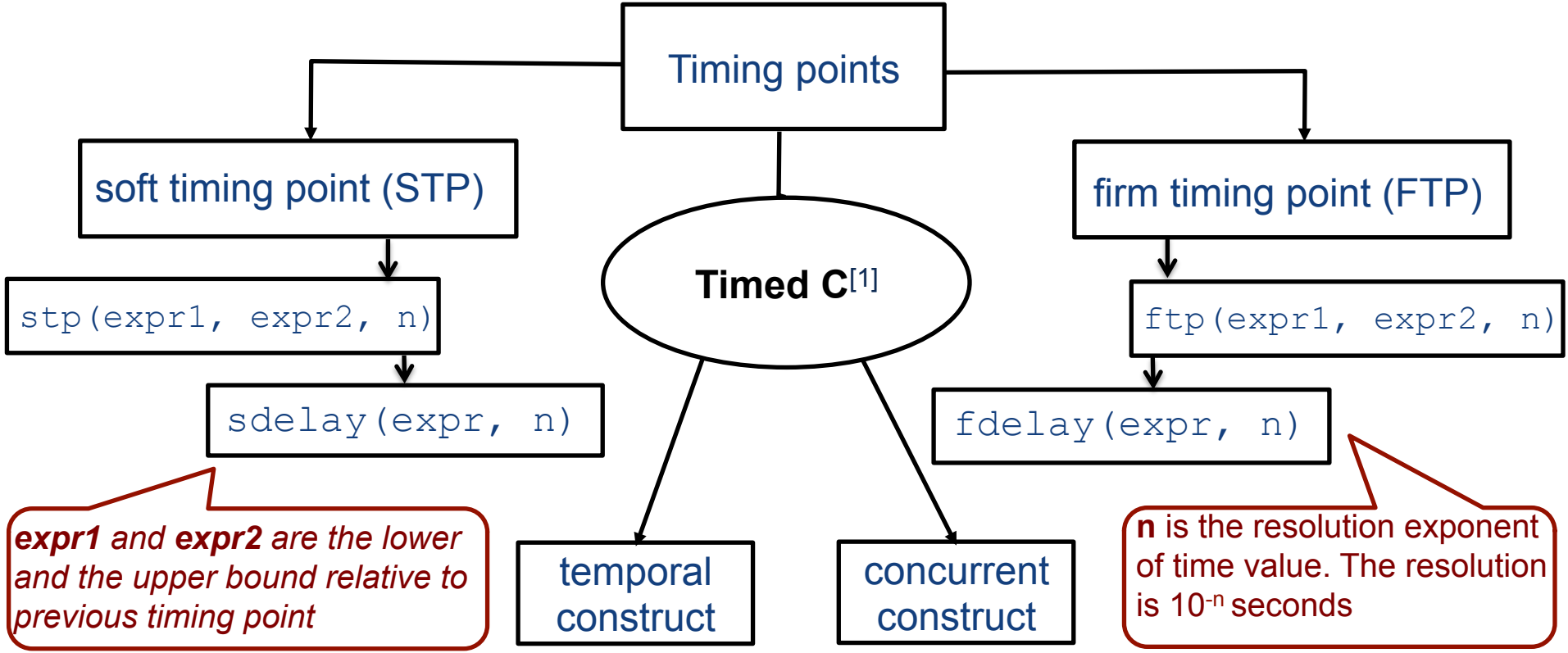


sensitivity analysis

Overview of the end-to-end toolchain



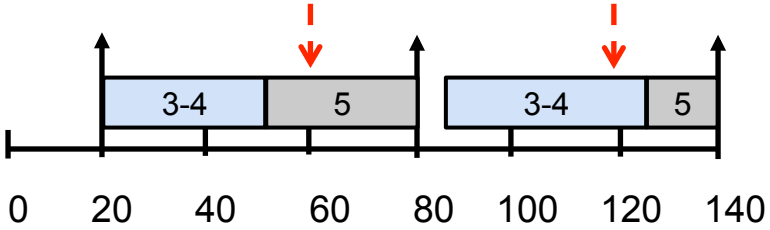
Background


Timing points
soft timing point (STP)
firm timing point (FTP)
Timed C[1]
stp(expr1, expr2, n)
ftp(expr1, expr2, n)
sdelay(expr, n)
fdelay(expr, n)
expr1 and expr2 are the lower and the upper bound relative to previous timing point
temporal construct
concurrent construct
n is the resolution exponent of time value. The resolution is 10-n seconds

Background

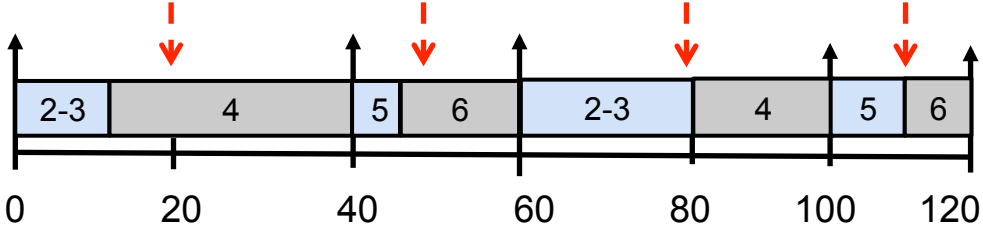
soft timing point

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



firm timing point

```
1 task work(){
2  while(1){
3     foo();
4     ftp(40,20,ms);
5     bar();
6     stp(20,10,ms);
7  }
8 }
```



System Model

In our analysis, we transform the instances of all the periodic GMF tasks to infinite set of jobs $\mathcal{J}$.

A task τ_i is defined by an offset O_i and finite sequence of N_i frames

$$<F_{i,1}, F_{i,2}, \ldots\ldots, F_{i,Ni}>$$

Each job refers to the execution of one frame

a frame $F_{i,j,}$ is characterized by,

$$(C_{i,j}^{max}, C_{i,j}^{min}, A_{i,j}, D_{i,j,} j_{i,j}, \Upsilon_{i,j})$$

a job $J_{i,}$ is characterized by,

$$(r_i^{min}, r_{i,}^{max}, c_i^{min}, c_{i,}^{max}, d_i, \Upsilon_i, \pi_i)$$

the functions $\delta(J_i)$, $\sigma(J_i)$, and $\lambda(J_i)$ return the task number, instance number, and frame number of J_i.

end-to-end toolchain

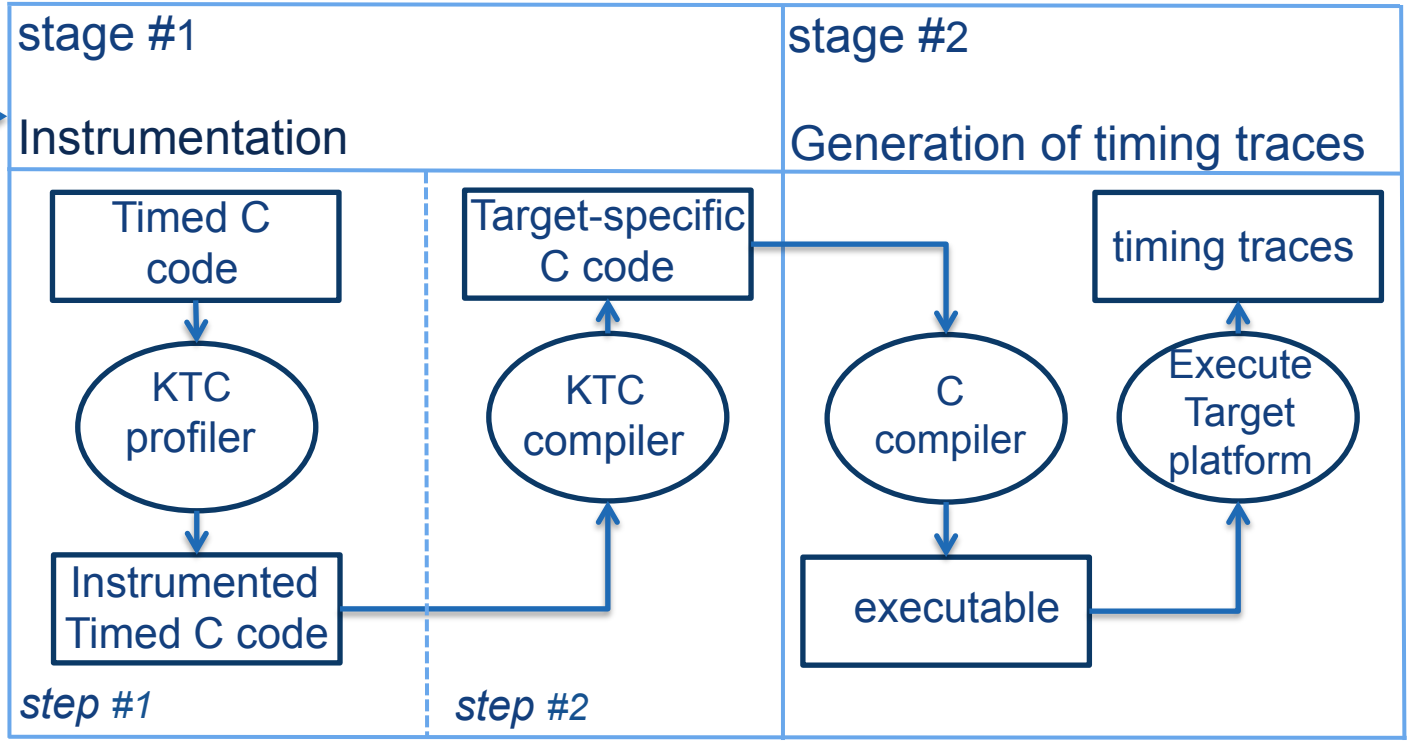
- uniprocessor platform
- generalized multiframe tasks (GMF)
- job-level fixed priority scheduling algorithm
- non-preemptively

In Timed C, a frame refers to the code segment between two consecutive timing points.

Timing Analysis

Timing Analysis

The end-to-end toolchain uses a measurement-based timing analysis to estimate BCET, WCET, maximum jitter, and trigger precision.



Instrumentation

TP0, TP1, and TP2 are timing points

```
1  task rts(int* a,int* b,int s){ // TP0
2    while(1){
3      sense(&s);
4      compute(b, a, s);
5      critical{
6        memcpy(a, b, 100);
7      }
8      fdelay(40, ms);              // TP1
9      actuate(a);
10     sdelay(10, ms);             //  TP2
11   }
12 }
```

Timed C construct **`critical`** ensure 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***

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

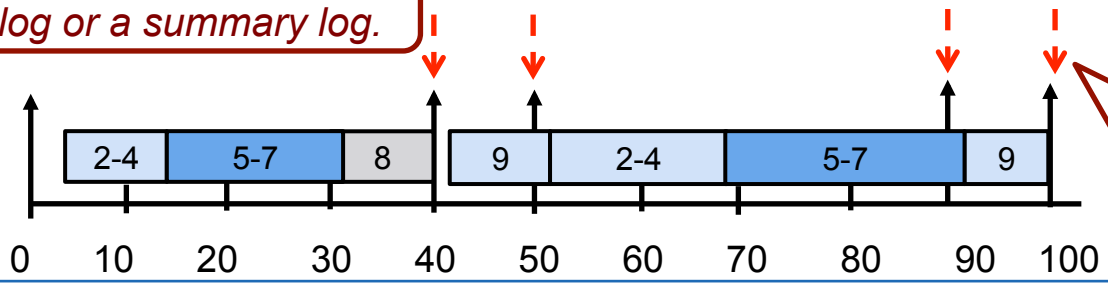
TP0-TP1, TP1-TP2, and TP2-TP1 are the code fragments.

Generation of Timing Traces

src	arrival	start	finish	precision	dst
0	0	4	30	0	1
1	40	42	52	0	2
2	50	52	91	23	1
1	90	91	100	0	2

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

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

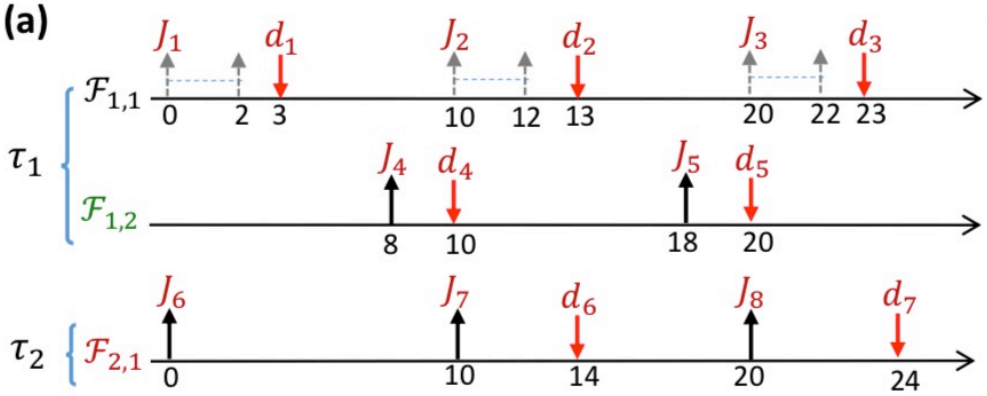
Schedulability Analysis

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

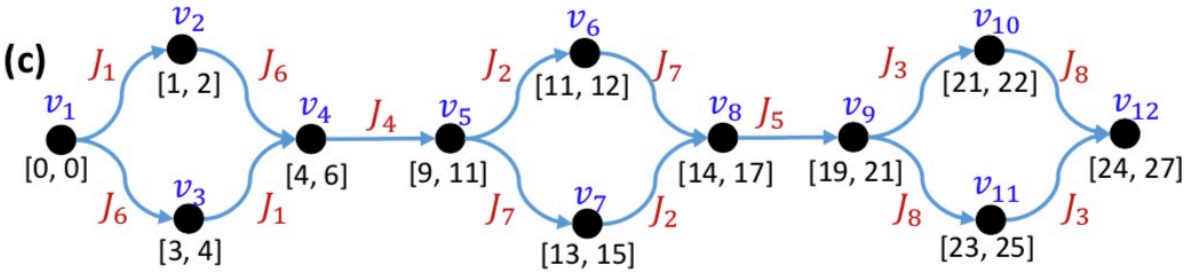
(a)



(b)

J_i	$r_i^{\{min\}}$	$r_i^{\{max\}}$	$c_i^{\{min\}}$	$c_i^{\{max\}}$	d_i	γ_i	π_i	f_i^{min}	f_i^{max}
J_1	0	2	1	2	3	soft	1	1	6
J_2	10	12	1	2	13	soft	1	11	17
J_3	20	22	1	2	23	soft	1	21	27
J_4	8	8	1	3	10	soft	1	9	11
J_5	18	18	1	3	20	soft	1	19	21
J_6	0	0	3	4	14	soft	2	3	6
J_7	10	10	3	4	24	soft	2	13	17
J_8	20	20	3	4	34	soft	2	23	27

(c)



$O_{1,1} = 0$ $\quad O^{max} = 8$

$O_{1,2} = 8$ $\quad H + O^{max} = 18$

$O_{2,1} = 0$ $\quad OW = 27$

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

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 τ_i is feasible **if any** k_i consecutive jobs of τ_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 τ_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

τ_1	0	0	0	0	0	0	0	0	0	0	$m_1 = 0$
τ_2	0	0	0	0	$m_2 = 0$						$M = 0$ Δ=1.24
τ_1	0	0	1	0	0	0	0	0	0	0	$m_1 = 1$
τ_2	0	0	0	0	$m_2 = 0$						$M = 1$ Δ=1.71
τ_1	0	0	1	0	0	0	0	1	0	0	$m_1 = 1$
τ_2	1	0	0	0	$m_2 = 1$						$M = 2$ Δ=1.98
τ_1	0	0	1	0	0	0	1	1	0	0	$m_1 = 2$
τ_2	1	0	0	0	$m_2 = 1$						$M = 3$ Δ=2.81

A GMF task set $\{\tau_1, \tau_2\}$, where τ_1 has two frames and τ_2 has one frame.

$A_{1,2}$=10 ms and $A_{2,1}$=30 ms

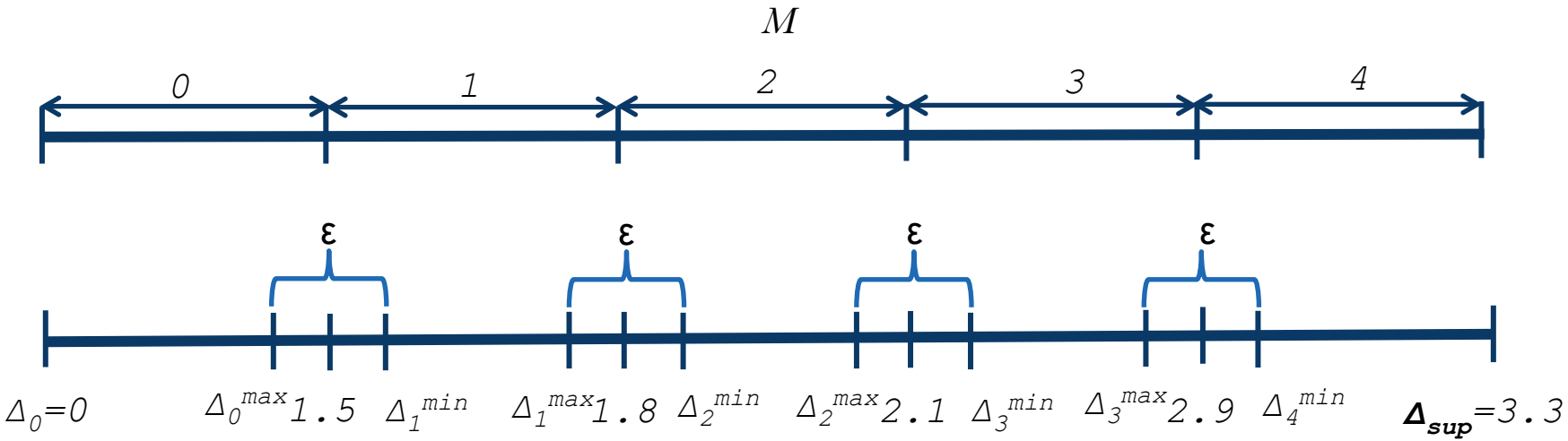
$A_{2,1}$=50 ms

H=200 ms

$k_1=k_2=2$

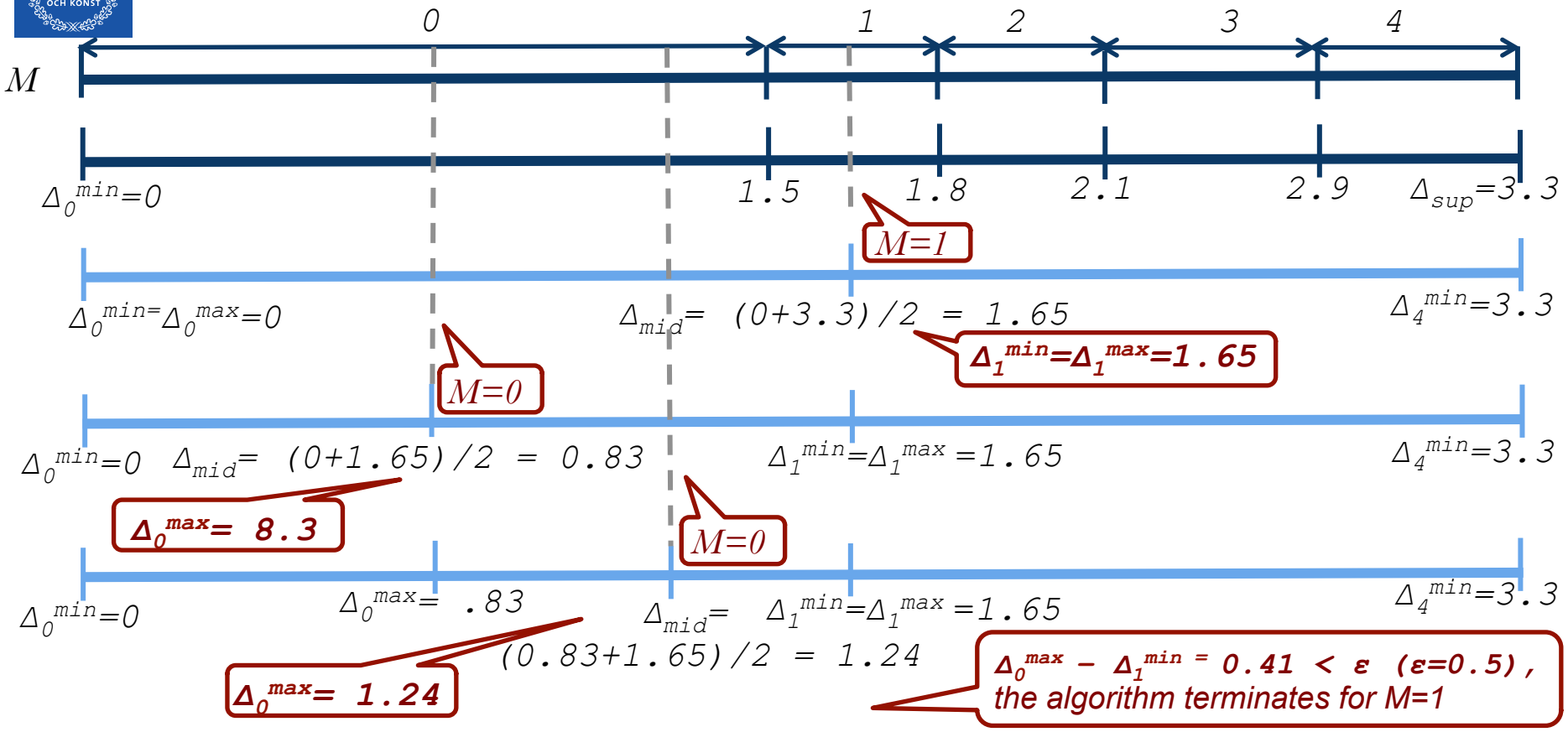
5 instances of τ_1 (each with 2 frames) and 4 instances of τ_2.

Sensitivity Analysis Overview



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



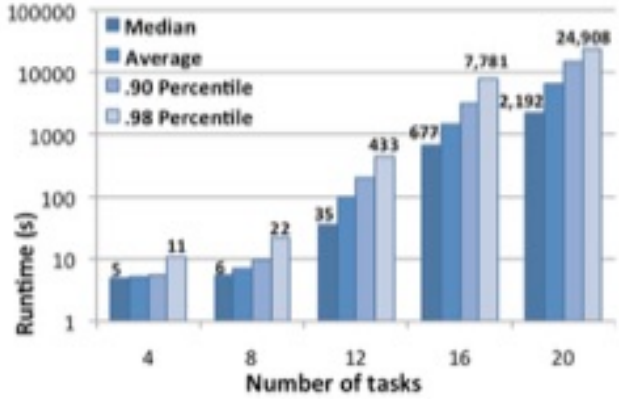
Evaluation

scalability

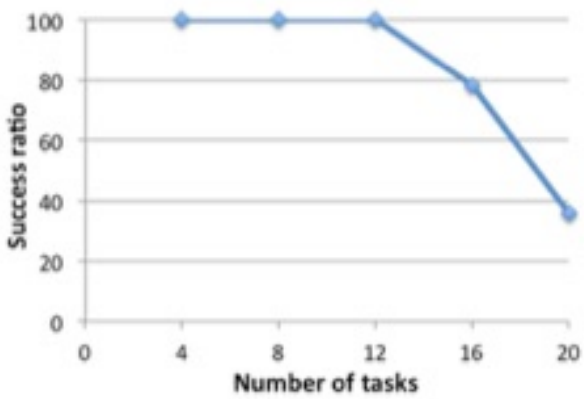
experimental setup

- Task sets were generated randomly following the period distribution of an automotive benchmark
- The experiment was carried out for 4 to 20 tasks 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*.

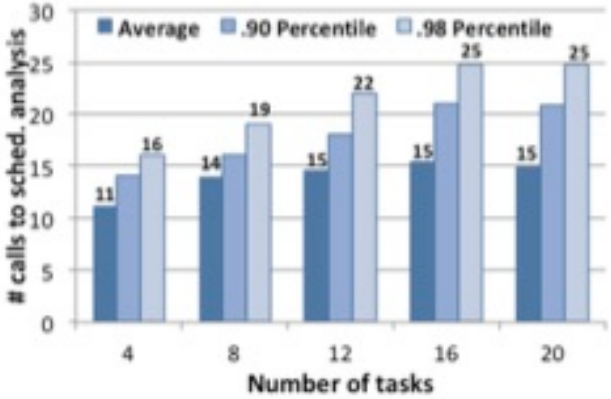
Evaluation



(a) Runtime



(b) Success ratio



(c) Calls to schedulability analysis

Evaluation

case study

experimental setup

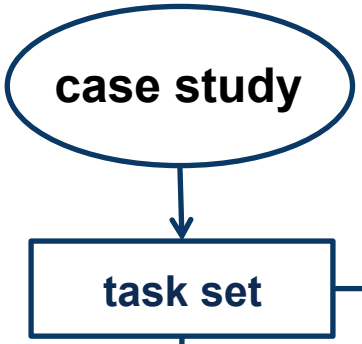
It is a synthetic case study inspired by an autonomous unmanned aerial vehicle (UAV).

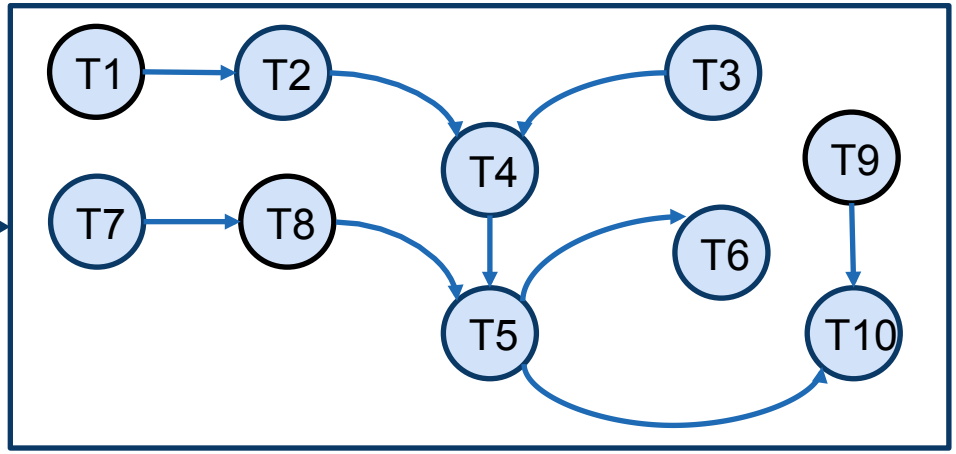
The task graph are based on the Paparazzi project.

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.*

Evaluation



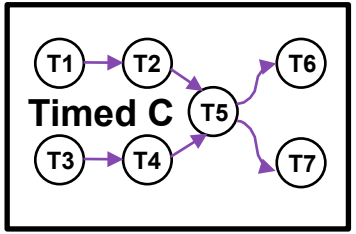


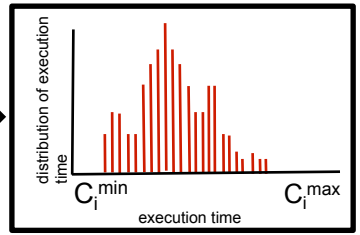
Misses	WCET Margin
0	1.55456852792
1	1.8654822335
2	2.02093908629
3	2.17639593909
5	2.25412436548
6	2.33185279188
7	2.40958121827

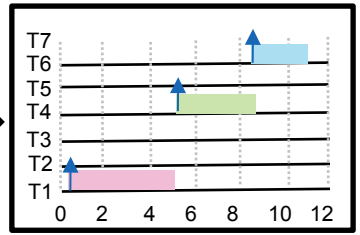
this brief case study illustrates how the information provided by the toolchain can be used to optimize the temporal robustness of a system.

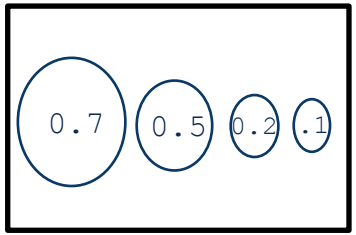
Limitation, Extension, and Conclusion

Any of these components can be replaced or improved individually.









A legacy program written in C does not have to be translated into Timed C.

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

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

sensitivity analysis is currently the primary scalability bottleneck.

Reference

[1] S. Natarajan and D. Broman, “Timed C: An Extension to the C Programming Language for Real-Time Systems,” in *RTAS*, 2018.

[2] M. Nasri and B. B. Brandenburg, “An exact and sustainable analysis of non-preemptive scheduling,” in *RTSS*, 2017.