Experiences from Industrial WCET Analysis Case Studies

Andreas Ermedahl, Jan Gustafsson and Björn Lisper
Dept. of Computer Science and Electronics, Mälardalen University
Box 883, S-721 23 Västerås, Sweden

Abstract

Static Worst-Case Execution Time (WCET) analysis is currently taking a step from research to industrial use. We present a summary of three case studies where static WCET analysis has been used to analyse production code for embedded real-time systems. The primary purpose has not been to test the accuracy of the obtained WCET estimates, but rather to investigate the practical and methodological difficulties that arise when applying current WCET analysis methods to these particular kind of systems.

In particular, we have been interested in how labor-intensive the analysis becomes, for instance by estimating the efforts to study the analysed code in detail, and measuring the number of manual annotations necessary to perform the analysis. From these observations, we draw some conclusions about what would be needed to turn static WCET analysis into a useful tool for embedded and real-time systems software development.

1 Introduction

To give timing guarantees for embedded and real-time systems, a key parameter is the *worst-case execution time* (WCET) of the executing tasks. Until now, the common method (if any) in industry to derive WCET values has been by measurements. A wide variety of measurement tools are employed in industry, including emulators, logic analyzers, oscilloscopes, and software profiling tools [14]. This is labor-intensive and error-prone work, and even worse, it is difficult to guarantee that the WCET has been found.

Static WCET analysis is an alternative method to determine the WCET of a program, relying on mathematical models of the software and hardware involved. The analysis avoids the need to run the program by considering the effects of all possible inputs, including possible system states, together with the program's interaction with the hardware. Given that the models are correct, the analysis will derive a timing estimate that is safe, that is greater than or equal to the actual WCET. The static WCET analysis research community has developed a number of prototype tools during the last couple of years, for example SWEET [18] and Heptane [10]. Recently also commercial WCET tools, such as aiT from AbsInt GmbH, Germany [1] and Bound-T from Tidorum, Finland [3], have appeared.

In this paper, we present experiences from three case studies where WCET analysis tools have been used. In the two first case studies, we analysed time-critical parts in a real-time operating system (Sections 3, 4). The third case study targeted code controlling automotive data communication (Section 5). We also report from two on-going case studies (Section 6).

All of these case studies have been performed as MSc theses works. The students can spend about five months on their work, and they are no experts on the code at the beginning. This means that these results can be seen as typical for WCET analysis made by an well-educated but external person; the work should probably have taken less time if an expert or the programmer had performed it.

We believe that doing case studies, with careful evaluations, provides valuable input both for WCET research and WCET tool development. Our hypothesis is that the studied software is representative for a large class of industrial embedded real-time code, making our results applicable to similar systems.

There are only a few other case studies of using WCET analysis in practice known to us; reporting on the use of Heptane [6], aiT [15], and Bound-T [11].

To make static WCET analysis industrially useful, it is desirable to automate the process on a "one-click-analysis" basis. Consequently, one of the main topics has been to investigate how labour-intense practical WCET analysis actually becomes.

We were also interested in the characteristics of the obtained WCET values. Most scheduling theories assume that each task has a single fixed WCET, and we wanted to find out whether this assumption is valid in

real industrial settings.

The rest of the paper is organized as follows. In Section 2, we present the used tools. In Section 3, we describe a case study with our own research prototype, SWEET (SWEdish Execution time Tool). Sections 4 and 5 describe two case studies where we used aiT to analyse commercial code. Section 6 presents two ongoing case studies. In Section 7 we draw some conclusions, and in Section 8, we point out further research.

2 Tools Used

SWEET is a prototype WCET tool developed at Uppsala and Mälardalen University [18]. SWEET consists of three main parts; a flow analysis which detects program flow constraints, a low-level analysis, and a final WCET calculation. The flow analysis part of SWEET analyses intermediate code produced by a research compiler. Our current focus is to develop automatic flow analysis methods, such as abstract interpretation-based methods [9].

The aiT tool is a commercial WCET analysis tool from AbsInt GmbH [1]. In contrast to SWEET, aiT does not rely on any specific compiler, but analyses executable binaries, with support for a number of target architectures. The tool also includes an automatic loop bound analysis, which can catch simple cases.

3 Case Study 1: Using SWEET to Find Time Bounds For DI Regions

This case study was performed with the low-level and calculation parts of an earlier version of SWEET. The purpose was to find upper bounds of the execution time for a number of Disable Interrupts (DI) regions in the delta kernel (ARM9 version) of the Enea OSE operating system [7]. The OSE operating system is a real-time operating system used in embedded applications, for example in mobile phones and aircrafts. The case study is described in closer detail in [4].

Having short DI regions is important, since the execution of these regions can potentially delay any other activity in the system. The goal of the study was to investigate if WCET analysis could provide a feasible way to bound the execution times of the DI regions at a reasonable cost.

The study was done in the following steps:

1. The DI regions was extracted from the binaries.
2. The control flow graph for these regions was constructed.
3. The WCET tool was used to calculate upper bounds for the WCET of each region.

We identified 612 DI regions in the Delta OSE kernel. Most of these were very simple. We selected ten DI regions that were potentially challenging for WCET analysis for a closer investigation. These regions had a more complex control structure than the others, and several contained loops. To find upper loop iteration bounds sometimes posed a problem, since no automatic loop bound analysis was available, and it was hard to deduce loop bounds manually from the code.

Experiences and conclusions. A lot of effort was used to identify DI regions and construct their control flow graphs. The tools developed to do this had some shortcomings. Even with these problems, there are some interesting conclusions we can draw:

- The problem of defining upper loop iteration bounds depends on the special type of code analysed here. Operating systems are often run in certain modes which may affect loop bounds, and therefore the WCET is typically mode-dependent. One would thus like to have different, tight WCET bounds for different modes, rather than a single WCET bound valid for all modes.
- The usefulness of analyses such as WCET analysis grows fast with the level of automation. In our experiment, even simple means of automation made a huge difference in the amount of engineering work.

4 Case Study 2: Using aiT to Find Time Bounds For Time Critical Code

The Enea OSE operating system for the ARM processor was studied also here. Some of the tools developed in the first case study were re-used in this case study. 180 of the previous DI regions were analysed, as well as four system calls. In this study, we used the aiT tool [1]. This commercial tool has a richer set of processor timing models, and a better user interface, than our prototype tool. The case study is described in closer detail in [13].

The aiT ARM7 tool analyses executables. This information is, however, often not sufficient to yield a good WCET bound for the analyzed code. In particular, information about program flow, such as bounds to loop iteration counts not caught by the loop bounds analysis, and knowledge of infeasible paths, has to be provided by the user. Therefore, aiT supports a set of *user annotations* to provide external information to the analysis [8]. Some of the more important annotations are: *loop bounds*, *maximal recursion depth*, *dead code*, and (static) *values of conditions*.

Experiences and conclusions. We soon discovered that the execution time of the system calls depended on many parameters. A global WCET bound, valid for all possible parameter values, could become very poor for for actual configurations and standard running modes.

We dealt with this problem in our experiments by assuming some "typical" scenarios for parameters affecting the WCET (after correspondance with the OSE designers). We also excluded uninteresting execution paths from the analysis by manual annotations.

We made the following observations:

- A significant amount of annotations were required for each system call; for the analysed routines of sizes between 78 and 143 instructions, the number of annotations were between 10 and 33.
- Another observation is that excluding the error handling code in the OSE system calls yielded significantly smaller code to analyze.
- Many loops in the OSE kernel depends on dynamic data structures. This had the consequence that the aiT loop bound analysis did not perform well for these loops.
- Providing upper bounds manually for these loops required a deep understanding of the code. Consequently, the analysis was quite labor-consuming, even if the analyzed code was small. Also, the analysis relied on information from the OSE designers.

We conclude that the usefulness of WCET analysis would improve with a higher level of automation and support from the tool. Especially, it would be important be to develop advanced flow analysis methods, that could find complex loop bounds automatically. Another important conclusion made is that absolute WCET bounds are not always appropriate for real-time operating system code. The reason is, as mentioned, that the WCET often depends on dynamic system parameters. An absolute WCET bound, covering all possible situations, will provide a gross overapproximation.

5 Case Study 3: Using aiT for Time-Critical Parts of Automotive Code

This case study targeted automotive code, namely the Volcano Tool Suite for design and implementation of in-vehicle communication over CAN and/or LIN networks. The company Volcano Communications Technologies AB (VCT) [16] provides tools for embedded network systems, principally used within the car industry. The Volcano LIN Target package (LTP) was selected as a suitable part of the Volcano LIN tool suite to analyse. The work is described in closer detail in [12].

The microcontroller used in this study was a MC9S12DP256 from Motorola, which includes a 16-bit Star12 CPU of the MC68HC12 family.

Results from analysis of nine different LIN API functions were presented. We were able to obtain WCET values for all analyzed functions. However, these values were often not a constant single value, but depended on some system parameters. Also, all functions needed manual annotations to be analysed. The number of annotations ranged between 6 and 14 for functions of sizes between 2 kb and 14 kb.

Experiences and conclusions. As for the OSE code, the WCET for the studied LIN functions often depends on some specific system configuration parameters and modes. Similarly, a single WCET bound valid for all parameter values would provide a very poor estimate in most situations. A mode- and input-sensitive WCET analysis would obtain a better resource utilization and provide better understanding of the system's timing characteristics.

For many parts of the LIN API it was possible to manually create parametrical WCET formulas. It seems interesting to develop methods to automatically derive these parametrical formulas.

Much work was required to set annotations manually. To do this required an understanding of the meaning of the code.

There is a need for ways to automate the analysis. For example, better flow analysis methods would be useful to avoid manual calculation of loop bounds.

After discussions with the VCT employees it turned out that not only the WCET, but also the jitter of a piece of code, is of large interest. (The jitter is the largest execution time variation a function can experience, that is the difference between the best-case execution time (BCET) and the WCET.)

6 On-going work

We are currently performing two case studies which are summarized below.

Comparison of Different Methodologies for Obtaining WCET Values. Two MSc students are currently studying real-time embedded systems code from CC-Systems [5], using aiT for the Infineon C167 processor. CC-Systems develops embedded software and hardware for welding machinery, as well as for trucks, ships, trains and other vehicles. This case study will compare the different methodologies for obtaining timing values, that is static analysis and measurement-based methods.

Evaluation of static WCET Analysis Methods for Time-Critical Real-Time Embedded Code. We have also just started a new case study at Volvo Construction Equipment (Volvo CE) [17]. Volvo CE uses the Rubus real-time operating system from Arcticus [2] in their embedded, time-critical systems for trucks and other vehicles. The target processor will be Infineon C167 and (if possible) Infineon XC161. The precision of the WCET analysis will be evaluated against the measurements which are performed regularly by Volvo CE.

7 Conclusions

Sections 3 to 5 provided a number of detailed results and experiences. Some common conclusions can be drawn from our case studies.

It is possible to apply static WCET analysis to code with properties similar to the analysed code. The tools used performs well, once the necessary preparatory work, such as defining annotations, has been done. However, the WCET analysis process is not automated on a 'one-click-analysis' basis. Much manual intervention, and detailed knowledge of the analyzed code, is required to perform the analysis.

A higher degree of support from the tool, for example with automatic loop bounds calculation, would be desirable. A graphical interface is also valuable, to obtain an overview of the analysed code and see how it executes.

Absolute WCET bounds are not always sufficient. Support for some type of parametrical WCET calculation is sometimes needed.

8 Future Work

We intend to continue with WCET analysis case studies. One direction is to use the flow analysis developed in our own tool, SWEET, both as a stand-alone tool and used in cooperation with the commercial tools. We are members of the Compilers and Timing Analysis cluster in the ARTIST2 Network of Excellence on Embedded Systems Design. One of the aims of the work in this group is to define common formats for cooperation between different parts of WCET tools.

Acknowledgements

This work was performed within the ASTEC competence center, `www.astec.uu.se`, supported by the Swedish Agency for Innovation Systems (VINNOVA), `www.vinnova.se`. We thank AbsInt GmbH for giving us access to their WCET analysis tool. We are also grateful to the ENEA, CC-Systems, Volcano, Volvo CE, and Arcticus companies for their support.

References

[1] AbsInt company homepage, 2005. `www.absint.com`.

[2] Arcticus Systems homepage. URL: `http://www.arcticus.com`, 2005.

[3] Bound-T tool homepage, 2005. `www.tidorum.fi/bound-t/`.

[4] M. Carlsson, J. Engblom, A. Ermedahl, J. Lindblad, and B. Lisper. Worst-case Execution Time Analysis of Disable Interrupt Regions in a Commercial Real-Time Operating System. In *Proc. 2nd International Workshop on Real-Time Tools (RT-TOOLS'2002)*, 2002.

[5] CC-Systems AB homepage. URL: `http://www.cc-systems.com`, 2004.

[6] A. Colin and I. Puaut. Worst-Case Execution Time Analysis for the RTEMS Real-Time Operating System. In *Proc. 13th Euromicro Conference of Real-Time Systems, (ECRTS'01)*, June 2001.

[7] Enea. Enea Embedded Technology homepage, 2004. URL: `http://www.enea.com`.

[8] C. Ferdinand, R. Heckmann, and H. Theiling. Convenient User Annotations for a WCET Tool. In *Proc. 3rd International Workshop on Worst-Case Execution Time Analysis, (WCET'2003)*, 2003.

[9] J. Gustafsson. *Analyzing Execution-Time of Object-Oriented Programs Using Abstract Interpretation*. PhD thesis, Department of Computer Systems, Information Technology, Uppsala University, May 2000.

[10] Homepage for the `Heptane` WCET analysis tool, 2005. `www.irisa.fr/aces/work/heptane-demo/heptane.html`.

[11] M. Rodriguez, N. Silva, J. Esteves, L. Henriques, D. Costa, N. Holsti, and K. Hjortnaes. Challenges in Calculating the WCET of a Complex On-board Satellite Application. In *Proc. 3rd International Workshop on Worst-Case Execution Time Analysis, (WCET'2003)*, 2003.

[12] S. Byhlin, A. Ermedahl, J. Gustafsson, B. Lisper. Applying Static WCET Analysis to Automotive Communication Software. In *Proc. 17th Euromicro Conference of Real-Time Systems, (ECRTS'05)*, July 2005.

[13] D. Sandell, A. Ermedahl, J. Gustafsson, and B. Lisper. Static Timing Analysis of Real-Time Operating System Code. In *Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*, Oct 2004.

[14] D. B. Stewart. Measuring Execution Time and Real-Time Performance. In *Proceedings of the Embedded Systems Conference (ESC SF) 2002*, Mar 2002.

[15] S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, and C. Ferdinand. An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software. In *Proc. of the IEEE International Conference on Dependable Systems and Networks (DSN-2003)*, June 2003.

[16] Volcano Technologies Communications AB homepage. `www.volcanoautomotive.com`, 2005.

[17] Volvo CE (Construction Equipment) homepage, 2005. URL: `http://volvo.com/constructionequipment`.

[18] Worst Case Execution Times (WCET) project homepage, 2005. `www.mrtc.mdh.se/projects/wcet`.