Appears in: Proceedings of the 37th Real-Time Systems Symposium (RTSS 2016)

Poster Abstract: Towards Code Metrics for Benchmarking Timing Analysis

Peter Wägemann, Tobias Distler, Phillip Raffeck, and Wolfgang Schröder-Preikschat
Friedrich-Alexander University Erlangen-Nürnberg (FAU)

***Abstract*—Comprehensive evaluations of the effectiveness of worst-case execution time (WCET) analyzers require a selection of benchmarks that pose a challenge to these tools. In this paper, we identify pitfalls that are associated with selecting such benchmarks based on complexity metrics (e.g., the number of loops contained in a program), which in part are caused by the fact that complexity measures are not necessarily stable in the face of compiler optimizations. To address these problems, we are developing a tool that automatically assesses the resilience of a benchmark against compiler optimizations by tracking complexity measures across different optimization levels. In combination with information on the data dependency of control flows, which is also provided by our tool, this allows users to find and discard benchmarks that appear challenging for WCET analyzers at the source-code level, but in fact are trivial at the machine-code level where the actual analysis is performed.**

I. Problem Statement

A common method to select WCET benchmarks is the use of complexity metrics such as the number of loops, function calls, or linearly independent paths in a program, following the rationale that such elements in general make WCET analyses more difficult. Having investigated this approach, we found that there are two major pitfalls associated with it.

Pitfall 1: The measures provided by such metrics are not necessarily stable across compiler optimizations. As a result, a benchmark that based on a certain metric is complex at the source-code level might end up trivial (according to the same metric) at the machine-code level at which the WCET analyzer operates. Due to compiler techniques such as loop unrolling and function inlining, typical examples of metrics affected by this issue are the numbers of loops or function calls.

Pitfall 2: Interpreting measures in isolation can lead to misleading assessments of the difficulty of benchmarks. This is especially true for programs whose control flows are independent of their input-data values and which consequently always execute the same path. Determining the execution times of such benchmarks is usually straightforward even though other measures (e.g., a high loop count) might suggest the opposite.

benchmark	has input?	loops on `O0`	loops `O3/O0` [%]	call chain on `O0`	calls `O3/O0` [%]	cyclomatic complexity (CC) on `O0`	CC `O0/O3` [%]	use float?
adpcm_dec	✓	6	0	3	100	32	16	✗
test3	✗	121	100	22	100	705432	100	✗

TABLE I
Complexity measures for TACLeBench (Excerpt)

II. Approach

To address the problems that arise when selecting WCET benchmarks, we are developing a tool to facilitate the selection process. Our tool runs on the target-agnostic intermediate representations of the LLVM compiler framework and automatically analyzes benchmarks at different optimization levels, currently `O0` through `O3`. As a result, it is able to evaluate the resilience of programs against optimizations, allowing users to identify benchmarks that do not pose a challenge to WCET analyzers. Instead of interpreting measures in isolation, our tool relies on a combination of several metrics (e.g., number of loops, number of paths, and inputs) and incorporates additional analyses for detecting data-flow–independent control flows.

III. Evaluation

Applying our tool to a preliminary version of the TACLeBench suite [1] shows that 19 of the 54 benchmarks (35%) are not resilient against compiler optimizations and lose loops due to loop unrolling. As illustrated in Table I, this is for example the case for the `adpcm_dec` benchmark, for which none of 6 loops at level `O0` remain at level `O3`. In addition, our analysis reveals that 12 benchmarks (22%) consist of a data-flow–independent control flow and thus implement a single-path program. Amongst these is the `test3` benchmark which has the longest call chain (22), the most loops (121), and the highest cyclomatic complexity (705432) of all benchmarks as well as a resilience of 100% in all these categories. As a result, it is the prime example of a benchmark that falsely appears challenging when only specific complexity measures are considered, confirming the need of combined metrics.

IV. Conclusion & Future Work

Our tool assists users in identifying and consequently discarding benchmarks that do not pose a challenge to WCET analyzers due to not being resilient against compiler optimizations and/or having input-independent control flows. As part of future work, we will exploit this functionality to improve the effectiveness of our benchmark generator GenE [2].

References

[1] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *Proc. of WCET '16*, 2016, pp. 1–10.

[2] P. Wägemann, T. Distler, T. Hönig, V. Sieh, and W. Schröder-Preikschat, "GenE: A benchmark generator for WCET analysis," in *Proc. of WCET '15*, 2015, pp. 1–10.

Acknowledgments: This work is supported by the German Research Foundation (DFG), in part by Research Grant no. SCHR 603/9-1, no. SCHR 603/13-1, and the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89, Project C1).