Discussion of Misconceptions about WCET Analysis

Article · January 2004

Source: CiteSeer

2 authors:

Raimund Kirner
University of Hertfordshire

Peter P. Puschner
TU Wien

Discussion of Misconceptions about WCET Analysis [*]

Raimund Kirner, Peter Puschner
Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3/182/1
A-1040 Wien, Austria
{raimund,peter}@vmars.tuwien.ac.at

Abstract

Worst-case execution time (WCET) analysis tools are needed for the development of hard real-time systems. Despite the theoretic advances in academic research in WCET there has been hardly any impact on the industrial practice of timing analysis. The essential question is why it was not possible to provide more influential research over the last one-and-a-half decades. This paper gives constructive answers to this question. It presents a number of misconceptions about current WCET analysis. These discussions will help to guide research to the development of more useful WCET analysis techniques. This paper deals with WCET analysis techniques for hard real-time systems.

1 Introduction

The knowledge of the worst-case execution time (WCET) of tasks is crucial for the design of real-time systems. Since about more than one and a half decades, research in WCET analysis has been done to support the industry by concepts for the development of WCET analysis tools. Still there is hardly any impact on the industrial practice of timing analysis. The numerous published WCET analysis techniques and several prototype tool implementations did not trigger any ground-breaking improvements for the wide-spread industrial use of more advanced WCET analysis techniques. But still there is a strong need for useful WCET analysis tools: simple runtime measurements or manual counting of instructions are no feasible solutions assessing the code timing of increasingly complex real-time systems. This leads to the question why there is still a lack of industrial-strength WCET analysis tools.

The focus of his paper is on WCET analysis techniques for *hard real-time systems* (HRTS). The construction of HRTS requires a validation that shows that the system meets all timing constraints under guarantee [5]. In contrast, *soft real-time systems* (SRTS) do not to fulfill such strict requirements.

In this paper we highlight misconceptions about WCET analysis to present starting points for future research in this area. One of the main challenges in WCET analysis is the increasing hardware complexity of processors. The variance between optimal and worst-case performance of processors is growing significantly. The advanced hardware features make the timing prediction of modern processors quite complex. As a result, approximations in static WCET analysis produce steady increasing pessimism in the calculated WCET bound. If it is not possible to test all relevant execution scenarios, the consequences for measurement-based WCET analysis approaches are similar. The implementation of precise WCET analysis tools becomes more and more complex and the computation time needed to analyze all variations for modern processors becomes tremendously long.

To overcome the problem of the increasing complexity in WCET analysis it is necessary to make useful restrictions that lead to more predictable systems. To achieve this, the fundamental misconceptions about current WCET analysis approaches have to be analyzed. Based on these elaborations one can identify WCET analysis approaches that are more promising for practical usability.

The rest of the paper presents current misconceptions about WCET analysis. Section 2 discusses the main misconceptions about WCET analysis. Section 3 concludes this document.

---

[*] This work has been supported by the IST research project "High-Confidence Architecture for Distributed Control Applications (NEXT TTA)" under contract IST-2001-32111.

2 Discussion of Misconceptions about WCET Analysis

For a better understanding of the existing problems, a short overview about some basic properties of static WCET analysis and runtime measurements is given.

Static WCET analysis methods usually provide safe upper bounds for the WCET. To guarantee safeness, any piece of information that is not available for the analysis has to be modelled in a conservative way. Therefore, overestimation becomes the price for the safeness of the calculated upper WCET bound. In a static WCET analysis framework, calculating the concrete execution time for fractions of the code is called *exec-time modeling.* The implementation of exec-time modeling for modern processors with features like caches or pipelines becomes quite complex. The advantage of measurement-based WCET analysis techniques is that they do not require exec-time modeling. However, the drawback of using simple measurements is that measured execution times may vary depending on the concrete values of the input data.

Misconception I: "Safe Upper WCET Bounds Need to be Known for Every Real-Time Task"

It is often argued that strict static WCET analysis has to be used to analyze the timing of any real-time system. In reality, only the design of *hard* real-time systems (HRTS) really requires the provision of safe upper WCET bounds. HRTS are only a small category of real-time systems, having usually simple software structures.

The timeliness of *soft* real-time systems (SRTS) is only a question of quality of service, as sporadic deadline misses usually do not cause serious consequences. Therefore, SRTS are built to handle only typical system load scenarios. Since the accurate timing analysis of SRTS is less stringend than for HRTS, SRTS tend to have relatively complex software structures, e.g, MPEG-based video streaming. As a consequence, for modern processors with pipelines or caches, the application of strict static WCET analysis techniques to SRTS may cause too much pessimism. Furthermore, for SRTS that use modern processors, the precision obtained by runtime measurements tends to be more precise than strict static WCET analysis techniques. And the common drawback of measurement-based analysis methods – the potential underestimation of the WCET – is not necessarily so critical for SRTS.

Misconception II: "Measurement is not an Adequate Technique for WCET Analysis"

It is often argued that runtime measurements are not an adequate technique to obtain the WCET for HRTS as they typically provide only a lower bound of the WCET. To discuss properties of runtime measurements in further detail, it is necessary to distinguish between pure runtime measurements and hybrid WCET analysis methods.

Performing pure runtime measurements with exhaustive search over the value space of the input data is in general not feasible and as a consequence, only a lower bound for the WCET can be found. But things become much more easier on programs with relatively few input-data dependent control flow.

For target architectures where instruction timing only depends on the previous program control flow and the values of the operands, it is sufficient to perform the measurements for all combinations of the input data that influence the control flow. For example, the instruction timing of an architecture having a pipeline but no instruction delays due to hierarchic memory depends only on the previous control-flow dependent and the parameters. Target architectures with features like caches have an instruction timing that depends on the previous control flow and instruction parameters. For these architectures it is required to perform the measurements for combinations of all input data.

Also hybrid WCET analysis methods based on static analysis and runtime measurements can be used to calculate safe upper bounds for the WCET. Hybrid methods are relatively new and they are typically designed to exploit available control-flow information.

As a consequence, runtime measurements are an adequate WCET analysis method for hybrid analysis methods or for the analysis of systems with strongly constrained input-data dependent control flow.

Misconception III: "WCET Analysis Is Simple To Use!"

The optimal WCET analysis tool would not require any special knowledge from the user about the analyzed code. Due to undecidability, the realization of such a tool is not possible. However, it is typically discussed whether static WCET analysis or a measurement-based approach can be provide more transparency to the user. In fact, both methods have their inherent limitations and, in general, will require additional knowledge about the runtime behavior of the code.

From the theoretic point of view, static WCET analysis has various advantages over measurement-based approaches. Also, the calculated WCET bound is automatically a safe upper bound if only partial knowledge about the possible control flow of a code is available. In practice, static WCET analysis has numerous limitations: One of them is due to *flow facts*, that describe the possible control flow paths (CFP) of a program. In general, flow facts cannot be fully automatically extracted from the program code by semantic analysis. Code inspection and manual code annotation by the programmer is required to specify the possible CFP more precisely. The flow facts together with the program code are used by the static WCET analysis tool to calculate a WCET bound. In practice, concrete flow facts specifications are not powerful enough to express the possible CFP of generic programs in a precise way. For relative simple processors without caches or pipelines it is sufficient to specify flow facts as restrictions over the execution frequencies of program blocks. For modern processors this information is not sufficient to calculate precise WCET bounds. As the footprints in pipelines and caches depend on the concrete execution *order* of instructions, flow facts need to have a semantics much closer to the program execution. The calculation of flow facts about the execution order of instructions would be even more complex than flow facts about the execution frequency, which might lead to additional pessimism.

Measurement-based approaches do not directly rely on flow facts as the knowledge about the control flow is not required to perform a runtime measurement. However, to obtain WCET bounds for hard real-time systems requires to test all relevant execution scenarios of the code. A concrete execution scenario for a code is determined by the initial state of the target hardware and the values for the input parameter. The key question is how to find the relevant values for the input data so that it is ensured that all relevant execution scenarios are tested. An exhaustive search over the whole value space of the input data is in general not feasible. Missing a relevant value instantiation of the input data can result into an underestimation of the WCET. Therefore, measurement-based approaches have an analogous limitation to static WCET analysis methods. As static WCET analysis methods require flow facts to describe the control flow of a given code, measurement-based approaches require the provision of precise information about execution scenarios to be tested.

For program code with limited complexity, static WCET analysis methods as well as measurement-based approaches can be designed to be simple to use. Due to undecidability, the analysis of generic code structures will, however, always require the provision of additional information about the execution behavior of the code.

Misconception IV: "Static WCET Analysis Provides Accurate Results"

An important factor for the accuracy of a WCET analysis tool is the construction of an accurate exectime model. To calculate a precise WCET bound, the WCET analysis tool has to use the underlying exectime model to consider all possible execution combinations - a task that becomes quite expensive and complex for modern processors. Static WCET analysis methods therefore use safe approximations, that inherently cause pessimism. For example, when modeling the behavior of a cache, it can happen due to approximations that the number of cache misses is highly overestimated. In practice, this means that the "effective cache size" is only a fraction of the real cache size. There exist numerous work about modeling of different hardware features by static WCET analysis tools. However, one has to be aware that the support of a certain hardware feature by a static WCET analysis tool in general cannot be done without inducing overestimations. Though a WCET analysis tool promises the support of a certain hardware feature, the user may not be satisfied by the provided accuracy.

Misconception V: "WCET Analysis Has to Consider Task Preemptions"

It is often argued that intra-task WCET analysis introduces too much pessimism. However, widening the WCET analysis to the inter-task level creates additional complexity in the analysis as the number of variable analysis parameters increases. The alternative approach is to construct more predictable systems that support a hierarchical timing analysis. Such an approach allows for the calculation of accurate results. There exist already research in the area of separating the execution context of tasks to make the execution time of a single task more predictable [3, 1, 4]. Further research in hardware and software paradigms is required to develop practicable solutions for constructing more predictable systems.

Misconception VI: "Too Much Reserved Time due to Pessimism in WCET Analysis can be Recycled as Gain Time by Soft Real-Time Tasks"

Due to undecidability, the calculation of safe upper WCET bounds often induces pessimism. It has been argued in literature that pessimism is not such a key problem for WCET analysis methods, since a waste of resources due to pessimism could be recycled as gain time by soft real-time tasks.

It is in general questionable whether it is a good strategy to mix hard and soft real-time computation patterns. An argument from the community of fault-tolerant computing is that it is a better strategy to split systems into smaller, redundant distributed parts to increase fault tolerance.

Another point is that such a combination increases the complexity of the system, as non-real-time tasks influence the predictability of the hard real-time tasks. The existence of non-real-time tasks also hampers the process of software certification as it becomes more difficult to argue about the predictability of a system that includes soft real-time tasks.

The lucid separation of hard real-time and soft real-time tasks may be also a system requirement. As hard real-time tasks typically have a quite simple software structure, their calculated WCETs have few possibilities for allocation of gain time. Therefore, the time budget for soft real-time tasks in most cases has to be allocated statically.

It is also a basic question whether the overestimation of WCET analysis tools is a real problem as computer systems used a safety-critical environment often have quite simple code. The overestimation of the WCET for the simple software in safety-critical systems tend to be significantly lower than that for generic software with more complicated code structures.

Misconception VII: "WCET Analysis Tools Have to Support Generic Programs"

It is often claimed that a WCET analysis tool has to support generic software structures. For example, some WCET analysis projects address the full support of a programming language like ANSI C.

A more promising strategy is to develop WCET analysis methods for specific application domains. As already mentioned in misconception I, hard real-time systems typically have a simple program control flow. Another point is that code generated automatically by a code generator often has a restricted shape that simplifies WCET analysis. The simplified structure code of programs targeting these application-specific domains makes WCET analysis easier. In contrast to this, WCET analysis tools are typically designed for generic programs, where their analysis limitations become apparent.

There are various ways for a WCET analysis tool to exploit simplifications from the concrete application context. As a potential benefit, the precision of the WCET analysis tool will improve and also the implementation complexity for the analysis tool will be reduced.

3 Summary and Conclusion

This paper discussed misconceptions in current WCET analysis approaches. An important result is that one has to analyze which activities of a real-time system are really time-critical. Only for these hard real-time activities a safe WCET analsis is required. For the soft real-time activities a probabilistic timing analysis is sufficient to guarantee aspects like quality of service.

To enable safe and precise WCET analysis for hard real-time tasks, mechanisms are required to ensure the predictability of them. A promising technique to achieve this is "WCET-oriented programming", i.e., reducing the number of input-dependent control flow paths in the code [6, 7, 8]. Development tools like an intelligent editor can assist the software developer in using this technique [2].

References

[1] B. Cogswell and Z. Segall. Macs: A predictable architecture for real time systems. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 296–305, 1991.

[2] J. Fauster, R. Kirner, and P. Puschner. Intelligent editor for writing wcet-oriented programs. Research Report 30/2003, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2003. submitted to EMSOFT'03.

[3] D. B. Kirk and J. K. Strosnider. Smart (strategic memory allocation for real-time) cache design using the mips r3000. pages 322–330, Lake Buena Vista, Florida, USA, Dec. 1990.

[4] M. Lee, S. L. Min, C. Y. Park, Y. H. Bae, H. Shin, and C. S. Kim. A Dual-mode Instruction Prefetch Scheme for Improved Worst Case and Average Case Program Execution Times. pages 98–105, 1993.

[5] J. W. S. Liu. *Real-Time Systems.* Prentice Hall, 1st edition, 2000. ISBN: 0130996513.

[6] P. Puschner. Is worst-case execution-time analysis a non-problem? – towards new software and hardware architectures. In *Proc. 2nd Euromicro International Workshop on WCET Analysis*, Technical Report, York YO10 5DD, United Kingdom, Jun. 2002. Department of Computer Science, University of York.

[7] P. Puschner. Transforming execution-time boundable code into temporally predictable code. In B. Kleinjohann, K. K. Kim, L. Kleinjohann, and A. Rettberg, editors, *Design and Analysis of Distributed Embedded Systems*, pages 163–172. Kluwer Academic Publishers, 2002. IFIP 17th World Computer Congress - TC10 Stream on Distributed and Parallel Embedded Systems (DIPES 2002).

[8] P. Puschner. Algorithms for Dependable Hard Real-Time Systems. In *Proc. 8th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, Jan. 2003.