Towards a Worst-Case Execution Time Calculation Platform with Certificate Production

Diogo Fialho, Nuno Gaspar and Simão Melo de Sousa
RELEASE - Reliable And Secure Computation Group
University of Beira Interior
Covilhã, Portugal
{*dfialho, nmpgaspar, desousa*}*@di.ubi.pt*

Rogério Reis
DCC-FC & LIACC
University of Porto
Porto, Portugal
rvr@ncc.up.pt

Jorge Sousa Pinto
DI/CCTC
University of Minho
Braga, Portugal
jsp@di.uminho.pt

***Abstract*—In real-time systems, time constraints must be satisfied in order to guarantee that deadlines will be met. The calculation of each task's worst-case execution time (WCET) is a prerequisite the for schedulability analysis, and hence of paramount importance for real-time systems. This can be difficult if the underlying hardware architecture possesses features like caches and pipelines.**

In this position paper we present our ongoing work towards an *Abstraction-Carrying Code* based platform for the derivation of the WCETs. Our approach has as starting point an extension of the C programming language with annotations that express the intended time behavior. Moreover, by placing these annotations directly into the source code, we are also able to express valuable informations for the subsequent WCET analysis, such as infeasible paths and loop bounds. Being one of the most used architectures in embedded systems, our platform targets the ARM processor.

Furthermore, in order to minimize the *trusted computing base*, we produce a checkable certificate whose validity entails compliance with the calculated WCET.

***Keywords*-real-time systems; worst-case execution time; abstract interpretation; abstraction-carrying code; fixpoint computation;**

I. Introduction

Real-time systems can be seen as a set of tasks, that are expected to perform some functionality. In general, in order to ensure a correct system behavior (schedulability analysis), an upper bound for the worst-case execution time (WCET) of each task is necessary.

In order to improve performance, modern processors include features like caches and pipelines which make instruction's execution time context dependent, increasing the difficulty of static timing analysis. One could be tempted to always assume the local worst case scenario (e.g. cache miss) in order to obtain safe predictions, but two problems could arise of such approach. First, it could lead to an excessive over-approximation of the actual WCET, resulting in a waste of hardware resources. Second, due to *timing anomalies* [1], the obtained prediction could in fact be unsafe (an under-approximation).

The determination of safe and tight upper bounds for the WCET has been the object of intensive study in the literature [2], yet, to the best of our knowledge, there is still no attempt to provide some insurances w.r.t. the correctness of the predicted WCET.

This work is partially supported by the RESCUE project PTDC/EIA/65862/2006 funded by FCT (Fundação para a Ciência e a Tecnologia)

II. Approach

The proposed platform is depicted in figure 1. We begin by extending the C programming language with annotations that define the intended time properties for each function. The time specification of the main function is of most importance, however one may also want to define some constraints on the auxiliary functions. Moreover, by placing these annotations directly into the source code, we are also able to express valuable informations for the subsequent WCET analysis, such as infeasible paths and loop bounds [3]. This is achieved by the use of a WCET-aware compilation process that preserve the annotations semantics.

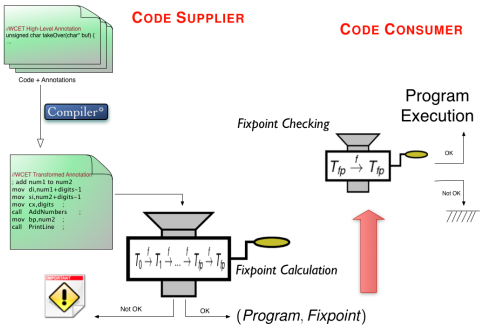

Figure 1. Abstraction-Carrying Code based WCET Platform

Only at the hardware level one can accurately calculate the WCET, but feedback about the compliance of the given time specification should be done at source-code level. Hence, in order to perform time validation w.r.t. the functions' time specification, we perform the WCET analysis at machine-code level, taking into account the effects of the hardware

specific features, and alert any non-compliance through to use of *back annotations* [4]. This is what the bottom left area of Figure 1 represents.

We denote the set of execution times of the program $\mathcal{P}$, by $[\![\mathcal{P}]\!]$. The problem of verifying the compliance of the given time specification can be formulated as follows:

$$\mathcal{P} \textit{ respects the time specification } \mathcal{I} \textit{ if } [\![\mathcal{P}]\!] \subseteq \mathcal{I},$$

where $\mathcal{I}$ stands for intended time behavior, i.e., the set of accepted execution times. The idea is to express the *collecting semantics* [5] of $\mathcal{P}$ as the fixpoint of a set of recursive equations.

In general, the state space to be considered is too large to exhaustively explore all possible executions, some abstraction of the application domain is required in order to make the time analysis feasible.

A. Abstract Interpretation

In the *abstract interpretation* [6] framework a program $\mathcal{P}$ is interpreted over a simpler *abstract domain* $\mathcal{D}_\alpha$. This abstract domain permits to trade efficiency over precision, i.e., although it is an approximation, by computing the fixpoint over this abstract domain, we will be able to produce precise, yet safe, (over-)approximations of the collecting semantics.

We denote by $[\![\mathcal{P}]\!]_\alpha$ the result of the fixpoint calculation over the abstract domain. Moreover, since comparison between actual and intended semantics is most easily done in the same domain, we assume that the intended time specification is also given in the abstract domain, i.e., $\mathcal{I}_\alpha \in \mathcal{D}_\alpha$.

The problem of verifying the compliance with the given time specification can now be reformulated as follows:

$$\mathcal{P} \textit{ respects the time specification } \mathcal{I}_\alpha \textit{ if } [\![\mathcal{P}]\!]_\alpha \subseteq \mathcal{I}_\alpha$$

B. Abstraction-Carrying Code

In the context of a code producer/code consumer software distribution model, i.e., in order to ensure the correctness of our calculation to a program consumer, we produce a certificate (or *proof*) whose validity entails compliance with the calculated WCET. Our methodology is based on the work introduced by Hermenegildo et al: *Abstraction-Carrying Code* [7].

The produced certificate allows a program consumer to locally check the calculated WCET, avoiding a blind confidence on the producer side. This checking operation is much more efficient than the certificate production process. While on the producer side one has to compute a fixpoint in a complex and possibly long iterative process, on the consumer side, a one pass process is enough to confirm that the certificate is indeed a fixpoint.

III. Conclusion

Abstract Interpretation has been widely used in the industry, being static timing analysis one of its most successful applications [5]. In our approach we also use the Abstract Interpretation framework as the underlying technique, however we obtain our WCET prediction by explicitly following a standard fixpoint computation strategy [8]. This fixpoint computation will allow us to infer an abstract model of the program, which can then be used as a certificate, i.e., a program consumer can locally validate the predicted WCET by simply checking that this abstract model is indeed a fixpoint (a one-pass process).

We presented our architecture proposal and ongoing work towards a platform for the calculation of WCETs with the production of certificates. We focus ourselves in the ARM architecture as target processor. By performing the WCET analysis on the resulting ARM binary, accounting with the hardware specific features (caches, pipelines, etc) behavior, we are able to verify if the program will comply with the given time specification.

To the best of our knowledge this is the first work applying the concepts of Abstraction-Carrying Code to the static timing analysis field.

References

[1] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker, "A definition and classification of timing anomalies," in *6th Intl Workshop on WCET Analysis*, 2006.

[2] R. Wilhelm and et al, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 1–53, 2008.

[3] R. Kirner, J. Knoop, A. Prantl, M. Schordan, and I. Wenzel, "Wcet analysis: The annotation language challenge," in *Proc. 7th International Workshop on WCET Analysis*, Pisa, 2005.

[4] T. Harmon and R. Klefstad, "Interactive back-annotation of worst-case execution time analysis for java microprocessors," in *Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. Washington: IEEE Computer Society, 2007.

[5] R. Wilhelm and B. Wachter, "Abstract interpretation with applications to timing validation," in *CAV '08: Proceedings of the 20th international conference on Computer Aided Verification*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 22–36.

[6] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL'77*. New York: ACM, 1977, pp. 238–252.

[7] M. V. Hermenegildo, E. Albert, P. López-García, and G. Puebla, "Abstraction carrying code and resource-awareness," in *PPDP*, 2005, pp. 1–11.

[8] G. A. Kildall, "A unified approach to global program optimization," in *POPL'73*. New York: ACM, 1973, pp. 194–206.