SIMULTime: Context-Sensitive Timing Simulation on Intermediate Code Representation for Rapid Platform Explorations

A. Cornaglia[†], A. Viehl[†], O. Bringmann[‡] and W. Rosenstiel[‡]

FZI Research Center for Information Technology[†]
University of Tübingen[‡]

ASP-DAC 2019
January 24, 2019

FZI FORSCHUNGSZENTRUM INFORMATIK

Outline

1 Introduction

2 Background

3 Methodology

4 Results

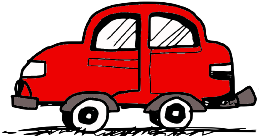
Outline

1. Introduction
2. Background
3. Methodology
4. Results

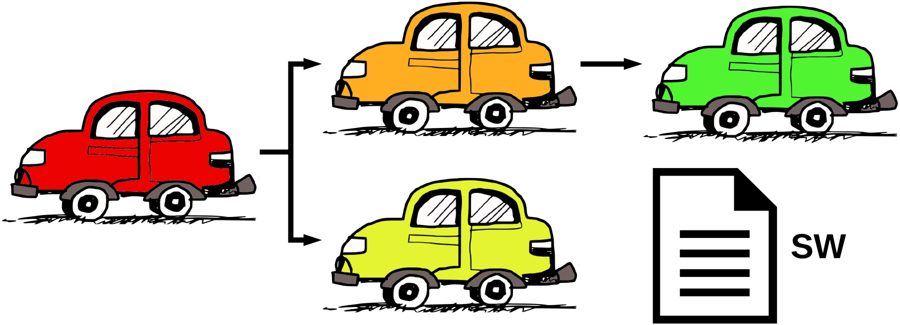
Timing Estimations for Embedded Software

- Necessity of Embedded Systems (ESs) properties assessment
  - Non-functional properties represent a key aspect
  - Different level of assessment during the development
- Timing predictions for the embedded application execution-time
  - Fast but accurate estimations (no Worst-Case Execution Time)
- Challenging and hard task due to several factors
  1. Predictions for multiple target platforms
  2. Hardware complexity and intellectual property restrictions
  3. Compiler optimizations
  4. Multiple input data
  5. Multiple soft-configuration of a single program instance

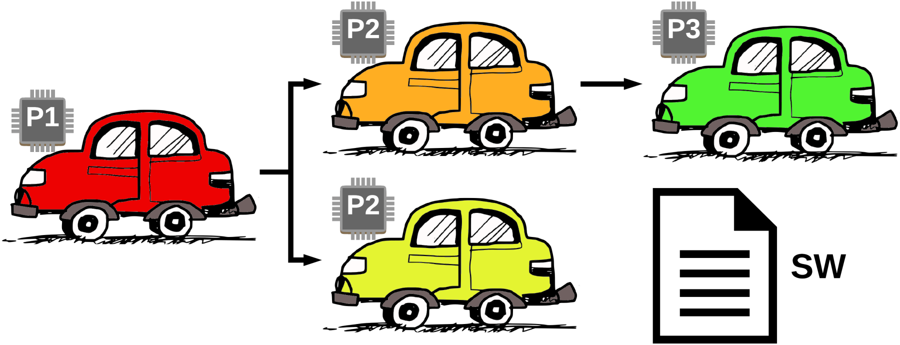
Variability in Embedded Systems



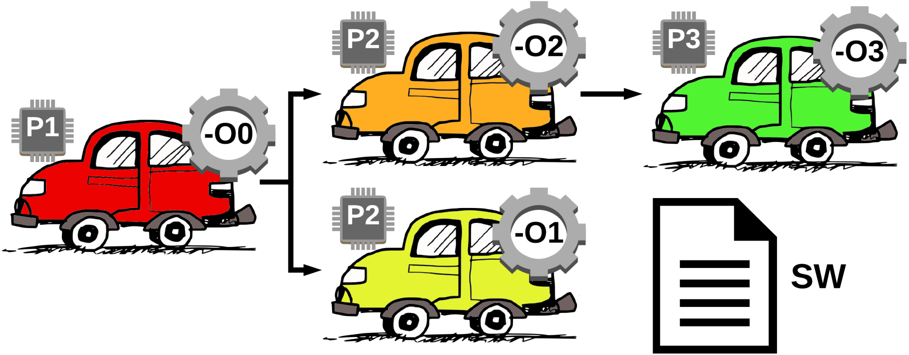
Variability in Embedded Systems



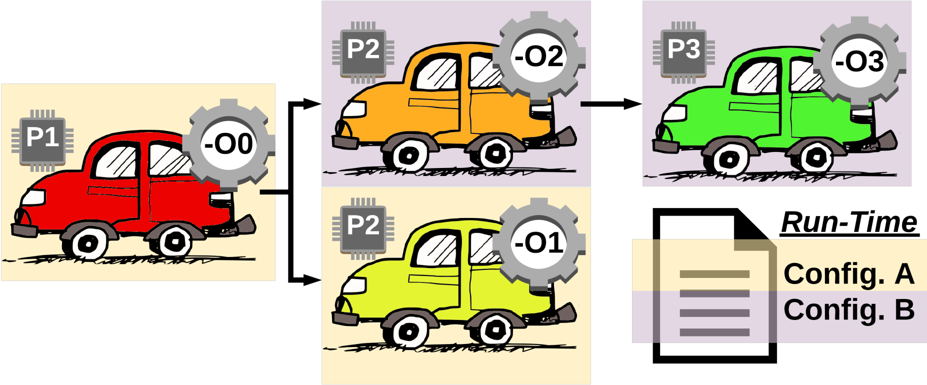
Variability in Embedded Systems



Variability in Embedded Systems



Variability in Embedded Systems



The Challenge

1. Timing estimations for multiple variants in a single run
   - Run-time variability: conditional statements driven by a configuration
   - Variability on target platforms
   - Different hardware dependent compiler optimizations
2. Fast and accurate timing estimations
   - Support for the development phase of the system
   - In depth modelling implies undesired slow-down
   - Speed easily achievable by sacrificing the accuracy

Outline

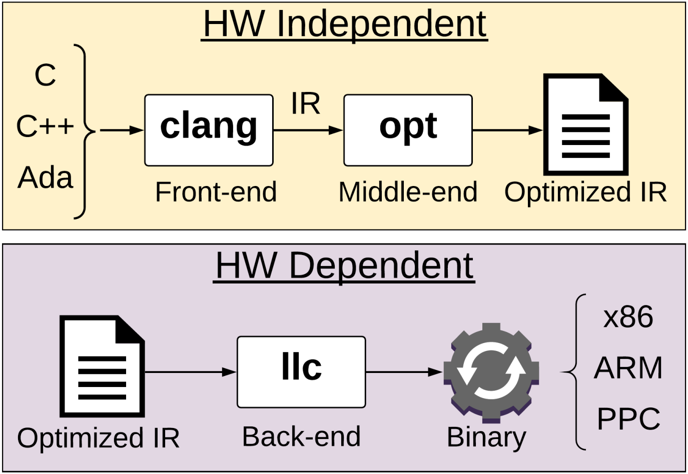
1 Introduction

2 Background

3 Methodology

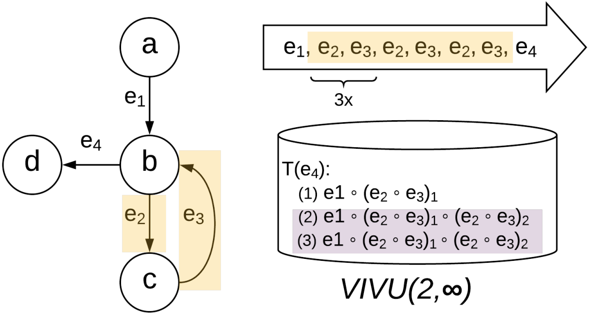
4 Results

The LLVM Compiler Infrastructure



Context-Sensitive Timing Database

- Context concept: string ruled by $\boxed{\textbf{\textit{VIVU(n,k)}}}$ mapping
  - Set of control flow paths in interprocedural control flow graph (CFG)
  - ***n***: maximum loop recursion count, ***k***: number of elements upper limit
- Timing database (TDB) for implicit target platforms modeling[1]
  - Relative execution times for the different program contexts
  - Accuracy for a single target system without needing a model of it
  - Generation from both static analysis and measurements



$T(e_4)$:

(1) $e1 \circ (e_2 \circ e_3)_1$

(2) $e1 \circ (e_2 \circ e_3)_1 \circ (e_2 \circ e_3)_2$

(3) $e1 \circ (e_2 \circ e_3)_1 \circ (e_2 \circ e_3)_2$

VIVU(2,∞)

[1] **S. Ottlik**, **et al. "Trace-based context-sensitive timing simulation considering execution path variations."**, **2016** .

LLVM-IR Execution Engine and Matching

- *lli* directly executes LLVM-IR programs
  - Similar to a virtual machine it is not an emulator
  - Executes only instructions for the host architecture
- Two different execution engines are provided
  - Complex just-in-time compiler (JIT)
  - Slower but easier interpreter
- Possibility for LLVM-IR context-sensitive simulations
  - lli determines the common execution path in the IR CFG
  - Function for IR to multiple binary CFGs mapping[2]
  - Association between HW independent code and multiple TDBs

[2] **C. Suhas**, **et al.** **“Automated**, **retargetable back-annotation for host compiled performance and power modeling.”**, **2013** .

Outline

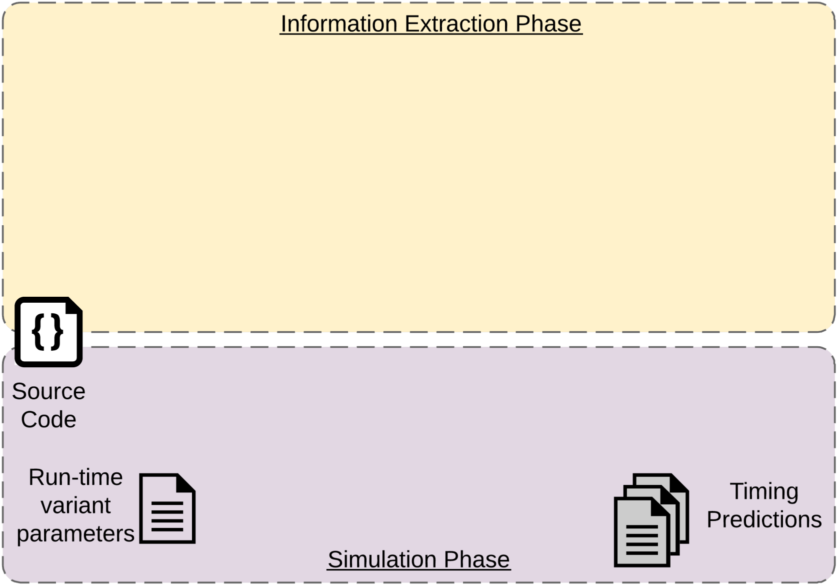
1 Introduction

2 Background

3 Methodology

4 Results

SIMULTime Workflow



Information Extraction Phase

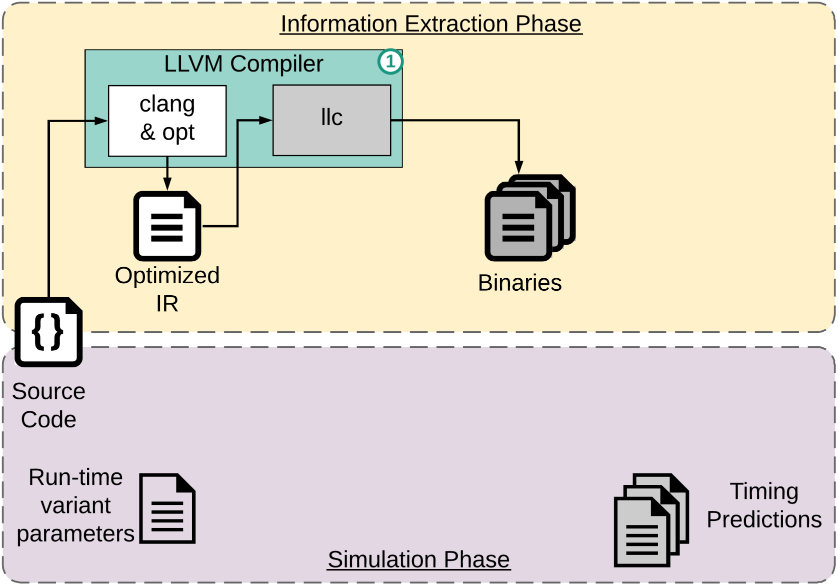
Source Code

Run-time variant parameters

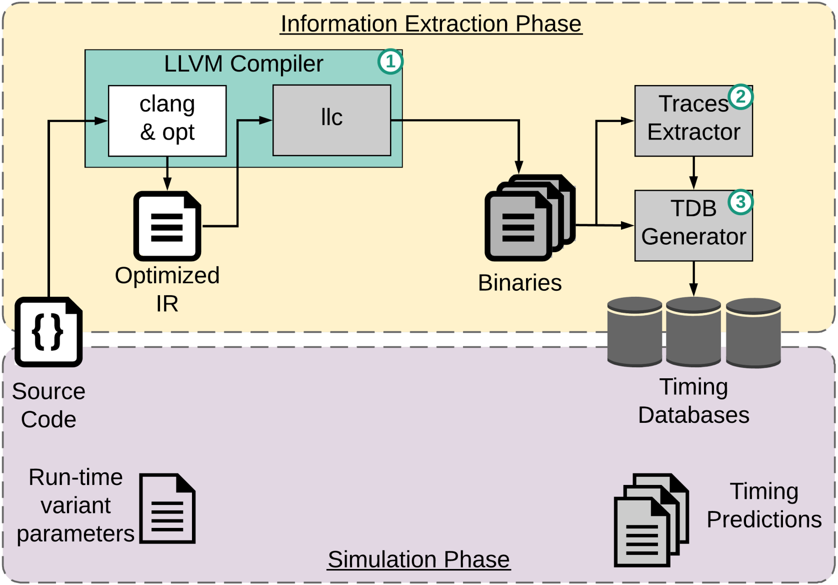
Simulation Phase

Timing Predictions

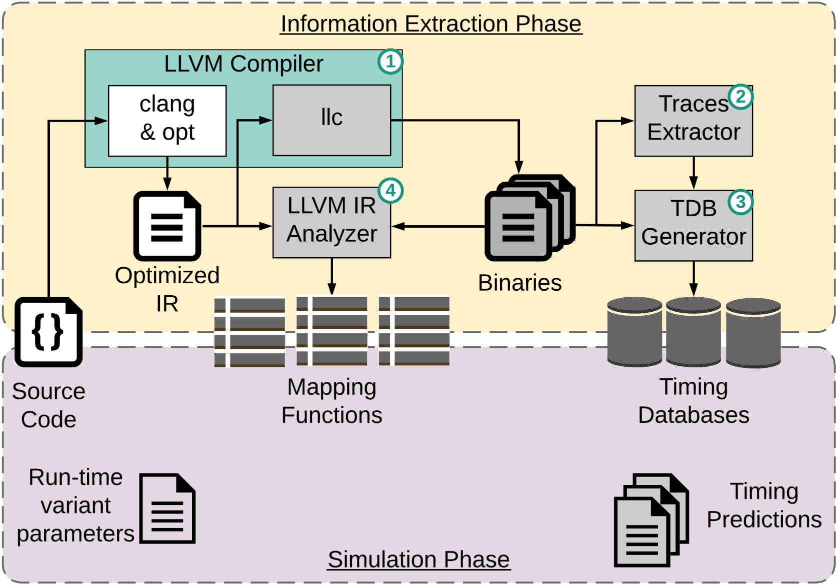
SIMULTime Workflow



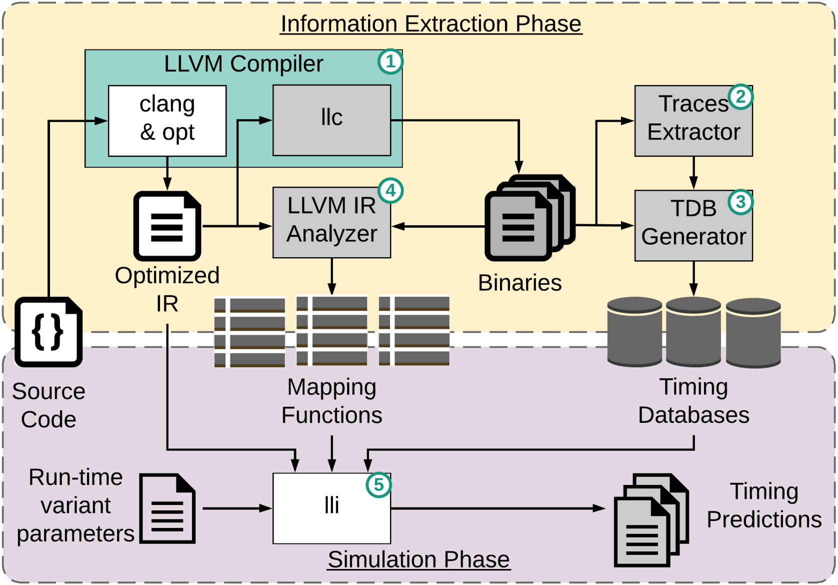
SIMULTime Workflow



SIMULTime Workflow



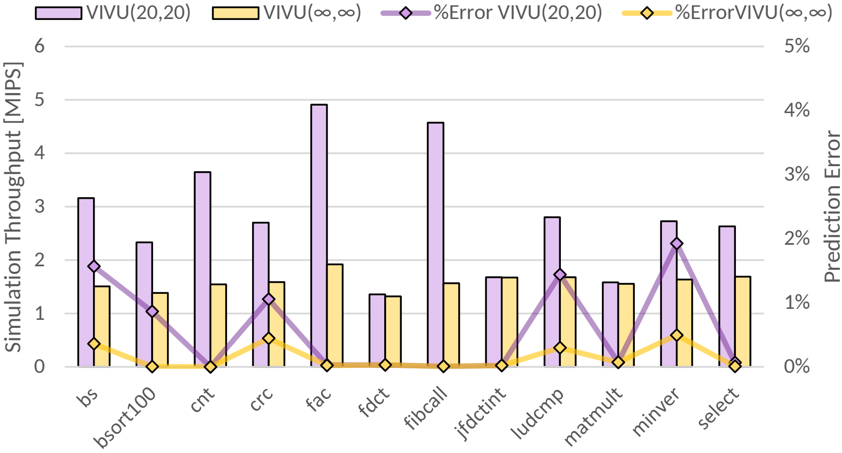
SIMULTime Workflow



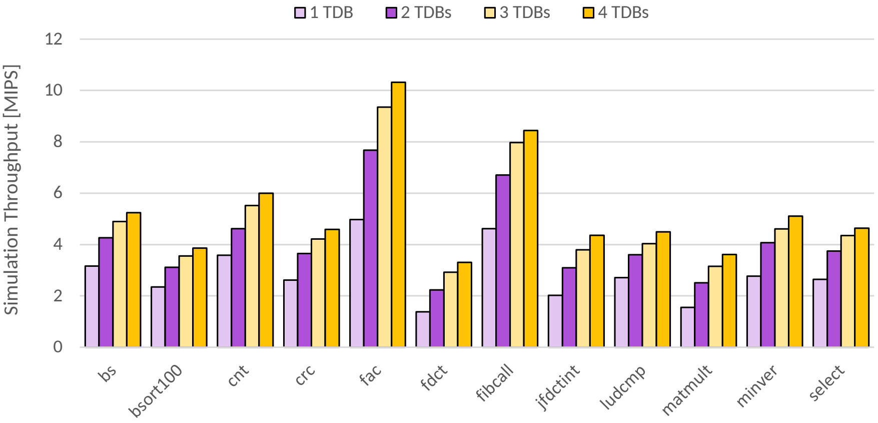
Outline

1. Introduction
2. Background
3. Methodology
4. Results

SIMULTime Throughput and Accuracy



Multiple Simultaneous Predictions Speedup



Conclusions and Future Work

- Multiple context-sensitive simulations based on the LLVM IR code
  - Exploration of different HW platforms and compiler optimizations
  - High accurate predictions even for complex architectures
  - One single run to provide significant speedup
- Prospective challenges
  1. JIT to increase the speedup by keeping the level of accuracy
  2. Support compile-time variability
  3. Reduce the TDB creation overhead
  4. Increase level of abstraction supporting model-based development

Thanks for your attention.

Questions?

Alessandro Cornaglia
cornaglia@fzi.de