Context-Sensitivity in IPET for Measurement-Based Timing Analysis*

Michael Zolda[1], Sven Bünte[1], and Raimund Kirner[2]

[1] Institute of Computer Engineering
Vienna University of Technology, Austria
{michaelz,sven}@vmars.tuwien.ac.at
[2] Department of Computer Science
University of Hertfordshire, Hatfield, United Kingdom
r.kirner@herts.ac.uk

Abstract. The *Implicit Path Enumeration Technique* (IPET) has become widely accepted as a powerful technique to compute upper bounds on the Worst-Case Execution Time (WCET) of time-critical software components. While the technique works fine whenever fixed execution times can be assumed for the atomic program parts, standard IPET does not consider the context-dependence of execution times. As a result, the obtained WCET bounds can often be overly pessimistic.

The issue of context-dependence has previously been addressed in the field of static timing analysis, where context-dependent execution times of program parts can be extracted from a hardware model. In the case of measurement-based execution time analysis, however, contexts must be derived from timed execution traces.

In the present extended abstract we present an overview of our work on the automatic detection and exploitation of context dependencies from timed execution traces.

1 Introduction

The well-known IPET approach [3,2] provides a scheme for formulating the problem of determining a WCET estimate of a software component as an integer linear programming (ILP) [1] problem. Working on the level of the *control flow graph* (CFG), the method introduces variables for the execution count of each *block*, as well as for each control flow edge between blocks. These variables are subject to linear constraints that can exclude some (but not all) infeasible control flow through the CFG. Assuming the availability of a fixed local upper WCET bound for each individual block, the determination of an upper bound of the

* The research leading to these results has received funding from the IST FP-7 research project "Asynchronous and Dynamic Virtualization through performance ANalysis to support Concurrency Engineering (ADVANCE)" and 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.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part II, LNCS 6416, pp. 487–490, 2010.

global WCET is reduced to the problem of maximizing the cost-weighted sum of execution counts over all blocks.

When we want to use IPET in practice, two important problems emerge. Both of them can be traced back to the fundamental assumption of a fixed local upper WCET bound for each individual block. They are:

1. How to determine the required local WCET bounds on a real processor that contains highly unpredictable hardware components, like caches, pipelines, branch predictors, out-of-order execution, etc.?
2. How to overcome the high pessimism introduced by using a single local WCET bound for each block, even for those blocks that show a broad spectrum of different execution times, the maximum of which possibly occurring only in very special situations?

As the creation of accurate, precise, and effective formal analyses of the behavior of modern microprocessors has become a highly complex and time-consuming task, *measurement-based timing analysis* (MBTA) has been proposed as a quick, effective, and easily deployable complementary approach.

MBTA allows for the derivation of empirical local WCET estimates from an elaborate choice of representative execution traces. Once such local WCET estimates have been determined for each block, they can be directly used as block costs in an IPET problem.

2 Context-Dependent Execution Times

We are currently working on a method for extending IPET to distinguish between different block execution times, based on empirically determined correlations with the blocks execution history and future.

Figure 1 illustrates the settings of our approach from a bird's eye view.

Our rationale is that the execution time of a block can show both, backward and forward dependencies, with respect to the execution traces.

Backward dependency is the more prominent case, where the execution time of a block depends on the concrete execution history. This is easily exemplified by the distinction of execution times of a block in the presence of a cold vs. a warm instruction cache: In a simple setting, a certain block might be absent from the instruction cache during the first iteration of a loop, but present during all subsequent iterations. A distinction of execution times of the block can then be based on whether the loop body is entered via the back edge, or from outside.

Our archetype for a forward dependency concerns the execution time of conditional jumps: The execution time of such a jump can depend on whether the jump condition is true or false. Because the jump condition controls the subsequent flow of control, we observe an apparent dependency of a block's execution time on its execution future (a circumstance that might seem counterintuitive at first thought).

It is infeasible and impractical to consider all possible dependencies of the execution time of a block on its concrete execution traces. Also, we have to

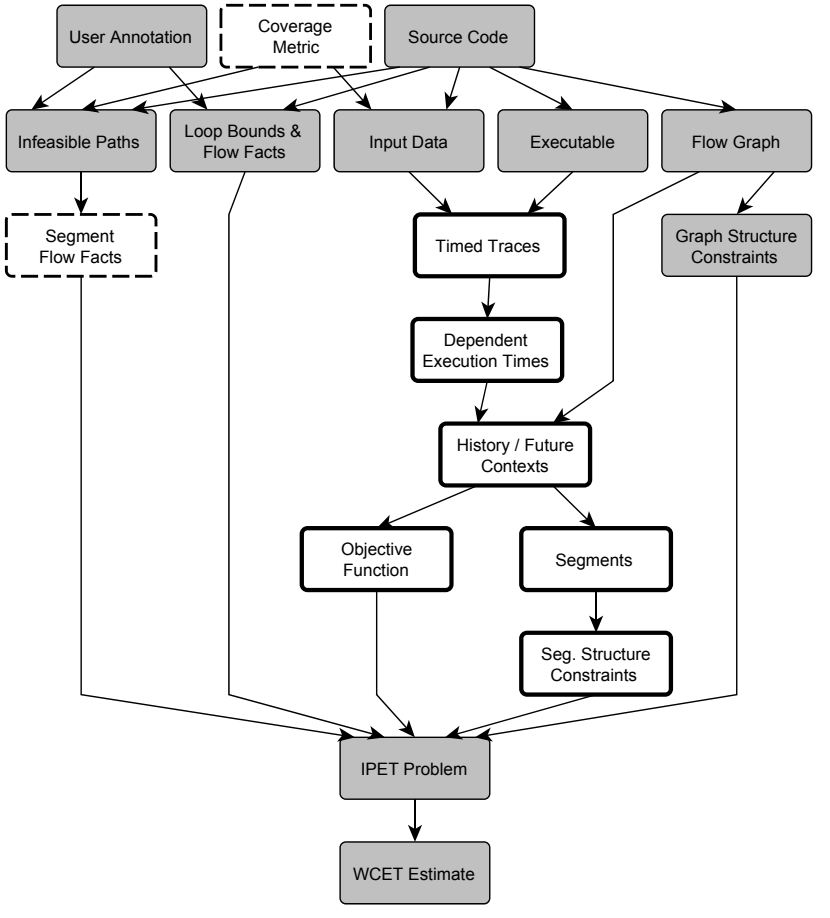

Fig. 1. Pieces of information used in the generation of a context-sensitive IPET problem. The highlighted pieces of information in the center represent our present state of research: *Timed traces* are obtained from runs of the software executable on the target hardware and *dependent block execution times* are extracted. Considering the possible flows in the software *flow graph*, suitable *history / future contexts* are derived that separate the different execution times of each block. These contexts can be easily mapped to graph *segments* (structural clusters of paths). It is then easy to derive *segment structure constraints* that model the control flow through each segment. Also, a new *objective function* is derived that consists of the cost-weighted segment execution frequencies. Adding the usual *graph structure constraints*, which can be derived automatically from the flow graph, the necessary *loop bounds*, and possibly additional *flow facts*, the context-sensitive *IPET problem* is derived.

consider a method to integrate such a distinction into IPET. In our approach, we therefore consider a subset of dependencies that we consider particularly interesting and apt to allow a reduction of the pessimism introduced by IPET.

3 Evaluation

To assess our approach, we used the following experimental setup: We analyzed 1000 traces obtained from running a slightly modified version of the *bsort100* benchmark from the Mälardalen WCET suite. The modifications of the benchmark consisted of reduction of the input array to 15 elements and code reformating for technical reasons. The program was compiled using *GCC* for the *TriCore 1796* processor without optimization. Our second benchmark was a core routine of an elevator control application. To generate the input data we used a mixed approach of systematic block coverage via model checking and a pseudo random data generation. The traces were recorded using a *Lauterbach Power Trace* device.

To estimate the overestimation introduced by IPET, we used the difference between the IPET result and the longest observed end-to-end execution time over all generated traces. Comparing the results of our context-sensitive approach with those of standard IPET, the experiments showed a marginal improvement for the (tiny) *bsort100* benchmark. For the second benchmark, the overestimation was reduced by 8%.

4 Conclusion and Outlook

We have presented our approach towards using context-dependent execution time measurements to reduce the pessimism in IPET, introducing history / future sensitivity. We have also presented first results that show that our approach can in fact help to reduce IPET pessimism in a experimental setting. The details of the approach shall be presented in a full paper.

To improve the effectiveness of our approach, we are currently working on the following two aspects[1]:

Firstly, the separation of contexts by history / future relies on the availability of a suitable set of timed execution traces. To this end, we are currently working on suitable coverage metrics and input-data generation methods.

Secondly, to exploit the full potential of context separation, it will be necessary to derive additional flow facts that restrict the possible combinations of contexts. With respect to this aspect, we are currently pursuing a method to extract such constraints from control flow paths that are known to be infeasible.

References

1. Chvátal, V.: Linear programming. W.H. Freeman, New York (1983)
2. Li, Y.T.S., Malik, S.: Performance Analysis of Embedded Software Using Implicit Path Enumeration. In: DAC 1995: Proceedings of the 32nd annual ACM/IEEE Design Automation Conference, pp. 456–461. ACM, New York (1995)
3. Puschner, P.P., Schedl, A.V.: Computing maximum task execution times - a graph-based approach. Real-Time Systems 13(1), 67–91 (1997)

[1] In Figure 1, the corresponding pieces of information are marked by a dashed borders.