Static Analysis Support for Measurement-based WCET Analysis

Stefan Schaefer [◇†] Bernhard Scholz [†] Stefan M. Petters [◇‡] Gernot Heiser [◇‡]

[◇] National ICT Australia*
[†] School of IT
University of Sydney
[‡] School of Computer Science
and Engineering
University of NSW

{*stefan.schaefer*,*stefan.petters*,*gernot*}@nicta.com.au
scholz@it.usyd.edu.au

Abstract

Guaranteeing that the worst-case scenario has been covered for each basic block individually is a major challenge in measurement-based WCET analysis. On the static analysis side the accuracy of hardware models used are also subject to doubt as they are based on available documentation provided by vendors which may be not detailed enough nor correct or both. Even for verified models the question remains whether subsequent chips adhere to the same specification. We introduce a new approach to enhance measurement-based WCET analysis by deploying static analysis to ensure test coverage on basic block level and reduce pessimism. In particular, our work focuses on the questions how detailed does the hardware specification have to be to make the measurements trustworthy.

1 Introduction

Response time analysis for real-time systems requires the *worst-case execution time* (WCET) of all code in the system to be known *a priori*, but finding the WCET of programs is difficult at best. Current microprocessor architectures are highly complex as they incorporate pipelines, branch prediction units, multiple execution units and caches.

There are currently two major approaches for WCET analysis: (1) static analysis of programs and (2) measurement-based techniques. Static analysis involves modelling of the target architecture. Current CPU architectures feature a rich set of mechanisms to accelerate the execution time of programs, e.g., caches, pipelines and branch prediction units. Most of these features are transparent for programmers and programs and act in a speculative way. Models of processors are pressed hard to keep track of all these features, but due to a complexity problem most models do not implement them and place extremely heavy restrictions on code, which results in very pessimistic upper WCET bounds and compensates the advantage that it does not depend on actual execution-time measurements of code.

Creating an accurate and precise model for static analysis is difficult at least, as:

- Vendors lose flexibility once they published specifications. Unpublished specifications can be subject of a future change much more easier.
- Handware manufacturers keep their economic benefit in mind. In most cases, manufacturers do not benefit from publishing specifications.
- Finally, the translation from documentation to the model used for static analysis is also an error prone process.

This has to be seen against the backdrop of an increasing number of real-time systems, in particular those deploying high performance architectures invading every day life.

Measurement-based analysis techniques are still current practice in industry. In principle, execution time measurements are taken with an assumed worst-case input. Besides being tedious, this does not give a satisfying answer for the WCET because it is impossible to enumerate all possible (infinite many) inputs for a pro-

*National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

gram and measure each execution time. Measurement-based analysis techniques always raise the question whether there exists a more pessimistic scenario, in which the execution time exceeds the WCET observed so far.

The major problem with that approach is that the execution time of a basic block differs when different input parameters are provided as different input causes different paths to be taken to reach this basic block during execution *and* the correlation between input and execution time is not obvious. The worst-case behaviour of a basic block can be exposed easily, but this is generally considered as not being sufficient since it is hard to prove that this is the case.

Our approach aims to combine the strength of the static and measurement-based approaches and involves two steps: In a first step, measurements of basic block execution time profiles (ETPs) are taken through measurements. Static analysis methods will validate these ETPs with the help of a simple hardware model. If the validation fails, the static analysis requests more measurements with different input. In a second step, ETPs of larger syntactical constructs are computed. Finally, static analysis comes in to improve the tightness of the upper WCET bound.

We develop and test our approach within NICTA's Potoroo project, which aims to establish the WCET of all code of the L4 microkernel. Performance and trustworthiness are the main objectives of the L4 kernel implementation used within the project. Trustworthiness covers functional as well as temporal behaviour. Being able to estimate the WCET of these systems' microkernels makes them trustworthy in the temporal sense, while minimality is an important step to achieve functional trustworthiness, as a formal proof of functional correctness, as anticipated by the project L4.verified, is only possible if the code base is small.

Our report is organised as follows. In Section 2 we survey the related work. In Section 3 we introduce our new hybrid approach and in Section 4 we draw our conclusion.

2 Related Work

Pure static WCET analysis methods try to find the greatest execution time explicitly by computing of implicitly by find the execution path that causes this execution time. It is well accepted that find this longest path is undecidable in the general case. This does not hold for real-time contexts due to heavy restrictions on the source code like bounded loops, absence of dynamic structures and forcing of annotation were made such as in the works of Chapman et al. [5] and Blieberger et al. [4]. Engblom et al. introduce *Co-Transformation* [7] to establish execution time informations between the source code and the object code in order to overcome the massive restructuring of code done by the optimisation stages of compilers.

Timing schemata were used by Lim et al. [11] to model the WCET behaviour of RISC processors. Our approach uses timing schemata as well. The general idea of a timing schema is to associate to each program construct a worst-case timing abstraction that accounts hardware features that affect the WCET.

Formal methods such as *Symbolic Analysis* [1], *Code Transformation* [12], *Model Checking* [15] and regular expressions [13] are other approaches to obtain an upper WCET bound through static analysis.

The most important technique is *Integer Linear Programming* (ILP). The problem of finding the longest execution path into an ILP problem is done by Li and Malik [10], which avoids the explicit enumeration of all feasible paths of a given control flow graph (CFG). Their implementation `cinderella` performs static WCET analysis by that approach and targets the Intel i960 processor.

Another ILP-based approach, `aiT`, is discussed by Ferdinand [8]. This tool computes an upper WCET bound based on a detailed hardware model. The main feature is its use of abstract interpretation. However, this tool puts some restrictions on the programmer, i.e., the absence of dynamic structures. Furthermore, it does not consider task switches, threads, parallelism and specified return addresses. The major drawback of ILP-based approaches in general is that all ILP solvers can only give an upper WCET bound, but cannot conclude to the actual path that caused this specific execution time.

Hardware simulation is a technique used by Colin and Puaut [6]. The authors introduce a framework for WCET analysis based on CFGs and *simulate* instruction caches, branch prediction units and pipelines statically to obtain the WCET of a program.

White et al. introduce a framework to bound worst-case instruction cache performances and a tool that bounds the worst-case data cache performance predictions [16]. While timing predictions for instruction caches with pipeline simulation are as tight as predictions for direct-mapped caches, the worst-case data

cache performance prediction can be tightened significantly.

Another tool for estimating WCET with a hybrid method is pWCET [3]. This tool performs probabilistic worst-case execution time analysis. It combines static analysis and measurement-based methods and consists of two parts. The basic block level ETPs obtained through measurements are combined with the help of the computational tree and timing schemata in order to obtain an overall ETP. The approaches developed within the Potoroo project and the approach described in this report are extensions of the approach in pWCET, in particular regarding obtaining WCETs for compound syntactical constructs, and the reader is referred to the publications above for details on the measurement-based approach.

The most relevant related work is [9]. Kirschner et al. introduce a new hybrid method that combines static analysis and measurement approaches. This technique guarantees that every feasible path of the target program is executed and its execution time is measured. The static analysis performs flow fact extraction from the source code and modelling of an execution-time model based on the target architecture. An abstract syntax tree (AST) traversal annotates sub-trees with upper WCET bounds. A WCET bound for a compound construct is computed from the WCET bounds of all sub-constructs with the help of timing formulas. Due to the nature of an AST, only local flow informations can be used. A path-based search focuses on a bottom-up WCET estimation. The third method, *Implicit Path Enumeration Technique* (IPET), transforms the structure of a program into a set of constraints of the CFG. ILP is then used as the back-end to solve the WCET problem. With the help of this technique, all possible execution paths are considered implicitely. However, this is essentially an exhaustive search which becomes quickly infeasible even for code with very moderate complexity. Furthermore, execution time is not just dependent on control flow, but is also substantially influenced by data locality (i.e. caching effects), which is not considered in that work.

The thesis [14] involves an exhaustive search combined with measurements. In this case, control flow during measurements was enforced. To deal with complexity, partintiong the application into measurement blocks provided a manageable number of paths to be explored. These blocks were insulated against each other by forcing a worst-case state of caches and branch prediction between measurement blocks, making sure the worst case has been captured.

3 Approach

3.1 Measurement-Based Analysis

The measurement-based approach this work complements relies on execution traces gathered during execution of the code. The traces may be generated using instrumentation code, debug tracing tools, or cycle accurate simulators. However, cycle accurate simulators suffer the same uncertainty of their models as pure static analysis and is hence not considered a good solution. The traces contain tuples of a time stamp and an basic block identifier, which connects the time stamp with the execution of a basic block. The basic blocks are matched to their corresponding nodes in the computational tree that is obtained from the CFG of the executable. In this manner, we get an execution-time profile (ETP) for each basic block. Such an ETP reveals cache misses occured during execution through their execution time penalty. Basic block ETPs are then combined to ETPs of higher order nodes in the computational tree with the help of timing schemata and surpremal convolutions. Surpremal convolutions are mathematical operators that describe the convolution of the ETPs such that the resulting ETP considers every dependency between the two argument ETPs. In this manner, we cover any possible dependency between the two basic blocks. The root node of the tree contains the ETP describing the WCET behaviour of the entire code. The details of the whole analysis method can be taken from [2]. Within this report, we will focus on the ETPs of basic blocks and issues when combining them. Picture 1 depicts an overview of the operations performed during WCET analysis.

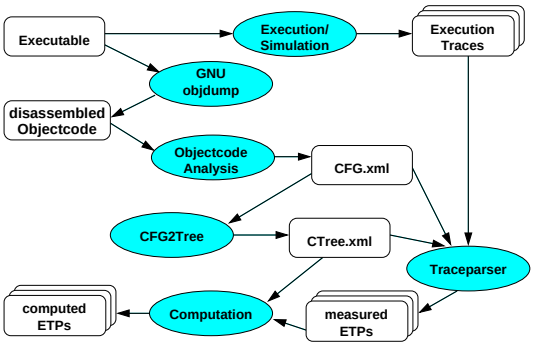

Figure 1: Our Current Tool Chain

3.2 Coverage Checking

As mentioned before, the major problem of approaches based on basic block level measurements is to guarantee that sufficient measurements have been taken. Our proposed approach is to supplement the measurement-based WCET analysis by static analysis. The static analysis part is focusing on first order effects like caches. Second order effects like pipelining are not considered. The main motivation of this limitation is that any risk involved in underestimating the WCET, if all cache effects have been surely covered, is very small. The variabilities of first order effects is large in relation to the those of second order effects.

In a first stage our aim is to predict possible numbers of cache misses in basic blocks. Creating a model of caching behaviour is easy compared to, for example, the interlocking dependencies between pipeline, cache and arithmetic model. It can be ported easily and the complexity pressed upon the computational stage is kept low. Cache size, associativity, and cache line size can usually be taken straight from the data sheets, but the replacement algorithm if often not specified or given as "pseudo-random". This allows the vendors for some leeway in changing implementations without changing documentation. In the end, only a small number of replacement algorithms are used, with the *clock algorithm* and *least recently used* (LRU) being very popular. All these can easily be identified and verified by simple test programs testing for the boundary cases.

The static analysis stage decomposes the object code of executables into basic blocks and keep track of memory accessing operations within each basic block. In this manner, a cache miss profile for each basic block is obtained. The results of the cache prediction model are then compared to the ETPs observed during measurements. The granularity difference between caching and other effects allows an easy comparison between the ETPs and the model results.

Our hybrid approach uses the idea of a feedback mechanism as shown in Figure 2. As indicated, more measurements are taken if the measured outcome deviates from predictions. If a basic block is not exhibiting the predicted worst-case behaviour for an extended period of time the process is stopped and manual intervention necessary. The manual intervention can be either to craft input data to create the worst-case scenario for an basic block or to dismiss such a worst-case scenario as being impossible to achieve. The latter may happen as the approach so far does not take infeasible paths into account in the computational stage. However, we aim at minimising such interventions to make the problem managable.

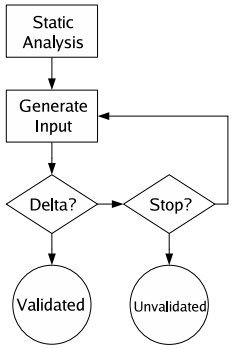

Figure 2: The Feedback Loop for our Hybrid Approach

Figure 3 shows the correlation of the cache misses of one basic block. The upper part shows the execution time penalties predicted by the static analysis. These penalities are variable as the cache itself has a variable execution time when writing back a cache line. The lower part shows the clusters of execution times around cache misses observed during running the basic block. These clusters arise from second order effects like pipelines. In order to etablish these correlations, matching the cache miss profiles and the ETPs is the key point. As indicated, the predictions obtained from the hardware model may differ from those obtained from the ETP in which case more measurements are necessary. The granularity that separates one cluster from a neighboring cluster is one of the most fundamental issues on this matching problem.

As pointed out, our aims are microkernels running on embedded systems. In particular, the L4 microkernel developed at NICTA has system call functions whose execution times in general are short since not much code is involved. This assists to minimise the space of possible cache states that might be achieved during execution. As a matter of fact, we have very little to no eviction of cache contents.

3.3 Overestimation Reduction

In order to obtain save results, the measurement-based approach makes no assumptions regarding dependency of execution times between basic blocks, but rather makes sure that any form of dependency is conservatively covered through surpremal convolutions. This

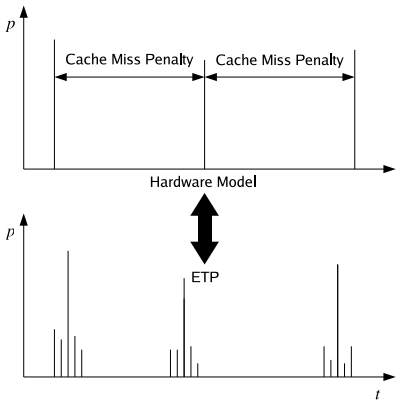

Figure 3: Correlation between Model and Execution Time Profile

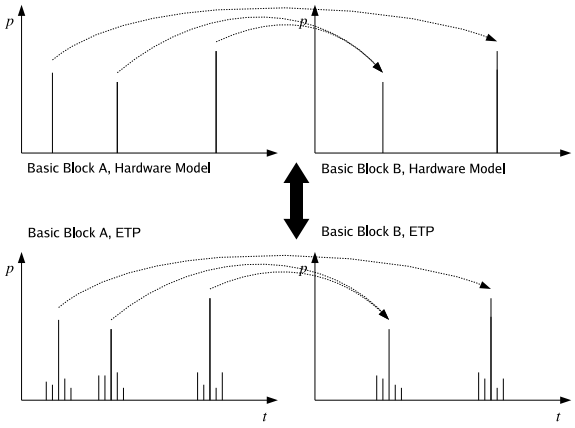

Figure 4: Establishing Dependency Structure

leads to inherent overestimation.

As a next step after verifying that sufficient measurement data has been obtained, our approach focuses on identifying dependencies between execution times of basic blocks. In the previous section we have described how to establish a connection between an ETP and a cache miss profile. In this step we establish now dependencies between the cache miss profiles of different basic blocks, which can than be taken into account in the computational stage of the approach using the measured ETPs.

Figure 4 depicts the established cache miss dependencies between two basic blocks A and B. The upper part shows the dependency establishment with the help of the hardware model. These dependencies may then be considered when combining the ETPs of the two basic blocks using timing schemata, which is shown in the lower part. The space of possible cache state transitions is reduced massively such that we can make statements like "If this cache miss occures in A, then it will not occure in B".

4 Conclusions

In this report we have detailed our work to enhance trustworthiness of measurement-based worst-case execution-time analysis by deploying static analysis techniques. In order to avoid a common problem of lack of trustworthiness of the models deployed in static analysis, we aim to use only a minimalistic model considering caching, opposed to modelling the entire processor and peripheral units. This model can be easily validated on any given hardware platform. Building on top of the trustworthiness analysis of the measured ETPs of basic blocks, we also aim to use the technique to tighten up on existing sources of overestimations by describing dependencies between basic blocks which can then be used in the later computational stages. Future work will focus on implementing and validating this approach.

References

[1] Guillem Bernat and Alan Burns. An approach to symbolic worst-case execution time analysis. In *25th IFAC Workshop on Real-Time Programming. Palma (Spain).*, May 2000.

[2] Guillem Bernat, Antoine Colin, and Stefan M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 279–288, Austin, Texas, USA, December 3-5 2002.

[3] Guillem Bernat, Antoine Colin, and Stefan M. Petters. pWCET: a tool for probabilistic worst-case execution time analysis of real-time systems. Technical report YCS353 (2003), University of York, Department of Computer Science, York, YO10 5DD, United Kingdom, April 2003.

[4] Johann Blieberger. *Timing Analysis of Object-Oriented Real-Time Programs.* (submitted), Institute for Informatics, Technische Universität Wien, Vienna, Austria, 1995.

[5] Roderick Chapman, Andy Wellings, and Alan Burns. Integrated Program Proof and Worst-Case Timing Analysis of SPARK Ada, June 1994.

[6] Antoine Colin and Isabelle Puaut. A Modular and Retargetable Framework for Tree-based WCET Analysis. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pages 191–198, Delft, Netherlands, June 13-15 2001.

[7] Jakob Engblom, Peter Altenbernd, and Andreas Ermedahl. Facilitating worst-case execution time analysis for optimized code. In *The 10th Euromicro Workshop on Real-Time Systems (ECRTS98)*, Berlin, Germany, June 1998.

[8] Christian Ferdinand. Worst-Case Execution Time Prediction by Static Program Analysis. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, New Mexico, USA, April 26-30 2004. IEEE Computer Society.

[9] Raimund Kirner, Ingomar Wenzel, Bernhard Rieder, and Peter Puschner. Using Measurements as a Complement to Static Worst-Case Execution Time Analysis. In *Intelligent Systems at the Service of Mankind*, volume 2. UBooks Verlag, Dec. 2005.

[10] Yau-Tsun Steven Li and Sharad Malik. Performance Analysis of Embedded Software Using Implicit Path Enumeration. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pages 456–461. ACM, June 1995.

[11] Sung-Soo Lim, Young Hyun Bae, Gyu Tae Jang, Byung-Do Rhee, Sang Lyul Min, Chang Yun Park, Heonshik Shin, Kunsoo Park, Soo-Mook Moon, and Chong-Sang Kim. An Accurate Worst-Case Timing Analysis for RISC Processors. *IEEE Trans. Software Eng.*, 21(7):593–604, 1995.

[12] Hemendra Singh Negi, Abhik Roychoudhury, and Tulika Mitra. Simplifying WCET analysis by code transformations. In *4th Intl. Workshop on Worst-Case Execution Time Analysis*, Catania, Italy, June 30 2004. Satellite Workshop of the 16th Euromicro Conference on Real-Time Systems.

[13] Chang Yun Park. Predicting Program Execution Times by Analyzing Static and Dynamic Program Paths. *Journal of Real–Time Systems*, 5(1):31–62, 1993.

[14] Stefan M. Petters. *Worst-Case Execution Time Estimation for Advanced Processor Architectures*. PhD thesis, Institute for Real-Time Computer Systems, Technische Universität München, Munich, Germany, September 2002.

[15] Sergio Vale Aguiar Campos and Edmund M. Clarke and Wilfredo R. Marrero and Marius Minea. Verus: A Tool for Quantitative Analysis of Finite-State Real-Time Systems. In *Workshop on Languages, Compilers and Tools for Real-Time Systems*, pages 70–78, 1995.

[16] Randall T. White, Frank Mueller, Christopher A. Healy, David B. Whalley, and Marion G. Harmon. Timing analysis for data and wrap-around fill caches. *Real-Time Systems*, 17(2-3):209–233, 1999.