WCET Analysis of ARM Processors using Real-Time Model Checking

Andreas Engelbredt Dalsgaard, Mads Christian Olesen, Martin Toft, René Rydhof Hansen and Kim Guldstrand Larsen

Department of Computer Science
Aalborg University, Denmark
{andrease,mchro,mt,rrh,kgl}@cs.aau.dk

Abstract. This paper presents a flexible method that utilises real-time model checking to determine safe and sharp WCETs for processes running on hardware platforms featuring pipelining and caching.

1 Introduction

In order to produce a reliable and efficient execution schedule for a real-time system (RTS), scheduling algorithms need safe and sharp worst-case execution times (WCETs) for the processes in the system [6]. The developed method utilises real-time model checking performed by the model checker UPPAAL [5] to determine these WCETs. The method is able to analyse program code found in real systems, and an extensive evaluation has been conducted using the benchmark programs published by Mälerdalen WCET Research Group [2].

The WCET for a process depends on the hardware platform that the process is executing on, thus the method is designed to allow a high degree of flexibility. For example, support for new processors only requires a model of the new processor.

Modern processors utilise techniques such as caching and pipelining, which increase the average number of calculations that can be executed per time unit [11]. Since these techniques are also found in many processors intended for embedded devices, such as members of the widely deployed ARM7 and ARM9 families [1], a modern WCET analysis method must take them into account to be useful. The presented method models these techniques in a modular and independent fashion.

It is important to be aware that caching and pipelining in certain instances can lead to timing anomalies [10], which complicate WCET analysis to a great extent. By introducing more non-determinism in the applied models, the presented method can be extended to handle the presence of timing anomalies.

2 Platform

Because the hardware platform must be taken into account when WCETs are determined, we have selected the ARM9TDMI processor core as the basis for an implementation of the method. This processor core is found in e.g. the ARM920T processor [8]. The execution time on a hardware platform generally depends on main memory, caches and pipeline, thus these elements must be modelled carefully.

Instructions and data are stored in main memory, which is very slow compared to the internal registers of a processor. Fortunately, this bottleneck can be mitigated by using one or more caches, which are small, fast memories for storing instructions or data temporarily. Caches increase a systems' overall execution speed by taking advantage of the principles of locality and by being faster than main memory. A cache is characterised on: speed, size, replacement policy and write policy.

The data path in a processor is the circuitry that executes instructions. In non-pipelined processors, an instruction must flow through the entire data path before the next instruction can enter. Since this takes time, a long data path forces the processor to run at a relatively low frequency. Conversely, a pipelined processor has its data path divided into a number of almost independent stages. Since the stages run in parallel, the cycle time becomes shorter and the processor may therefore run faster. For example, the ARM9TDMI processor core has a five stage pipeline [4], where the stages' names are fetch, decode, execute, memory and writeback. Sometimes the stages have inter-dependencies, which give rise to stalls and other situations, where the stages must interact. A stall occurs when an instruction uses the result of the previous instruction, which is delayed due to e.g. memory access. Some events, like branching, also require special handling.

3 WCET Analysis

The presented method is mainly inspired by [3,7,9]. In this work, WCET analysis is split into four analyses: cache, pipeline, path and value analysis. The first three analyses are modelled using model checking of a network of timed automata (NTA), whereas value analysis is performed separately on the disassembled process after reconstructing its control flow.

Besides an automaton for each function in the process' control flow graph (CFG), the NTA contains automata for the hardware platform's pipeline, caches and main memory. Each transition in the function automata simulates an abstract execution of an instruction. The simulation of an instruction is done by synchronising with the automata modeling the pipeline, which in several places synchronise with the automata modeling the caches. Finally, the caches synchronise with the automaton modeling the main memory.

To determine the WCET, we perform real-time model checking on the NTA using the model checker UPPAAL. There are, however, other steps that must precede the UPPAAL verification process. Figure 1 provides an overview of the method and the applied tools.

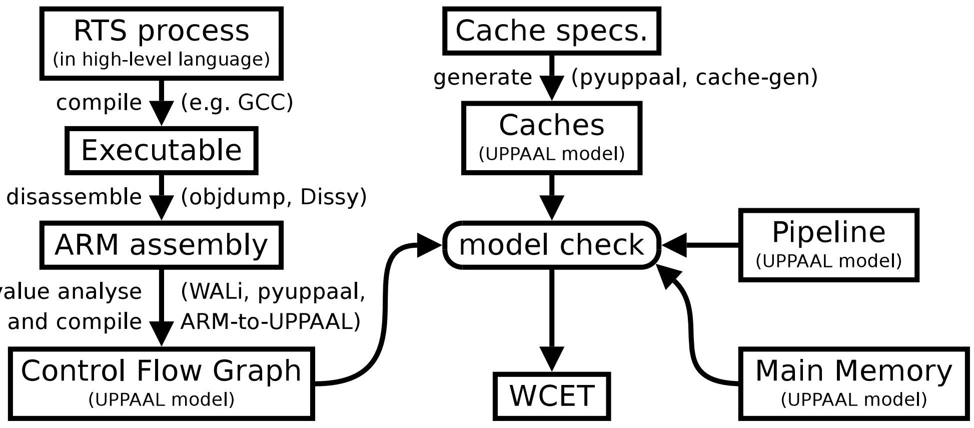

Fig. 1. Overview of the method and the applied tools.

The method is flexible, as it is easy to add support for new processors — the models for pipelines and caches can be re-used with little or no work. The modularisation of the platform model facilitates re-use and easy testing of components independently.

4 Toolchain

WCET analysis must be done at machine code level, as this is the only level with enough information [7]. By taking as input a process written in its original, high-level language and subsequently compiling and disassembling it, we are able to extract required memory addresses and take compiler optimisation into account.

We utilise the existing tools GCC, objdump, Dissy, WALi and UPPAAL, which provide compilation, disassembly, disassembly front-end, value analysis and model checking, respectively. In addition, a number of tools have been written partly or completely for the presented method. This includes pyuppaal for importing, exporting and layouting UPPAAL models, an ARM-to-UPPAAL compiler for parsing ARM assembly and generating CFG automata, "cache-gen" for generating cache automata, and "combine" for putting together all automata in an NTA. UPPAAL identify the WCET by finding the highest value of a global cycle counter clock during a full exploration of the final NTA's state space.

5 Conclusion

This paper offers a highly flexible WCET analysis method based on real-time model checking, taking caching and pipelining into account. Currently, the implementation is able to analyse 14 of the 25 working, non-floating point WCET benchmarks from Mälerdalen, which demonstrates the applicability of the method. The biggest challenge is recovering enough control-flow information, as a lack thereof is handled as non-determinism, leading to state-space explosion. Support for floating point is planned.

References

1. ARM9 - ARM Processor Family. `http://www.arm.com/products/CPUs/families/ARM9Family.html`.
2. WCET Challenge 2006. `http://www.mrtc.mdh.se/projects/wcet/benchmarks.html`.
3. Martin Alt, Christian Ferdinand, Florian Martin, and Reinhard Wilhelm. Cache Behavior Prediction by Abstract Interpretation. In *SAS '96: Proceedings of the Third International Symposium on Static Analysis*, pages 52–66, London, UK, 1996. Springer-Verlag.
4. ARM. *ARM9TDMI Technical Reference Manual*, 2000.
5. Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.
6. Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages.* Pearson Education Limited, third edition, 2001.
7. Reinhold Hechmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 2003.
8. ARM Limited. ARM920T Technical Reference. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0151c/ARM920T_TRM1_S.pdf`, 2001.
9. Alexander Metzner. Why Model Checking Can Improve WCET Analysis. In *Proceedings of Computer Aided Verification*, pages 334–347. Springer Berlin / Heidelberg, 2004.
10. Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A Definition and Classification of Timing Anomalies. In *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, 2006.
11. Andrew S. Tanenbaum. *Structured Computer Organization.* Pearson Education, fourth edition, 4 Nov 1998.