22nd International Workshop on Worst-Case Execution Time Analysis

WCET 2024, July 9, 2024, Lille, France

Edited by

Thomas Carle

OASICS

Editors

Thomas Carle
IRIT - Univ. Toulouse 3 - CNRS, France
thomas.carle@irit.fr

ACM Classification 2012
Computer systems organization → Real-time systems; Theory of computation → Program analysis; Software and its engineering → Software verification and validation; Software and its engineering → Software safety; Software and its engineering → Software performance

ISBN 978-3-95977-346-1

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-346-1.

Publication date
August, 2024

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/OASIcs.WCET.2024.0

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

22nd International Workshop on Worst-Case Execution Time Analysis (WCET 2024).
Editor: Thomas Carle
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

Welcome to the proceedings of the 22nd International Workshop on Worst-Case Execution Time Analysis (WCET 2024). This year's edition of the WCET workshop is held on July 9th and is co-located with the Euromicro Conference on Real-Time systems (ECRTS 2024) in Lille, France. The WCET workshop is the main venue for research on the topic of worst-case execution time in the broad sense.

This year's edition starts with a keynote by Prof. Isabelle Puaut from IRISA–Univ. of Rennes, that presents the main issues she, her students and her colleagues encountered in building machine learning solutions for WCET analysis. The rest of the program consists in 2 regular papers that were peer-reviewed by 3 members of the program committee each, and 4 invited papers of excellent quality.

Many people were involved in the process of putting together a high quality program, organising the workshop and publishing the proceedings. First, I would like to thank all the authors that submitted their work to the WCET 2024 workshop. I also want to thank all the members of the program committee for their time and energy in reviewing the papers and making constructive feedback to the authors. I want to thank as well the steering committee for trusting me to organize the workshop this year, and for their support all along the way. The team at Schloss Dagstuhl has been of tremendous help in putting up the proceedings, so I want to thank them and in particular Michael Wagner and Michael Didas for their careful work and for their support at each step of the publishing process.

Last but not least, I would like to warmfully thank Peter Wägemann, who organized the workshop last year and has taken time to help and guide me for the organization of this year's edition.

I hope that you will find this year's program as interesting as I do, and that we will manage to create once again the friendly yet studious atmosphere that characterizes the WCET workshop.

Toulouse, France
June 21, 2024
Thomas Carle

Committees

Program Chair

- Thomas Carle, IRIT–Université Toulouse 3, France

Program Committee

- Konstantinos Bletsas, Polytechnic Institute of Porto (ISEP/IPP)
- Hugues Cassé, IRIT - Université de Toulouse
- Liliana Cucu-Grosjean, INRIA, Statinf
- Björn Frosberg, RISE, Sweden
- Benjamin Lesage, ONERA, France
- Björn Lisper, Mälardalen University
- Isabelle Puaut, University of Rennes/IRISA
- Peter Puschner, Vienna University of Technology
- Martin Schoeberl, Technical University of Denmark
- Peter Wägemann, Friedrich Alexander Universität Erlangen-Nürnberg

External Reviewers

- Emad Jacob Maroun

Steering Committee

- Björn Lisper, Mälardalen University
- Isabelle Puaut, University of Rennes/IRISA
- Jan Reineke, Saarland University

WORTEX: Worst-Case Execution Time and Energy Estimation in Low-Power Microprocessors Using Explainable ML

Hugo Reymond
Univ. Rennes, INRIA, CNRS, IRISA, Rennes, France

Abderaouf Nassim Amalou
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Isabelle Puaut
Univ. Rennes, INRIA, CNRS, IRISA, Rennes, France

Abstract

Real-time and energy-constrained systems heavily rely on estimates of the worst-case execution time (WCET) and worst-case energy consumption (WCEC) of code snippets to ensure trustworthy operation. Designing architecture-specific analytical models for time and energy is often challenging and time-consuming. In situations where analytical models are unavailable or incomplete, machine learning (ML) techniques emerge as a promising solution to build WCEC/WCET models. This paper introduces WORTEX, a toolkit for WCEC/WCET estimation of basic blocks based on ML techniques. To ensure the real-world applicability of its models, WORTEX extracts large datasets of basic blocks from real programs and precisely measures their energy consumption/execution time on the physical target platform. The dataset is used to train various WCEC/WCET models using different ML techniques. Experimental results on simple and time-predictable hardware show that even the most basic ML techniques provide accurate results, that never underestimate actual values. We also discuss the use of explainability techniques to gain trustworthiness for the models.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases Worst-Case Execution Time (WCET), Worst-Case Energy Consumption (WCEC), Machine Learning, Explainable ML models

Digital Object Identifier 10.4230/OASIcs.WCET.2024.1

Supplementary Material *Dataset*: `https://zenodo.org/doi/10.5281/zenodo.11066622` [22]

Funding *Hugo Reymond*: This work has received a French government support granted to the Labex CominLabs excellence laboratory and managed by the National Research Agency in the "Investing for the Future" program under reference ANR-10-LABX-07-01.
Abderaouf Nassim Amalou: This work was supported by the Agence Nationale de la Recherche under grant number ANR-22-CE92-0066.

Acknowledgements The authors would like to thank Hector Chabot for his essential work on the creation of this paper dataset.

1 Introduction

Worst-case execution time (WCET) and worst-case energy consumption (WCEC) estimation techniques play a crucial role in the validation of real-time systems. They provide upper bounds of WCEC/WCET and allow to guarantee correct operation of real-time embedded devices. Estimations of WCEC/WCET are particularly useful in battery-less devices, that harvest energy from their environment, store it in a capacitor, and execute short bursts of computation using the stored energy. For such devices, WCEC/WCET estimations can be used by static checkpointing strategies [6, 21, 31], that select checkpoint locations in the code to ensure that there is enough energy to reach the next checkpoint.

The state-of-the-art methods for statically estimating the WCET/WCEC of a program commonly use the Implicit Path Enumeration Technique (IPET) [18, 13]. IPET estimates WCET/WCEC by finding the longest path in the program's Control Flow Graph (CFG), using Integer Linear Programming (ILP) to avoid enumerating all paths in the CFG. The IPET approach requires time and energy consumption data for the basic blocks [1] of the program, which are usually derived from detailed architectural documentation of the processor. Such information is not always available due to intellectual property restrictions. In cases where they are accessible, it would require a detailed analysis of the documentation and extensive validation of the obtained model, which is time-consuming. Moreover, the documentation often does not include energy consumption information. As an alternative to an analytical model, it is possible to directly measure the basic block time/energy consumption and provide it as an input to IPET. However, the applicability of this technique is limited in practice, because the measurement campaign has to ensure that every basic block is executed sufficiently often to identify its WCET/WCEC. Moreover, this process needs to be performed for every modification of the program.

To overcome these shortcomings, a new class of approaches has been introduced, that leverages machine learning (ML) to estimate the WCEC/WCET of individual basic blocks. These approaches involve training an ML model using a large dataset of representative and varied basic blocks. The WCEC/WCET of individual basic blocks can then be used as inputs to IPET to calculate WCET/WCEC at the program level.

Previous works in this area feature some limitations. Some use small datasets or operate at the source code or intermediate code level [5, 11, 12, 30], which affects the model's accuracy. All previous ML models, including the most elaborated ones [3, 2] act as black boxes and lack interpretability of the results. To address these challenges, we propose WORTEX, for WORst-case execution Time and Energy consumption estimation using eXplainable machine learning, a toolkit for WCEC/WCET estimation, that aims at overcoming the limitations of existing techniques. WORTEX uses large and diverse datasets from AnghaBench [7] to train the models. Furthermore, we leverage a local model-agnostic explainable artificial intelligence technique [20] and discuss how such a technique can be used to understand the predictions, find bias in the training dataset, and therefore gain confidence in the model.

As a case study, we selected the MSP430FR5969 low-energy processor [27], which is extensively used in the battery-less community. To evaluate our solution, we conducted experiments at both the basic block and program levels. To do so, WORTEX generated models were integrated into the Heptane static WCET estimation tool [10].

Overall, our contributions are the following:

- We propose WORTEX, a toolkit for WCEC/WCET estimation of basic blocks using ML techniques. The simple ML models integrated in WORTEX (linear regression, gradient boosting, multi-layer perceptron neural network) were selected because of their very fast prediction, and good accuracy for the simple MSP430 platform. The loss function used when training the models is tailored to the prediction of worst-case values.
- We open-source a dataset of 30 000 basic blocks for the training of ML models [22]. All of them have been measured and have energy consumption and execution time information.
- We discuss the use of the LIME explainable AI technique [23] to validate the ML models and gain confidence in the trained models.

[1] A basic block is a CFG node, and is defined as a sequence of consecutive instructions with a single entry point and no branching inside

The remainder of the paper is organized as follows. Section 2 first compares WORTEX with related works. Section 3 then gives an overview of WORTEX in an architecture-independent way. The components of WORTEX that are specific to targeted architecture (MSP430) are described in Section 4. Section 5, presents experimental results. Finally, Section 6 discusses the use of explainable AI to validate the ML models. Section 7 concludes and outlines future works.

2 Related work

This section reports previous research leveraging machine techniques for WCET and WCEC estimation.

ML-based WCET estimation. Several methods to estimate the WCET using Machine Learning (ML) have been proposed [5, 1, 16, 17, 5, 3, 11, 2]. The studies reported in [5, 1, 11, 12] utilize worst-case event counts (number of multiplications, number of memory accesses...) extracted from the intermediate representation of the code to train different ML models, which are then used to calculate the WCET of a program in early development phases. Similarly, Kumar's research, described in [16, 17] estimate WCET by examining features taken from the program's source code. All these methods ignore important information about how the generated code and hide the effects of compilation by operating at the source code or intermediate code level, which might affect timing predictions. In contrast, the research presented in [3, 2], like WORTEX, extract features from the binary code and use ML techniques to predict the WCET of individual basic blocks. All the approaches cited above treat the ML techniques as a black boxes. WORTEX add to them the use of explanations [20] to gain more confidence in the predictions.

ML-based WCEC estimation. While numerous studies have explored the use of ML for estimating average-case energy/power consumption (as reviewed in [8]), the application of ML to estimate WCEC was only introduced in [12] and [30]. The work by Huybrechts et al. [12] specifically focuses on intermediate code representation. In contrast, WORTEX enhances the accuracy of estimation by operating on binary code. Moreover, Huybrechts et al. ML-based WCEC estimation technique typically relies on a limited number of benchmarks for training, whereas WORTEX leverages a large collection of basic blocks, ensuring a superior quality of training. In the study [30], simulation (cycle accurate model) or FPGA synthesis (Register Transfer Level RTL) is employed to capture the performance counters (such as cache misses and branch predictor misses) and the corresponding energy consumption on their target. By creating a dataset through this process, the authors employ linear regression as an energy model that can forecast energy consumption when provided with performance counters for code snippets. Contrary to this kind of approach, WORTEX does not need the RTL nor the cycle-accurate model to create an energy model that makes it usable for any target.

3 Overview of WORTEX

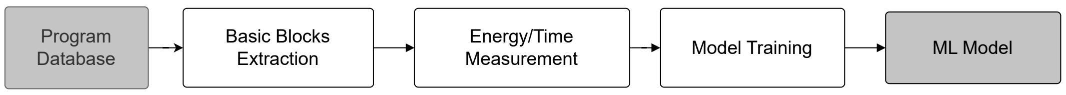


Figure 1 WORTEX's prediction workflow.

WORTEX comprises three components, depicted Figure 1. The first component *Basic Blocks Extraction* is responsible for extracting a large and unique dataset of basic blocks from different programs. The second component measures the energy consumption and execution times of all basic blocks from this dataset. The third component *Model Training* trains an ML model and deploys explainability [20] to understand the impact of each instruction on basic block WCEC/WCET. The obtained ML model can predict the WCEC/WCET of previously unseen basic blocks. Our tool offers flexibility in the choice of machine learning models, that can be used in different tools with minimal integration efforts. We have integrated the generated models into the HEPTANE static WCEC/WCET estimation tool [10]. This section describes the different components of WORTEX.

3.1 Dataset generation

To accurately predict WCEC/WCET using ML techniques, it is crucial to train the model with an appropriate dataset. This section outlines the requirements for generating such a dataset. Their implementation for a specific architecture (MSP430FR5969, from Texas Instruments) will be described in Section 4.2. The requirements that we consider important for accurate training are the following:

- **RD1: representative and diverse dataset**. This requirement aims at training the ML models on a set of basic blocks that provide good coverage of the instruction set of the target architecture but also of the code constructs of the programming language used. This requirement is met in WORTEX by using the AnghaBench benchmark suite [7] to train the models. AnghaBench contains 1 million compilable programs, mined from the largest public C repositories on GitHub [2].
- **RD2: balanced dataset**. Compilers tend to generate basic blocks (e.g. for function entry, exit, array indexing) that are identical or almost identical to other basic blocks (e.g. same instructions but different registers). Such duplicated basic blocks do not improve the quality of training and should be removed from the dataset that is used during training.
- **RD3: ability to execute basic blocks in isolation**. Basic blocks are executed in isolation to measure their energy consumption and execution time. Executing basic blocks out of their original context may lead to exceptions (e.g. divisions by zero, invalid memory accesses). A pre-processing phase has to be applied to each basic block to ensure that it executes without error. For example, the contents of the registers used for indirect accesses to memory have to be controlled to avoid indirect memory accesses.

3.2 Energy and time measurement

WORTEX is designed to produce models for *worst-case* timing and energy consumption. Hence, the measurement process must meet the following two requirements:

- **RM1: measurement in worst-case scenario:** The execution context of basic blocks, in particular in architectures with caches and pipelines, largely influences the execution time and energy consumption of basic blocks. For instance, cache misses cause longer execution times and higher energy consumption than cache hits. Enforcing RM1 highly depends on the targeted architecture, it is addressed in Section 4.3.
- **RM2: precise measurements:** The precision of measurements obviously has an impact on the quality of the energy/timing estimations. Addressing RM2 is particularly challenging because basic blocks usually comprise a small number of instructions, which makes measurements tricky on individual basic blocks. This is detailed in Section 4.3, by performing measurements on a series of basic blocks instead of individual basic blocks.

[2] AnghaBench dataset: `https://github.com/brenocfg/AnghaBench`

3.3 Model creation

Using the collected dataset, WORTEX includes three ML models capable of conservatively estimating basic block's WCEC/WCET:

- **Quantile Linear Regression (QLR) [15].** It is a statistical technique that shows how the input variables (basic block) affect linearly the output variable (WCEC/WCET). They offer a more detailed view compared to classic linear regression by looking at the full range of possibilities for the dependent variable (WCEC/WCET), not just its average.
- **Gradient Boosting (GB) [26].** Gradient boosting constructs a model based on a set of decision trees. In contrast to linear regression GB can capture non-linear relationships.
- **Multi-Layer Perceptron (MLP) [25].** An MLP is a type of artificial neural network featuring layers of interconnected nodes or neurons. It is particularly skilled at modeling intricate, non-linear relationships in the data.

Deploying the ML models is performed in two classical phases: the *training phase* and the *estimation phase*.

Training. During the training phase, WORTEX trains ML models on a large dataset to derive a WCET/WCEC model at the basic block granularity. Training is performed once for each specific micro-architecture (in our use case, the MSP430FR5969). The algorithms learn from *features* extracted from the basic blocks. Due to the simplicity of the targeted architecture, WORTEX currently uses *static* features. The features are mainly the proportions of different types of machine instructions across their various operand addressing modes divided by the total number of instructions in the basic block, for example [29]. The algorithms analyze *features* extracted from basic blocks, utilizing all MSP430 machine instructions across their various operand addressing modes. By representing instruction types as proportions, the timing/energy model created by WORTEX becomes independent of the basic block's length.

To mitigate the risk of underestimation, we leverage a *quantile loss* function, that penalizes overestimations and underestimations asymmetrically. The quantile loss evaluates the disparity between predicted and actual quantiles and for a given quantile q where $0 < q < 1$ is defined as:

$$L_q(y, \hat{y}) = \sum_{i=1}^{n} (q - \mathbf{1}\{y_i - \hat{y}_i < 0\}) \cdot (y_i - \hat{y}_i)$$

where:

- y_i and $\hat{y}_i$ are respectively the true value and the predicted for the i-th observation in the dataset
- $\mathbf{1}\{y_i - \hat{y}_i < 0\}$ is an indicator function that equals 1 if the condition $y_i - \hat{y}_i < 0$ is true and 0 otherwise.

Prediction. WORTEX estimates the WCEC/WCET at the basic block level. To estimate WCEC/WCET at the program level, the program is split into basic blocks, the features of basic blocks are extracted and the ML model is then applied. The estimated energy/timing can then be given as input to a static WCEC/WCET analysis tool.

4 Specifics of WORTEX on MSP430

This Section describes the components of WORTEX that are specific to the low-power microcontroller MSP430FR5969.

4.1 The MSP430FR5969 microcontroller

The MSP430FR5969 is an energy-efficient and low-cost microcontroller based on a 16-bit CPU, popular for applications powered by ambient energy. The micro-architecture of the MSP430X CPU series is simple. The architecture has no data cache, no branch predictor, and features a very simple pipeline with only 3 stages. The MSP430FR5969 features two main memories: Volatile Memory (VM), more precisely 2KB of SRAM, and non-volatile memory (NVM), more precisely 64kB of non-volatile ferromagnetic RAM (FRAM)[28]. The FRAM controller uses a small 2-way associative cache that has a 64-bit line size (4 16-bit instructions) to store pre-fetched instructions. In our use case, the code is stored in NVM and the VM contains the program data and stack.

4.2 Dataset generation

WORTEX first cross-compiles the numerous C sources from the AnghaBench benchmark suite into assembly language. At this stage, the generated basic blocks do not yet meet requirements RD2 and RD3 described in Section 3.1: many redundant basic blocks exist in the data set (RD2), and some of them may cause errors when executed in isolation (RD3).

Regarding the elimination of redundant basic block (RD2), preliminary experiments on the MSP430FR5969 have shown that the specific general-purpose register used in an instruction has negligible impact on its WCEC/WCET. Therefore, basic blocks are pre-processed to use a specific general-purpose register for all instructions. Duplicated basic blocks are then removed from the dataset.

Further pre-processing is required to ensure that basic blocks can be executed in isolation (requirement RD3) without addressing errors. This is achieved as follows:

- For *absolute addressing*, each accessed symbol/address is replaced with a predetermined location in NVM.
- For *indirect accesses*, where the address to be accessed is stored in a register, we face an additional challenge. In such cases, we need to have control over the value stored in the register itself. To address this, we have selected one of the registers to be a *controlled base register* used as a target for every indirect access. This register holds the address of a chosen memory location in the VM. We further replace any written instruction targeting this register with another one.

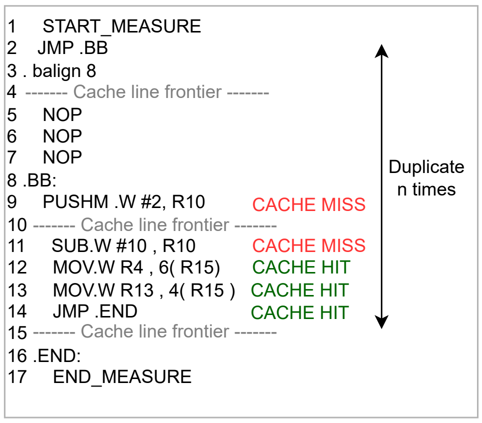
Simulation shipped with the MSP430 GNU Debugger (GDB) was used to check that no run-time error occurs and that all memory accesses fall in the VM.

4.3 Energy and timing measurement

Measurement in worst-case scenario (RM1). Enforcing the worst-case execution scenario on the MSP430FR5969 architecture is achieved through a control of its two main sources of variability, its instruction cache and its pipeline.

The worst-case scenario regarding the instruction cache is enforced by controlling the layout of basic blocks to maximize the number of cache misses. The end of the first instruction of each basic block is aligned on a cache-line boundary, as depicted in Figure 2a. Fetching the first instruction (line 9) will thus trigger a first cache miss. Then, the second instruction (line 10), belonging to another cache line, will also generate a cache miss, loading the subsequent instructions.

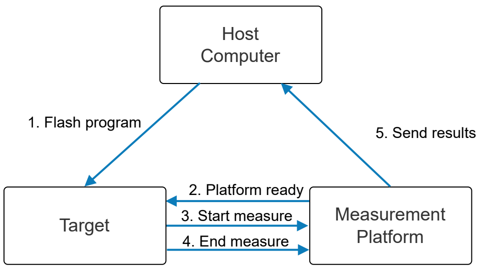
Regarding pipeline effects, adding JMP (unconditional jump) and NOP instructions before the basic block induces a pipeline flush (lines 5-7), as the NOP instruction, while loaded in the pipeline, will not be executed.



```
  START_MEASURE
 JMP .BB
. balign 8
------- Cache line frontier -------
  NOP
  NOP
  NOP
.BB:
  PUSHM .W #2, R10        CACHE MISS
------- Cache line frontier -------
   SUB.W #10 , R10        CACHE MISS
   MOV.W R4 , 6( R15)     CACHE HIT
   MOV.W R13 , 4( R15 )   CACHE HIT
   JMP .END               CACHE HIT
------- Cache line frontier -------
.END:
   END_MEASURE
```

Duplicate
n times

(a) Ensuring the worst case scenario for the MSP430FR5969 instruction cache and pipeline.



(b) Basic block measurement process.

Figure 2 Measuring a basic block energy consumption and execution time.

Precision of measurements (RM2). To measure the energy consumption and execution time of the MSP430FR5969, we need to be able to handle the scale difference between the measurement tool and the basic block execution time. This is achieved by duplicating the basic block multiple times while making sure to keep the worst-case memory layout as explained previously. Furthermore, to synchronize the basic block execution with the measurement platform, we add instructions to signal the start and the end of the execution of the basic block using General-Purpose Input/Output ports (GPIOs), (START_MEASURE and END_MEASURE on Figure 2a).

The measurement process involves three actors, as depicted in Figure 2b: the target (MSP430 under analysis), a measurement platform, and the host computer. The host computer initiates the process by flashing the instrumented code on the target (1). Once flashed, the target actively waits for the measurement platform *Platform Ready* signal (2). As it receives it, it sends back a signal *Start Measure* and starts executing the code (3). As soon as the *Start Measure* signal is received, the measurement platform measures the energy consumption of the target, until it receives the signal *End Measure* (4), marking the end of basic block execution. The measurement platform then computes the basic block execution time (defined by the time between *Start Measure* and *End Measure*). Finally, the measurement platform sends the measurements to the computer (5), which, in turn, will flash a new program to the target (1).

5 Experimental evaluation

Section 5.1 first describes the experimental setup used. Then WORTEX is evaluated according to different metrics: quality of predictions at the BB level (Section 5.2) and quality of predictions at the program level (Section 5.3)

5.1 Experimental setup

The target platform for our experiments is the MSP430FR5969 microcontroller. All code is stored in NVM and the VM is used to store program data and stack. The basic blocks, extracted from the AnghaBench benchmark suite [7] are compiled using the GCC MSP430 compiler version 9.3.1 with no optimization (-O0). Obtained basic blocks are pre-processed and filtered as explained in Section 4. Basic blocks are duplicated 50 times. The measurement platform, similar to the one used in [4], includes the following components:

- A stable power supply (N6705A [14]) providing 3.3V to power the MSP430.
- A shunt resistor with a resistance of 4.7Ω to measure the current consumed by the MSP430. This resistor has been chosen to ensure a low voltage drop (around $4.7mV$) and is connected to an operational amplifier (INA214CIDCKR) to amplify the voltage drop amplitude, allowing precise measurement. The amplified voltage is then measured with an ADS8661 analog-to-digital converter.
- A relay array to isolate the MSP430 from the computer when measuring energy. It prevents any energy interference coming from the JTAG connection.
- A bare metal Raspberry PI (RPI) 3B+ is used for synchronization with the MSP430, for controlling the isolation of the MSP430, and for time/current consumption measurement.

Section A.1 in the Appendix explains how we deduce the energy consumption from the measured current.

For each (duplicated) basic block, we perform 100 measurements and retain the highest value. Subsequently, we divide this value by the duplication factor to get the basic block energy consumption and execution time. A dataset of 30 000 unique basic blocks with their energy consumption and execution time was used, where 80% of the basic blocks were used for training and cross-validation and 20% were used for testing.

5.2 Analysis of WCET and WCEC predictions at BB level

Table 1 qualifies the results of the predictions at the BB level for the three models (QLR, GB, and MLP), by employing quantile values of 0.99 and 0.99999[3]. The quality of predictions is evaluated using two metrics.

- Mean Average Percentage Error (MAPE): MAPE $= \frac{1}{n}\sum_{i=1}^{n}\left|\frac{A_i - P_i}{A_i}\right| \times 100\%$, where A_i is the actual value, P_i is the predicted value and n is the number of basic blocks. The lower the MAPE, the more accurate the model, disregarding under/overapproximations.
- Underestimations: percentage of basic blocks whose WCEC/WCET is underestimated: Underestimation (%) $= \frac{N_{\text{underestimated}}}{N_{\text{total}}} \times 100$

Table 1 Comparison of MAPE and Underestimation Percentages for WCET and WCEC Predictions at Quantiles 0.99 and 0.99999 Across Models.

Model	MAPE				Underestimation (%)			
	Quantile 0.99		Quantile 0.99999		Quantile 0.99		Quantile 0.99999	
	WCET	WCEC	WCET	WCEC	WCET	WCEC	WCET	WCEC
QLR	56.7 %	58.0 %	101.3 %	**97.3 %**	0.69 %	**0.52 %**	**0 %**	**0 %**
GB	42.1 %	39.8 %	93.0 %	**97.3 %**	**0.56 %**	0.65 %	**0 %**	**0 %**
MLP	**8.2 %**	**16.2 %**	**41.1%**	99.1 %	0.65 %	1.07 %	**0 %**	**0 %**

The results show that the MLP model outperforms QLR and GB in terms of accuracy, with the lowest MAPE for both time and energy and both quantile values, except at WCEC for 0.99999 quantile where the QLR and GB model are more accurate. As we shift from

[3] A quantile of 0.99 (resp. 0.99999) means that we aim at predictions that are larger or equal to the actual WCEC/WCET in 99% (resp. 99.999%) of the cases.

quantile 0.99 to quantile 0.99999, accuracy degrades for all models, as shown by the increase in the MAPE metric. However, this degradation comes with a sharp decrease in the ratio of underestimations (no underestimation is actually observed), which is essential when estimating WCEC/WCET. More complete results about the quality of predictions for the different techniques are given in Section A.2 in the appendix.

5.3 Analysis of WCET and WCEC predictions at program level

This experiment aims at evaluating the error made by WORTEX when it is used by the Heptane static WCET analysis tool to estimate WCEC/WCET of entire programs, from the Mälardalen benchmark suite [9]. In this experiment, the WCET/WCEC predicted by WORTEX (for 0.99999 quantile) is fed to Heptane that uses the standard Implicit Path Enumeration Technique (IPET, [18]). We selected a small subset of benchmarks for which we can generate input data that systematically execute the longest execution path found by Heptane, ensuring that the pessimism comes from WORTEX and not from Heptane.

Table 2 Observed vs predicted execution time/energy consumption (time in µs, energy in nJ).

Program		Maximum observed		QLR based estimations		GB based estimations		MLP based estimations	
		Time	Energy	Time	Energy	Time	Energy	Time	Energy
bs		293	358	507	516	445	459	334	376
fibcall		1 035	1 059	1 131	1 132	1 181	1 214	1 197	1 380
lcdnum		919	1 019	1 506	1 516	1 372	1 484	1 077	1 260
nsichneu		24 672	25 179	36 252	37 256	33 367	35 141	27 248	31 967
Overestimation			mean	48.3%	36.9%	37.6%	32.0%	**14.3%**	21.5%
			min	9.2%	6.9%	14.1%	14.6%	10.4%	**5.0%**

Table 2 shows the maximum observed execution time and energy consumption for each program (executed its worst-case input) and the prediction from the different ML techniques. Regardless of the benchmarks studied, we can make several observations. First, thanks to the loss used during the training phase, the prediction always overestimates the observed execution time and energy consumption. Second, MLP-based techniques significantly outperform QLR and GB. Third, the overestimation for the best-performing technique MLP is reasonable (14% on average for time, 21% for energy).

5.4 Inference time

One of the interests of using ML techniques for WCEC/WCET estimation if that the predictions are fast enough to be used in WCET/WCEC analysis tools. Table 3, reports the average inference time of one basic block for each technique.

Table 3 Average inference time per basic block, on an Intel i7-11850H CPU.

Method	QLR	GB	MLP
Average time (µs)	1.95	48.95	35.88

Regardless of the chosen technique, the inference time is within the micro-second range. As anticipated, the QLR model, due to its simplicity, exhibits the shortest inference time, surpassing the other techniques by approximately a factor of 20. In contrast, GB and MLP show higher inference time but offer better precision.

6 Discussion on explainable AI

Most ML techniques, while making accurate predictions, are black-box techniques, meaning their users cannot easily understand *why* they make their decisions. *Explainability* [20] techniques help understand why a model makes certain predictions, by revealing the factors that influence the estimations. Explainability helps detect misbehaviors of ML models, caused for example by biased datasets or sub-optimal selection of features. Once the misbehaviors are detected and corrected, confidence can be gained in the models.

Several techniques for explainability exist for black-box models. Among them, we focus in this paper on LIME [23] (Local Interpretable Model-agnostic Explanations). LIME creates interpretable models (e.g., linear regression) that approximate the predictions of the black-box model around a specific data point of interest. It achieves this by generating a synthetic dataset (i.e., neighborhood points) and training a simple, interpretable model on it (a linear regression in our case). By analyzing the behavior of this *new simple local model*, we can gain insights into the factors influencing the model's decision at that particular data point.

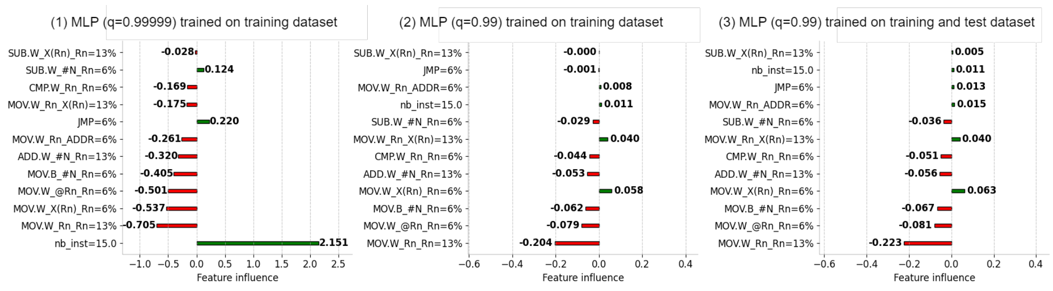


Figure 3 LIME Explanations for MLP on a BB under different training settings (quantile value, training dataset).

Figure 3 illustrates the explainability of the MLP model generated by LIME for a single BB sample. This MLP is trained under different configurations of quantile loss and dataset compositions to investigate the model's decisions. The figure is divided into three generated LIME explanations, where the x-axis represents the direction of influence that each feature has on the model's prediction (which we retrieve from a linear regression local model): positive values are associated with an increase in the predicted outcome, while negative values suggest a decrease. The y-axis lists the non-null features of the BB: instructions (type and addressing mode), with a percentage indicating their occurrence rate within the BB.

In Part (1) of Figure 3, the MLP is trained with a quantile loss of 0.99999. The results show a significant negative influence of certain features like `MOV.W_X(Rn)_Rn` and `MOV.W_Rn_X(Rn)`, which is counter-intuitive considering the known indirect access mode instruction latencies for the MSP430 [29]. This suggests that the extreme quantile value, aiming to capture the worst-case predictions, may be skewing the model's focus towards only the `nb_inst` (number of instructions) feature to make a conservative prediction independently of the actually executed instructions.

Part (2) of Figure 3 presents the LIME explanations when the MLP is trained with a quantile loss of 0.99. Here, the influence of features appears to align more closely with the MSP430 instruction latencies [29], except for `SUB.W_X(Rn)_Rn`. Although it should have the same influence factor as the `MOV.W_X(Rn)_Rn` instruction, it does not. Upon investigation, we found that the subtraction instruction appears less frequently than the `MOV` instruction within our training dataset.

Finally, to investigate whether the hypothesis that the rarity of the `SUB.W_X(Rn)_Rn` instruction influenced the explanation, we explore in part (3) of Figure 3 LIME explanations with post-augmentation of the training dataset (used in the previous two experiments) with the test set, under the same quantile loss of 0.99. This data enrichment leads to a further refined interpretation of feature influences, correcting earlier misalignments between LIME explanation and the MSP430 documentation [29], for instance `SUB.W_X(Rn)_Rn` has now a positive influence.

These findings highlight the importance of selecting an appropriate quantile loss value ensuring the comprehensiveness and the empirical safety of the ML models. They also show that XAI [20] techniques can help us diagnose our model and dataset.

7 Conclusion

This paper has introduced WORTEX, a toolkit for WCEC/WCET estimation of basic blocks using ML techniques. WORTEX was shown to produce safe yet precise WCEC/WCET estimates for low-power processors. As future work, we believe that integrating WORTEX in a compiler toolchain would allow us to explore program optimizations with both time and energy in mind would be interesting. In addition, further exploring other explainability techniques like SHAP [19] or Anchors [24] will help the design of WCEC/WCET estimation techniques on more complex machine learning techniques, for example, Transformers [2].

References

1 Peter Altenbernd, Jan Gustafsson, Björn Lisper, and Friedhelm Stappert. Early execution time-estimation through automatically generated timing models. *Real-Time Systems*, 2016. `doi:10.1007/s11241-016-9250-7`.

2 Abderaouf Nassim Amalou, Elisa Fromont, and Isabelle Puaut. CAWET: Context-Aware Worst-Case Execution Time Estimation Using Transformers. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, 2023. `doi:10.4230/LIPIcs.ECRTS.2023.7`.

3 Abderaouf Nassim Amalou, Isabelle Puaut, and Gilles Muller. WE-HML: hybrid WCET estimation using machine learning for architectures with caches. In *IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2021. `doi:10.1109/RTCSA52859.2021.00011`.

4 Gautier Berthou, Kevin Marquet, Tanguy Risset, and Guillaume Salagnac. Accurate power consumption evaluation for peripherals in ultra low-power embedded systems. In *Global Internet of Things Summit (GIoTS)*, 2020. `doi:10.1109/GIOTS49054.2020.9119593`.

5 Armelle Bonenfant, Denis Claraz, Marianne De Michiel, and Pascal Sotin. Early WCET prediction using machine learning. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2017. `doi:10.4230/OASIcs.WCET.2017.5`.

6 Jongouk Choi, Larry Kittinger, Qingrui Liu, and Changhee Jung. Compiler-directed high-performance intermittent computation with power failure immunity. In *28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'22)*, 2022. `doi:10.1109/RTAS54340.2022.00012`.

7 Anderson Faustino Da Silva, Bruno Conde Kind, José Wesley de Souza Magalhães, Jerônimo Nunes Rocha, Breno Campos Ferreira Guimaraes, and Fernando Magno Quinão Pereira. Anghabench: A suite with one million compilable c benchmarks for code-size reduction. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021. `doi:10.1109/CGO51591.2021.9370322`.

8 Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 2019. `doi:10.1016/j.jpdc.2019.07.007`.

9 Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET benchmarks: Past, present and future. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2010. doi:10.4230/OASIcs.WCET.2010.136.

10 Damien Hardy, Benjamin Rouxel, and Isabelle Puaut. The heptane static worst-case execution time estimation tool. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2017. doi:10.4230/OASIcs.WCET.2017.8.

11 Thomas Huybrechts, Siegfried Mercelis, and Peter Hellinckx. A new hybrid approach on WCET analysis for real-time systems using machine learning. In *18th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2018. doi:10.4230/OASIcs.WCET.2018.5.

12 Thomas Huybrechts, Philippe Reiter, Siegfried Mercelis, Jeroen Famaey, Steven Latré, and Peter Hellinckx. Automated testbench for hybrid machine learning-based worst-case energy consumption analysis on batteryless iot devices. *Energies*, 2021. doi:10.3390/en14133914.

13 R. Jayaseelan, T. Mitra, and X. Li. Estimating the worst-case energy consumption of embedded software. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2006. doi:10.1109/RTAS.2006.17.

14 KEYSIGHT. N6700 Modular Power System Family. URL: https://www.keysight.com/fr/en/assets/7018-05443/data-sheets/5992-1857.pdf.

15 Roger Koenker. *Quantile regression*, volume 38. Cambridge university press, 2005.

16 Vikash Kumar. Deep neural network approach to estimate early worst-case execution time. In *IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021. doi:10.1109/DASC52595.2021.9594326.

17 Vikash Kumar. Estimation of an early WCET using different machine learning approaches. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 2022. doi:10.1007/978-3-031-19945-5_30.

18 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *SIGPLAN workshop on Languages, compilers, & tools for real-time systems*, 1995. doi:10.1145/216636.216666.

19 Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 2017.

20 Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable machine learning–a brief history, state-of-the-art and challenges. In *ECML PKDD 2020 Workshops*, 2021. doi:10.1007/978-3-030-65965-3_28.

21 Hugo Reymond, Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou, Isabelle Puaut, and Erven Rohou. SCHEMATIC: compile-time checkpoint placement and memory allocation for intermittent systems. In *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2024*, 2024. doi:10.1109/CGO57630.2024.10444789.

22 Hugo Reymond, Hector Chabot, Abderaouf Nassim Amalou, and Isabelle Puaut. MSP430FR5969 basic block worst case energy consumption (WCEC) and worst case execution time (WCET) dataset, April 2024. doi:10.5281/zenodo.11066623.

23 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *SIGKDD international conference on knowledge discovery and data mining*, 2016. doi:10.1145/2939672.2939778.

24 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, 2018. doi:10.1609/aaai.v32i1.11491.

25 David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1986. doi:10.1038/323533a0.

26 Robert E. Schapire. The boosting approach to machine learning: An overview. *Nonlinear estimation and classification*, 2003. doi:10.1007/978-0-387-21579-2_9.

27 Texas Instruments. MSP430FR5969 user's guide. URL: https://www.ti.com/lit/ug/slau367p/slau367p.pdf.

28 MSP430FR5969 product information. URL: https://www.ti.com/product/MSP430FR5969.

29 Msp430 family instruction set summary. URL: `https://www.ti.com/sc/docs/products/micro/msp430/userguid/as_5.pdf`.

30 Simon Wegener, Kris K. Nikov, Jose Nunez-Yanez, and Kerstin Eder. EnergyAnalyzer: Using Static WCET Analysis Techniques to Estimate the Energy Consumption of Embedded Applications. In *21th International Workshop on Worst-Case Execution Time Analysis (WCET 2023)*, 2023. `doi:10.4230/OASIcs.WCET.2023.9`.

31 Bahram Yarahmadi and Erven Rohou. Compiler Optimizations for Safe Insertion of Checkpoints in Intermittently Powered Systems. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2020. `doi:10.1007/978-3-030-60939-9_12`.

A Appendix

A.1 Computing the energy consumption from the current consumption

The energy consumption of a basic block is computed from voltage and current consumption using equation 1.

$$E = \int U \times i(t)dt \tag{1}$$

As we work with discrete samples, equation 1 becomes equation 2:

$$E = \sum_{k=0}^{n} U \times i(k) \times t_{sample} = U \times t_{sample} \times \sum_{k=0}^{n} i(k) \tag{2}$$

t_{sample} defines the time needed to get a sample, is supposed to be the same for each sample, and is computed as shown in Equation 3.

$$t_{sample} = \frac{T_{measure}}{n_{sample}} \tag{3}$$

A.2 Detailed results on BB prediction

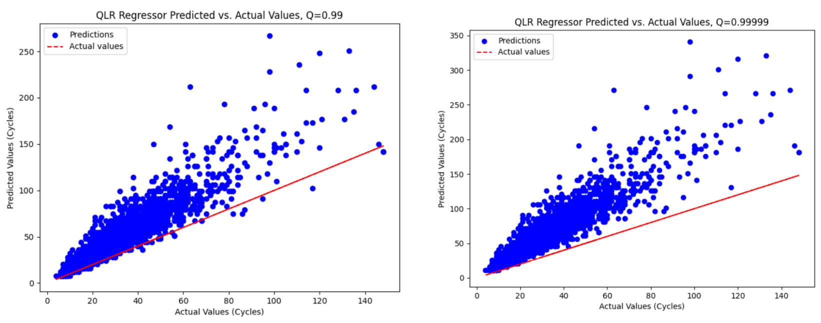


Figure 4 Predictions using QLR for quantile values of 0.99 (left) and 0.99999 (right).

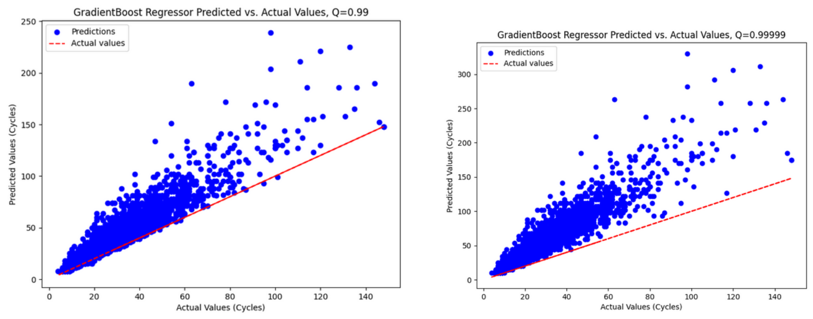


Figure 5 Predictions using GB for quantile values of 0.99 (left) and 0.99999 (right).

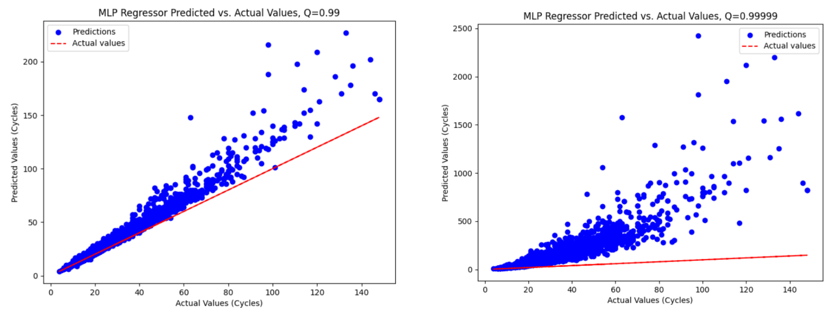


Figure 6 Predictions using MLP for quantile values of 0.99 (left) and 0.99999 (right).

The Platin Multi-Target Worst-Case Analysis Tool

Emad Jacob Maroun
Technical University of Denmark, Lyngby, Denmark

Eva Dengler
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Christian Dietrich
Technische Universität Braunschweig, Germany

Stefan Hepp
Technische Universität Wien, Austria

Henriette Herzog
Ruhr-Universität Bochum, Germany

Benedikt Huber
Technische Universität Wien, Austria

Jens Knoop
Technische Universität Wien, Austria

Daniel Wiltsche-Prokesch
Hitachi Rail, Wien, Austria

Peter Puschner
Technische Universität Wien, Austria

Phillip Raffeck
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Martin Schoeberl
Technical University of Denmark, Lyngby, Denmark

Simon Schuster
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Peter Wägemann
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

With the increasing number of applications that require reliable runtime guarantees, the relevance of static worst-case analysis tools that can provide such guarantees increases. These analysis tools determine resource-consumption bounds of application tasks, with a model of the underlying hardware, to meet given resource budgets during runtime, such as deadlines of real-time tasks.

This paper presents enhancements to the Platin worst-case analysis tool developed since its original release more than ten years ago. These novelties comprise Platin's support for new architectures (i.e., ARMv6-M, RISC-V, and AVR) in addition to the previous backends for Patmos and ARMv7-M. Further, Platin now features system-wide analysis methods and annotation support to express system-level constraints. Besides an overview of these enhancements, we evaluate Platin's accuracy for the two supported architecture implementations, Patmos and RISC-V.

2012 ACM Subject Classification Software and its engineering → Automated static analysis; Software and its engineering → Compilers; Computer systems organization → Real-time systems

Keywords and phrases worst-case resource consumption, WCET, static analysis tool

Digital Object Identifier 10.4230/OASIcs.WCET.2024.2

Supplementary Material *Software (Platin Source Code)*: `https://github.com/t-crest/platin`

Funding The contributions of Emad Jacob Maroun have been funded by Huawei Technologies Sweden AB under agreement number TC20220819029. Authors from the Friedrich-Alexander-Universität Erlangen-Nürnberg have been funded in parts by the German Research Foundation (DFG) under the project number 502947440 (Watwa).

1 Introduction

Safety-Critical Applications & Worst-Case Analysis. The relevance of solving the worst-case execution time (WCET) problem [57] is higher than ever when considering the increasing number and the complexity of today's safety-critical application requirements running on modern processors. From small medical (implantable) devices over automotive and avionics applications to large industrial-control scenarios, systems that require resource budget guarantees for safe execution need tooling support to determine resource bounds analytically. The basic principle of worst-case analysis is to combine a representation of the system's program paths with a cost model of the underlying hardware. With this knowledge, the worst-case analysis generates a mathematically sound problem formulation, such as an integer linear program (ILP). When given to a mathematical solving tool, the solution of this formulation yields resource-consumption bounds, which are used for offline budgeting of runtime resources.

Resource Other Than Time: Worst-Case Energy Consumption. This method of combining a program-path model with a cost model was introduced in the 1990s [35, 39] and referred to as the *Implicit Path Enumeration Technique (IPET)*. The original purpose of the IPET targeted the timeliness of real-time systems. However, Jayaseelan et al. later demonstrated the applicability of this approach for determining worst-case energy consumption (WCEC) bounds [28]. In the same way, WCET bounds are crucial for meeting deadlines in real-time systems; WCEC estimates are helpful in energy-constrained settings to guarantee the safe completion of tasks under energy budgets. This paper addresses the two resources: time and energy within the PLATIN tool with WCET/WCEC analyses.

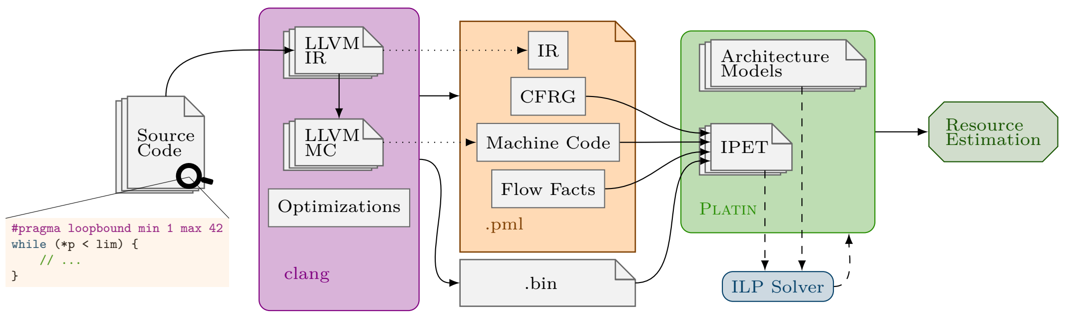
Necessity for Open Architectures & Open Tooling Infrastructures. The PLATIN tool was originally introduced more than ten years ago [23, 40] as a portable LLVM annotation and timing toolkit. PLATIN's development started with the T-CREST project [44], targeting time-predictable multi-core architectures. With the entire technology available as open source, we argue that the research community requires both open processor architectures and the respective worst-case tooling support to advance state-of-the-art without unnecessary barriers (i.e., licensing, closed-source infrastructures). In line with this rationale, all our improvements and extensions to the PLATIN tool have been published as open-source over the last few years.

Contributions. The core contribution of this work is an overview of these novelties compared to PLATIN's initial release [23]. The novelties include both pillars of worst-case analysis: (1) Regarding the hardware-dependent cost modeling, PLATIN now supports four new architectures (ARMv6-M, ARMv7-M, RISC-V, AVR). (2) Given the hardware-agnostic program-path analysis, we give insight into the introduced support for system-wide resource-consumption analysis and PLATIN's annotation infrastructure. Besides the overview of existing work, we evaluate PLATIN for the Patmos and RISC-V (RV32IMC ISA) architectures.

Paper Organization. The paper is structured as follows: Section 2 gives a general overview of the tooling infrastructure of and around PLATIN. The existing and newly introduced architectures are part of Section 3. Section 4 describes extensions of PLATIN for whole system time and energy analysis. Section 5 presents evaluation results. Section 6 discusses related work. Section 7 concludes the paper.

2 Overview of the PLATIN Analysis Tool

The PLATIN ecosystem displayed in Figure 1 combines compilation and WCET analysis to make use of high-level information that the compiler already has [40]. The source code, potentially enriched with user-annotated control-flow information (so-called *flow facts*) such



```
#pragma loopbound min 1 max 42
while (*p < lim) {
    // ...
}
```

Figure 1 Overview of PLATIN's ecosystem for compiler-analysis integration providing analysis-aware compilation to improve accuracy with automatically collected program meta-information.

as loop bounds, is compiled by an extended version of the `clang` compiler into the final binary and a meta-information file (`.pml`). This meta-information contains the program control flow and flow facts in the YAML[1]-based Program Metainfo Language (PML) format specific to PLATIN. Specifically, it contains the program's control-flow graph (CFG) in an intermediate representation (IR) and on machine-code (MC) level. A control-flow–relation graph (CFRG) [25] matches program paths between the two representations even across different optimization-induced control-flow transformations. The CFRG is thus a useful tool to lower IR-level flow facts (both annotated and compiler-inferred) to the machine-code level, where the actual timing analysis is performed. PLATIN uses the CFGs and the lowered flow facts to derive an IPET formulation and, finally, transforms this formulation together with a target-specific cost model from PLATIN's architecture models into an integer linear program (ILP). An external ILP solver (e.g., `lp_solve`, `gurobi`) then yields the resource bounds.

At the heart of PLATIN are the architecture models, which provide the translation from control-flow information to platform-specific resource demands. The core component for each of PLATIN's architecture models is a cost model of the machine instructions, informing PLATIN in how many processor cycles each instruction is executed in the worst case. Modeling of the microarchitecture, such as processor pipelining or caches, refines the model, allowing for more accurate bounds than pessimistic assumptions about cache misses and pipeline stalls.

In the following, we give further insights into the analysis-aware compilation process (Section 2.1) and other tools of PLATIN besides the analysis (Section 2.2).

2.1 Analysis-Aware Compilation with Clang

For PLATIN to perform its analyses, it needs the control-flow and flow-fact information provided in the PML format. PLATIN uses this data-serialization format to store and retrieve relevant program information for the worst-case analysis. Our fork of the LLVM compiler framework [32] includes support to automatically create the accompanying PML files for each compilation unit with a mixture of user-annotated and compiler-generated knowledge. The LLVM/clang infrastructure's code base is rapidly changing, which leads to the challenge of keeping our analysis infrastructure up to date with new LLVM/clang releases. To make

[1] YAML data-serialization language: `https://yaml.org`

Table 1 Overview of PLATIN's supported architectures with respective processor implementations.

Architecture	Processor Implementation
Patmos	Chisel-based implementation with FPGA synthesis (Altera DE2-115)
ARMv6-M	NXP FRDM-KL46Z with ARM Cortex M0+ [2, 3, 20]
ARMv7-M	XMC4500 with ARM Cortex M4 [4, 5, 27]
RISC-V	ESP32-C3 supporting RV32IMC extensions [15, 43]
AVR	ATmega1284p [6, 7]

forward compatibility and version updating as easy as possible, we strive not to change the core LLVM code and only add code specific to our use cases. That way, we can benefit from improvements in the LLVM infrastructure (e.g., novel analysis passes) with minor changes (e.g., adapting the PML export logic).

The compiler first takes the C source code and compiles it into LLVM intermediate representation (LLVM-IR). Any flow fact information, including loop bounds provided as pragmas, is embedded in the LLVM-IR to maintain it through the compilation pipeline [25]. Besides the program code, required standard libraries for the target can be linked on the LLVM-IR level with `llvm-link`. This enables a whole program view for the remaining steps, including optimization and PML export.

For targets that cannot be linked with `clang` or where libraries are not available as source code, the compiler produces object files, which can then be linked (without further optimizations) with an external linker. Library functions are only available at link time; however, they are challenging if they are part of any program path beginning from the analysis entry point. A timing bound can be derived solely from the machine code or must be known from external sources.

The backend exports the control flow and flow-fact information at the last stage of the compilation, where the machine instructions and their final order have been determined. The PML format and the compiler code managing its export are architecture-independent, giving seamless support for all current and future architectures.

2.2 PLATIN's Supporting Tools

Besides worst-case analyses, the PLATIN ecosystem provides several accompanying tools that support the analysis. Visualization of the CFRG allows debugging in cases where the one-on-one mapping is violated. Likewise, ILP visualization makes understanding the analysis results possible, while an interactive version enables live analysis in large projects.

Integration with external analysis tools (e.g., aiT) and transformation tools from and to the PML format allow PLATIN to profit from existing analyses. A configuration tool inspired by `pkg-config` helps to invoke tools with the correct options (e.g., target-specific flags, analysis entry) to guarantee interoperability with PLATIN.

3 PLATIN's Support for Multiple Architectures

The original version of PLATIN had full support for the Patmos processor and initial support for the ARM architecture, expressing the hope that using LLVM as a basis would allow for quick development of further backends [23]. This hope proved warranted, as PLATIN now supports multiple architectures. Table 1 gives an overview of the available architecture backends and corresponding processor models, further described in the following.

Patmos Architecture. PLATIN was initially developed for the Patmos architecture as part of the T-CREST project [44]. It provides full support for the architecture, which is also the default target [45]. As Patmos was developed explicitly for real-time systems, it has a unique cache structure: It uses a method instead of an instruction cache [11], which loads complete methods or subsets of them at predefined points in the program. It also has a dedicated stack cache for caching stack-local data. PLATIN supports modeling both of these caches [26, 29]. Patmos also has a traditional data cache. As PLATIN does not have native data-cache modeling, it assumes all data-cache accesses miss. However, when integrating with the AbsInt aiT analysis tool, AbsInt's data-cache modeling can be leveraged for improved WCET bounds.

ARM Architectures. PLATIN currently has support for the ARMv6-M and ARMv7-M versions of the ARM architecture. For the ARMv6-M backend, namely for the NXP FRDM-KL46Z with a Cortex M0+ processor, we demonstrated the feasibility of automatic derivation of the cycle costs of the timing models [50, 51]. The Cortex M0+ has a comparatively simple microarchitecture, which is not modeled explicitly but is part of the derived model. The ARMv7-M backend, which is for the XMC4500 with a Cortex M4 processor, features integrated modeling of the processor pipeline and the instruction cache [41] building upon the concept of microarchitecture execution graphs [52].

RISC-V Architecture. Additionally, we introduced support for the open-source hardware standard RISC-V [43] as an additional backend [12]. The supported ESP32-C3 [15] system-on-chip, which uses the RV32IMC instruction set, features a 4-stage pipeline and zero-wait-state memory for both instruction and data access. Due to the lack of documentation on the timing behavior, the timing model is derived from measurements, including the effects of pipelining.

AVR Architecture. We further extended PLATIN to support the AVR architecture, often utilized for embedded systems and popular Arduino projects. AVR microcontrollers typically have a relatively simple microarchitecture that allows straight-forward hardware models and their integration into PLATIN, in our case for the ATmega1284p [6]. Almost all instructions are executed with constant timing, documented in the AVR Instruction Set manual [7]. As the ATmega1284p does not have integrated caches, PLATIN's AVR backend allows for accurate WCET-bound predictions. To underpin this statement based on the exemplary benchmark `count_negative` from the TACLeBench suite [17]: This benchmark avoids overestimations from the program-path analysis and, consequently, helps to reveal pessimism originating from the architecture modeling. PLATIN's AVR backend reports 24009 cycles while the (straight-line code) measurement counts 22560 cycles: These results indicate minor analysis pessimism with the overestimation by 6 % and highlight PLATIN's applicability for the predictable AVR architecture.

4 PLATIN's Path-Analysis & Annotation Extensions

Besides PLATIN's support for several architectures, several works extended the analysis toolkit to support whole-system analyses (see Section 4.1 and 4.2) and express semantic annotations across the system stack (see Section 4.3).

4.1 SysWCET: Whole-System Response-Time Analysis

With static real-time analysis, we calculate the response-time bounds of digital systems for (external) events. Usually, we first calculate the WCET of each task in isolation before the worst-case response-time (WCRT) analysis takes the surrounding execution context (i.e., other tasks, the operating system, IRQs) into account. While this two-step approach reduces complexity, it accumulates pessimism as program-level flow constraints cannot interact with system-level constraints. With SysWCET [14], PLATIN can express the WCRT analysis for a task as a WCET analysis of the whole system while executing that task.

SysWCET formulates an ILP that encodes not only the intra- and inter-procedural control flow graphs but also the system-state transition graph (SSTG) [13], thus allowing for function- and system-level flow constraints. To calculate the SSTG, we perform an abstract interpretation of the complete system, including the operating system, preempting interrupts, and all tasks with their (fixed-priority) scheduling semantics. Hence, the SSTG includes all synchronous and asynchronous control-flow transitions between tasks and interrupt handlers.

For SysWCET, we extended PML to store the different control-flow levels (function, task, system) and generalized the IPET to encode those levels into a single ILP simultaneously. Hence, ILP-encoded flow-fact constraints can include variables from all control-flow levels. For example, SysWCET can express that two branches in different tasks are mutually exclusive, further tightening the WCRT bounds. Furthermore, parametric annotation languages (see Section 4.3) allow for more accurate, context-sensitive timing bounds.

4.2 SysWCEC: Whole-System Energy-Consumption Analysis

A further extension to the system-state graph provides PLATIN with knowledge about the devices present in the system, their state (on/off), and how much power they draw in the respective state. Combined with the timing analysis, this enables PLATIN to yield worst-case bounds on the energy consumption of the analyzed systems [54]. Comparable to SysWCET, the energy-related analysis can determine the code's worst-case energy demand between two arbitrary program points. Thereby, the analysis determines the *worst-case response energy consumption* of tasks, that is, the demand from start to finish of an operation, including all power-state changes and the scheduling semantics. This interplay between types of worst-case analyses underlines the usability of analysis techniques originally introduced for timeliness to also work for the energy resource.

Beyond the modeling of simple on/off states of devices, an additional enhancement keeps track of internal device states and configurations, enabling fine-grained modeling of device behavior across system states [42]. As a result, this enables PLATIN to derive more accurate resource bounds, for example, for modeling the states of transceiver devices. The scope of these energy-related extensions goes beyond the worst-case execution-time analysis of real-time systems since these analyses are beneficial for highly energy-constrained systems, such as intermittently-powered embedded systems. One example is systems with intermittent power supply that, for example, harvest their energy through solar cells within the battery-free Internet of Things [1].

4.3 System-Wide Annotation Support

Within system-wide analyses, the operating system's kernel represents an interesting target for the static timing analysis, as the WCET of a system call is not static but heavily depends on the system state. One solution proposed [37] to resolve this lack of application information within kernel-level analyses is to move to a parametric analysis that can jointly

```c
void func(void *data, size_t len) {
  for (size_t i = 0; i < len; i++) {
    #pragma platina lbound "max_len"
    /* ... */
  }
}
```

```c
#pragma platina let "max_len=12"
func(input, 12);

#pragma platina let \

"max_len=NUM_TASKS"
func(tasks, numReadyTasks());
```

Figure 2 Parametric loop annotation in function `func` assigning context-sensitive values to the symbolic variable `max_len` at the call sites as manual loopbound (`12`) and system fact (`NUM_TASKS`).

Table 2 Reasoning for excluding some of the TACLeBench benchmarks from the evaluation.

Reason for exclusion	Benchmarks
Recursion	ammunition, anagram, bitcount, bitonic, fac, huff_enc, quicksort, recursion
Not self-contained	DEBIE, PapaBench, rosace
Infeasible loop bound	rijndael_dec, rijndael_enc

consider both applications and the backing RTOS. SWAN [48] addresses this challenge by introducing system facts, a unit of information gained from system-level analysis and referenced in source-level, parametric annotations within the operating system code. By evaluating the annotation expressions over the interaction's system facts and lowering the flow facts gained to the machine-code level with the aid of CFRGs [25], Platin can thus yield system-context–specific timing bounds for individual system calls. PragMetis [49] extends this parameterization from a per-system-interaction level to smaller structural contexts such as call- and loop-contexts, as shown by the example in Figure 2. This allows Platin to express parametricity within a single system call and eases the use of parametric annotations within application-level libraries that often exhibit similar context sensitivity.

5 Evaluation

We evaluate the performance of Platin on the TACLeBench benchmark suite [17]. However, we had to exclude some programs. As Table 2 shows, eight programs were excluded for using recursion, as Platin cannot handle recursion. The three `Parallel` benchmarks (`DEBIE, PapaBench, rosace`) were excluded for not being self-contained and needing OS support for threading. Finally, two benchmarks (`rijndael_dec, rijndael_enc`) include invalid loop bounds. We excluded the benchmarks as Platin requires correct bounds to produce meaningful results. After these necessary exclusions, 47 benchmarks remain for the evaluation.

In Table 3, we give the Platin-provided bound for the remaining benchmarks of the TACLeBench suite. We compare the measured execution time for each architecture against the Platin bound. The measured times for the Patmos target are with the data cache disabled, equivalent to Platin assuming all data-cache accesses miss. This can give us a slight sense of the efficiency of Platin, though we must stress that our measured times are not guaranteed to be the true WCETs since TACLe does not guarantee the input exercises the worst-case path. For the RISC-V target, we work with an ESP32-C3 processor, which features an RV32IMC instruction set and a single-cycle–accessible SRAM [15]. The SRAM has a storage capacity of 400 KB, which was large enough for our tests. We employed a measurement-based approach to determine the WCET of each instruction available on the processor. The benchmarks for the ESP32-C3 are executed on the `ESP32-C3-DevKitM-1 v1.0` development board. We use the CPU cycle counter available on the ESP32-C3 to determine the run time of each benchmark. For floating-point operations, a software-float implementation is used for the ESP32-C3. We omitted a WCET for the floating-point library instructions in Platin, and, therefore, the related benchmarks (marked with ≋) are skipped. For the Patmos target, five additional benchmarks were excluded for incorrect compilation (marked with 🐞). The 🖩 mark is used when Platin failed to provide a valid bound.

The third column of each target is the pessimism, i.e., by how much the given bound is higher than the measured execution time. For the Patmos target, we can see that the pessimism ranges widely. For some of the simpler programs, Platin was able to identify the measured execution time as the WCET. This is not surprising in simple cases like `countnegative`, however, it is surprising in more complex cases like `h264_dec`.[2] On the other hand, some benchmarks have very high pessimism. For example, the pessimism of `fft` is 25 820 %, which is likely down to this program having nested loops with big ranges between minimum and maximum loop bounds.[3] A similar picture emerges for the RISC-V evaluations: most of the pessimism ranges from 80 % to 300 %. The pessimism of `fft` again marks an outlier with 42 590 %, with the same reasoning as for the Patmos target. Unlike the Patmos target, Platin does not produce any bound equal to the WCET for the RISC-V target. This is mainly due to pessimism introduced by the "C" extension of RISC-V: The `compressed` instruction-set extension offers shorter codes (2-byte instead of 4-byte) for often-used instructions. Therefore, the control flow during branching instructions can enter at addresses that are not 4-byte-aligned. As the bus only supports loading 4-byte-aligned code, a 4-byte instruction may need two loads instead of one to fetch the entire instruction from memory. This pessimism at branching instructions leads to more overestimates than the Patmos target.

6 Related Work

Worst-Case Analysis Tools. To this point, numerous WCET analyzer tools have been developed for different hardware platforms. Several analyzers stem from academia [8, 16, 19, 21, 22, 24, 30, 31, 34, 36, 40, 46] and, based on these results, commercially available products are available [10, 19, 30]. This underlines the importance of WCET analysis in safety-critical real-time systems.

Hybrid WCET Analysis. With the increasing complexity of modern high-performance multi-core microarchitectures, the use of hybrid WCET tools is gaining in importance: Determining accurate timing models of the target architecture can become practically infeasible with the lack of documentation and unpredictable components. In this context, the TimeWeaver tool [30] presents a hybrid approach: This approach combines timing information from measurements with static analysis techniques. Such hybrid resource-consumption approaches are also interesting in the context of Platin's system-wide analysis techniques [14, 54].

LLVMTA. The infrastructure of LLVMTA [21] is related to the Platin infrastructure, with both projects relying on the LLVM framework. LLVMTA focuses on microarchitectural analysis and implements its analyses on the final assembler representation in the LLVM backend. LLVMTA has no integration into the clang compiler, comparable to Platin's support of control-flow-relation graphs. That is, LLVMTA cannot exploit high-level source code information within the resource-consumption analysis.

[2] Remember, no data caches are used.
[3] Remember, the suite does not guarantee the programs exhibit WCET.

Table 3 Comparison of measured execution times and WCET bounds provided by Platin. Bounds with a ’*’ use the `gurobi` optimizer instead of the default `lp_solve`.

	Patmos			RISC-V		
benchmark	Measured	Bound	Pessimism	Measured	Bound	Pessimism
lift	2 567 285	6 506 322	153 %	1 738 754	3 846 697	121 %
powerwindow	12 601 467	24 599 225	95 %	3 930 880	10 387 998	164 %
binarysearch	369	449	22 %	232	409	76 %
bsort	492 507	961 942	95 %	322 824	1 086 659	237 %
complex_updates	591 526	1 047 923	77 %		💨	
cosf	12 755 728	37 280 642	192 %		💨	
countnegative	13 000	13 000	0 %	21 463	37 353	74 %
cubic	89 339 041	256 701 960	187 %		💨	
deg2rad	7 017 167	11 842 517	69 %		💨	
fft	2 474 043	641 279 936	25 820 %	1 288 291	549 970 763*	42 590 %
filterbank	1 687 548 481	4 175 572 460	147 %		💨	
fir2dim	1 874 513	3 742 975	100 %		💨	
iir	90 327	301 992	234 %		💨	
insertsort	10 080	16 160	60 %	2 804	7 205	157 %
isqrt		🐞		1 821 949	3 322 333*	82 %
jfdctint	8 000	8 000	0 %	4 553	8 396	84 %
lms	76 108 993	144 454 450	90 %		💨	
ludcmp		🐞			💨	
matrix1	98 586	98 586	0 %	37 517	66 523	77 %
md5	71 415 634	258 563 319	262 %	36 390 690	192 082 391	428 %
minver	351 107	1 568 068	347 %		💨	
pm		🧮			💨	
prime		🐞		1 580	4 879	209 %
rad2deg	7 065 136	11 809 717	67 %		💨	
sha		🐞		5 526 979	12 060 259	118 %
st	78 350 471	134 464 808	72 %		💨	
adpcm_dec	13 315	13 786	4 %	5 594	11 852	112 %
adpcm_enc	17 724	19 454	10 %	11 615	20 526	77 %
audiobeam	137 166 947	241 512 652	76 %		💨	
cjpeg_transupp	13 591 503	125 542 054	824 %	8 047 222	129 576 647	1 510 %
cjpeg_wrbmp	275 719	279 767	1 %	280 943	508 787	81 %
dijkstra	192 399 577	32 480 864 150*	16 782 %	121 641 591	16 510 486 058	13 473 %
epic	1 406 490 126	486 510 736 766*	34 490 %		💨	
fmref	256 117 759	795 672 591	211 %		💨	
g723_enc	2 488 842	3 756 308	51 %	1 402 196	4 911 683	250 %
gsm_dec	4 387 842	10 534 654	140 %	4 688 449	25 143 501	436 %
gsm_enc	14 680 073	20 184 547	37 %	9 834 175	31 178 349	217 %
h264_dec	49 621	49 621	0 %	144 429	445 724	209 %
huff_dec	820 259	2 293 420	180 %	529 762	2 104 049	297 %
mpeg2	1 040 665 400	56 804 616 615	5 358 %	535 872 047	41 835 888 109*	7 707 %
ndes	246 594	260 249	6 %	143 728	248 207	73 %
petrinet	8 960	36 728	310 %	2 517	7 648	204 %
statemate	67 364	117 113	74 %	81 270	290 443	257 %
susan		🐞			💨	
cover	2 098	2 758	31 %	58 740	199 374	239 %
duff	2 565	2 628	2 %		🧮	
test3	1 943 674 306	2 029 568 191	4 %	487 414 145	941 044 973	93 %

Compiler & WCET-Analysis Integration. Bernat and Holsti presented a wish list of compiler features that could aid WCET analysis [9]. The list included various features that would aid analysis, such as providing program control flow structure, various properties of the code, and controlling code generation. Many of the essential features are supported in our compiler, with data export through the PML format to aid PLATIN. This, of course, includes the control flow, flow facts, and user annotations. The compiler is missing most feature sets for source code-to-object code mapping and other features, such as the logical effects of code sub-sequences. Other compilers also implement dedicated support for WCET analysis: Li et al. introduced a framework for maintaining flow information during compiler optimizations [33]. Falk et al. introduced the WCET-aware C Compiler (WCC), which can automatically call the aiT WCET analyzer and change the code generation to minimize the WCET [18]. Schommer et al. extend the CompCert certified C compiler with support for AIS annotations [47]. These annotations are then embedded in a dedicated section in the ELF, which the WCET analyzer can consume.

Worst-Case Energy-Consumption Analysis. The original use of the IPET targeted the time resource for real-time systems. Later, Jayaseelan et al. [28] introduced the usability of WCET techniques for the energy resource to yield worst-case energy-consumption estimations, leading to further research on WCEC analysis [28, 38, 42, 53, 54, 55, 56]. With the PLATIN toolkit, we explored system-wide WCEC analysis [54] and the modeling of context-sensitive device states [42]. We consider the PLATIN framework a fundamental basis for our further work in this area for addressing energy-constrained systems.

7 Conclusion

Real-time systems need to prove the absence of deadline misses. To ensure this property, we need schedulability analysis and static WCET analysis of the individual tasks. This paper presented PLATIN, an open-source worst-case analysis tool targeting Patmos, RISC-V, ARM, and AVR processors. The PLATIN toolkit, initially introduced for WCET analysis, has also proven to be suitable for analyzing tasks' worst-case energy consumption to address energy-constrained systems. Extensions to PLATIN include system-wide analyses and expressive annotation support. In our evaluations, we show the multi-target capabilities of PLATIN by providing WCET bounds for two different processors (Patmos and a RISC-V variant) for the TACLe benchmarks. We envision that PLATIN will be extended to other real-time processors, e.g., the FlexPRET processor [58]. PLATIN is available as open-source software, simplifying cooperation between research groups on developing static worst-case analysis tools.

References

1 Saad Ahmed, Bashima Islam, Kasim Sinan Yildirim, Marco Zimmerling, Przemysław Pawełczak, Muhammad Hamad Alizai, Brandon Lucia, Luca Mottola, Jacob Sorber, and Josiah Hester. The internet of batteryless things. *Communications of the ACM*, 67(3):64–73, February 2024. doi:10.1145/3624718.
2 ARM Limited. ARMv6-M Architecture Reference Manual, 2010.
3 ARM Limited. Cortex-M0+ Technical Reference Manual, 2012.
4 ARM Limited. *ARMv7-M Architecture Reference Manual*, 2014.
5 ARM Limited. *ARM Cortex-M4 – Technical Reference Manual*, 2015. Revision: r0p1.
6 Atmel Corporation. *AATmega1284P – 8-bit Microcontroller with 128K Bytes In-System Programmable Flash*, 8059d–avr–11/09 edition, 2009. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/doc8059.pdf.

7 Atmel Corporation. *AVR Instruction Set Manual*, atmel-0856l-avr-instruction-set-manual_other-11/2016 edition, 2016. URL: `https://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf`.

8 C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. OTAWA: An open toolbox for adaptive WCET analysis. In *Proceedings of the 8th International Workshop on Software Technolgies for Embedded and Ubiquitous Systems (SEUS '10)*, pages 35–46, 2010.

9 Guillem Bernat and Niklas Holsti. Compiler support for wcet analysis: a wish list. In *WCET*, pages 65–69, 2003.

10 Adam Betts, Nicholas Merriam, and Guillem Bernat. Hybrid measurement-based wcet analysis at the source level using object-level traces. In *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET '10)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

11 Philipp Degasperi, Stefan Hepp, Wolfgang Puffitsch, and Martin Schoeberl. A method cache for Patmos. In *Proceedings of the 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2014)*, pages 100–108, Reno, Nevada, USA, June 2014. IEEE. `doi:10.1109/ISORC.2014.47`.

12 Eva Dengler, Phillip Raffeck, Simon Schuster, and Peter Wägemann. Fusionclock: Energy-optimal clock-tree reconfigurations for energy-constrained real-time systems. In *Proceedings of the 35th Euromicro Conference on Real-Time Systems (ECRTS '23)*, volume 262, pages 6:1–6:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

13 Christian Dietrich, Martin Hoffmann, and Daniel Lohmann. Cross-kernel control-flow-graph analysis for event-driven real-time systems. In *Proceedings of the 2015 ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '15)*, New York, NY, USA, June 2015. ACM Press. `doi:10.1145/2670529.2754963`.

14 Christian Dietrich, Peter Wägemann, Peter Ulbrich, and Daniel Lohmann. SysWCET: Whole-system response-time analysis for fixed-priority real-time systems. In *Proceedings of the 23nd Real-Time and Embedded Technology and Applications Symposium (RTAS '17)*, pages 37–48, 2017.

15 Espressif Systems. *ESP32-C3 Series Datasheet Ultra-Low-Power SoC with RISC-V Single-Core CPU*, 2024. Version 1.6. URL: `https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf`.

16 H. Falk and P. Lokuciejewski. A compiler framework for the reduction of worst-case execution times. *Real-time Systems*, 46(2):251–300, 2010.

17 Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. TACLeBench: A benchmark collection to support worst-case execution time research. In Martin Schoeberl, editor, *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:10, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.WCET.2016.2`.

18 Heiko Falk, Paul Lokuciejewski, and Henrik Theiling. Design of a wcet-aware c compiler. In *2006 IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia*, pages 121–126. IEEE, 2006.

19 Christian Ferdinand and Reinhold Heckmann. aiT: Worst-case execution time prediction by static program analysis. *Building the Information Society*, 156:377–383, 2004.

20 Freescale Semiconductor, Inc. *KL46 Sub-Family Reference Manual*, 2013.

21 Sebastian Hahn, Michael Jacobs, Nils Hölscher, Kuan-Hsun Chen, Jian-Jia Chen, and Jan Reineke. LLVMTA: An llvm-based wcet analysis tool. In Clément Ballabriga, editor, *Proceedings of the 20th International Workshop on Worst-Case Execution Time Analysis (WCET '22)*, pages 2:1–2:17, 2022.

22 D. Hardy, B. Rouxel, and I. Puaut. The Heptane static worst-case execution time estimation tool. In Jan Reineke, editor, *Proceedings of the 17th International Workshop on Worst-Case Execution Time Analysis (WCET '17)*, volume 57 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.WCET.2017.8.

23 Stefan Hepp, Benedikt Huber, Jens Knoop, Daniel Prokesch, and Peter P. Puschner. The platin tool kit - the T-CREST approach for compiler and WCET integration. In *Proceedings 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach, Austria, October 5-7, 2015*, 2015.

24 N. Holsti and S. Saarinen. Status of the Bound-T WCET tool. In *Proceedings of the 2nd International Workshop on Worst-Case Execution Time Analysis (WCET '02)*, pages 36–41, 2002.

25 B. Huber, D. Prokesch, and P. Puschner. Combined WCET analysis of bitcode and machine code using control-flow relation graphs. In *Proceedings of the 14th Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '13)*, pages 163–172, 2013.

26 Benedikt Huber, Stefan Hepp, and Martin Schoeberl. Scope-based method cache analysis. In *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, pages 73–82, Madrid, Spain, July 2014. doi:10.4230/OASIcs.WCET.2014.73.

27 Infineon Technologies AG. *XMC4500 Microcontroller Series for Industrial Applications — Reference Manual*, 2016. V1.6 2016-07.

28 Ramkumar Jayaseelan, Tulika Mitra, and Xianfeng Li. Estimating the worst-case energy consumption of embedded software. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '06)*, pages 81–90, 2006. doi:10.1109/RTAS.2006.17.

29 Alexander Jordan, Florian Brandner, and Martin Schoeberl. Static analysis of worst-case stack cache behavior. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS 2013)*, pages 55–64, New York, NY, USA, 2013. ACM. doi:10.1145/2516821.2516828.

30 Daniel Kästner, Markus Pister, Simon Wegener, and Christian Ferdinand. TimeWeaver: A tool for hybrid worst-case execution time analysis. In Sebastian Altmeyer, editor, *Proceedings of the 19th International Workshop on Worst-Case Execution Time Analysis (WCET '19)*, volume 72 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:11, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.WCET.2019.1.

31 Raimund Kirner. The wcet analysis tool CalcWcet167. In *Proceedings of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '12)*, pages 158–172, 2012.

32 C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO '04)*, pages 75–86, 2004.

33 Hanbing Li, Isabelle Puaut, and Erven Rohou. Traceability of flow information: Reconciling compiler optimizations and wcet estimation. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, pages 97–106, 2014.

34 Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1):56–67, 2007.

35 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *ACM SIGPLAN Notices*, volume 30(11), pages 88–98. ACM, 1995.

36 B. Lisper. SWEET – a tool for WCET flow analysis. In *Proceedings of the 6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '14)*, pages 482–485. Springer, 2014.

37 Mingsong Lv, Nan Guan, Yi Zhang, Qingxu Deng, Ge Yu, and Jianming Zhang. A survey of WCET analysis of real-time operating systems. In *Proceedings of the 2009 International*

Conference on Embedded Software and Systems (ICESS'09), pages 65–72, 2009. doi:10.1109/ICESS.2009.24.

38 James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. Data dependent energy modeling for worst case energy consumption analysis. In *Proceedings of the 20th Workshop on Software and Compilers for Embedded Systems*, pages 51–59, 2017. doi:10.1145/3078659.3078666.

39 P. Puschner and A. Schedl. Computing maximum task execution times: A graph-based approach. *Real-Time Systems*, 13:67–91, 1997.

40 Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, and Gernot Gebhard. The t-crest approach of compiler and wcet-analysis integration. In *Proceedings of the 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, pages 1–8, 2013.

41 Phillip Raffeck, Christian Eichler, Peter Wägemann, and Wolfgang Schröder-Preikschat. Worst-case energy-consumption analysis by microarchitecture-aware timing analysis for device-driven cyber-physical systems. In *Proceedings of the 19th International Workshop on Worst-Case Execution Time Analysis (WCET '19)*, pages 6:1–6:12, 2019. doi:10.4230/OASIcs.WCET.2019.4.

42 Phillip Raffeck, Johannes Maier, and Peter Wägemann. WoCA: Avoiding intermittent execution in embedded systems by worst-case analyses with device states. In *Proceedings of the 25th ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '24)*, 2024. URL: https://sys.cs.fau.de/publications/2024/raffeck_24_lctes.pdf.

43 RISC-V Foundation. *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA*, December 2019. Document Version 20191213. URL: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf.

44 Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015. doi:10.1016/j.sysarc.2015.04.002.

45 Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, April 2018. doi:10.1007/s11241-018-9300-4.

46 Martin Schoeberl, Wolfgang Puffitsch, Rasmus Ulslev Pedersen, and Benedikt Huber. Worst-case execution time analysis for a Java processor. *Software: Practice and Experience*, 40/6:507–542, 2010. doi:10.1002/spe.968.

47 Bernhard Schommer, Christoph Cullmann, Gernot Gebhard, Xavier Leroy, Michael Schmidt, and Simon Wegener. Embedded program annotations for wcet analysis. In *WCET 2018: 18th International Workshop on Worst-Case Execution Time Analysis*, volume 63. Dagstuhl Publishing, 2018.

48 Simon Schuster, Peter Wägemann, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Proving Real-Time Capability of Generic Operating Systems by System-Aware Timing Analysis. In *Proceedings of the 25th Real-Time and Embedded Technology and Applications Symposium (RTAS '19)*, pages 318–330, Montreal, 2019. ieee.

49 Simon Schuster, Peter Wägemann, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Annotate once – analyze anywhere: Context-aware wcet analysis by user-defined abstractions. In *Proceedings of the 22nd ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '21)*, pages 54–66, 2021. doi:10.1145/3461648.3463847.

50 Volkmar Sieh, Robert Burlacu, Timo Hönig, Heiko Janker, Phillip Raffeck, Peter Wägemann, and Wolfgang Schröder-Preikschat. An end-to-end toolchain: From automated cost modeling to static WCET and WCEC analysis. In *Proceedings of the 20th International Symposium on Real-Time Distributed Computing (ISORC '17)*, pages 1–10, 2017.

51 Volkmar Sieh, Robert Burlacu, Timo Hönig, Heiko Janker, Phillip Raffeck, Peter Wägemann, and Wolfgang Schröder-Preikschat. Combining automated measurement-based cost modeling with static worst-case execution-time and energy-consumption analyses. *IEEE Embedded Systems Letters*, 11(2):38–41, 2019. `doi:10.1109/LES.2018.2868823`.

52 Ingmar Jendrik Stein. *ILP-based path analysis on abstract pipeline state graphs.* epubli, 2010.

53 D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla. Worst-case energy consumption: A new challenge for battery-powered critical devices. *IEEE Transactions on Sustainable Computing*, pages 1–8, 2019. `doi:10.1109/TSUSC.2019.2943142`.

54 Peter Wägemann, Christian Dietrich, Tobias Distler, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS '18)*, volume 106, pages 24:1–24:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.ECRTS.2018.24`.

55 Peter Wägemann, Tobias Distler, Timo Hönig, Heiko Janker, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Worst-case energy consumption analysis for energy-constrained embedded systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS '15)*, pages 105–114, 2015. `doi:10.1109/ECRTS.2015.17`.

56 Simon Wegener, Kris K. Nikov, Jose Nunez-Yanez, and Kerstin Eder. EnergyAnalyzer: Using static wcet analysis techniques to estimate the energy consumption of embedded applications. In Peter Wägemann, editor, *Proceedings of the 21th International Workshop on Worst-Case Execution Time Analysis (WCET '23)*, volume 114 of *Open Access Series in Informatics (OASIcs)*, pages 9:1–9:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.WCET.2023.9`.

57 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (ACM TECS)*, 7(3):1–53, 2008.

58 Michael Zimmer, David Broman, Chris Shaver, and Edward A Lee. FlexPRET: A processor platform for mixed-criticality systems. In *Proceedings of the 19th Real-Time and Embedded Technology and Applications Symposium (RTAS '14)*, pages 101–110, 2014.

Invited Paper: Assessing Unchecked Factors for Certification: An Experimental Approach for GPU Cache Parameters

Cédric Cazanove
ONERA, Toulouse, France

Benjamin Lesage
ONERA, Toulouse, France

Frédéric Boniol
ONERA, Toulouse, France

Jérôme Ermont
IRIT – INP – ENSEEIHT, Toulouse, France

Abstract

The certification objectives for airborne electronic hardware defined in AMC20-152A [9] and in AMC20-193 [8] capture some of the activities required for an applicant to embed a hardware platform in a safety-critical avionic system. For COTS (Commercially available Off-The-Shelf) platforms in particular, these objectives require applicants to identify functions, configuration settings, and resources present on the platform, and assess their use by the system. AMC20-152A however recognizes that documentation regarding the behavior of a COTS may be incomplete.

There is thus a strong push for applicants to the certification of a COTS to demonstrate their mastery of the platform, to highlight relevant *factors* (functions, settings, resources, etc.), and their use in their system. We outline in the following a standard approach to the exploration of unchecked factors of a platform, considering existing approaches in the literature, to build such a mastery. Our approach incrementally incorporates and validates knowledge of various factors by including them in micro-simulations compared to experimental ground truth.

2012 ACM Subject Classification Computer systems organization → Embedded hardware; Computer systems organization → System on a chip; Computer systems organization → Real-time system architecture; Computer systems organization → Multicore architectures

Keywords and phrases GPU, benchmarks, simulation, certification

Digital Object Identifier 10.4230/OASIcs.WCET.2024.3

Funding The work presented in this paper is part of the PHYLOG 2 project supported by the Directorate General of Civil Aviation (DGAC). It is funded by the French government through the France Relance program, based on the funding from the European Union through the NextGenerationEU program.

1 Introduction

Recent advances in machine learning allow the development of complex embedded functions which show a lot of potential for future algorithms in the avionic domain like ATTOL (Autonomous Taxi, Take-Off, and Landing). These algorithms have increasingly large computation requirements. This led to the emergence of a new generation of multi-core, hybrid architectures to offer greater computing power through increased compute parallelism. Nevertheless, hybrid platforms need to go through the same stringent certification process as other platforms before they are embedded in an avionic system. The European Union Aviation Safety Agency (EASA) defines Acceptable Means of Compliance (AMC) to guide the certification process. Each AMC defines a number of certification objectives the applicant must satisfy, but it does not prescribe actual tools and methods to that end. Architecture mastery, as an overarching certification requirement, needs to find an experimental method to assess and understand architectural factors.

Prior work has highlighted specific, sometimes previously unknown, factors on COTS platforms, and proposed methods to understand or capture their behaviour. Experimental results, and observations collected from the platform, often provide a ground truth or reference to assess factors. Mastering the complexity of COTS platforms may require dealing with multiple, complex interactions between these factors. As an example, the objective of computing the Worst-Case Execution Time (WCET) of a task on a COTS platform requires a mastery of all factors of said platform. We instead focus on understanding individual factors as a stepping stone towards more general models (WCET, simulator, etc.).

Contributions

This paper presents an experimental approach to reinforce knowledge and mastery of a COTS platform, more specifically to understand platform factors relevant to certification, i.e. used functions, configuration settings, shared resources, etc. We follow a widespread approach in practice: comparing expected behaviours (from the documentation or literature) to ones observed on the platform, for each factor or for increasingly large combinations thereof. The rationale is to provide supporting evidence for certification objectives related to these factors.

Where micro-benchmarks aim to support the characterisation of a platform by exercising specific factors, we propose the use of micro-simulators as reference models to compare against. The goal is to catalogue factors and related models, as micro-simulators and micro-benchmarks, to bootstrap the mastery of a new COTS platform for applicants. We focus the following on discovering the factors related to the cache hierarchy of an embedded GPU.

Organisation

The contribution is organised as follows. We first outline in Section 2 the certification objectives for COTS as per AMC20-152A and AMC20-193. Section 3 presents the overall approach to build a mastery of a complex COTS platform. The Jetson AGX Xavier is introduced in Section 4, as a supporting example of embedded COTS platform, alongside some of its factors up for validation as support for our evaluation (Section 6). The means of evidence for said factors are described in Section 5. We finally conclude by discussing related and future work (respectively in Sections 7 and 8).

2 Certification

As discussed in the introduction, certification is a key issue for safety-critical avionic systems. Processors involving several general-purpose cores and accelerators must undergo a stringent certification process before they are deployed. The European Union Aviation Safety Agency (EASA) and Federal Aviation Administration (FAA) respectively define Acceptable Means of Compliance (AMC) and Advisory Circulars (AC), setting down objectives that applicants to the certification process must satisfy. The joint A(M)C AMC20-193 and AMC20-152A in particular define objectives for the respective certification of hardware platforms and multi-core processors.

Overview of the AMC20-193. The AMC20-193 addresses the issue of multi-core platforms. It defines a Multi-Core Processor as a device with two or more activated processing cores, with a core being a device that executes software. It mainly focuses on both temporal and functional interference, that is, situations when two or more cores compete for shared resources. Interference may cause additional delays due to the arbitration of accesses to the resource or control flow variations due to external modifications of a shared variable. It may thus cause a loss of deterministic behaviour for the application. The impact of interference channels on applications in the system should be assessed. To address this issue, the AMC20-193 defines three main requirements: 1) the identification of the hardware and software resources of the processor, 2) the identification of their configuration settings (e.g., the L1 and L2 cache sizes, their replacement policy, if they are partitioned or not, etc.), and 3) the worse-case impact of interference in these resources.

Overview of the AMC20-152A. The AMC20-152A is complementary to the AMC20-193. It discusses the certification of complex COTS platforms. The clarifications proposed by AMC20-152A are important, as devices, especially COTS, become more complex and integrate in a single chip more functions and resources (such as GPUs) than older ones. The key objectives to certify COTS items according to AMC20-152A are 1) identifying the used function, 2) assessing the correct use of the COTS item, and 3) assessing the correct behaviour of the COTS item if used outside vendor specifications. AMC20-152A focuses on the risks inherent to the use of COTS, and that of incomplete or incorrect documentation. The issue when embedding COTS is the use of undefined or undocumented configurations, that may lead to unexpected behaviours.

From AMC20-193 and AMC20-152A we identified 4 activities for GPU platforms. **Activity 1:** It is necessary to master *complex* core architectures, that is, to identify all the configurations settings (e.g., resource capacities, arbitration policies, etc.) that are not clearly established and to assess them with certification evidence. To that end, we believe that stressing benchmarks would be needed in addition to documentation reviews. **Activity 2:** An assessment should be performed for each device of the platform. In particular, one should consider how the device is configured and accessed through hardware and software means, how it interacts with the rest of the system, and whether or not existing analysis techniques and tools apply. **Activity 3:** The utilisation of a COTS must be within the limit of the device manufacturer specification. This means that we need a specification of the COTS and its limits to check the compliance of usage. **Activity 4:** It is mandatory to qualify the COTS behaviour and all micro-code (e.g., the scheduling policies).

In this article, we focus on activity 1: architecture mastery for GPU platforms.

3 Mastering COTS platforms

Certification requirements, as discussed in Section 2, define a number of certification objectives for embedding a platform. Those revolve around the mastery of the platform, especially complex core architecture for COTS platforms. Industry practice and related work tend to rely on a similar approach, outlined in Figure 1, of iterating over identified factors to compare observed and expected behaviours. Discrepancies lead to refinements of the platform description and may guide further exploration. We consider the following activities:

- **Identify relevant platform factors** to capture functions, resources, and configuration settings available on the platform. The list may not be exhaustive and initially stems from vendor-provided documentation. Additional knowledge from the literature or the applicant's experience may also identify common factors to be assessed.
- **Identify unchecked factors** amongst the ones captured by the platform description. Each factor which has not been covered, or for which evidence is insufficient should be included. Even if documentation is available on the expected behaviour of a factor, it may be insufficient and benefit from further investigation.

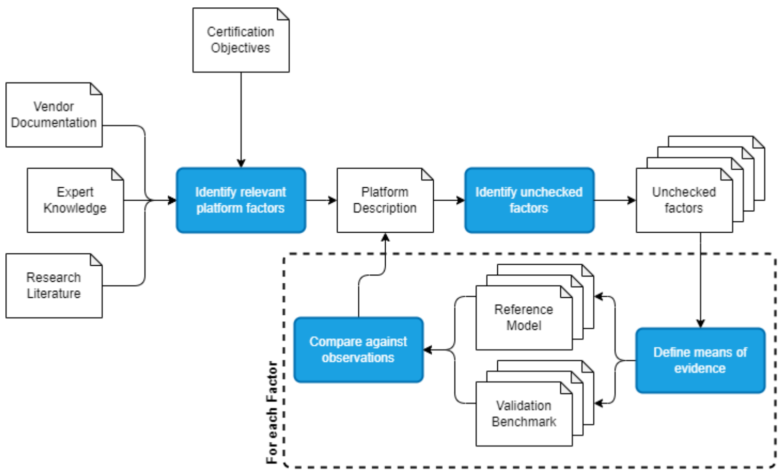


Figure 1 Overview of the process for mastering a complex COTS platform.

- **Define means of evidence** for each factor to provide evidence for certification, i.e. observe its behaviour, and capture expectations. This relies on the ability first to exercise the factor and capture relevant metrics, and second to model said factor under various configurations contending as capturing its behaviour. Models could take as an example the form of analytical models, simulations, etc.
- **Comparing against observations** should help discriminate the behaviour of the factor between possible candidates. It also provides evidence for certification that the factor is well understood. Discrepancies, however minor, may highlight unknown factors, or configurations which have yet to be described in the platform.

Our instantiation of the process relies in particular on the use of dedicated micro-simulators as a mean of collecting evidence of platform knowledge. Where possible, independent simulators and benchmarks should be defined to keep the complexity of the platform description manageable and to ease the argumentation for certification. The method will need a catalogue of benchmarks to validate and discover factors on a platform. Where dependencies exist, each simulator is built to include the impact of dependent factors as they are mastered by the applicant.

We illustrate the process in the remainder of the paper, focusing on the capturing parameters of the L1 cache of the Jetson AGX Xavier GPU, from platform description (Section 4), through evidence means (Section 5), to validation through comparison (Section 6).

4 Jetson AGX Xavier Platform

The Jetson AGX Xavier features the Xavier System-on-Chip (SoC). The Xavier SoC embeds a Volta GPU, an 8-core Carmel CPU complex, and dedicated accelerators for video and audio processing applications. Figure 2 presents a block diagram of the SoC as described in the platform Technical Reference Manual (TRM). This section presents the execution model of the Volta GPU to identify notable resources. The TRM provides little information on the Volta GPU architecture and related factors. We rely on supplemental vendor documentation and literature [10, 17, 3, 20, 16, 15, 7, 12], regarding the Volta architecture and its execution model in general. To highlight our approach we consider factors related to the GPU private L1 Data caches.

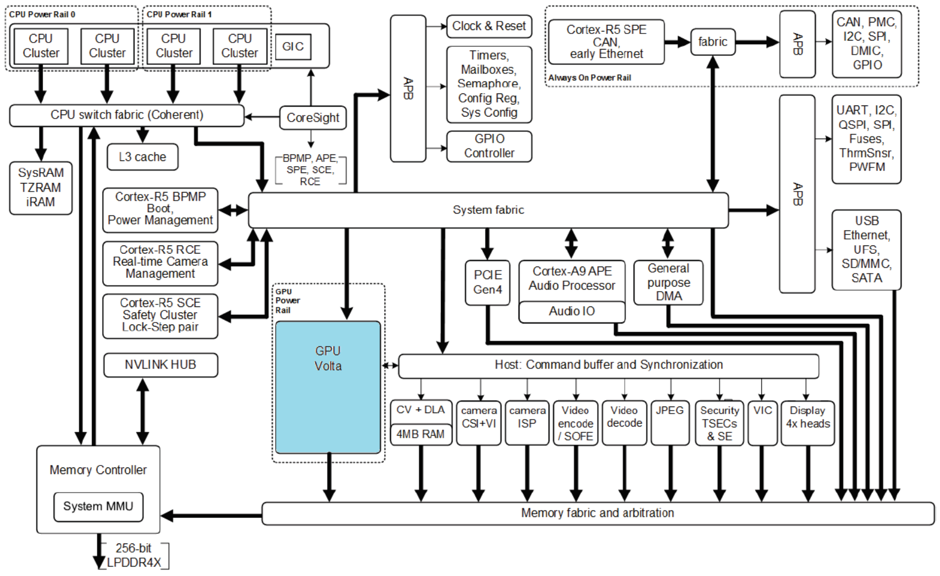

CPU Power Rail 0
CPU Power Rail 1
CPU Cluster
CPU Cluster
CPU Cluster
CPU Cluster
GIC
CPU switch fabric (Coherent)
CoreSight
BPMP, APE, SPE, SCE, RCE
L3 cache
SysRAM TZRAM iRAM
Cortex-R5 BPMP Boot, Power Management
Cortex-R5 RCE Real-time Camera Management
Cortex-R5 SCE Safety Cluster Lock-Step pair
NVLINK HUB
Memory Controller
System MMU
256-bit LPDDR4X
APB
Clock & Reset
Timers, Mailboxes, Semaphore, Config Reg, Sys Config
GPIO Controller
Cortex-R5 SPE CAN, early Ethernet
Always On Power Rail
fabric
APB
CAN, PMC, I2C, SPI, DMIC, GPIO
APB
UART, I2C, QSPI, SPI, Fuses, ThrmSnsr, PWFM
USB Ethernet, UFS, SD/MMC, SATA
System fabric
GPU Power Rail
GPU Volta
PCIE Gen4
Cortex-A9 APE Audio Processor
Audio IO
General purpose DMA
Host: Command buffer and Synchronization
CV + DLA
4MB RAM
camera CSI+VI
camera ISP
Video encode / SOFE
Video decode
JPEG
Security TSECs & SE
VIC
Display 4x heads
Memory fabric and arbitration

Figure 2 Overview of the Xavier SoC as described in vendor documentation [13].

4.1 Volta Execution Model

A user defines computation *kernels* for execution on the GPU, and other GPU operations through the CUDA Runtime or related high-level libraries, on top of the Runtime, such as cuDNN. We focus in the following on the definition and execution of computation kernels. A kernel call is shaped by the computation *grid* definition, i.e. the number of thread *blocks* and their shape. All thread blocks in a call are equally shaped [14]. The size of the kernel, the total number of threads invoked on a call, is thus the product of the number of threads per block and the number of blocks. Each thread is given a unique identifier to address different data segments. It is composed of its block index in the grid, and its thread index in the block. Both are accessible during kernel definition.

The block scheduler hierarchically dispatches each kernel block to a Graphics Processing Cluster (GPC), then picks a Thread Processing Cluster (TPC) on the GPC, and a Streaming Multiprocessor (SM) on the TPC. Dispatch depends on the block's resource requirements and the SM occupancy. The block will remain on its allocated SM until its threads are complete (unless a rescheduling occurs on preemption). The block's threads are further dispatched onto the SIMT Units (SU) which compose the SM. Each SU holds a register file and functional units to execute threads.

The Jetson AGX Xavier Volta GPU is composed of 8 Streaming Multiprocessors (SM), depicted in Figure 3, each with its own private L1 instruction and data caches (respectively ICache and DCache). SMs on the Jetson Volta GPU are partitioned into 4 TPC, each composed of 2 SMs. The GPU features a single GPC. All SMs share a unified L2 cache and 2 levels of TLB. Each SM is further divided into 4 SU. The register file on a SU, or L0 data cache, holds the context of multiple threads. Each SM is composed of 64 CUDA cores and 8 Tensor cores, split evenly across the SU. Tensor cores are a class of Deep Learning Accelerators supporting multiply-add operations on matrices, as instructed by the user (or a library). Restrictions on the number of registers in each SM imply that not all threads in a kernel call can execute concurrently. Blocks can hold a maximum of 1024 threads, at most 64 warps can reside at once on a single SM.

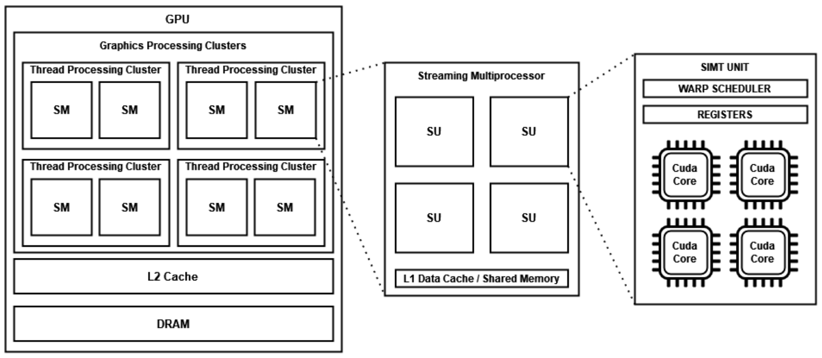


Figure 3 Overview of the Volta GPU architecture on the Jetson AGX Xavier.

SU follow the Single Instruction Multiple Thread (SIMT) execution model: the same instruction is executed across multiple threads at the same time, possibly addressing different data[1]. Threads are scheduled and executed in groups of 32, called a *warp*, such that each thread belongs to a single warp across its lifetime. Warp size and the allocation of threads in warps are not user-controlled parameters but platform-specifics. The warp scheduler on each SU can schedule up to 1 warp every cycle. Instructions from two distinct warps may coexist on the same core provided they rely on different functional units, e.g. a long-running load due to a cache miss and an integer addition. The combined register files on each SM can accommodate up to 64 warps to ensure the warp schedulers can maximise core occupancy.

4.2 DCache factors

The GPU embedded in the Jetson AGX Xavier relies on the Volta GPU architecture, and it is referenced as the GV10B model. While the GPU factors may be well documented at the architecture- or model-level, it is important to validate the documented factors as discussed in Section 2. There are furthermore factors in the GPU that are not documented and that an applicant needs to master to certify the platform, such as scheduling policies for the kernel components or the cache policy. We focus on the following factors related to the DCache private to each SM. Caches act as small buffers between a core and the comparatively slower SoC memory. They hold a copy of the most recently used data (and thus most likely to be reused) by a core. Table 1 presents a collection of cache-relevant factors, and their configuration as documented for the Volta GPU on the Jetson AGX Xavier. Bold factors are the ones considered in the subsequent evaluation.

5 Evidence means

To assess factors related to the DCache on the Volta GPU, we measure the behaviour of purpose-built characterisation benchmarks on the platform. We rely on a micro-simulator for the cache as our reference model. The micro-simulator allows us to explore different factors and their candidate configurations to find the best match for the observed behaviour.

[1] Note that the Volta GPU architecture introduces thread divergence, where threads may have different program counters. The scope and impact of thread divergence on the SIMT model are unclear.

Table 1 Selection of platform factors related to the private L1 DCache on the Volta GPU.

Factor	Description	Configuration
Line size	Number of (aligned) Bytes loaded upon a cache miss, i.e. when the accessed data is absent from the cache. Subsequent access to the same cache line may thus be served by the cache (hit).	32B [10]
Size	Number of cache lines held by the cache. Accesses to cache lines beyond the cache size will cause the evictions of older lines to make room for more recent ones.	128KB [11, 10]
Associativity	Number of candidates positions for a line in the cache. A cache line can only be stored in a limited number of places, based on its address. Simplifies the history required to maintain the ordering of cache lines.	1024 [10]
Replacement policy	Policy for selecting a position amongst the candidates based on the cache history upon inserting a line in the cache (miss). The policy also defines how the cache history is updated upon subsequent accesses (hit).	Not LRU [10]
Scratchpad allocation	Cache space allocated as a software-managed scratchpad for data shared between threads in a block. The accessible cache space is reduced as a result of scratchpad allocation.	8KB slices [12]
Prefetch policy	Policy allowing cache lines to be loaded in anticipation of future memory accesses. Which cache lines are prefetched depends on prior cache miss patterns.	N.A.
Coalescing policy	Policy for batching cache accesses caused by simultaneous thread execution into one or more cache accesses. Reduces the number of cache accesses required to serve a single warp.	N.A.
Write policies	Policies for dealing with data writes in the cache. Includes whether writes are blocking, cause insertion in the cache, and propagate all the way to the main memory.	N.A.
Inclusion policy	Captures how accesses, insertions, and evictions to (and from) the lower cache levels, e.g. the L2, affect the DCache.	N.A.
Coherency protocol	Policy capturing how concurrent accesses to the same data affect copies held in different private and shared, e.g. evictions, updates, etc.	N.A.

5.1 Experimental setup

We need to isolate the contribution of specific DCache factors on the GPU from other factors. To that end, we measure the L1 Cache Hit Rate during the execution of a single kernel running in isolation. The Cache Hit Rate captures the portion of accesses served by the cache over the total number of accesses performed by an application. It is thus mostly independent of the effects of the L2 factors. Focusing on a single-block, single-warp benchmark isolates observations from the highest level dispatch and scheduling policies (kernel, block and warp). It is still dependent on the scheduling of memory accesses by the SU, that is how threads within a warp are dispatched to the functional units.

On the Jetson AGX Xavier we use the Jetpack 5.10 to set a Linux environment of the platform. Concerning the Cuda version, we use Cuda 11.4 to develop our benchmark. Measurements are collected using Nvidia Nsight Compute (ncu).

5.2 Benchmark

It is necessary to exercise the DCache in a controlled fashion to understand its factors. CPU-based approaches [2] rely on configurable micro-benchmarks to generate a stream of memory access in which cache locality is known or behaviour is easy to predict. GPUs introduce additional instruction-level parallelism over CPU. We thus propose the following benchmark.

In a GPU several threads may execute concurrently and cause memory accesses. To discover the memory hierarchy of the GPU we introduce two notions: Step and Stride. Stride defines the distance in bytes between concurrent accesses of two consecutive threads; Step establishes the distance between two consecutive accesses from the same thread. All threads read from the same array. This allows us to define a simple read benchmark, per Algorithm 2, taking into account the instruction-level parallelism for GPU. By varying Step and Stride, we can control the way the threads of a kernel access the memory and the cache, and thus assess various parameters of the cache hierarchy. The implementation relies on an index-chasing pattern [2], with the pattern initialized as per Algorithm 1.

Algorithm 1 Array Initialization with Index Chasing.

```
1: Input: $n$                                   ▷ Size of the array
2: Input: $Step$                                ▷ Value of the Step
3: Output: $A$                                  ▷ Initialized array of size $n$
4: $A \leftarrow$ new array of size $n$
5: for $i \leftarrow 0$ to $n - 1$ do
6:     $A[i] \leftarrow (i + Step) \bmod n$
7: end for
```

Algorithm 2 Pseudo-Code of the Kernel of the read benchmark.

```
1: i ← threadIndex × STRIDES
2: for op ← 0 to NB_OP − 1 do
3:     i ← ptr[i]
4: end for
```

5.3 Simulation

As a reference model for the platform, we use a micro-simulator for the cache. The simulator focuses on implementing a single cache layer. It supports several policies and configuration points to explore possible factor configurations. We also build a surrogate model of our benchmark to mimic the sequence of memory accesses it generates on the platform under specific strides and step values. The generated sequence is fed into the simulator to estimate the L1 Cache Hit Rate.

This allows us to compare the output of the simulator and the benchmark. We compare our model of the platform under varied factor configurations with the measurements made on the real platform. The comparison allows to identify which configurations better represent the platform, and master undocumented factors as identified in Section 4. Where variations occur, we can refine the simulation configuration, or identify policies which better match the observed behaviours.

6 Evaluation

6.1 Assessing the DCache size

Configuration

We first aim to assess the DCache size. First, we need to isolate our experiment from the cache replacement policy effects. The Stride and Step parameters are set to prevent threads from reusing cache lines loaded by concurrent threads, within and between iterations. We thus set the Stride to the DCache Line Size (32B), with a 32B Line Size Step for each of the 32 threads in a warp, Step = 1024B. The accessed array is kept within the L2 cache size to limit the impact of inclusion and coherency policies. We use a read benchmark to remove the impact of write policies. As discussed in Section 5 focusing on a single-block single-warp kernel, we can mostly assume the memory accesses are issued in the order defined in the benchmark, from a single SU, so we will not have interference from an other SU. Moreover having a single-block single-warp kernel allows the focus on only one L1 Cache because the block will only run on one SM. We repeat the experiment with the kernel each time iterating over an increasingly bigger array. The DCache hit rate should suddenly drop whenever the array exceeds the cache capacity.

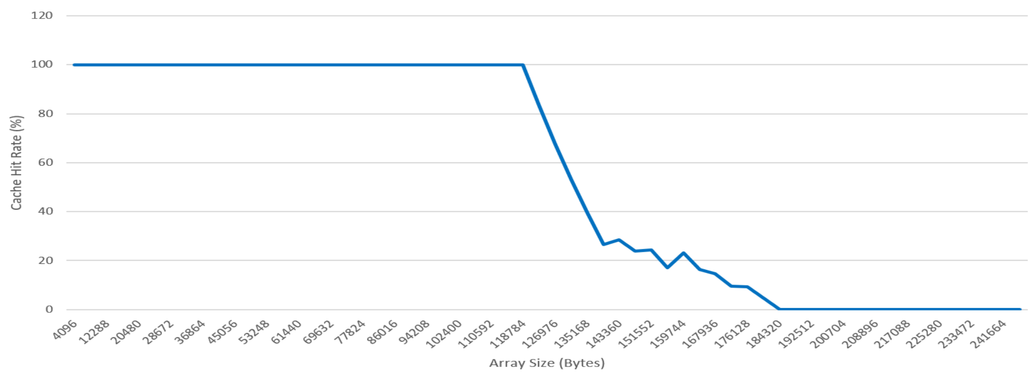


Figure 4 L1 Cache Hit Rate of a Read Benchmark with Chasing Index.

Observations

Figure 4 presents the observed DCache hit rates (y-axis) over an increasing array size (x-axis). We observe that the drop-off point does not correspond to the documented L1 size of 128KB, instead stalling when the array reaches a size of 116KB.

To validate our measurements, we considered a different collection tool (NVIDIA `nsys`). `nsys` does not support collecting Cache Hit Rates, but it did capture a drop-off point in execution time around 120KB. We think the NVIDIA instrumentation tools alocate space in DCache to collect measurements effectively reducing its capacity. However the 12 KB loss in cache size does not correspond to a valid scratchpad allocation, which operates in 8KB increment per the documentation.

6.2 Assessing of the DCache Associativity

Configuration

We repeat the same configuration as in the previous section. Beyond the drop-off point, the rate and shape of the hit rate decrease should provide information on the replacement policy and the associativity. To assess the DCache associativity, we consider different simulation configurations, namely varying the replacement policy and associativity to match configurations from the literature. Our simulator is configured using a 116KB DCache size as previously observed.

Observations

Figure 5 proposes a comparison of the observed hit rates (solid blue line) and simulated ones (dashed lines) while increasing the array size (x-axis). For simulations, we consider different configurations of associativity using the LRU or Random replacement policies. Under the current memory access pattern, the LRU, PLRU and FIFO replacement policies exhibit the same behaviour and we omit redundant results.

The random replacement policy results in an asymptotic behaviour with the hit rate gradually decreasing as the array size increases. This does not correspond to the relatively sharper observed drop. The 4-way LRU cache perfectly matches the observed drop initially, until the two diverge around 25% hit rate. The tail of the observed behaviour matches the slope of the 2-way LRU cache. This possibly hints at the use of adaptive replacement policies [18], a complex mechanism which may fit high-performance caches.

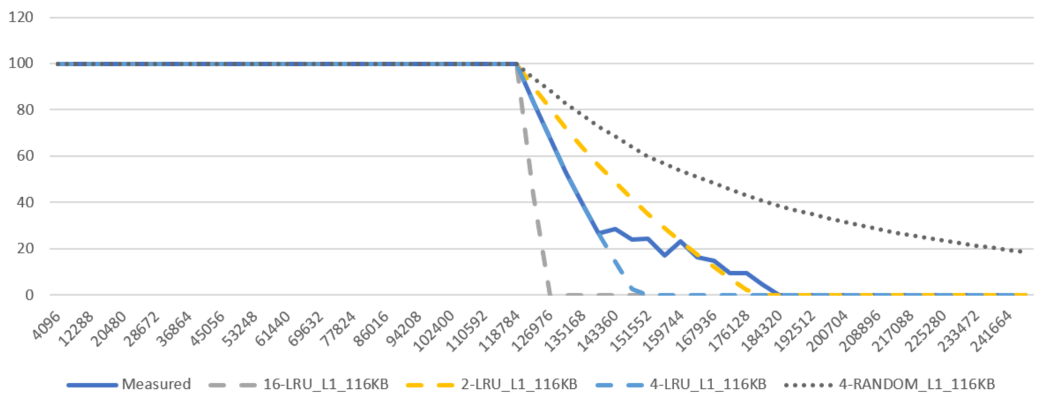


Figure 5 L1 Cache Hit Rate Simulation for several configurations.

7 Related work

With the increasing need and integration of GPUs in embedded systems, recent works have focused on understanding GPUs. There is a need to master said GPU to deploy Machine Learning in certified critical systems. NVIDIA represents today one of the leading GPU vendors in the market, including GPU for embedded systems. Hence there is consequent effort on mastering GPUs from NVIDIA such as the JETSON AGX XAVIER. However, one major problem persists: the GPU mainly works as a black box. Of course, some documented configurations are provided by the vendor but it is insufficient to fulfil certification objectives for critical systems. But thanks to prior works, configurations emerged from reverse-engineering or stressing benchmarks. Thanks to micro-benchmarking, [10, 6, 17, 3, 20, 16, 19, 4] discovered a number of factors of GPU platforms, especially regarding scheduling policies at different levels and related latencies. All of those works represent factors that applicants need to master for certification, as well as related means of evidence.

AMC20-193 recognises resource partitioning, as a suitable tool to alleviate specific objectives. In 2023, work by [6] pointed out the possibility of partitioning hardware resources of the most recent NVIDIA GPUs. The authors discovered the capability through public patent information as the feature is not documented by the vendor, its use poses additional challenges for certification with regards to AMC20-152A.

Other approaches aim to provide a model of the GPU as a whole, especially to predict its timing behaviour. This is crucial to WCET computation and related certification objectives. To the best of our knowledge, PasTiS [1] is one of the few efforts to build a GPU model suitable for WCET computation. While our approach does not aim to provide a model suitable for timing analysis, the platform description and knowledge derived from means of evidence do feed into the definition of such a model. Similarly, micro-simulators do not aim to replace full-fledged solutions such as GPGPUSim [5]. But they provide a quick reference.

8 Conclusion and future work

This paper presents an experimental approach to address the requirement for certification of the COTS platform according to AMC20-152A and AMC20-193. The approach, based on existing practices, aims to incrementally builds an accurate description of a platform. We suggest the use of micro-benchmarks to isolate each factor, and purpose-built micro-simulators to assess the behaviour of said factors. The approach thus benefits from existing work to capture factors to consider in a GPU, and the related means of generating evidence for certification. We illustrated the approach by considering the private cache level in an embedded COTS GPU. Our evaluation highlighted the applicability of existing work, for CPU-related factors, to the GPU characterisation.

As part of future work, we aim to catalogue factors identified in the literature, and reference the related micro-benchmarks and reference models, with the goal of easing the certification of existing or new COTS platforms. The work will include understanding how to collect observations from micro-benchmarks without impacting the platform behaviour or its configuration, or at the very least mastering the impact of the tools on the observations. We will continue the work on qualifying cache-related factors for embedded GPU, as well as factors related to instruction- and kernel-level parallelism.

References

1 Michaël Adalbert, Thomas Carle, and Christine Rochange. PasTiS: building an NVIDIA Pascal GPU simulator for embedded AI applications. In *11th European Congress on Embedded Real-Time Systems (ERTS 2022)*, 2022. URL: `https://ut3-toulouseinp.hal.science/hal-03684680`.

2 Alif Ahmed and Kevin Skadron. Hopscotch: a micro-benchmark suite for memory performance evaluation. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, pages 167–172, 2019. `doi:10.1145/3357526.3357574`.

3 Tanya Amert. *Enabling Real-Time Certification of Autonomous Driving Applications*. PhD thesis, The University of North Carolina at Chapel Hill, 2021. AAI28650154.

4 Tanya Amert, Zelin Tong, Sergey Voronov, Joshua Bakita, F. Donelson Smith, and James H. Anderson. TimeWall: Enabling Time Partitioning for Real-Time Multicore+Accelerator Platforms. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 455–468, 2021. `doi:10.1109/RTSS52674.2021.00048`.

5 Ali Bakhoda, George Yuan, Wilson Fung, Henry Wong, and Tor Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174, 2009. doi:10.1109/ISPASS.2009.4919648.

6 Joshua Bakita and James H. Anderson. Hardware Compute Partitioning on NVIDIA GPUs. *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 54–66, 2023. URL: https://api.semanticscholar.org/CorpusID:259235797.

7 Alejandro J. Calderón, Leonidas Kosmidis, Carlos F. Nicolás, Francisco J. Cazorla, and Peio Onaindia. Gmai: Understanding and exploiting the internals of gpu resource allocation in critical systems. *ACM Trans. Embed. Comput. Syst.*, 19(5), September 2020. doi:10.1145/3391896.

8 EASA. AMC (Acceptable Means of Compliance) 20-193 on the use of multi-core processors (MCPs), 2020.

9 EASA. AMC (Acceptable Means of Compliance) 20-152A Development Assurance for Airborne Electronic Hardware (AEH), 2021.

10 Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele Paolo Scarpazza. Dissecting the NVIDIA volta GPU architecture via microbenchmarking. *CoRR*, abs/1804.06826, 2018. arXiv:1804.06826.

11 NVIDIA. Jetson AGX xavier 32 GB specs. https://www.techpowerup.com/gpu-specs/jetson-agx-xavier-32-gb.c4088.

12 NVIDIA. Volta tuning guide. https://docs.nvidia.com/cuda/volta-tuning-guide/.

13 NVIDIA. *NVIDIA Xavier Series System-on-Chip: Technical Reference Manual.* NVIDIA Corporation, Santa Clara, California, April 2020.

14 NVIDIA. Nvidia heterogeneous computing on cuda platforms. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#heterogeneous-computing, 2022. Accessed: 2022-11.

15 Ignacio Sañudo Olmedo, Nicola Capodieci, Jorge Luis Martinez, Andrea Marongiu, and Marko Bertogna. Dissecting the CUDA scheduling hierarchy: a Performance and Predictability Perspective. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 213–225, 2020. doi:10.1109/RTAS48715.2020.000-5.

16 Nathan Otterness, Ming Yang, Tanya Amert, James H. Anderson, and F. D. Smith. Inferring the scheduling policies of an embedded CUDA GPU. In *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT)*, 2017.

17 Nathan Michael Otterness. *Developing Real-Time GPU-Sharing Platforms for Artificial-Intelligence Applications.* PhD thesis, The University of North Carolina at Chapel Hill, 2022.

18 Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, 2007. doi:10.1145/1250662.1250709.

19 Tyler Yandrofski, Jingyuan Chen, Nathan Otterness, James H. Anderson, and F. Donelson Smith. Making Powerful Enemies on NVIDIA GPUs. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 383–395, 2022. doi:10.1109/RTSS55097.2022.00040.

20 Ming Yang, Nathan Otterness, Tanya Amert, Joshua Bakita, James H. Anderson, and F. Donelson Smith. Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems. In *ECRTS*, 2018.

Invited Paper: Worst-Case Execution Time Analysis of Lingua Franca Applications

Martin Schoeberl
Technical University of Denmark, Lyngby, Denmark

Ehsan Khodadad
Technical University of Denmark, Lyngby, Denmark

Shaokai Lin
University of California, Berkeley, CA, USA

Emad Jacob Maroun
Technical University of Denmark, Lyngby, Denmark

Luca Pezzarossa
Technical University of Denmark, Lyngby, Denmark

Edward A. Lee
University of California, Berkeley, CA, USA

Abstract

Real-time systems need to prove that all deadlines will be met. To enable this proof, the full stack of the system must be analyzable, and the right tools must be available. This includes the processor (execution platform), the runtime system, the compiler, and the WCET analysis tool.

This paper presents a combination of the time-predictable processor Patmos, the coordination language Lingua Franca, and the WCET analysis tool Platin. We show how carefully written Lingua Franca programs enable static WCET analysis to build safety-critical applications.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Automated static analysis

Keywords and phrases worst-case execution time, coordination language, real-time systems, lingua franca

Digital Object Identifier 10.4230/OASIcs.WCET.2024.4

Supplementary Material
Software (Lingua Franca Source Code): `https://github.com/lf-lang/`
Software (T-CREST Source Code): `https://github.com/t-crest/`
Software (Experiment Instructions): `https://github.com/lf-lang/lf-patmos-template/`
archived at `swh:1:dir:65a90fb54c7c1753a6fc062f1e19c41c0ca5dadd`

Funding *Emad Jacob Maroun*: Huawei Technologies Sweden AB, Agreement Number TC20220819029.

1 Introduction

In safety-critical applications, the reliability of a system is of utmost importance to ensure no catastrophic failures happen. Real-time systems must guarantee that they respond to events within a given deadline. Designing such a system includes analyzing all its tasks and ensuring its worst-case execution time (WCET) and that the resulting schedule will meet its deadlines. However, it is not enough that individual tasks meet their deadlines. We must also ensure that all the tasks always meet their deadlines, regardless of the runtime conditions or other tasks executing in parallel. As such, schedulability analysis must consider not only the individual tasks but also their runtime environment and execution platform.

A reliable real-time system must be built from a stack of analyzable components. It starts with a platform based on a time-predictable processor. Then comes the execution environment, which includes a real-time operating system or real-time runtime environment that can allocate resources as needed. Next comes programming language and compiler that must support the writing of analyzable code. Lastly comes the analysis tool, which should be able to account for all the components to produce safe WCET bounds, i.e., bounds that are guaranteed not to be lower than the actual WCET.

This paper presents a complete real-time system and toolchain for safety-critical applications. It is based on the time-predictable RISC processor Patmos [33]. Instead of a complicated real-time operating system, we use the Lingua Franca (LF) reactor language as our runtime environment [24]. LF allows us to write individual *reactions* in the C language, which are automatically compiled with the LF runtime using the Patmos compiler. The LF runtime handles the provisioning of resources to each reaction and schedules them when needed. Using the PLATIN WCET analysis tool, we can analyze the WCET of individual reactions and feed it to a quasi-static schedule generator that ensures reactions are scheduled such that they will never miss a deadline. The quasi-static schedule generator and its underlying virtual machine, PretVM, have recently been introduced to LF to enable fine-grained timing analysis of the LF runtime.

The individual components presented in this paper are not new; they have been presented in other papers. However, this paper's contribution is the presentation of the combination of those components to provide a complete time-predictable execution environment. This is intended as a step towards building correct-by-construction real-time systems.

Lingua Franca, Patmos, and PLATIN are open-source. The links to the GitHub repositories are given on the title page. To reproduce the evaluation, consult the `README` file in the following repository: `https://github.com/lf-lang/lf-patmos-template/`.

The rest of this paper is organized into the following sections. Section 2 provides the background on Lingua Franca, the Patmos time-predictable processor, the compilation pipeline, and the PLATIN WCET analysis tool. Section 3 discusses the most important aspects of analyzable code in LF applications, as well as the benefits and issues of LF for WCET analysis. Section 4 presents the experimental evaluation results of the proposed approach. Section 5 discusses related work. Section 6 concludes the paper.

2 Background

This paper combines several technologies to build a complete real-time platform: the coordination language Lingua Franca [22], the Patmos processor [33] and compiler, and the WCET analysis tool PLATIN [25].

2.1 Lingua Franca

LF is a coordination language and framework based on concurrent actors called Reactors. LF adds deterministic reactive concurrency to target languages. Currently, it supports C, C++, Python, TypeScript, and Rust. The generated code can be deployed on almost every computer system, including embedded systems [5, 22, 23].

Coordination languages and frameworks are designed based on coordination models of computations. These models of computation provide a technology that supports the interaction between software components. Moreover, they generally enhance modularity, reuse of existing (sequential or even parallel) components, portability, and language interoperability [27, 34].

In the coordination languages and frameworks, actors are usually used as the primary programming model. This programming model was first introduced in 1973 by Hewitt for concurrent systems. They are independent entities like objects that can communicate through asynchronous message passing without any locking mechanism [11].

To make actors deterministic, a new model, named *reactors*, was introduced in 2019 as the building block of the Lingua Franca language. In reactors, messages are guaranteed to be delivered to a reactive component in order. For these purposes, logical timestamps are used [24].

Procedures inside reactors are called reactions invoked in response to a trigger event. The reactions are atomic to one another, meaning they are mutually exclusive. The reactions can be written in the LF's target programming languages, and they can read input and produce outputs. Timers, ports, actions, and built-in triggers (such as startup or shutdown) can trigger reactions.

Ports are the types inside the reactors responsible for communicating with other reactors. We have two kinds of ports, input, and output, for receiving and sending messages. LF uses a timestamp tag on a logical timeline for messages to make them ordered. Unlike physical time, logical time does not elapse during reaction execution. In LF, timers use logical time to invoke reactions periodically.

2.2 Patmos

Patmos [33] is a RISC-style processor developed as part of the T-CREST project [32]. Patmos is designed for real-time systems with ease of analysis in mind. It has features that make it easy to analyze, such as an in-order pipeline and special caches. Instead of a traditional instruction cache, Patmos includes a method cache that stores complete function bodies or explicit parts of functions (sub-functions) [6]. At function calls/returns or at explicit points in the function, the method cache is triggered to load the next executed (sub-)function. This means instruction fetching can only miss in the method cache at this point, making it easy for an analyzer to reason about. Patmos also includes a stack cache that stores stack-local data exclusively and is explicitly controlled by the compiler [15]. Accessing stack data, therefore, never misses except at the start or end of a function. The remaining data accesses go through the conventional data cache or can circumvent all caching to access main memory directly.

2.3 The Compiler and the WCET Analyzer

In this paper, we use C as the target language and describe the compilation pipeline of LF for that target. LF, as shown in Listings 1 and 2, contains target code in C, wrapped with the markers `{=` and `=}`. The code around those C fragments is written in LF, which the LF compiler compiles into C functions. The generated functions include those C fragments. Additionally, LF provides the runtime (e.g., the reactor scheduler, functions to set outputs, and other utility functions) as C source files. The LF library code also contains platform-specific low-level functions. LF can execute on various platforms, from systems with full-blown operating systems (e.g., Linux or MacOS) down to the bare metal. In the latter case, LF *is* the operating system, and since no complex operations are used, the timing of the full application can be analyzed.

The generated C code is then compiled with a C compiler into an executable. This paper uses Patmos as the execution platform, which has had LF ported to it [16]. The Patmos compiler is based on the LLVM framework, which compiles C language code to Patmos machine code. It supports adding flow fact information to the code using annotations, the

Listing 1 The `Source` reactor of `SimpleConnection`.

```
reactor Source {
  output out: int
  timer t(0, 1 sec)
  state s: int = 0

  reaction(t) -> out {=
     lf_set(out, self->s);
     self->s++;
  =}

  reaction(t) -> out {=
    int v = -1 * self->s;
    lf_set(out, v);
  =}
}
```

Listing 2 The `Sink` reactor (incomplete) and reactor connections of `SimpleConnection`.

```
reactor Sink {
  input in: int
  state last_received: int = 0

  reaction(in) {=
    self->last_received = in->value;
  =}
  ...
}

main reactor {
  source = new Source()
  sink = new Sink()
  source.out -> sink.in after 2 sec
}
```

simplest of which are loop-bound annotations. The flow facts are maintained through the compilation pipeline and can be exported as part of the compilation to be used by the PLATIN WCET analyzer. The compiler automatically inserts code to manage the Patmos method and stack caches so that programs experience misses only at the predefined points.

We use the open-source analysis tool PLATIN to derive WCET bounds for code executing on Patmos [10, 25]. Through the tight integration with Patmos' compiler, PLATIN gets details about the program control flow and flow facts, which it uses to estimate the WCET bound. PLATIN includes a detailed model of the Patmos processor architecture, its method cache, and its stack cache [13, 15]. It does not have a dedicated data-cache analysis, meaning all data-cache accesses are assumed to be cache misses.

A requirement for using PLATIN is that the code is analyzable and the compiler can provide it with a *.pml* file containing control flow and flow fact information. As such, the LF runtime and scheduling code must be analyzable and include sufficient information and annotations for PLATIN to estimate the WCET of the LF runtime functions.

3 Analysis-Friendly Applications with Lingua Franca

The execution time of general programs is usually not statically analyzable. We need restrictions in the algorithms, e.g., maximum bounds on loop iterations, and a runtime system amenable to timing analysis, e.g., Lingua Franca, thanks to its determinism.

3.1 Analyzable Code

Code for real-time systems must be carefully written to enable static analysis tools, like PLATIN, to determine upper bounds on execution times, the WCET bounds [28]. The most apparent restriction on real-time code is the prohibition of indeterminate loop iteration. As such, annotations must be added to the code to provide such a bound when the tools or compilers cannot infer an upper bound on the number of iterations a loop may execute. Likewise, recursion is often prohibited in real-time code, as it also introduces the possibility of indeterminable execution time through infinite recursion. Recursion depth bounds can also be used to limit recursion. However, PLATIN does not support analysis of recursion, meaning we also prohibit recursion.

Dynamism must also be strictly regulated in real-time code. Dynamically sized arrays or data structures must have an upper bound on their size, such that iteration over those arrays is also bounded. Dynamic function pointers are also often prohibited, as knowing which functions they call is difficult. PLATIN does not allow any function calls through function pointers.

Furthermore, the C standard library contains several functions that are not analyzable. One prominent one is `printf`. Besides being a complex function with probably unbounded loops, `printf` may block. In the case of Patmos, the standard output stream is mapped to a serial port. When the send buffer of the serial port is full, `printf` will block until characters are sent out.

Another functionality to avoid in analyzable code is dynamic memory management. Standard implementations of `malloc` do not have execution time bounds. A better solution for some dynamic memory management is using pools with a bounded size [26].

3.2 Benefits of LF for WCET Analysis

In Lingua Franca, applications are developed modularly as networks of communicating reactors, where each reactor defines reactions to individual events. The reaction bodies tend to be small pieces of code, making them more manageable by static analysis tools. Moreover, the program specifies real-time requirements by attaching deadlines to the reactions. Most importantly, the program explicitly specifies the dependencies between reactions, so the analysis tool knows every piece of code that can affect the ability to meet the deadlines. The LF syntax encourages breaking apart complex application code into a set of simple reactions, which helps generate WCET values from timing analysis tools. In addition, LF's deterministic semantics enable the generation of predictable quasi-static schedules, which help analyze timing behavior at the system level, given individual WCETs of reactions.

It is also known *how* it affects the ability to meet a deadline. A piece of code may need to be executed before the deadline expires, or it may only need to be completed before the next event arrives. Ignoring that code in certain circumstances may be reasonable in the latter case. It may, for example, be performing logging functions that utilize the difficult-to-analyze `printf` function.

For example, suppose that an arriving event triggers several reactions, that some reactions have deadlines or are dependent on reactions that have deadlines, and some do not. Then, if we assume that reactions with deadlines will be prioritized in some specified manner, we can focus the WCET analysis on the bodies of those reactions. In principle, the reactions without deadlines can be deferred indefinitely, although doing so could create performance or memory problems. Those can be guarded against by, for example, dropping or modifying log entries when problems arise. Hence, our technique's ability to include some code that is more difficult to analyze in a program makes it much more practical than techniques that require full modeling of every part of the application.

3.3 Challenges with Dynamic Scheduling

The timing behavior of some functions in the standard LF runtime is not yet analyzable using PLATIN [16]. These functions typically include print statements or allocate memory with `malloc` or similar memory management functions.

By default, LF programs are scheduled by a dynamic scheduler, which maintains an event queue at runtime and ensures that events are processed in timestamp order. However, the dynamic scheduler presents challenges in timing analysis. The first challenge is the use of dynamic memory allocation. For example, a common user-facing library function is `lf_set`, which sets the value of an output port of a reactor and calls `calloc` when no events allocated on the heap can be recycled. A standard solution in real-time systems is to use only statically allocated objects. A program-managed pool of preallocated objects can be used if dynamic buffer management is needed. To enable WCET analysis of standard LF programs, we propose changing the runtime to use explicit memory management.

We need to analyze individual reactions and their scheduling, which affects the overall timing behavior. Since the dynamic scheduler makes all decisions at runtime, predicting its behavior at compile time requires building an accurate model that captures its intended behavior, which is challenging.

The above challenges motivate an alternative technique for scheduling LF programs amenable to timing analysis at compile time and suitable for hard real-time systems, which we will discuss next.

3.4 Quasi-Static Scheduling of LF Applications

It is common practice for safety-critical systems to use a static schedule, usually called a cyclic executive. The pros and cons of cyclic executives have been discussed [21]. The main disadvantage of a cyclic executive is that long-running tasks often need to be split into smaller tasks to construct a feasible, static schedule. This restriction can be overcome by using multiple processor cores and scheduling long-running tasks on a dedicated processor core [29].

Recently, Lin et al. [18] developed a technique for generating quasi-static schedules from LF programs. Quasi-static schedules are encoded into bytecode programs, composed of an instruction set developed for *PretVM*, a virtual machine executing the schedules within the LF runtime. The schedules are "quasi-static" instead of "static" because parts of them can be enabled or bypassed depending on the execution context. Compared to the user-written LF reactions, which represent the *application* logic, a quasi-static schedule represents the *coordination* logic of an LF program, encoding scheduling decisions satisfying task dependencies and timing constraints. LF's quasi-static scheduling is experimental and limited to timer-driven programs. Yet, as we will show next, it offers promising analyzability. LF's quasi-static scheduler supports multiple cores. However, this initial work focuses on a single-core system. In future work, we plan to use multiple cores to execute reactors in parallel and a network-on-chip to exchange messages between reactors [14].

Unlike the default dynamic scheduler, which collects events and determines which to process next at runtime, LF under quasi-static scheduling makes all the scheduling decisions at compile-time. The user first annotates the LF program with a WCET estimate for each reaction using the `@wcet` attribute. Then, based on LF's deterministic semantics, the compiler computes the LF program's state space, identifying various execution phases and finding a hyperperiod of reaction invocations. Once the state space is determined, a set of (unpartitioned) directed acyclic graphs (DAGs) is generated. These DAGs encode dependencies among reaction invocations and between reaction invocations in real-time. A quasi-static scheduler is invoked by the LF compiler to schedule the unpartitioned DAG and generate partitions of the DAG based on the number of workers specified by the program. From that DAG partition we produce a bytecode program using the virtual instruction set.

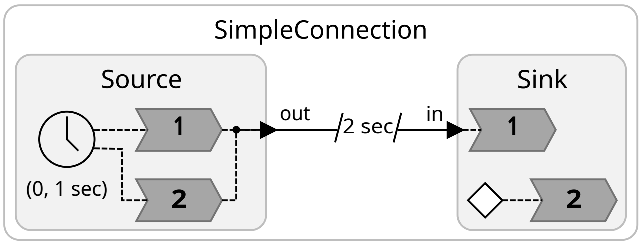


Figure 1 The graphical representation of the reactors and their connection generated by the Lingua Franca framework for the LF application shown in Listings 1 and 2.

3.5 Analyzing the Runtime System

Compiling an LF program results in a standard .elf file containing the LF runtime and the individual reactions. We need the WCET of the individual reactions to make the quasi-static schedule, which can be obtained using Platin. We run Platin on the .elf, prompting it for the WCET bound of each reaction. This is then fed to the quasi-static scheduler to try and create a schedule. For a full WCET bound of a reaction, we must also account for the time the scheduler executes. For this work, the PretVM issues specific *instructions* to schedule each reaction. These instructions take time to execute and add to the WCET of a reaction. Each instruction is executed by a dedicated function in the LF runtime. This allows us to use Platin again to analyze these functions to associate each instruction with its WCET bound. To calculate a reaction's WCET bound, we take the Platin bound and add the bounds of each instruction used to schedule the reaction, giving us a total bound for the WCET of a reaction, which also accounts for the scheduling time.

4 Evaluation

The proposed approach is demonstrated and evaluated by targeting the Patmos processor and by using the Platin tool for WCET analysis. Our evaluation focuses on demonstrating the feasibility and effectiveness of using LF to produce a predictable real-time application. We start by presenting a simple example application to illustrate the basic analyzability of LF reactors. Subsequently, we scale up to a medium-sized application to showcase the proposed approach's ability to handle more complex and realistic examples. For the medium-sized application, we aim to comprehensively assess the end-to-end workflow from source code to schedulability analysis, including the overhead introduced by the LF runtime.

4.1 A Simple Example Application

As an initial example for WCET analysis of a complete LF program, we use the `SimpleConnection` program, shown in Figure 1. This program has four reactions, divided into two reactors. The `Source` reactor (Listing 1) has a timer that every second triggers its two reactions. The first reaction outputs the reactor shared state variable, `s`, and then increments it. The second reaction negates the value of `s` before outputting it. The result of these reactions is that each trigger outputs the negated count that the reactor has reached. I.e., $0, -1, -2, -3$ etc. After a delay of two seconds – which is managed by the LF runtime – the `Sink` reactor's first reaction is triggered with the previous output value, which it stores in a reactor variable (Listing 2). We ignore the second `Sink` reaction as its only meant to run when the program times out.

Table 1 Individual WCET bounds in clock cycles of the reactions of Lingua Franca applications.

Application	Function	WCET Bound
`SimpleConnection`	Source reaction 1	969
	Source reaction 2	925
	Sink reaction 1	540
`ADASModel`	Brakes reaction	497
	Lidar reaction	946
	Camera reaction	946
	Dashboard reaction	497
	Processor reaction 1	1786
	Processor reaction 2	2130

Analyzing the WCET bounds of these reactions is done as previously described. In Table 1, we see the WCET bounds produced by Platin for each function (called "Sink reaction X" or "Source reaction X"). These numbers are given to the quasi-static scheduler, which attempts to construct a feasible schedule for them. To account for its overhead, i.e., the execution of PretVM instructions, we also analyze the functions executing those instructions and provide the numbers to the scheduler. The scheduler then uses the WCET number of its instructions and those of the reactions and attempts to construct a feasible schedule for the target, in this case, a single Patmos core. For example, the schedule would use the scheduler instructions {`EXE; ADDI`} when triggering the Sink 1 reaction. The `EXE` instruction requires 112 cycles to run (excluding the execution time of the called function), while the `ADDI` instruction requires 403 cycles. So, the cumulative execution time of Sink 1 when accounting for the scheduler is 1055 cycles.

The scheduler's output is a quasi-static schedule that, within some hyperperiod, will schedule all tasks on the available hardware. Reactions can be given deadlines that are bound by their release times. If there exists no schedule that satisfiws all deadlines, the scheduler throws an error. For example, if a reaction depended on Sink 1 with a deadline of 1000 cycles, the scheduler would trivially fail because it simply could not schedule Sink 1 early enough to meet the deadline regardless of the rest of the schedule.

4.2 Medium-sized Application

As a medium-sized application, we experimented with and modeled the "Advanced Driver Assistance System (ADAS)", a ubiquitous system in the automotive industry [19]. In this model, we have a processor that receives events from a Light Detection and Ranging (LiDAR) device and a camera and sends commands to the brakes and the dashboard. In this system, the dashboard shows a message when an object approaches the vehicle. It also triggers brakes automatically when the object is too close. In Figure 2, we model each system part as reactors connected by ports.

We again give the Platin-produced WCET bounds of each reaction in Table 1. Notice how the bounds for the brake and dashboard reactions are the same. This is because these reactions are only stubs, as we do not have a physical system to interact. This is also the case for the LiDAR and Camera reactions. Therefore, only the processor reactions have actual code. We use this application only to show the feasibility of implementing a real-work application, which would differ from this one only by implementing the physical end-point reactions and proper sensor fusion.

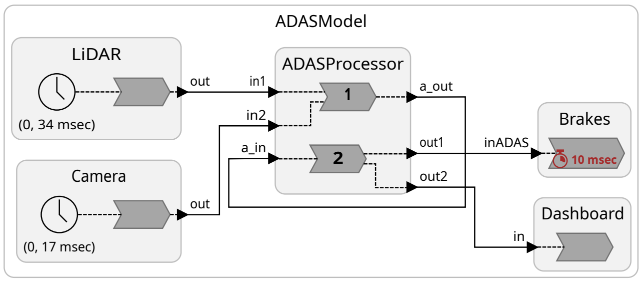


Figure 2 The graphical representation of the reactors and their connection generated by the Lingua Franca framework for the ADASModel LF application.

After we get the schedule devised by the quasi-static scheduler, we can evaluate whether it satisfies our requirements. For example, we might want to be able to detect 100 times per second whether breaking is needed using our default T-CREST test platform (Altera Cyclone IV FPGA mounted on DE2-115 evaluation board). Our FPGA runs at 80 MHz, therefore the deadline of 10 ms translates to 800 000 clock cycles. As part of the schedule construction, the overhead of each reaction is accounted for, as well as the execution time of other reactions that might run between the reaction's arrival and its beginning execution. In Figure 2, we have added the 10 ms as a deadline for releasing the brake reaction. In LF, all other tasks preceding the braking must execute within the 10 ms, cumulatively.

To check whether the schedule can meet the deadline, we look at the execution chains of the schedule. Any chain ending in the brake reaction must have a cumulative WCET bound below the 800 000 cycles. The first chain in the scheduled hyperperiod executes the following reactions: LiDAR, Camera, Processor 1, Processor 2, Brake, and Dashboard. The chain until the braking uses the following PretVM instruction counts: 3x`BEQ`, 7x`EXE`, 2x`JAL`, and 4x`ADDI`. Based on the Platin bounds on executing these instructions, they cumulatively add 4 086 cycles to the WCET. Coupled with the WCET of the reaction until and excluding the brake, we get a total WCET bound of 9 894 for the reactions before the braking. This is well below our limit, and so the schedule is valid. If the schedule had not been valid, the scheduler would have thrown an error, saying it could not meet the deadline. For example, we could trigger this error by reducing the deadline to 9 000 clock cycles.

In practice, the analysis must be done for all the execution chains in the scheduled hyperperiod. However, for brevity, we will omit this for the other chains in the `ADASModel` schedule.

4.3 Automated Solution

In the long term, we aim to streamline the development process for real-time systems by implementing a one-click automated solution that integrates compilation, WCET analysis, schedulability analysis, and scheduling. The automated solution will start by compiling the reaction code and the LF runtime functions (which include the functions executing scheduling instructions.) This code is analyzed using Platin to get the WCET bound of the reactions and LF runtime functions. Next, the LF quasi-static scheduler organizes reaction execution (classically called tasks) based on the provided WCET bounds and the deadlines and periods according to the use case's constraints. Finally, the scheduler performs schedulability analysis automatically to ensure a feasible schedule exists before creating one. Thus, it produces a schedule as an object file linked with the previously produced code to become the final application executable (ELF file). All these steps have been carried out manually for this paper. We aim to automate that process to ensure that only a correct-by-construction solution is output, meaning the resulting executable will adhere to all specified time constraints.

5 Related Work

Several projects aim to build time-predictable processors. One example is FlexPRET [36], the latest version of the so-called precision timed machines [7]. FlexPRET also aims to be a platform that supports LF applications. FlexPRET includes timing instructions for the precise timing of reactions. Furthermore, FlexPRET implements fine-grain multithreading. FlexPRET is not yet supported by any WCET analysis tool. However, as Platin now also supports the RISC-V architecture [25], we will be able to adapt Platin for FlexPRET. Like the multicore version of Patmos, the InterPRET [14] projects aim for a multicore version of FlexPRET supporting parallel execution of LF actors on multiple cores.

ForSyDe (Formal System Design) [30, 31] is a methodology enabling high-level abstraction modeling and design of heterogeneous systems-on-chip and cyber-physical systems. The idea is to integrate formal methods from the specification phase and use formal refinement techniques to bridge the gap between specification and implementation. Thus, creating a correct-by-design system. The most interesting aspect related to our solution is the ability to employ formally analyzable models of predictable platforms and applications to provide service guarantees, which are essential in time-predictable applications.

MIRSA C [3] is a set of software development guidelines for the C programming language to ensure that C code is safe, reliable, and maintainable. Even if these guidelines do not directly address time-predictable systems, enforcing a strict coding standard helps avoid undefined behaviors that can lead to unpredictable execution times. One very concrete guideline is the prohibition of the use of dynamic memory allocation functions such as `malloc()`, `calloc()`, and `free()`. These are restricted to avoid memory leaks, fragmentation, and unpredictable behavior. The latter is particularly relevant for real-time systems since the time taken to allocate or deallocate memory can vary significantly and can be difficult to predict.

The review presented in [35] discusses the current challenges in WCET analysis and surveys several WCET tools, highlighting their methods, functionalities, and limitations. Here, two main categories of WCET tools are identified: static analysis and measurement-based or hybrid tools. Static analysis tools determine WCET by analyzing the code without executing it. These tools construct a detailed model of the program and the processor to estimate execution time bounds. They mainly focus on control-flow and data-flow analysis to provide guaranteed upper bounds on execution times. Measurement-based or hybrid tools (using measurements and static analysis) estimate the WCET by executing the program or its parts on actual hardware or simulators. They measure execution times of code segments and use these measurements to infer timing bounds. This approach often results in more accurate estimates for complex systems but may lack formal guarantees. Commercial tools examples include aiT [1, 9] from AbsInt, Bound-T from Tidorum [2, 12], and RapiTime from Rapita Systems [4]. Examples from academia include Heptane [8], Chronos [17], and SWEET [20].

6 Conclusion

This paper presented the integration of the time-predictable processor Patmos with the Lingua Franca (LF) coordination language and the Platin WCET analysis tool. More specifically, we used the WCET analysis tool Platin to analyze individual reactions of LF reactors and the runtime of the quasi-static schedule. Using a simple and a medium-sized application, our evaluation confirmed that carefully written LF programs can be analyzed for WCET, creating a quasi-static schedule that fulfills all timing requirements for safety-critical applications.

References

1 aiT webpage. Online at `https://www.absint.com/ait/`. Accessed: 07 June 2024.

2 Bound-T webpage. Online at `http://www.bound-t.com/`. Accessed: 07 June 2024.

3 MISRA webpage. Online at `https://misra.org.uk/`. Accessed: 07 June 2024.

4 RapiTime webpage. Online at `https://www.rapitasystems.com/products/rapitime`. Accessed: 07 June 2024.

5 Lingua franca website:. `http://www.lf-lang.org/`, 2024.

6 Philipp Degasperi, Stefan Hepp, Wolfgang Puffitsch, and Martin Schoeberl. A method cache for Patmos. In *Proceedings of the 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2014)*, pages 100–108, Reno, Nevada, USA, June 2014. IEEE. `doi:10.1109/ISORC.2014.47`.

7 Stephen Edwards and Edward Lee. The case for the precision timed (PRET) machine. In *2007 44th ACM/IEEE Design Automation Conference*, pages 264–265, July 2007. URL: `https://ieeexplore.ieee.org/abstract/document/4261184`.

8 Damien Hardy, Benjamin Rouxel, and Isabelle Puaut. The Heptane Static Worst-Case Execution Time Estimation Tool. In Jan Reineke, editor, *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*, volume 57 of *Open Access Series in Informatics (OASIcs)*, pages 8:1–8:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.WCET.2017.8`.

9 Reinhold Heckmann and Christian Ferdinand. Worst-case execution time prediction by static program analysis. Technical report, AbsInt Angewandte Informatik GmbH, 2013. [Online, last accessed November 2013].

10 Stefan Hepp, Benedikt Huber, Jens Knoop, Daniel Prokesch, and Peter P. Puschner. The platin tool kit - the T-CREST approach for compiler and WCET integration. In *Proceedings 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach, Austria, October 5-7, 2015*, 2015.

11 Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 235–245, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

12 Niklas Holsti, Thomas Långbacka, and Sami Saarinen. Bound-T time and stack analyzer, reference manual. Technical report, Tidorum Ltd, 2013. available at `http://www.bound-t.com/manuals/ref-manual.pdf`.

13 Benedikt Huber, Stefan Hepp, and Martin Schoeberl. Scope-based method cache analysis. In *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, pages 73–82, Madrid, Spain, July 2014. `doi:10.4230/OASIcs.WCET.2014.73`.

14 Erling R. Jellum, Shaokai Lin, Peter Donovan, Chadlia Jerad, Edward Wang, Marten Lohstroh, Edward A. Lee, and Martin Schoeberl. Interpret: A time-predictable multicore processor. In *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, CPS-IoT Week '23, pages 331–336, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3576914.3587497`.

15 Alexander Jordan, Florian Brandner, and Martin Schoeberl. Static analysis of worst-case stack cache behavior. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS 2013)*, pages 55–64, New York, NY, USA, 2013. ACM. `doi:10.1145/2516821.2516828`.

16 Ehsan Khodadad, Luca Pezzarossa, and Martin Schoeberl. Towards lingua franca on the patmos processor. In *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*, 2024.

17 Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1):56–67, 2007. Special issue on Experimental Software and Toolkits. `doi:10.1016/j.scico.2007.01.014`.

18 Shaokai Lin, Erling Jellum, Mirco Theile, Tassilo Tanneberger, Binqi Sun, Chadlia Jerad, Ruomu Xu, Guangyu Feng, Christian Menard, Marten Lohstroh, Jeronimo Castrillon, Sanjit Seshia, and Edward Lee. Pretvm: Predictable, Efficient Virtual Machine for Real-Time Concurrency, 2024. arXiv:2406.06253.

19 Shaokai Lin, Yatin A. Manerkar, Marten Lohstroh, Elizabeth Polgreen, Sheng Jung Yu, Chadlia Jerad, Edward A. Lee, and Sanjit A. Seshia. Towards building verifiable cps using lingua franca. *Acm Transactions on Embedded Computing Systems*, 22(5 s):155, 2023. doi:10.1145/3609134.

20 Björn Lisper. Sweet – a tool for wcet flow analysis. In Bernhard Steffen, editor, *6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 482–485. Springer-Verlag, October 2014. URL: http://www.es.mdu.se/publications/3693-.

21 C. Douglas Locke. Software architecture for hard real-time applications: Cyclic executives vs. fixed priority executives. *Real-Time Systems*, 4(1):37–53, 1992.

22 Marten Lohstroh, Christian Menard, Soroush Bateni, and Edward A. Lee. Toward a lingua franca for deterministic concurrent systems. *Acm Transactions on Embedded Computing Systems*, 20(4):36, 2021. doi:10.1145/3448128.

23 Marten Lohstroh, Christian Menard, Alexander Schulz-Rosengarten, Matthew Weber, Jeronimo Castrillon, and Edward A. Lee. A language for deterministic coordination across multiple timelines. *Forum on Specification and Design Languages*, 2020-:9232939, 2020. doi:10.1109/FDL50818.2020.9232939.

24 Marten Lohstroh, Martin Schoeberl, Andrés Goens, Armin Wasicek, Christopher Gill, Marjan Sirjani, and Edward A. Lee. Actors revisited for time-critical systems. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, pages 152:1–152:4, New York, NY, USA, June 2019. ACM. doi:10.1145/3316781.3323469.

25 Emad Jacob Maroun, Eva Dengler, Stefan Dietrich, Chistian Hepp, Benedikt Herzog, Henriette Huber, Jens Knoop, Daniel Prokesch, Peter Puschner, Phillip Raffeck, Martin Schoeberl, Simon Schuster, and Peter Wägemann. The platin multi-target worst-case analysis tool. In *22th International Workshop on Worst-Case Execution Time Analysis (WCET 2024)*, 2024.

26 Miguel Masmano, Ismael Ripoll, Alfons Crespo, and Jorge Real. Tlsf: A new dynamic memory allocator for real-time systems. In *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.*, pages 79–88. IEEE, 2004.

27 George A. Papadopoulos and Farhad Arbab. Coordination models and languages. *Advances in Computers*, 46:329–400, 1998. doi:10.1016/S0065-2458(08)60208-9.

28 Peter Puschner and Christian Koza. Calculating the maximum execution time of real-time programs. *Real-Time Syst.*, 1(2):159–176, 1989.

29 Anders P. Ravn and Martin Schoeberl. Safety-critical Java with cyclic executives on chip-multiprocessors. *Concurrency and Computation: Practice and Experience*, 24:772–788, 2012. doi:10.1002/cpe.1754.

30 Ingo Sander. *System Modeling and Design Refinement in ForSyDe.* PhD thesis, KTH, Microelectronics and Information Technology, IMIT, 2003. NR 20140805.

31 Ingo Sander and Axel Jantsch. Modelling adaptive systems in forsyde. *Electronic Notes in Theoretical Computer Science*, 200(2):39–54, 2008. Proceedings of the First Workshop on Verification of Adaptive Systems (VerAS 2007). doi:10.1016/j.entcs.2008.02.011.

32 Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015. doi:10.1016/j.sysarc.2015.04.002.

33 Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, April 2018. doi:10.1007/s11241-018-9300-4.

34 Robert Tolksdorf. Models of coordination. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1972:78–92, 2000. `doi:10.1007/3-540-44539-0_6`.

35 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution time problem – overview of methods and survey of tools. *Trans. on Embedded Computing Sys.*, 7(3):1–53, 2008. `doi:10.1145/1347375.1347389`.

36 Michael Zimmer, David Broman, Chris Shaver, and Edward A. Lee. FlexPRET: A processor platform for mixed-criticality systems. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, Berlin, Germany, April 2014.

Invited Paper: On the Granularity of Bandwidth Regulation in FPGA-Based Heterogeneous Systems on Chip

Gianluca Brilli
University of Modena and Reggio Emilia, Italy

Giacomo Valente
University of L'Aquila, Italy

Alessandro Capotondi
University of Modena and Reggio Emilia, Italy

Tania Di Mascio
University of L'Aquila, Italy

Andrea Marongiu
University of Modena and Reggio Emilia, Italy

Abstract

Main memory sharing in commercial, FPGA-based Heterogeneous System on Chips (HeSoCs) can cause significant interference, and ultimately severe slowdown of the executing workload, which bars the adoption of such systems in the context of time-critical applications. *Bandwidth regulation* approaches based on *monitoring* and *throttling* are widely adopted also in commercial hardware to improve the system quality of service (QoS), and previous work has shown that the finer the granularity of the mechanism, the more effective the QoS control. Different mechanisms, however, might exploit more or less effectively the available residual memory bandwidth, provided that the QoS requirement is satisfied. In this paper we present an exhaustive experimental evaluation of how three bandwidth regulation mechanisms with coarse, fine and ultra-fine granularity compare in terms of exploitation of the system memory bandwidth. Our results show that a very fine-grained regulation mechanism might experience worse system-level memory bandwidth exploitation compared to a coarser-grained approach.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture

Keywords and phrases Bandwidth Regulation, System-on-Chip, FPGA

Digital Object Identifier 10.4230/OASIcs.WCET.2024.5

Funding *Andrea Marongiu*: This work was supported by project AI4CSM, which has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007326. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Austria, Belgium, Czech Republic, Italy, Netherlands, Lithuania, Latvia, Norway. The work was also supported by the European Union under the NextGenerationEU Programme within the Plan "PNRR - Missione 4 "Istruzione e Ricerca" - Componente C2 Investimento 1.1 "Fondo per il Programma Nazionale di Ricerca e Progetti di Rilevante Interesse Nazionale (PRIN)" by the Italian Ministry of University and Research (MUR)". Project Title: "Simplifying Predictable and energy-efficient Acceleration from Cloud to Edge (SPACE)", Project code: E53D23007800006. MUR D.D. financing decree n. 959, 30th June 2023.

1 Introduction

Heterogeneous Systems-on-Chip (HeSoCs) coupling general purpose multi-cores and accelerators of various types are widely adopted across several application domains. Commercial off-the-shelf HeSoCs constitute a convenient and cheap solution to providing the necessary computing power to run modern software applications, but also pose novel challenges. Main interconnect and memory sharing, which is a key architectural trait of such products, causes a significant slowdown of the application tasks [6], ultimately barring their adoption in time-critical domains. Main memory bandwidth regulation strategies are being increasingly adopted in commercial products, to provide some degree of QoS control. Several research approaches have also been proposed in this area to improve the effectiveness of such techniques [8, 15, 20].

Focusing on FPGA-based based HeSoCs, previous work has shown that the combination of bandwidth *monitoring* and *throttling* mechanisms is key not only to providing QoS guarantees to the CPU workloads – in terms of maximum slowdown experienced – but also to allowing FPGA accelerators to effectively use the residual bandwidth (without slowing down the CPU beyond the tolerated QoS thresholds) [3]. The degree of coupling between *monitoring* and *throttling* is pivotal to achieving fine-grained QoS control, suggesting that the finer the granularity, the better the results. We observed that while this is certainly true in terms of control capability (how fast the mechanism can adapt to varying workload characteristics due to memory interference), a very fine-grained bandwidth regulation mechanism might adversely impact the behavior of the memory controller, ultimately resulting in worse system-level memory bandwidth exploitation.

In this paper we study and compare three state-of-the-art mechanisms for joint bandwidth monitoring and throttling: (i) a coarse-grained, software-based approach that can be implemented on top of widely available HW performance counters; (ii) a fine-grained, FPGA-based, integrated runtime bandwidth regulator (RBR [3]); (iii) an ultra fine-grained, interconnect-level hardware approach, available in a number of commercial HeSoCs (ARM QoS-400 [1]). We conduct an extensive experimental evaluation on a representative FPGA-based HeSoCs, the AMD/Xilinx Zynq Ultrascale+, using real benchmarks: the Polybench benchmark suite [18]. Our setup measures the amount of residual bandwidth that various bandwidth regulation mechanisms allow the FPGA accelerators to exploit, while maintaining the slowdown of CPU programs within the tolerated QoS requirements. Experimental results show that the best results are achieved for the medium-granularity approach, while the ultra fine-grained one is often surpassed even by the coarse-grained one in terms of exploitation of the system bandwidth. We thus present a more in-depth investigation of the problem, that highlights how idleness insertion at a very fine-grained level[1] triggers worse DRAM controller behavior than a slower regulation mechanism.

2 Related Work

Memory interference significantly impacts the performance of modern HeSoCs. This has driven extensive research in recent years, examining its effects on various components such as the main CPU [7], GPGPU accelerators [5, 19], and FPGA accelerators [4, 12].

Memory bandwidth regulation is a practical and effective technique typically adopted on Commercial Off-the-shelf (COTS) HeSoCs. This method is essential for providing guarantees on application Quality of Service (QoS) and formitigating issues related to memory contention. Providing an accurate memory bandwidth regulation reduces contention and enforces execution time predictability, particularly in scenarios where multiple applications with diverse and competing bandwidth demands are executed simultaneously [17, 20, 22].

[1] QoS-400 operates at the granularity of a small number of back-to-back transactions (or *beats*), 16 on the target hardware, equivalent to 256 bytes only.

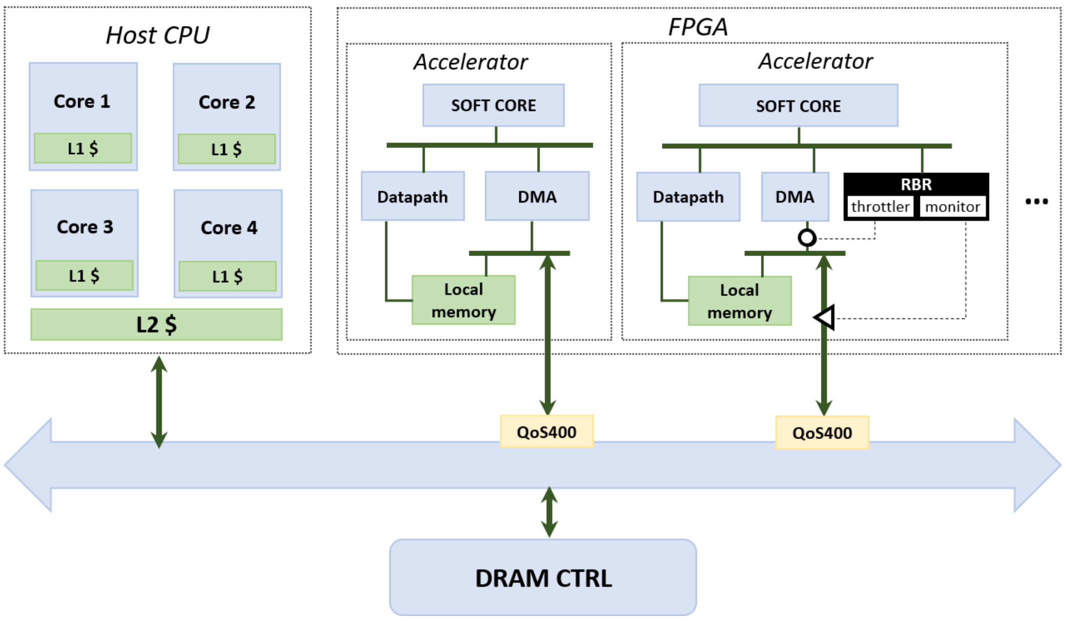


Figure 1 Architectural template of the target HeSoC.

Several bandwidth regulation techniques exist both from hardware and software perspectives. Considering software-based memory bandwidth regulators, Yun et al. proposed *MemGuard* [20] a memory bandwidth throttler that is based on a joint action between bandwidth monitoring (using core performance counters) and a software throttling mechanism based on interrupts. Controlled Memory Request Injection (CMRI) has been originally proposed as a software-based bandwidth regulation technique, to regulate CPU-based workloads [7] and in a more complex setup where also FPGA-based accelerators are involved [4]. All these software-based bandwidth regulators suffer of non-negligible overheads due to software interactions between loosely-coupled monitoring and throttling components.

Finer bandwidth regulation mechanisms could be designed by leveraging tighter hardware components to implement bandwidth regulation. For example, Zuepke et al. proposed *MemPol* [22] a hardware memory bandwidth regulator that can regulate application cores with 6.25μs granularity, where the throttling phase is implemented using a hardware debugging interface. Similarly, Farshchi et al. proposed a hardware-based fine-grained regulator for application cores evaluated on the *FireSim* simulator [8]. A complementary hardware approach has been proposed to regulate FPGA-based accelerators with a similar granularity [3]. Several other works try to understand and effectively exploit the available hardware knobs for controlling QoS and regulating memory bandwidth at different interconnect levels and memory hierarchy. As it was shown in recent works, using these knobs is typically not straightforward, due to the varying degrees of support on different products and to the many different (and in some cases obscure) configurations available [9, 16, 21]. Furthermore, these mechanisms lack generality, as they are typically closed solutions specific to a given vendor or hardware platform (e.g., ARM MPAM [14], ARM QoS-400 [1]).

Although it is known that a fine-grained mechanism adapts better to bandwidth variations, our work proposes a preliminary investigation that highlights that, in some conditions, using a coarser-grained bandwidth regulator (i.e. even software-based), could better utilize the bandwidth from the FPGA, given a QoS requirement on a task running on CPU cores.

3 Target Architecture and Bandwidth Regulation Schemes

We consider the architectural template of an FPGA-based HeSoC shown in Figure 1, which is composed of a *host* multi-core CPU coupled to an FPGA subsystem. The two subsystems communicate via the main DRAM. This template captures the main traits of several existing commercial products. Within the FPGA, one or more *accelerators* are deployed. A generic template for an *accelerator* includes a *datapath*, namely the core logic that performs the computation, and an efficient *DMA engine*, used to facilitate the staging of data from the DRAM into a fast, local memory. To simplify the development of FPGA applications it is common to also enrich the accelerator template with a *soft core* for local control of the *datapath* and *DMA* operation, without the need for the costly intervention of the main CPU [2, 10, 11, 13].

In modern HeSoCs, the DMAs inside FPGA accelerators generate much higher DRAM bandwidth request than what happens on the CPU cores, and more than a single master port is typically available to individually attach accelerators to the main interconnect fabric (for example, the AMD/Xilinx Ultrascale+ device that we use for our experimental setup features three independent ports). If CPU cores and FPGA accelerators run in parallel without DRAM access control, the execution time of the CPU tasks can slow down by over 10× [12]. On the other hand, enforcing mutually exclusive CPU/FPGA DRAM accesses causes severe under-utilization of the available memory bandwidth. Since the main interface of an *accelerator* to the DRAM is the *DMA*, previous work has shown that bandwidth monitoring and throttling can efficiently happen at this level [3].

3.1 Bandwidth Regulation Schemes

In this section we describe three bandwidth regulation mechanisms that rely on integrated monitoring and throttling cycles, ranging from coarse-grained, full-SW solutions to ultra-fine-grained, full-HW solutions.

3.1.1 SW-DMA

Previous research has investigated the throttling of FPGA *accelerators* by utilizing *soft cores* to program the DMA in a duty-cycled loop [4]. Each DMA transfer request is divided into multiple smaller transfers, which can be interspersed with a programmable amount of $idle_{cycles}$, computed as shown in Eq. (1):

$$idle_{cycles} = \frac{100 - THR_{\%}}{THR_{\%}} * copy_{cycles} \tag{1}$$

Here, $THR_{\%} \in [1, 100]$ denotes the throttling factor[2] applied to the transfer, and $copy_{cycles}$ is the number of clock cycles required to complete the smaller transfer. A disadvantage of throttling accelerators via software (even when executed on the local *soft core*) is the substantial programming overhead, which prevents very fine-grained operation.

3.1.2 Runtime Bandwidth Regulator

The Runtime Bandwidth Regulator (RBR) [3] has been proposed as a non-intrusive component introduced in the the accelerator template, as shown in Figure 1. It contains two main blocks: (i) a monitor that probes the outgoing channel to the DRAM to unobtrusively measure

[2] THR=100 corresponds to 100% bandwidth, while THR=1 corresponds to 1% bandwidth.

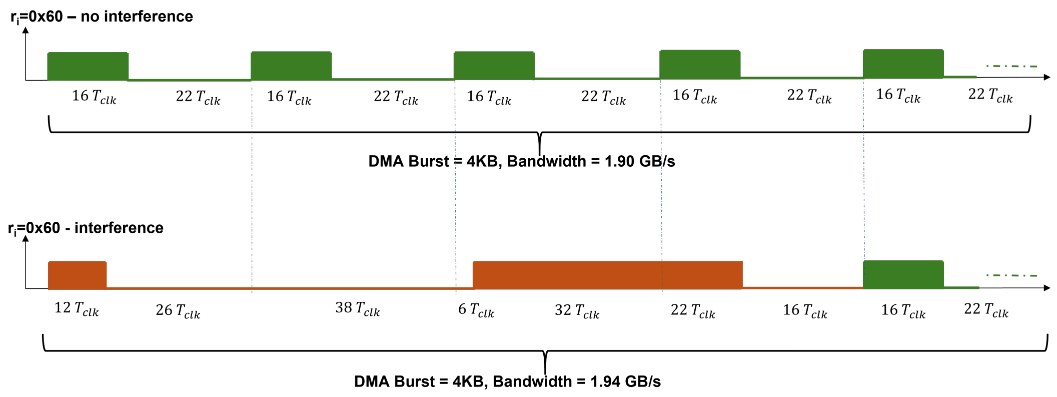


Figure 2 Waveforms of a memory transaction generated from an accelerator, composed of 256 beats and regulated using QoS-400 with r_i = 0x60. The red and green waveforms represent the case with and without memory interference.

the time (copy cycles from Eq. (1)) to transfer a given amount of bytes. The size of this transfer defines the *granularity* of the technique; (ii) a throttler that computes the idle cycles from Eq. (1) and stops DMA operation for that amount of time. The *granularity* and the throttling factor (THR) can be dynamically (re)configured at any time by the software.

The tight coupling between the monitor and the throttler in the RBR guarantees very fast QoS regulation, which is convenient in presence of dynamically varying QoS requirements.

3.2 ARM QoS-400 regulator

The ARM CoreLink QoS-400 regulator [1] is a hardware component designed to manage bus traffic generated from various actors sharing main memory on the HeSoC. The official ARM documentation does not provide detailed descriptions of the QoS-400 regulator behavior, nor does it specify the granularity of the regulation. By means of a thorough experimental characterization of the QoS-400 behavior, it is possible to observe that the regulation operates with a fixed, very fine granularity.

Fig. 2 shows an example of how the QoS-400 manages long outstanding transactions from a DMA in absence of interference (top plot) and in presence of interference (bottom plot). The example DMA transfer is a burst of 4096 bytes. The burst is split in 256 *beats*, as the physical size of the channel is 16 bytes. In absence of interference, the QoS-400 steadily fragments the 256-beat transfer into blocks of 16 beats each, followed by a idle period whose length is determined according to Eq. (1). In presence of interference, it is evident that the QoS-400 is capable of adapting, employing bandwidth monitoring and adjusting the throttling. From the plot on the bottom we can see that the QoS-400 detects the bandwidth drop due to interference and maintains the desired QoS level by adjusting the throttling . By allowing twice the number of beats to be transmitted in the third sub-transaction, the bandwidth is maintained at the target value.

4 Experimental Results

This experimental section aims to analytically study the behavior of the different bandwidth controllers under investigation. Previous work has highlighted already that finer-grained approaches are more effective at guaranteeing the desired QoS levels in presence of fast dynamic changes in the QoS requirements or/and in the traffic characteristics [3]. However, different approaches are more or less effective at redistributing the bandwidth unused by the actors executing the QoS-constrained workload to the remaining (best-effort) actors. Our experiments focus on analyzing the capability of each method of redistributing the unused bandwidth, provided that they all are configured to always meet the QoS requirements. Further, the experiments examine how the different architectural designs of the controllers and their inherent operational granularities impact their performance and the level of interference they generate in the system.

4.1 Experimental Setup

The experimental evaluation is conducted on an AMD/Xilinx Zynq Ultrascale+, *XCZU9EG* HeSoC. The accelerator template for the SW-DMA and RBR mechanisms was modeled after the setup described in [3] [4], using AMD/Xilinx IPs for the DMA, soft-core and interconnects. To capture worst-case interference effects, we instantiate three accelerators (one per DRAM controller port), each of which is configured to operate as a generator of steady R/W traffic[3] [12]. The resulting design was synthesized with a target frequency of 300 MHz using AMD/Xilinx Vivado 2020.2.

In our setup, the CPU is considered the actor with a QoS constraint, while the FPGA accelerators are the best-effort actors. As it is often done in the literature [16, 17, 22], we consider two fixed thresholds to the maximum tolerated QoS degradation: 10% and 20% slowdown with respect to non-interfered execution. We measure the maximum amount of memory bandwidth that FPGA accelerators can utilize without causing performance degradation in software applications that surpass the aforementioned QoS thresholds. As a target workload executing under the described QoS constraints, we execute 31 different applications from the Polybench benchmark suite [18].

4.2 Exploitable Residual Bandwidth Evaluation

Figure 3 describes how the experimental evaluation is conducted. One such plot is derived for every benchmark. The X-axis represents the overall throttling factor $\text{THR}_{\%}$ applied to the three FPGA accelerators. Note that the percentage shown on the X-axis refers to the cumulative bandwidth used by the three accelerators. Thus, a $\text{THR}_{\%} \leftarrow 100\%$ implies that the i-th accelerator is configured with one-third of the total $\text{THR}_{\%}$ (i.e., $ACT_i = 33.3\%$ $\text{THR}_{\%}$). A $\text{THR}_{\%} \leftarrow 100\%$ corresponds to utilizing the entire memory bandwidth of the FPGA, as denoted by a vertical dotted line. The red area depicts the bandwidth used by the FPGA accelerators and refers to the left Y-axis of the plot. The horizontal black curves mark the two tolerated CPU slowdowns (10% and 20%) and refer to the right Y axis of the plot. Two black arrows originate at the points where the measured CPU slowdown curve (the one with red markers) intersects the horizontal black curves and are projected up vertically to the point where they intersect the red area. The latter intersection points indicate the memory bandwidth used by the FPGA for the target regulation mechanism under both QoS constraints (10% and 20%). Intuitively, a higher FPGA bandwidth means a better capability of redistributing the unused bandwidth by the CPU to the FPGA accelerators.

These memory bandwidth values are plotted in Figure 4 for all the PolyBench benchmarks. Subplots a) and b) refer to 10% and 20% max QoS degradation, respectively. The RBR regulator generally allows for greater bandwidth usage compared to other regulators, both at

[3] Note that this is without loss of generality, as well-designed accelerators overlap memory transactions with computation.

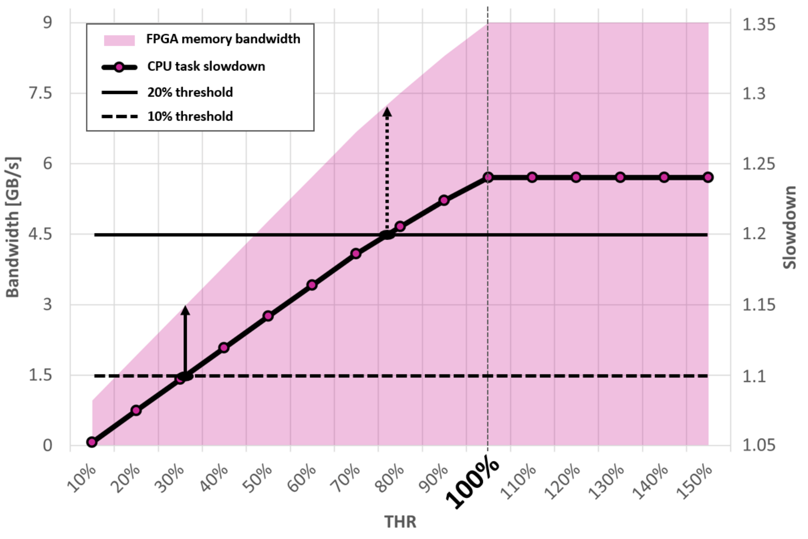


Figure 3 Application slowdown and FPGA memory bandwidth varying the THR$_{\%}$ parameter. The intersections between the slowdown curve and the two horizontal lines determine the maximum tolerated FPGA bandwidth.

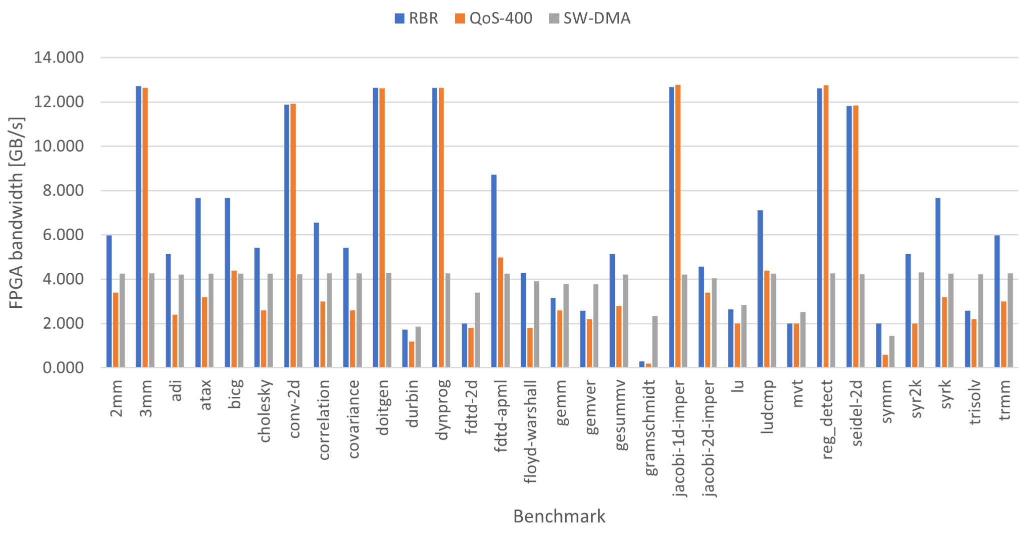
10% and 20% QoS thresholds. This is evident in Figure 5, where we summarize the average memory bandwidth utilization of the 31 benchmark kernels, normalized to the memory bandwidth achieved by the RBR mechanisms. Medium-grained bandwidth regulation (RBR) allows the exploitation of +37.57% and +16.08% more of the memory bandwidth compared to SW-DMA and QoS-400, respectively, for the 10% QoS requirement. A similar improvement is also seen when relaxing the QoS to 20%. In that case, the RBR can allow +35.45% and +8.54% higher memory bandwidth utilization.

This advantage is particularly pronounced when stringent regulation is necessary, such as when FPGA bandwidth falls below 10 GB/s. Looking at individual benchmarks, however, it might come as a surprise that the SW-DMA sometimes offer better residual bandwidth exploitation than QoS-400 or even RBR. This seem to happen in scenarios where the residual bandwidth is low (below 6 GB/s), indicating that the CPU benchmark is very sensitive to interference. It appears that a slower idleness insertion mechanism in these situations can better exploit the residual bandwidth.

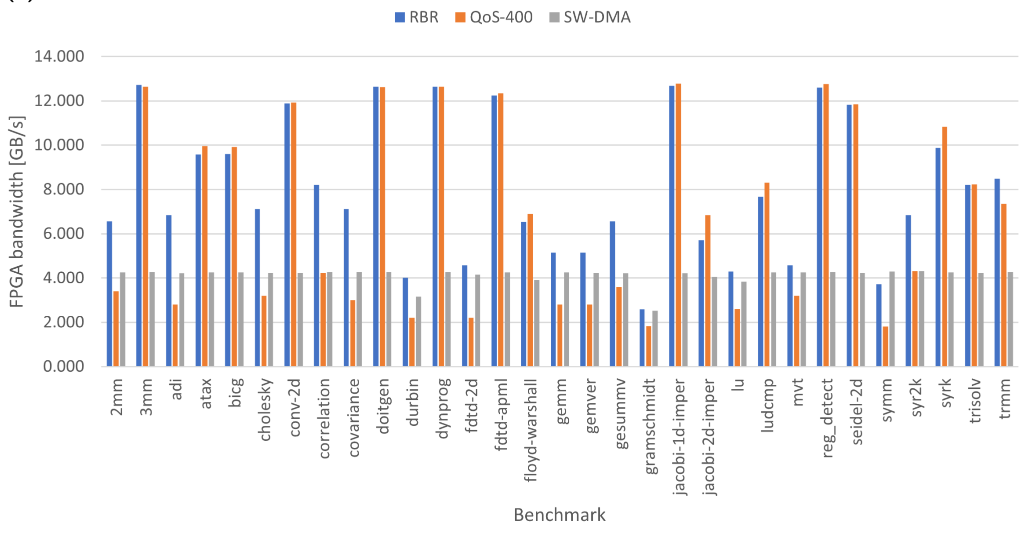
To confirm this intuition we conduct a new experiment, where we focus on a synthetic benchmark, running on the CPU, which performs only DRAM read accesses with a 100% L2 cache miss rate (maximum DRAM bandwidth requirement).

Figure 6 illustrates the usable bandwidth for FPGA accelerators (colored area, left Y-axis) and the respective slowdown experienced by the application under test (black line, right Y-axis) for each regulator type: *fine-grain* QoS-400, *medium-grain* RBR, and *coarse-grain* SW-DMA, as the THR varies (X-axis). Analyzing the FPGA bandwidths at different THR levels, it is evident that the QoS-400 and RBR controllers can both precisely control the amount of DRAM bandwidth used by the FPGA accelerators under identical THR configurations, peaking at 8.6 GB/s. In contrast, the SW-DMA controller can only utilize half of that bandwidth (4.2 GB/s) due to the overhead introduced by the software-based monitoring and control loop between different data transactions.

If we now focus on the impact on the application under test, we can notice that given any THR configuration, the CPU traffic suffers a significantly higher slowdown under QoS-400 regulation compared to RBR. Using RBR the FPGA can exploit up to 1.73 GB/s and 3.16



(a) 10% slowdown increase.



(b) 20% slowdown increase.

Figure 4 FPGA memory bandwidth usage given a maximum tolerated slowdown (X% increase) of each computational kernel of the PolyBench benchmarking suite.

GB/s of DRAM bandwidth at QoS levels of 10% and 20%, respectively. In contrast, the more fine-grained QoS-400, only exploits residual bandwidth of 0.40 GB/s and 1.40 GB/s for the same QoS settings (over 50% worse than RBR). Surprisingly, the coarse-grain controller SW-DMA allows for a higher bandwidth exploitation than the other two controllers, meeting the QoS requirements at 2.35GB/s and 3.81GB/s. This preliminary investigation thus indicates that using the finest bandwidth regulation component (e.g., the QoS-400) is not always beneficial for maximizing memory bandwidth utilization, especially when the CPU has a strict slowdown requirement.

The intuition is that a very fine-grained memory bandwidth regulator imposes smaller regulation intervals on the memory buses, resulting in reduced opportunities for the CPU to interleave software-generated memory transactions on the memory controller. The intuition seems to be confirmed by simply doubling the regulation period of the RBR mechanism and observing that it can use higher residual bandwidth.

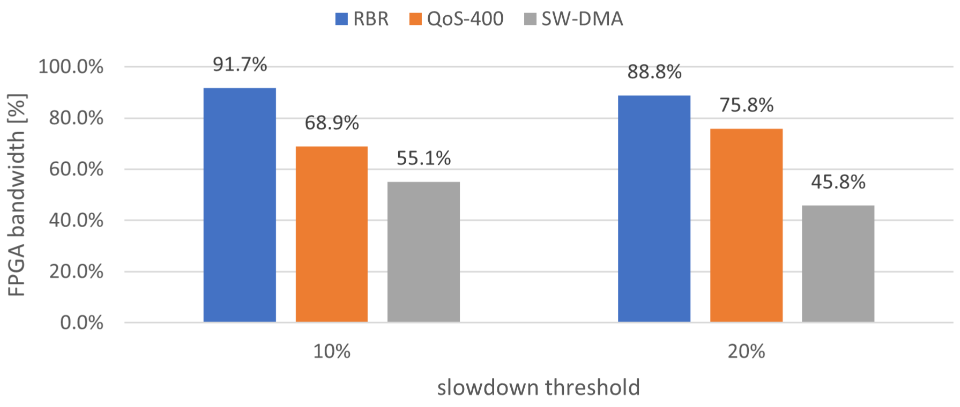


Figure 5 Average FPGA memory bandwidth utilization while the PolyBench applications are subject to 10 and 20% of slowdown increase.

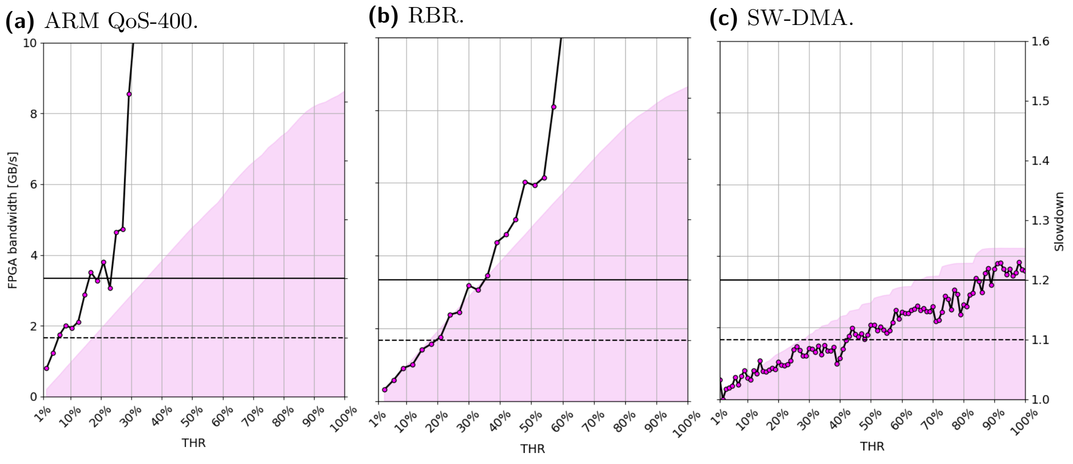


Figure 6 Synthetic benchmark that executes 100% cache read-miss operations, while the FPGA accelerators are regulated using the three different bandwidth regulators.

A coarser granularity has clearly the downside of making the approach slower to adapt to dynamically varying QoS requirements or traffic characteristics [3], so the sweet spot changes across different applications and hardware platforms. This insight opens the door to studying novel, potentially dynamic solutions that exploit different regulation granulates optimized for allowing the system to use the maximum performance from the HeSoC in all operational scenarios (e.g., traffic patterns, QoS levels, etc.), which is the focus of our ongoing work.

5 Conclusion

Bandwidth regulation mechanisms based on integrated monitoring and throttling are being increasingly used both in commercial products and in research, as they allow to mitigate the unpredictable behavior of HeSoCs where several CPUs and accelerators share the main memory. Although previous work has already highlighted that fine-grained bandwidth regulation allows for very precise QoS control and fast adaptation to varying QoS requirements and traffic characteristics, our preliminary investigation indicates that using the finest bandwidth regulation is not always beneficial for maximizing residual memory bandwidth utilization, especially when the CPU has a strict slowdown requirement. Configurable-granularity approaches like RBR can be exploited to devise software policies to exploit the sweet spot between fast control and maximal residual bandwidth exploitation, dynamically adapting at runtime to the characteristics of the workload and target hardware.

References

1 ARM. *CoreLink QoS-400 network interconnect advanced quality of service*, 2016. Accessed: November 5th, 2024. URL: `https://developer.arm.com/documentation/dsu0026/latest/`.

2 Gianluca Bellocchi, Alessandro Capotondi, Francesco Conti, and Andrea Marongiu. A RISC-V-based FPGA Overlay to Simplify Embedded Accelerator Deployment. In *24th Euromicro Conf. on Digital System Design*, pages 9–17, 2021. `doi:10.1109/DSD53832.2021.00011`.

3 G. Brilli, G. Valente, A. Capotondi, P. Burgio, T. Di Masciov, P. Valente, and A. Marongiu. Fine-grained qos control via tightly-coupled bandwidth monitoring and regulation for fpga-based heterogeneous socs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023. `doi:10.1109/DAC56929.2023.10247840`.

4 Gianluca Brilli, Alessandro Capotondi, Paolo Burgio, and Andrea Marongiu. Understanding and Mitigating Memory Interference in FPGA-based HeSoCs. In *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1335–1340, 2022. `doi:10.23919/DATE54114.2022.9774768`.

5 N. Capodieci, R. Cavicchioli, I. S. Olmedo, M. Solieri, and M. Bertogna. Contending memory in heterogeneous SoCs: Evolution in NVIDIA Tegra embedded platforms. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10, 2020. `doi:10.1109/RTCSA50079.2020.9203722`.

6 CAST. *Position Paper CAST-32A Multi-core Processors*, 2016. Accessed: November 21st, 2021. URL: `https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/cast/media/cast-32A.pdf`.

7 Roberto Cavicchioli, Nicola Capodieci, Marco Solieri, Marko Bertogna, Paolo Valente, and Andrea Marongiu. Evaluating Controlled Memory Request Injection to Counter PREM Memory Underutilization. In *Workshop on Job Scheduling Strategies for Parallel Processing*, page 85. Springer, 2020.

8 Farzad Farshchi, Qijing Huang, and Heechul Yun. BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors. In *2020 IEEE Real-Time and Emb. Tech. and Applications Symposium (RTAS)*, pages 364–375, 2020. `doi:10.1109/RTAS48715.2020.00011`.

9 Sergio Garcia-Esteban, Alejandro Serrano-Cases, Jaume Abella, Enrico Mezzetti, and Francisco J. Cazorla. Quasi Isolation QoS Setups to Control MPSoC Contention in Integrated Software Architectures. In Alessandro V. Papadopoulos, editor, *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, volume 262 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:25, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ECRTS.2023.5`.

10 Xiaoyi Ling, Takahiro Notsu, and Jason Anderson. An Open-Source Framework for the Generation of RISC-V Processor + CGRA Accelerator Systems. In *2021 24th Euromicro Conference on Digital System Design (DSD)*, pages 35–42, 2021. `doi:10.1109/DSD53832.2021.00015`.

11 Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. Agile SoC Development with Open ESP: Invited Paper. In *2020 IEEE/ACM Inter. Conf. On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

12 Maxim Mattheeuws, Björn Forsberg, Andreas Kurth, and Luca Benini. Analyzing Memory Interference of FPGA Accelerators on Multicore Hosts in Heterogeneous Reconfigurable SoCs. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1152–1155. IEEE, 2021.

13 Hossein Omidian, Nick Ivanov, and Guy G.F. Lemieux. An Accelerated OpenVX Overlay for Pure Software Programmers. In *2018 International Conference on Field-Programmable Technology (FPT)*, pages 290–293, 2018. `doi:10.1109/FPT.2018.00056`.

14 Andrea Pellegrini. Arm Neoverse N2: Arm's 2 nd generation high performance infrastructure CPUs and system IPs. In *2021 IEEE Hot Chips 33 Symposium (HCS)*, pages 1–27. IEEE, 2021.

15 Francesco Restuccia, Alessandro Biondi, Mauro Marinoni, Giorgiomaria Cicero, and Giorgio Buttazzo. AXI HyperConnect: A Predictable, Hypervisor-level Interconnect for Hardware Accelerators in FPGA SoC. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. `doi:10.1109/DAC18072.2020.9218652`.

16 Alejandro Serrano-Cases, Juan M Reina, Jaume Abella, Enrico Mezzetti, and Francisco J Cazorla. Leveraging hardware QoS to control contention in the Xilinx Zynq UltraScale+ MPSoC. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

17 Parul Sohal, Rohan Tabish, Ulrich Drepper, and Renato Mancuso. E-WarP: A System-wide Framework for Memory Bandwidth Profiling and Management. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 345–357, 2020. `doi:10.1109/RTSS49844.2020.00039`.

18 The Ohio State University. PolyBench-ACC/C the Polyhedral Benchmark suite. `https://github.com/cavazos-lab/PolyBench-ACC`, 2011.

19 H. Wen and W. Zhang. Interference Evaluation In CPU-GPU Heterogeneous Computing. *IEEE High Performance Extreme Computing Conference (HPEC)*, 2017.

20 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Emb. Tech. and Applications Symposium (RTAS)*, pages 55–64, 2013. `doi:10.1109/RTAS.2013.6531079`.

21 Matteo Zini, Giorgiomaria Cicero, Daniel Casini, and Alessandro Biondi. Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms. *Software: Practice and Experience*, November 2021. `doi:10.1002/spe.3053`.

22 Alexander Zuepke, Andrea Bastoni, Weifan Chen, Marco Caccamo, and Renato Mancuso. MemPol: Policing Core Memory Bandwidth from Outside of the Cores. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 235–248, 2023. `doi:10.1109/RTAS58335.2023.00026`.

Invited Paper: Statistical, Stochastic or Probabilistic (Worst-Case Execution) Execution Time? – What Impact on the Multicore Composability

Liliana Cucu-Grosjean
Kopernic team, Inria, Paris, France
StatInf, Paris, France

Abstract

The problem of estimating worst-case execution times of programs on processors has appeared within the context of critical industries like avionics or space. Rapidly adopted by the real-time scheduling community, worst-case execution time estimates of programs or tasks are mandatory to understand the time behaviour of a real-time system. Analyzing such time behaviour is done, often, with an important pessimism due to the consideration of worst-case scenarios. A decreased pessimism has been obtained by understanding that large execution times of a program have low probability of appearance. Probabilistic (worst-case) execution time notion has been proposed. Nevertheless, independence hypotheses makes difficult today to calculate the probabilistic worst-case execution time of a program and current approaches are built, often, on statistical estimators based on the use of Extreme Value Theory or concentration inequalities. Thus, future probabilistic time analyses are expected to consider worst-case execution times estimates obtained by using statistical estimators on measured execution times instead of probabilistic (worst-case) execution times estimations. Within this paper, we discuss the opportunity of differentiating probabilistic (worst-case) execution times from statistical (worst-case) execution times and how dependence between execution times are better or easier captured by each of the definition, while stochastic execution times could be, also, an appropriate alternative.

2012 ACM Subject Classification Computer systems organization → Embedded systems

Keywords and phrases Worst-case execution time, probabilistic analyses, statistical estimator

Digital Object Identifier 10.4230/OASIcs.WCET.2024.6

Acknowledgements The author would like to thank Marwan Wehaiba, Kevin Zagalo, Ismail Hawila, Slim Ben Amor, Benjamin Rouxel and Yves Sorel for discussions on the appropriate definition of execution times for probabilistic real-time systems. The author thanks, also, Ismail Hawila and StatInf, for providing outputs of the statistical WCET estimator tool, RocqStat, for KDBench programs. This paper puts in writing the keynote presented by the author at the Capital workshop 2024 in Toulouse and the author would like, also, to thank participants of this workshop for their feedback that has, clearly, influenced this paper.

1 Introduction and motivation

Originally introduced to answer DO-178B certification requirements, the problem of estimating worst-case execution time (WCET) of programs or tasks has received increased attention from the real-time community. Static methods for the WCET estimation of a program on a processor, analyzing the program without any execution of the program, has been intensively proposed for different processor architectures [12] and dynamic methods requiring execution of programs have received a recent interest due to the arrival of more complex processors [4]. This interest has motivated the introduction of new methods to overcome the possible optimism of dynamic methods. Indeed, while static methods are recognized to

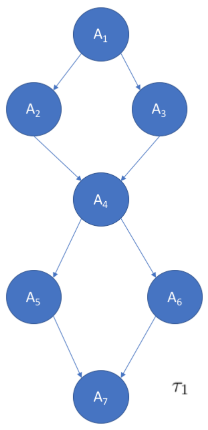


Figure 1 A possible view for a program τ_1.

be safe, but pessimistic, dynamic methods have the reputation of being unsafe since their estimation is based, only, on what it has been observed or measured. Probabilistic worst-case execution time has been proposed within this context by Bernat and Petters [1] in order to underline that based on some probabilistic reasoning, the observed execution times obtained by dynamic methods are enriched in order to achieve safeness. As such, this definition generalizes the static WCET definition but it comes with a strong hypothesis: probability distributions at some granularity level of programs, e.g., basic blocks, should be independent or any combinations of them should include a safe description of possible dependence between those basic blocks. We understand here by basic blocks, linear sequences of instructions, without any branch. One may note that this is different from the problem of dependence between probability distributions describing probabilistic execution times of two instances of the same program or instances of different programs as described in [2], even if a relation does exist between them (see Section 3). Indeed, probabilistic worst-case execution definition has appeared as the result of probabilistic operations which are done in order to obtain the probabilistic distribution of a (worst-case) execution time for a program. For instance, if one considers a program τ_1 with its basic blocks $A_1, A_2, A_3, \cdots, A_7$ as described in Figure 1, then probabilistic operations (convolutions, dominance relation for instance) between the probability distributions of those basic blocks could be done to propose a probabilistic (worst-case) execution time estimation for the program τ_1. Another alternative is building or calculating a dominance relation between two probability distributions, which is possible in absence of any particular mathematical properties, while the convolution between the probability distribution of A_1 and the $max(A2, A3)$ requires to understand the dependence that may exist between them. Such understanding may be built on input variables the two basic blocks share but also on states at which the execution of A_1 leaves processor features after its execution. Within this paper and for the sack of simplicity, we focus on the variation of input variables and/or the existence of several cores.

Our main contributions are the following:

- we provide discussions on the fact that probabilistic (worst-case) execution times and statistical (worst-case) execution times should co-exist. Moreover, we discuss the opportunity of using the identically distributed hypothesis within the context of probabilistic (worst-case) schedulability analyses:

- we underline the misleading relation between dependence hypothesis at basic block level and at the instances of a program and conclude on the possibility that new execution time models are needed to include both types of dependence;
- we propose a first discussion on how statistical WCET estimators may bring time composability to the multicore problem.

Organization of the paper. We provide in Section 2 main definitions and notations used to introduce our contribution. In Section 3 we compare probabilistic (worst-case) execution time and statistical (worst-case) execution time definitions as well as the place of identically distributed and independent hypotheses. In Section 4, we provide hints on how statistical WCET estimators are good candidates for the multicore problem.

2 Notations and related work

Within this paper, we consider a set n programs (or tasks) τ_i executed on a processor π[1]. A program or task τ_i, $\forall i \in \{1, 2, .., n\}$ is defined by C_i its execution time defined by a cumulative distribution function F_{C_i} (see Equation (1)), where $\Omega_0 = Q \times I$ is the product space between Q is the set of possible states of the processor π and I is the set of all possible input values of tasks τ_i [13].

$$F_{C_i}(c) = P_{C_i}((-\infty, c)) = P(\omega_0 \in \Omega_0 : C_i(\omega_0) \leq c) \tag{1}$$

One may underline that $F_{C_i}(c)$ defines the probability for the execution time C_i to be smaller than c. Indeed, within the real-time community, one is interested in the exceedance function $1 - F_{C_i}$ which defines the probability for the execution time C_i to be larger than c. Such exceedance function is, often, addressed as the probabilistic (worst-case) execution time of a task or program. One may consider the obtention of such function by two main classes of methods:

- probabilistic approaches - they combine information at some granularity level, e.g., basic blocks. For instance, for the program τ_1 introduced in Section 1 (see Figure 1), one may calculate this distribution using the distributions of all basic blocks under appropriate mathematical hypotheses;
- statistical approaches - they use statistical estimators on measured execution time at some granularity level, e.g., between the beginning and the end of the program τ_1.

Few results [4] are proposed for the first class of approaches as they require an important understanding on how the probability distribution of a basic block has an impact on the probability distributions of another basic block. This first method has been proposed within the literature together with a definition for the probabilistic worst-case execution time [1] and this definition has been, often, used when execution times are described by exceedance functions. Nevertheless, in [1], no independence, nor identically distributed hypothesis is discussed. The second class of approaches [6] has been proposed in parallel with [1], but the authors do not introduce a definition for the freshly proposed notion of probability for the execution time to exceed a given value. They do provide theoretical bases for an important existing observation from [11], the distribution of execution times is heavily tailed, i.e., the probability of appearance of large values for the execution times are low and large values

[1] For the sake of the simplicity, we consider a simple processor as more complex architectures do not modify the conclusions of our paper

are much larger than average ones. In parallel, a schedulability analysis proposed for a set of tasks with independent and identically distributed (i.i.d) random variables describing execution times is proposed [5]. Its strong i.i.d. hypothesis is required to prove a stationarity property allowing to conclude on the schedulability of the system. This full i.i.d hypothesis is, often, used within schedulability analyses by more recent authors, without necessarily being used entirely. Indeed, while the independence hypothesis is required to operate a convolution, there is no need for two distributions to be identically distributed for such convolution to be operated [3]. Usually the identically distributed hypothesis is required to prove or to use convergence results mainly in statistics or stochastic processes. More precisely, one important and correct use of the identically distributed hypothesis is done within the application of statistical estimators on ordered sequences of measured execution times obtained by following some measurement protocols. Within this context, we resume below latest results on statistical approaches as proposed in [7]. The WCET of a program or task τ_i is defined as its largest execution time for any valid execution scenario S. During each scenario S_j, execution times are collected as ordered sequences of execution times. Statistical estimators are applied on these sequences and i.i.d properties are checked. In [7], an Extreme Value Theory estimator[2] is applied to sub-sequences where a sub-sequence contains i.i.d. execution times. Finally, the WCET estimation is obtained by building an envelop on all sub-sequences WCET estimations. Such mathematical operation introduces **a time composability between sWCET estimations** of different sub-sequences of execution times. One may notice that these sub-sequences correspond to different execution modes.

When dealing with dynamic approaches for WCET estimation of a program on a processor, one may be interested in identifying the paths within the program that produces an execution time. Following the definition proposed in [7], a path P_j of a program τ_i is critical with respect to a pWCET estimation of A if, at least, one measurement of the execution of that path appears within the sub-sequence contributing to the pWCET estimate of A. Always in [7], the authors define a domination relation that formally underlines that a path within one execution scenario may produce execution times that participate to the WCET estimation, while within another execution scenario, this is not the case. For instance, in Figure 2, we consider an ordered sequence (from the left to the right of the figure) measured during a simulated flight for the Sensors program (KDBench programs, more details in [9]). Execution times obtained by exercising the same paths are colored with the same color while execution times within two consecutive vertical lines constitute a sub-sequence on which the statistical estimator is applied.

In this paper and in order to distinguish between WCET estimations obtained by using probabilistic approaches and WCET estimations obtained by using statistical approaches, we call the second ones as the statistical worst-case execution time (sWCET) estimates. In Figure 3, we illustrate the sWCET estimation obtained as an envelop built on top of sWCET estimations obtained for each sub-sequence given in Figure 2.

3 Two definitions and one identically distributed and independent hypothesis within the context of estimating (worst-case) execution times and probabilistic schedulability analyses

In this section, we propose a deeper discussion on hypotheses of independent and identically distributed and their relation with pWCET and sWCET notions as well as their impact on a schedulability analysis.

[2] An interested reader may find more details on such estimators in [4].

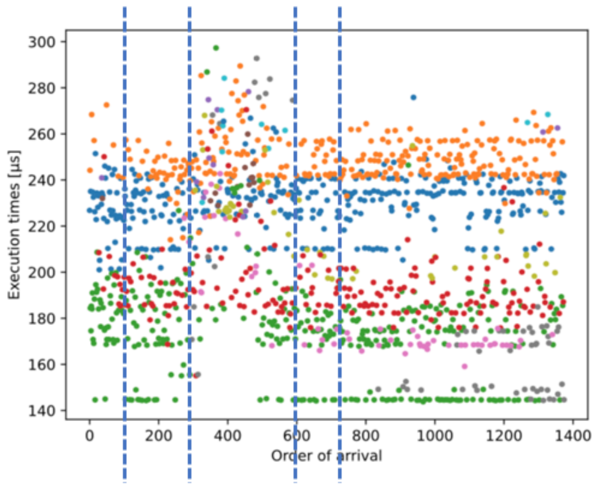


Figure 2 An ordered sequence of execution times for the Sensors program (KDBench programs [8]).

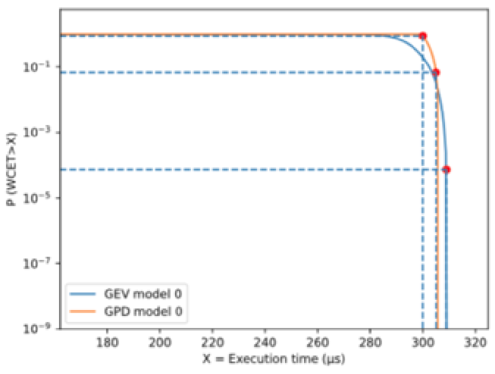


Figure 3 sWCET illustration obtained as an envelope (KDBench programs, see [9]).

Probabilistic approaches do not require any identically distributed hypothesis among basic blocks (or other parts of the program) to produce the pWCET of that program, but they require probability distributions of those basic blocks to be independent, or if not, one has to describe the dependence between these blocks. If we move at a higher level, then a schedulability analysis requires to understand the dependence relation between pWCET of different tasks or their instances. Is a probabilistic approach able to provide such understanding? With our current understanding of the literature, the answer is negative. Indeed, probabilistic approaches are static analysis-based and they do not provide information on different instances of a program, e.g., what path is executed at some time instant. Let us consider the set of tasks $\tau = \{\tau_1, \tau_2, \tau_3\}$, where the internal structure of τ_1 is illustrated in Figure 1, of τ_2 in Figure 4 and of τ_3 in Figure 5.

We consider now a possible schedule illustrated in Figure 6, where exercised paths are colored in green. A pWCET estimation does not distinguish between different paths, neither of different execution contexts. Actually, if one looks at path level, then it does not exist any current probabilistic schedulability analysis considering paths to be associated with some program instances (or jobs). Indeed, considering dependence relations between consecutive

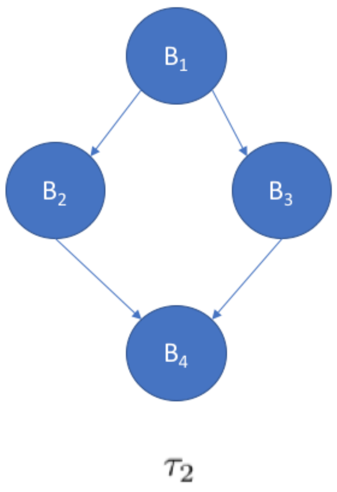


Figure 4 A possible view for a program τ_2.

execution times requires, also, to understand if exercising $B_1 - B_3 - B_4$ in τ_2 increases the probability for τ_3 to exercise $C_1 - C_2 - C_4 - C_5$, for instance. pWCET estimations as defined today within the literature do not include such information. Moreover, the independence hypothesis at basic block level does not help to advance towards estimating inter-programs or inter-instances dependence.

Coming back to the identically distributed hypothesis, its utilization for integrating pWCET estimations within a schedulability analysis is not necessary to provide correct results. As underlined within the introduction, the convolution between probability distributions does not need this mathematical property. Looking at realistic executions, while different paths do indicate the existence of multimodal distributions [14][3], the static analysis-based reasoning of pWCET estimators do not allow to differentiate among these paths and consecutive instances of the same program may have different distributions if they exercise different paths.

Statistical approaches may require the i.i.d. hypothesis for sequences of measured execution times in order to obtain a sWCET estimated. For instance, in Figure 2 i.i.d sub-sequences of execution times are considered in order to apply statistical estimators only to the execution times within the same sub-sequence. Finally, the sWCET may be obtained by building a probability distribution upper-bounding all sWCET estimations obtained per sub-sequence. Thus, a dominance relation is built within the final sWCET estimation. To the best of our knowledge, using such sWCET within a schedulability analysis has never been considered in the literature. Even if pWCET and sWCET provide a probability distribution to the schedulability analysis, their estimations require different level of information on the variation of execution times. Since sWCET estimation is expected to consider as input, execution times obtained in real execution conditions, then the sequences of execution times are measured with respect to a given scheduling algorithm. The evolution of execution times do capture dependence relations introduced by the scheduling algorithm. While the paths exercised by a program during its execution are, mainly, imposed by the variation of input variables of that program, the variation of execution times per path may be impacted by the choice of the order in which programs are executed.

[3] We introduce intuitively the notion of multimodal distribution as a distribution with several peaks

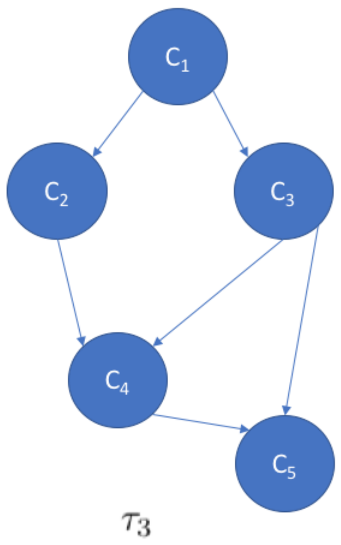


Figure 5 A possible view for a program τ_3.

If one wants to capture this evolution and introduce it within a schedulability analysis, stochastic processes are an alternative - existing results as [10] (to cite the latest, to the best of our knowledge) are promising and one may consider what is a stochastic (worst-case) execution and its relation to pWCET and sWCET definitions. Moreover, a formal pWCET definition provided in [2] indicates that stochastic processes fulfill hypotheses allowing to built correct schedulability analysis.

4 From single to multicore processors

While the probabilistic WCET estimation has received important attention in the case of programs executed on one core processor, the statistical WCET estimation is, in our opinion, a more promising candidate for programs executed on multicore processors or in presence of operating systems. The main limitation is the strong pWCET hypothesis of independence between probability distributions describing the execution time of basic blocks. By adding new interference sources, the multicore case increases the complexity of describing such dependence relations. In the case of sWCET estimators, dependence relations between consecutive executions of a program does help to detect different sub-sequences of execution. These sub-sequences identify different execution modes that could be provoked either by the execution of different paths, but also by the evolution of hardware states like multicore interferences. In Figure 7, the execution times of a program are obtained on a 4 cores processor with a variation of the execution time dependent on an input variable. Each core has local cache (data and instruction) memories and there is one global cache data memory that is shared by all cores. The execution times are presented from the left to the right in the order of their measurement, while on the vertical axis, the values of execution times are provided in cycles.

We underline the existence of 4 groups of execution times - the lower group is obtained when only the core executing the program is active, the second from the bottom is obtained when a second core is active, etc. Within the same group, execution times are obtained respectively, without active cache memory, with active cache instruction memory and with active cache data memory. The highest value is obtained in presence of cache memory shared

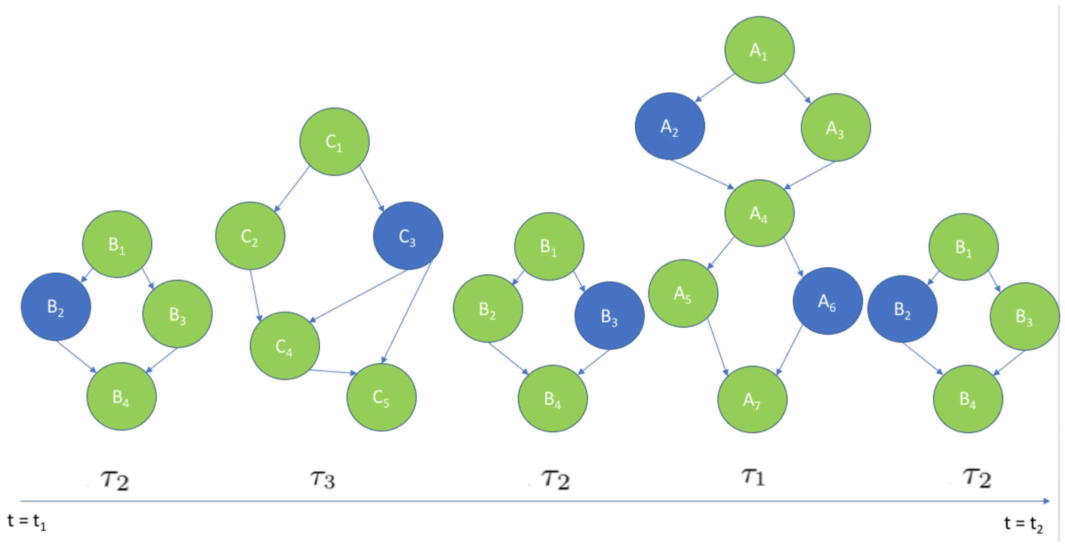


Figure 6 A schedule from t_1 to t_2 of programs τ_1, τ_2 and τ_3 and their paths. The exercised paths are illustrated in green.

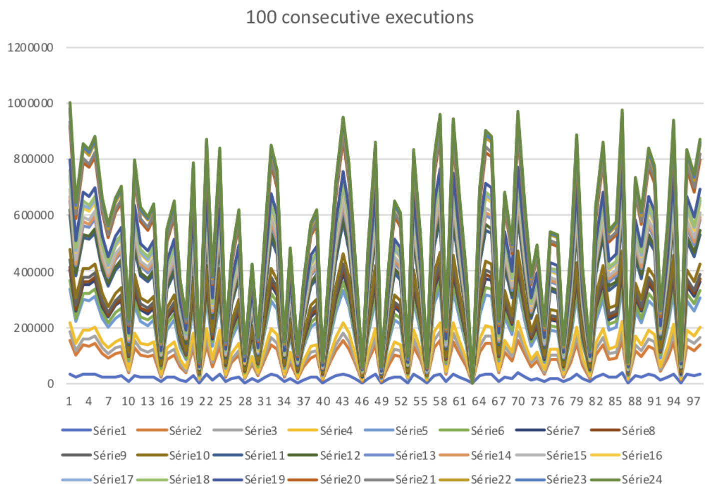


Figure 7 The evolution of execution times for a program executed 100 times with the same sequence of input variables and the execution time is dependent of the variation of one input variable illustrated in blue.

by all cores. Since the processor does not include pipelines, nor branch predictors, the execution times are ordered in layers and a statistical estimator provides sWCET estimations with a nice dominance property. In reality, these layers of execution times may cross each other and the sWCET estimations per core could be visualized as in Figure 8. Comparing the envelop built per core is a possible way to estimate the penalty of a core but the most important is that building a global envelop ensures **a time composability between sWCET estimated for a program executed in the presence of several cores**. Considering stochastic WCET estimation for programs executed on several cores is an open problem, but its common hypotheses with the sWCET estimation is promising. Including such estimations (stochastic or statistical) within multicore schedulability analysis is an open problem.

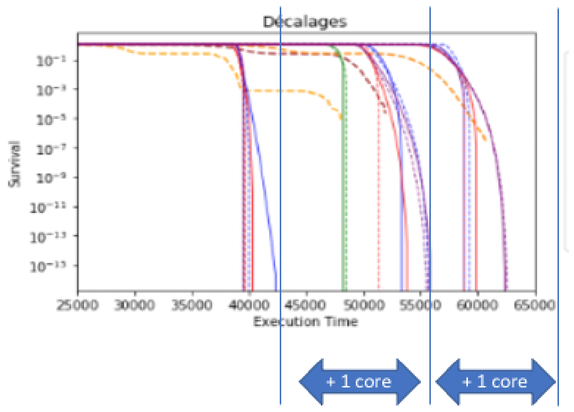


Figure 8 SWCETs cross each other and the envelop per core allows to calculate the penalty for each new core.

5 Conclusions

In this paper, we propose a discussion on how three definitions for describing the probability that the execution time is larger than a given value have been introduced within the real-time community. Understanding their estimation is an important step towards their correct integration within higher-level time analysis and their hypotheses may prevent some of them from such integration. We conclude the paper by presenting hints on how sWCET estimations may provide a time composability answer to the WCET multicore estimation problem.

References

1 Guillem Bernat, Antoine Colin, and Stefan M. Petters. pWCET, a tool for probabilistic WCET analysis of real-time systems. In Jan Gustafsson, editor, *Proceedings of the 3rd International Workshop on Worst-Case Execution Time Analysis, WCET 2003*, volume MDH-MRTC-116/2003-1-SE, pages 21–38, 2003.

2 S. Bozhko, F. Markovic, G. von der Bruggen, and B. Brandenburg. What really is pWCET? In *2023 IEEE Real-Time Systems Symposium (RTSS)*, pages 13–26, 2023.

3 L Cucu-Grosjean. Independence-a misunderstood property of and for probabilistic real-time systems. *Real-Time Systems: the past, the present and the future, the Alan Burns 60th anniversary*, pages 29–37, 2013.

4 R. I. Davis and L. Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *LITES*, 6(1):03:1–03:60, 2019.

5 José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02),*, pages 289–300. IEEE Computer Society, 2002.

6 S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium*, 2001.

7 Marwan Wehaiba el Khazen, Slim Ben-Amor, Kossivi Kougblenou, Adriana Gogonel, and Liliana Cucu-Grosjean. Work in progress: Towards a statistical worst-case energy consumption model. In *29th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2023*, pages 333–336. IEEE, 2023.

8 Marwan Wehaiba el Khazen and et al. Work in progress: Kdbench - towards open source benchmarks for measurement-based multicore WCET estimators. In *28th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 309–312. IEEE, 2022.

9 Marwan Wehaiba el Khazen, Kevin Zagalo, Hadrien Clarke, Mehdi Mezouak, Yasmina Abdeddaïm, Avner Bar-Hen, Slim Ben-Amor, Rihab Bennour, Adriana Gogonel, Kossivi Kougblenou, Yves Sorel, and Liliana Cucu-Grosjean. Work in progress: Kdbench - towards open source benchmarks for measurement-based multicore WCET estimators. In *28th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 309–312. IEEE, 2022.

10 Anna Friebe, Filip Markovic, Alessandro Vittorio Papadopoulos, and Thomas Nolte. Continuous-emission markov models for real-time applications: Bounding deadline miss probabilities. In *29th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 14–26. IEEE, 2023.

11 Mark K. Gardner and Jane W.-S. Liu. Analyzing stochastic fixed-priority real-time systems. In *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99*, volume 1579 of *Lecture Notes in Computer Science*, pages 44–58. Springer, 1999.

12 R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G.Bernat, C. Ferdinand, R.Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, G. Staschulat, and P. Stenströem. The worst-case execution time problem: overview of methods and survey of tools. *Trans. on Embedded Computing Systems*, 7(3):1–53, 2008.

13 Kevin Zagalo, Yasmina Abdeddaïm, Avner Bar-Hen, and Liliana Cucu-Grosjean. Response time stochastic analysis for fixed-priority stable real-time systems. *IEEE Trans. Computers*, 72(1):3–14, 2023.

14 Kevin Zagalo, Liliana Cucu-Grosjean, and Avner Bar-Hen. Identification of execution modes for real-time systems using cluster analysis. In *25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 1173–1176. IEEE, 2020.

Machine Learning for Timing Analysis: The Good, the Bad and the Ugly

Isabelle Puaut
Univ. Rennes, INRIA, CNRS, IRISA, France

Abstract

The microarchitecture of processors is becoming increasingly complex and less documented, making the design of timing models for WCET calculation increasingly complicated, if not impossible. We have recently experimented with the use of machine learning techniques (ML) to predict the WCET of basic blocks [3, 1, 2, 4, 5]. Predicted WCETs can then be integrated into static WCET calculation tools, resulting in a hybrid WCET calculation.

In this keynote, we present our experience using ML for WCET calculation, across a range of architectures, from very simple ones (MSP430, Cortex M4) to more complex architectures. Rather than presenting only what worked, we also discuss in this keynote the bad, and even very bad, surprises encountered during the process, and how we overcame (most of) them.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Computer systems organization → Real-time systems; Computing methodologies → Machine learning

Keywords and phrases Worst-Case Execution Time (WCET) estimation, Machine Learning, Explainable ML models

Digital Object Identifier 10.4230/OASIcs.WCET.2024.7

Category Invited Talk

References

1 Abderaouf N. Amalou, Élisa Fromont, and Isabelle Puaut. CATREEN: context-aware code timing estimation with stacked recurrent networks. In Marek Z. Reformat, Du Zhang, and Nikolaos G. Bourbakis, editors, *34th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2022, Macao, China, October 31 – November 2, 2022*, pages 571–576. IEEE, 2022.

2 Abderaouf N Amalou, Elisa Fromont, and Isabelle Puaut. Cawet: Context-aware worst-case execution time estimation using transformers. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

3 Abderaouf N Amalou, Isabelle Puaut, and Gilles Muller. We-hml: hybrid wcet estimation using machine learning for architectures with caches. In *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 31–40. IEEE, 2021.

4 Abderaouf Nassim Amalou. *Machine learning for timing estimation. (apprentissage automatique pour l'estimation du temps d'exécution)*. PhD thesis, University of Rennes 1, France, 2023. URL: `https://tel.archives-ouvertes.fr/tel-04406029`.

5 Abderaouf Nassim Amalou, Elisa Fromont, and Isabelle Puaut. Fast and accurate context-aware basic block timing prediction using transformers. In Gabriel Rodríguez, P. Sadayappan, and Aravind Sukumaran-Rajam, editors, *Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction, CC 2024, Edinburgh, United Kingdom, March 2-3, 2024*, pages 227–237. ACM, 2024.