Science of Computer Programming 111 (2015) 505–534

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Certifying execution time in multicores

Vítor Rodrigues [a,*], Benny Akesson [b], Mário Florido [c], Simão Melo de Sousa [d], João Pedro Pedroso [e], Pedro Vasconcelos [c]

[a] Rochester Institute of Technology, Computer Science Department, New York, United States
[b] Faculty of Electrical Engineering, Czech Technical University, Prague, Czech Republic
[c] DCC-Faculty of Sciences & LIACC, University of Porto, Portugal
[d] DI-University of Beira Interior & LIACC, University of Porto, Portugal
[e] DCC-Faculty of Sciences & INESC TEC, University of Porto, Portugal

ARTICLE INFO

Article history:
Received 26 August 2013
Received in revised form 15 June 2015
Accepted 16 June 2015
Available online 23 June 2015

Keywords:
Abstract interpretation
Abstraction-carrying code
WCET
LP
$\mathcal{LR}$-servers

ABSTRACT

This article presents a semantics-based program verification framework for critical embedded real-time systems using the *worst-case execution time* (WCET) as the safety parameter. The verification algorithm is designed to run on devices with limited computational resources where efficient resource usage is a requirement. For this purpose, the framework of *abstract-carrying code* (ACC) is extended with an additional verification mechanism for *linear programming* (LP) by applying the certifying properties of *duality theory* to check the optimality of WCET estimates. Further, the WCET verification approach preserves feasibility and scalability when applied to multicore architectural models.

The certifying WCET algorithm is targeted to architectural models based on the ARM instruction set and is presented as a particular instantiation of a compositional data-flow framework supported on the theoretic foundations of *denotational semantics* and *abstract interpretation*. The data-flow framework has algebraic properties that provide algorithmic transformations to increase verification efficiency, mainly in terms of verification time. The WCET analysis/verification on multicore architectures applies the formalism of *latency-rate* ($\mathcal{LR}$) servers, and proves its correctness in the context of abstract interpretation, in order to ease WCET estimation of programs sharing resources.

1. Introduction

The design of embedded real-time systems is, in general, guided by some *timeliness* criteria. Timeliness in embedded real-time systems means that programs have operational deadlines, i.e. strict run-time constraints, and that the system must guarantee such requirements to ensure *safety*. Timeliness evaluation is performed at hardware level and is defined as the system ability to assure that execution deadlines are met at all times. Therefore, when the risk of failure, in terms of system responsiveness, may endanger human life or substantial economic values [21], the determination of upper bounds for the execution times of programs becomes a safety requirement.

The timeliness safety criteria is most commonly specified by the *worst-case execution time* (WCET) [74]. This timing property is an over-approximation of the execution time of the path inside the program that takes the longest to execute. In general, the particular input data that causes the exact WCET is unknown at compile time. Therefore, the exclusive use

* Corresponding author.
E-mail address: vgrcs@rit.edu (V. Rodrigues).

http://dx.doi.org/10.1016/j.scico.2015.06.006
0167-6423/

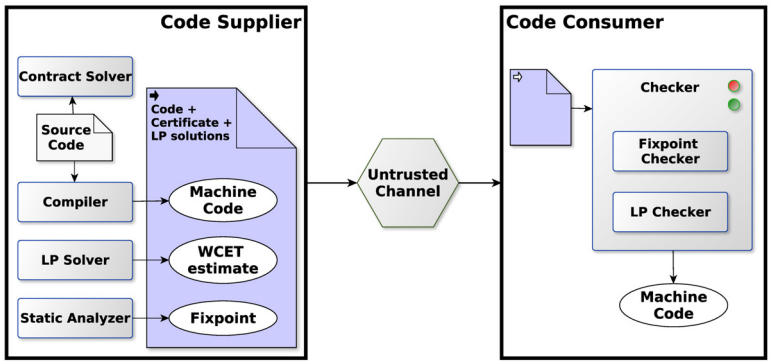

Fig. 1. Overview of the extended ACC certifying platform.

of measurement-based techniques to determine the WCET for *any* possible run of the program is not feasible in terms of computational cost (exception made, for example, to straight-line programs, for which input data never changes execution order). Alternative solutions use *static analysis* methods to guarantee sound estimates of the WCET in finite time [73]. Nonetheless, *state of the art* WCET tool suites like `aiT` [1] and `Otawa` [11] are conservative by nature and may require manual intervention in order to predict tight WCET estimates. Also, end-to-end measurement-based approaches can be combined with game-theoretic learning approaches using SMT solvers in order to generate predictable WCET estimates based on probabilistic guarantees [63].

In addition, embedded real-time systems often require incremental updates, where critical "patches" may be required after deployment [6]. Traditionally, this is done using manual and heavyweight processes, specifically dedicated to a particular modification. However, incremental updates of real-time systems can only be achieved if the system design abandons its traditional monolithic and closed conception. We propose a novel approach to an ideal scenario, where the WCET analysis is complemented with a *verification mechanism*, whose task is to verify whether the computed WCET estimate is compliant with safety requirements of an embedded real-time system.

State of the art WCET analyzers like `aiT` [1] make use of the theoretical foundations of *abstract interpretation* [19] combined with *linear programming* [55]. Although such type of tool suites excel in computing tight and precise WCET estimates using real-world hardware models, they do not easily fit to the task of formal WCET verification. The reason is that, here, the focus needs to be put more on the search of highly efficient mechanisms for WCET *checking*. Although it is not our objective to reproduce the quality of state-of-the-art WCET analysis, we have designed a comprehensive WCET tool prototype,[1] based on declarative programming, where the underlying concepts of abstract interpretation can be elegantly implemented [57]. The analysis is restricted to source-to-assembler code compiled for the ARM target architecture.

We perform a *flow-sensitive*, *path-sensitive* and *context-sensitive* timing analysis. Apart from the induction of abstract interpreters that perform *value* and *cache* static analysis based on state-of-the-art domains [57], in this article we focus on a new approach to *pipeline analysis* that can be parametrizable by the timing model of a generic processor. Sound upper bounds of execution time are computed as the combined result of a simultaneous *value*, *cache* and *pipeline analysis*. Along the lines of [1], estimates of the *worst-case* execution time of the program are computed *a posteriori* by a *path analysis* using linear optimization.

Despite the common use of complex hardware features to increase instruction throughput, most embedded systems have limited computing resources. Therefore, mechanisms for WCET verification face new challenges due to the design complexity of WCET estimation. For this reason, the computational burden resulting from the integration of the complete WCET toolchain into the *trusted computing base* (TCB) of embedded systems with real-time constraints would be unacceptable. Well-known solutions that address this issue are Proof-Carrying Code (PCC) [50], Typed-Assembly Languages (TAL) [48] and Abstraction-Carrying Code (ACC) [9].

The main objective of this article is to present a verification mechanism that efficiently checks if a program satisfies a given safety specification in terms of WCET [60]. Along the lines of the previously mentioned approaches, we propose a lightweight and standalone method, which does not depend on a third-party certifying entity to monitor the transmission of one program through an "untrusted" communication channel. We extend the existent ACC framework with an efficient mechanism to check the solutions of the linear optimization problem. This mechanism is illustrated in Fig. 1 by the component "LP Checker".

The verification mechanism uses *fixpoint theory* [40] to check the least fixpoint solution of the static analyzer, complemented with *duality theory* [47] applied to the *simplex method* [36]. The application of the verification mechanism to multicore architectures is based on the sound abstractions of the concrete timing model provided by the formal model of $\mathcal{LR}$-servers [67].

[1] Available at https://github.com/esmifro/wcetac.