Is Worst-Case Execution-Time Analysis a Non-Problem? — Towards New Software and Hardware Architectures *

Peter Puschner
Institut für Technische Informatik, Technische Universität Wien
A1040 Wien, Austria
Email: peter@vmars.tuwien.ac.at

Abstract

Despite the scientific advances in the worst-case execution-time (WCET) analysis, there is hardly any industrial impact of the research solutions presented so far. This seems to be due to the high complexity of implementing and using the proposed WCET approaches.

This paper discusses what makes WCET analysis complex and proposes to use adequate hardware and software architectures to improve the predictability of program timing, thus simplifying WCET analysis.

1 Introduction

Research in worst-case execution-time (WCET) analysis has been around for one and a half decades. During this period a number of different approaches to WCET analysis, including solutions for modelling hardware features and characterizing possible execution paths of real-time tasks have been found [5]. Still, the results of WCET-analysis research have hardly any impact on the industrial practice of timing analysis. This seems to be due to the high complexity of the implementation and use of WCET analysis tools. In addition, WCET research always seems to lag one step behind the advances in microprocessor technology – whenever WCET research manages to deal with the features of one hardware generation the next generation of processor and hardware architectures, equipped with novel speedup features, are already there.

This paper proposes to use new, adequate hardware and software architectures to improve the temporal predictability of programs, and thus reduce the complexity of WCET analysis. Hardware architectures for future real-time systems must allow to determine instruction execution times locally by inspecting single instructions and only a small number of instructions preceding them. Software architects have to investigate into programming techniques that reduce the number of input-data dependent branching decisions in the software, thus reducing the number of different execution paths of a program. The further sections presents our considerations in more detail and proposes possible solutions.

*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 The Complexity Dilemma of WCET Analysis

There is no doubt that WCET analysis as it is currently used is a complex problem. It has been shown that, in general, the number of paths to be analyzed for an exact WCET analysis of a piece of code grows exponentially with the number of consecutive branches in the control flow of the analyzed code. This statement assumes that the code (a) is coded in traditional style (i.e., not applying programming techniques that focus on ease of WCET prediction) and (b) is to be executed on a modern high-performance processor architecture that includes caches and pipelines. Except for very simple programs this high complexity makes the full path enumeration needed for an exact WCET analysis intractable [3].

Current approaches to WCET analysis deal with this complexity in two ways:

- Calculate a high-quality WCET bound by accepting long computation times for the analysis.
- Trade the feasibility or speed of analysis for quality, by using simplified but pessimistic models of the possible software behaviours and the hardware timing.

The dilemma of WCET analysis is that neither of these approaches is acceptable in a commercial setting. On the other hand, using current hardware and software architectures does not allow for a better solution – the complexity of the problem simply is there. This raises the question if the current approaches to WCET-analysis are the correct answer to the problem of task timing analysis, or if current WCET research is focussing on the wrong problem.

2.1 Sources of Complexity

Puschner and Burns [5] identified two central factors that determine the WCET of a program in a given application context, (a) the possible sequences of program actions in a given application, and (b) the time needed for each action in each of these possible sequences. Clearly, both factors do not only determine the WCET of the code, but also the complexity of WCET analysis. Possible sequences of actions (instructions) depend of the algorithm that has been chosen to implement a solution to a problem and the code manipulations the compiler performs during compilation. The time needed for each action (instruction) depends on the features and configuration of the machine (hardware) on which the actions are executed. A number of principles applied in typical modern hardware and code design can be made responsible for WCET complexity. In the following we focus on two such principles, one for hardware and one for software.

Hardware Speedup by Speculation: This is the principle found in hierarchical memory architectures, e.g., cache. Instructions or data are loaded into (and kept in) small buffers (caches) with short access times — these access times are typically much shorter than the access times of the larger store that holds larger portions of instructions and data — with the intention to speed up future memory accesses. The decisions about which items are to be loaded, kept, and replaced in cache are usually guided by heuristics, i.e., speculation about which items might be accessed in the near future.

The use of speculative decision mechanisms leads to variable memory access times. The duration of each particular memory access, in turn, depends on the state in which the preceding operations have left the cache. Both effects (the fact that memory access times vary and the dependency of actual memory access times on the execution history) taken together contribute to the complexity of WCET analysis.

Software Optimization for Frequent Scenarios: Real-time programmers use algorithms and programming techniques that have proven to be effective for non real-time applications. In non real-time applications, speed optimization for the most probable (i.e., frequent) scenarios is the primary goal. Temporal predictability is not an issue. In order to favour frequent cases, non real-time algorithms choose the actions to be performed based on input data. Input-data dependent control decisions, however, cause programs to execute on different execution paths with different execution times. As a consequence the number of different cases to be considered by the WCET analysis is potentially high.

3 Possible Ways Out of the WCET Dilemma

This section illustrates the potential of alternative hardware and software architectures to simplify WCET analysis significantly. It provides an alternative to each of the two mentioned design principles.

3.1 Hardware Speedup: Control Instead of Speculation

In contrast to non real-time applications, (hard) real-time applications primarily require temporal predictability. Appropriate hardware designs therefore support WCET analysis via predictability. This can be achieved by using memory hierarchies that exercise absolute control on the contents of fast buffers instead of relying on speculation. Rather than hoping that future memory accesses result in a cache hit, adequate prefetching strategies make the contents and thus access times of high-speed memory easy to predict. A memory architecture that achieves predictability by prefetching has been proposed a number of years ago [2]. Unfortunately, alternative memory architectures have not been further explored.

3.2 Software: Getting Rid of Input-data Dependencies

The second problem we mentioned is that traditional algorithm design and optimization yields code that behaves differently for different input data. To circumvent this problem and allow for a simple analysis, program behaviour must be less dependent on input-data values. By reducing input-data dependencies the number of paths to be considered during WCET analysis gets smaller and, as a consequence, the complexity decreases.

Following this concept we developed the single-path paradigm [6]. The single-path paradigm yields programs that are fully temporally predictable. The central idea of the paradigm is to generate programs whose behaviour is completely independent of input data and which thus always execute on the one and only possible execution path.

Single-path programming builds upon a code transformation that removes data-dependent branching statements from the code. This code transformation is capable of transforming every WCET-analyzable piece of code into code with a single path. The transformation uses two different strategies to convert statements with if-then-else and loop semantics, respectively. If-then-else and other sequential branching statements with an input-data dependent branching condition are transformed into strictly sequential code by using *if-conversion*, [1]. Loops with input-data dependent termination are converted into loops with a constant — the maximum — iteration count. The termination condition of such loops is built into the head of a new *if* statement that is generated in the body of the loop being transformed. As a last step, if-conversion is applied to the newly generated *if* statement, see [4].

The fact that programs only have a single execution path makes WCET analysis trivial: First, path analysis is superfluous: observing the execution path of any code execution with any input data yields

the singleton execution path. Second, the analysis does not need complex and accurate hardware timing models for static WCET analysis. Since programs following this paradigm only have a single path, this singleton path is necessarily the worst-case path. Thus, obtaining the WCET by measurements is possible (either by measurements on the target or on a cycle-accurate hardware simulator) and there is no need to build any specific tools for static analysis. The latter also provides a solution to dealing with new hardware features in the analysis (see above). As the WCET analysis of single-path programs does not require hardware modelling, software developers do not have to wait until tool vendors incorporate the new features into their models in order to perform WCET analysis for their new platforms.

4 Conclusion

"Is WCET analysis a non-problem?" is the question posed in the title. To answer this question we investigated whether highly sophisticated WCET analysis techniques are the correct way to deal with the complexity of task timing analysis. We discussed hardware and code design practices that cause complexity and proposed an alternative memory architecture and the single-path programming paradigm as possible ways out.

The answer to the original question seems to be "Yes and No": As long as real-time code is coded for speed rather than temporal predictabiliy and hardware manufacturers continue to use memory hierarchies that rely on speculation then the answer is "no" — and we will certainly have to deal with such systems for at least one more decade. On the other hand, if people get aware of the importance of temporal predictability and build systems correspondingly, then WCET analysis indeed becomes trivial. So the new question is if it will be possible to convince people to change their way of thinking and put temporal predictability first.

Acknowledgments

The author would like to thank Raimund Kirner for his valuable comments on an earlier version of the paper.

References

[1] J. Allen, K. Kennedy, C. Porterfield, and J. Warren. Conversion of Control Dependence to Data Dependence. In *Proc. 10th ACM Symposium on Principles of Programming Languages*, pages 177–189, Jan. 1983.

[2] M. Lee, S. Min, C. Park, Y. Bae, H. Shin, and C. Kim. A Dual-mode Instruction Prefetch Scheme for Improved Worst Case and Average Case Program Execution Times. In *Proc. 14th Real-Time Systems Symposium*, pages 98–105, 1993.

[3] T. Lundqvist and P. Stenström. Timing Anomalies in Dynamically Scheduled Microprocessors. In *Proc. 20th IEEE Real-Time Systems Symposium*, pages 12–21, Dec. 1999.

[4] P. Puschner. Transforming Execution-Time Boundable Code into Temporally Precdictable Code. In *Proc. IFIP World Computer Congress, Stream on Distributed and Parallel Embedded Systems*, Aug. 2002.

[5] P. Puschner and A. Burns. Guest Editorial: A Review of Worst-Case Execution-Time Analysis. *Real-Time Systems*, 18(2/3):115–127, May 2000.

[6] P. Puschner and A. Burns. Writing Temporally Predictable Code. In *Proc. 7th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 85–91, Jan. 2002.