Divide and Measure: CFG Segmentation for the Measurement-Based Analysis of Resource Consumption

Michael Zolda and Raimund Kirner (Faculty Mentor)
Institut für Technische Informatik
Technische Universität Wien
Wien, Austria
Email: {michaelz, raimund}@vmars.tuwien.ac.at

Abstract — *A computer system is a good computer system if it correctly performs the task it was intended to perform. This is not even half of the truth: Non-functional requirements are abundant in the world of software and system engineering, even if they are not always stated explicitly. In our work we are concerned with the measurement-based analysis of resource consumption. Examples of resources are time, energy, or memory space. In the context of our measurement-based approach for software analysis, we face the problem of breaking the software under examination into smaller parts of manageable size, a process dubbed CFG Segmentation.*

I. Introduction

In our measurement-based approach for resource-consumption of embedded software, we perform measurements on the real physical computer system, and subsequently integrate the measurement results into a resource consumption model that is ready for expert inspection or use in a higher level analysis of the system.

In order to obtain an accurate picture of the system behavior, we would like to achieve full measurement coverage of all feasible operation sequences of the software under test. We therefore start from the *Control Flow Graph (CFG)* of the software under examination, a fundamental program representation where nodes represent the operations of the software, and where directed edges represent possible successive execution (see Figure 1 for an example of a CFG). Thus, each feasible sequence of operations of the software corresponds to a path in the CFG.

In theory, it would therefore suffice to examine all CFG paths, but due to the huge number of paths in the CFGs of real software, this approach is practically infeasible. For example, the CFG of a simple actuator controller that is part of our benchmarks contains about 10^{44} paths, which is clearly beyond measurement feasibility.

To cope with such huge numbers of paths, we split the CFG into connected subgraphs of manageable size ("segments"), deal with these subgraphs individually, and subsequently merge the data that was obtained for each segment into a global resource consumption model.

A *(simple) segment* is a connected subgraph of a CFG, and is characterized by its sets of *entry edges* (edges leading into the subgraph), its set of *exit edges* (edges leading out of the subgraph), and the number of *segment paths* (paths leading through the subgraph).

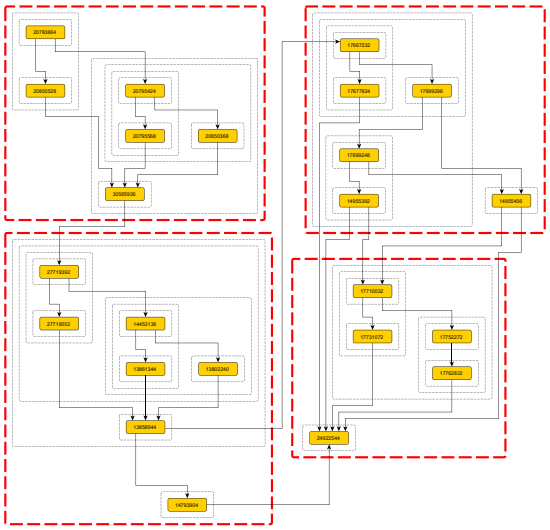

Figure 1: A CFG with four segments (indicated by thick, dashed borders). Thin borders, on the other hand, mark intermediate segments generated by our proposed greedy segmentation algorithm.

Our goal is to partition the CFG into as few segments as possible, such that the segment path total does not exceed a given feasibility limit.

To illustrate our idea, consider Figure 1, which shows a tiny CFG that has been partitioned into four segments. The total number of segment paths to be examined is $3 + 6 + 7 + 8 = 24$. Without segmentation we would have to examine all 99 CFG paths. The number of paths to be examined is thus lowered by a factor of 4 in this tiny example, but the effect is much more pronounced for realistically-sized CFGs, where we use segments in the magnitude of hundreds or thousands of paths.

II. Proposed Approach

In [1, 2], Wenzel et al. present an approach for CFG segmentation that produces segments with a single entry edge. Such segments are advantageous from a composi-

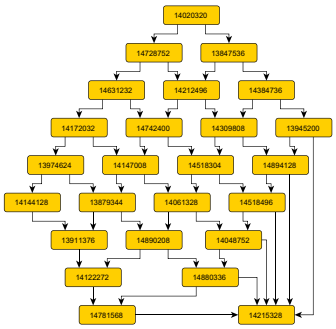

Figure 2: A CFG with a lattice-shaped structure.

bility point of view, but yield problems with large *proper interval* structures [3]. For example, the lattice-shaped CFG in Figure 2 allows only the two trivial segmentations of either putting everything into one single segment, or putting each node an individual segment.

We therefore abandon the single entry restriction and allow any number of entry edges. However, for composability reasons, we do not want the entry/exit interface to become unjustifiably large. Because we later want to associate measurement results with all entry/exit combinations of a segment, a reasonable size metric for the interface of a segment is

$$h(S) = entries(S) \cdot exits(S). \quad (1)$$

To perform a segmentation of this more general form, we propose an iterative greedy algorithm that builds segments bottom-up from smaller segments, starting with single-node segments. For each iteration, the algorithm considers all segments that could be created by merging pairs of adjacent segments and picks the least-cost one. As costs functions, we currently consider the following class of functions:

$$costs(S) = h(S)^a \cdot paths(S)^b, \quad (2)$$

We have already mentioned that we would like to keep the entry/exit interfaces of segments as small as reasonably possible. This explains the $h(S)$ part of the costs function, which is weighted by the tunable exponent a. On the other hand, the $paths(S)$ part of the costs function assures that the algorithm will produce roughly equally-sized segments. Again, this part of the costs function can be tuned by adjusting an exponent.

The example in Figure 1 indicates both, the final segments produced by our algorithm (thick, dashed borders), and the intermediate segments generated during the individual iterations (thin borders).

III. Preliminary Results and Future Work

We have implemented the segmentation algorithm presented above as a component of the timing analysis framework that is currently being developed within the FORTAS project. Our implementation is already capable of handling real industrial application code. A logical next step would be a detailed evaluation of the produced segments for a larger number of benchmarks and the usage of the obtained insight to tune the costs function presented above. It is even conceivable that careful examination of such results point at completely new forms of costs functions.

Performing segmentation is just one tasks in measurement-based analysis of resource consumption. Other tasks include the generation of suitable test data that can trigger the execution of individual paths [1], performing the actual measurements, construction of the complete timing model, and others. All these tasks, and especially their integration, are important future research issues.

IV. Summary

In this work, we have explained the problem of CFG segmentation, as it arises in the context of measurement-based resource consumption analysis. We have introduced segmentation as a possible solution to handle the explosion of CFG paths and have pointed at the shortcomings of previous segmentation approaches. Lastly, we have sketched a novel segmentation algorithm that overcomes these shortcoming.

Acknowledgments

The research leading to these results has received funding from the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) within the research project “Formal Timing Analysis Suite of Real-Time Systems” (FORTAS-RT) under contract P19230-N13.

References

[1] Ingomar Wenzel, Bernhard Rieder, Raimund Kirner, and Peter Puschner. Automatic Timing Model Generation by CFG Partitioning and Model Checking. In *Proc. Conference on Design, Automation, and Test in Europe*, Mar. 2005.

[2] Ingomar Wenzel, Raimund Kirner, Bernhard Rieder, and Peter Puschner. Measurement-Based Timing Analysis. To be presented at the 3rd Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Oct. 2008.

[3] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, August 1997.