Computer Languages, Systems & Structures 40 (2014) 155–170

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Refinement of worst-case execution time bounds by graph pruning

Florian Brandner [a,*], Alexander Jordan [b]

[a] Computer Science and System Engineering Department, ENSTA Paris Tech, 828, Boulevard des Maréchaux, 91762 Palaiseau, France
[b] Embedded Systems Engineering Section, Technical University of Denmark, Richard Petersens Plads, Building 322, 2800 Lyngby, Denmark

ARTICLE INFO

Article history:
Received 19 March 2014
Received in revised form
9 September 2014
Accepted 12 September 2014
Available online 28 September 2014

Keywords:
Worst-case execution time analysis
Iterative refinement
Graph pruning

ABSTRACT

As real-time systems increase in complexity to provide more and more functionality and perform more demanding computations, the problem of statically analyzing the Worst-Case Execution Time (WCET) bound of real-time programs is becoming more and more time-consuming and imprecise.

The problem stems from the fact that with increasing program size, the number of potentially relevant program and hardware states that need to be considered during WCET analysis increases as well. However, only a relatively small portion of the program actually contributes to the final WCET bound. Large parts of the program are thus irrelevant and are analyzed in vain. In the best case this only leads to increased analysis time. Very often, however, the analysis of irrelevant program parts interferes with the analysis of those program parts that turn out to be relevant.

We explore a novel technique based on *graph pruning* that promises to reduce the analysis overhead and, at the same time, increase the analysis' precision. The basic idea is to eliminate those program parts from the analysis problem that are known to be irrelevant for the final WCET bound. This reduces the analysis overhead, since only a subset of the program and hardware states have to be tracked. Consequently, more aggressive analysis techniques may be applied, effectively reducing the overestimation of the WCET. As a side-effect, interference from irrelevant program parts is eliminated, e.g., on addresses of memory accesses, on loop bounds, or on the cache or processor state.

First experiments using a commercial WCET analysis tool show that our approach is feasible in practice and leads to reductions of up to 12% when a standard IPET approach is used for the analysis.

1. Introduction

Real-time systems have seen a steady increase in complexity during the last decades [1]. Due to the increased processing power of modern processors, more elaborate algorithms are implemented, new functionality added, and existing functionality *migrated* to software. As the complexity of real-time software grows, the accurate static analysis of the software becomes more and more demanding. This is particularly problematic for real-time systems, where a provable

* Corresponding author. Tel.: +33 1 81 87 20 63.
E-mail address: florian.brandner@ensta-paristech.fr (F. Brandner).

http://dx.doi.org/10.1016/j.cl.2014.09.001
1477-8424/

bound of the program's execution time — its *Worst-Case Execution Time* (WCET) — is needed to guarantee the timely operation of the system.

Foremost size and complexity of software are causing the analysis overhead to grow rapidly, as the number of potential states of the program under analysis increases. This is even amplified for WCET analysis, as the number of potential software states is further increased by potential hardware states, which also have to be considered to safely bound the WCET. Even when only small portions of a complex software program are relevant for the final WCET estimation, the analysis has to account for *all* of the program's code to derive a safe bound. This often reduces the precision of the WCET analysis, as irrelevant code parts interfere with the analysis of relevant code parts and lead to unnecessary overestimation of the statically determined WCET bound compared to the actual worst-case behavior.

To address these two issues, we propose a novel technique based on graph pruning. An iterative algorithm allows us to discard irrelevant program parts and apply a standard WCET analysis to the remaining (relevant) program parts only. The iterative processing ensures safe bounds, while focusing only on relevant program parts reduces the size of the analysis problem and promises to reduce the overall analysis overhead. At the same time, more aggressive analysis techniques can be applied to the smaller remaining program. Together with the reduced interference from irrelevant code parts, this leads to a more precise and tighter WCET bound.

The basic idea of our approach is to find the longest path for every basic block in the *Control-Flow Graph* (CFG) of a real-time program [2] using a fast (and potentially imprecise) WCET analysis. The basic blocks are grouped into sets according to the length of their respective paths. The sets are then processed iteratively by decreasing path length. During each iteration a subgraph of the original CFG is formed by unifying the subgraph of the previous iteration with the basic blocks from the currently considered set. A potentially more advanced WCET analysis is then applied to the program represented by the new subgraph. The algorithm terminates, with a possibly refined WCET estimate, as soon as a safe bound, valid for the original program, has been reached.

The advantages of this approach are twofold. First of all, the analysis problems defined by the subgraphs at each iteration are much smaller than the original analysis problem. This promises to reduce the analysis overhead, while still providing tight bounds. Secondly, processing the sets of basic blocks according to their decreasing path lengths, eliminates the interference from other basic blocks, whose longest paths are known to be shorter. This improves the precision of the WCET analysis precisely for those code parts of the real-time program that impact the WCET estimate the most.

We evaluate our approach using the well-established benchmark programs Debie1 [3] and PapaBench [4] for two embedded PowerPC processors from Freescale. The WCET analysis is performed by an unmodified version of the commercial tool aiT[1] using scripting and analysis annotations. Our approach treats the WCET analysis tool as a black box and is thus compatible with other tools. Our experiments show improvements of up to 12%, depending on the processor model, when a standard implicit path enumeration approach [5] is used. Targeting an advanced analysis technique, which accounts for architecturally infeasible execution paths, but does not scale to large analysis problems, the improvement is still as high as 5%. We also notice a general trend that the benefit from our pruning method increases with the size of the analysis problems. This property is promising, since it suggests *accelerating returns* for future analysis applications. Since we use an unmodified, and for our purposes unoptimized, version of the analysis tool, the iterative processing causes some overhead. We expect that this overhead can be eliminated by adapting the WCET analysis tool to the iterative processing, for example, by using incremental analysis techniques.

The main contributions of this paper are as follows:

- We present a novel WCET analysis technique based on graph pruning that focuses the analysis effort on relevant code parts of the real-time program.
- Due to reduced analysis overhead, more elaborate analysis techniques can be applied to the smaller sub-programs, leading to improved precision.
- We evaluate our approach using a commercial off-the-shelf WCET analysis tool and demonstrate considerable improvements of WCET bounds.

The remainder of this paper is structured as follows. We first give some background and motivation in Section 2. We then describe our novel graph pruning technique in Section 3. Section 4 presents a detailed evaluation of our approach for two well-established real-time benchmarks and a realistic processor architecture. Related work is covered in Section 5 before concluding in Section 6.

2. Background and motivation

This section covers some basic definitions of control-flow graphs, paths, and WCET analysis, followed by a brief discussion of recent findings motivating this work.

[1] http://www.absint.com/ait/