Static Timing Analysis for Hard Real-Time Systems*

Reinhard Wilhelm, Sebastian Altmeyer, Claire Burguière,
Daniel Grund, Jörg Herter, Jan Reineke,
Björn Wachter, and Stephan Wilhelm

Saarland University, Saarbrücken, Germany

Abstract. Hard real-time systems have to satisfy strict timing constraints. To prove that these constraints are met, timing analyses aim to derive safe upper bounds on tasks' execution times. Processor components such as caches, out-of-order pipelines, and speculation cause a large variation of the execution time of instructions, which may induce a large variability of a task's execution time. The architectural platform also determines the precision and the complexity of timing analysis.

This paper provides an overview of our timing-analysis technique and in particular the methodological aspects of interest to the verification community.

1 Introduction

Hard real-time systems have to satisfy strict timing constraints. Traditionally, measurement has been used to show their satisfaction. However, the use of modern high-performance processors has created a severe problem. Processor components such as caches, out-of-order pipelines, and all kinds of speculation cause a large variability of the execution times of instructions, which induces a potentially high variability of whole programs' execution times. For individual instructions, the execution time may vary by a factor of 100 and more. The actual execution time depends on the architectural state in which the instruction is executed, i.e. the contents of the caches, the occupancy of the pipeline units, contention on the busses etc.

Different kinds of timing analyses are being used today [1]; measurement-based/hybrid [2,3,4] and static analysis [5] being the most prominent. Both methods compute estimates of the worst-case execution times for program fragments like basic blocks. If these estimates are correct, i.e. they are upper bounds on the worst-case execution time of the program fragment, they can be combined to obtain an upper bound on the worst-case execution time of the task.

* The research leading to these results has received funding from the following projects (in alphabetical order): the Deutsche Forschungsgemeinschaft in SFB/TR 14 *AVACS*, the German-Israeli Foundation (GIF) in the *Encasa* project, and the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement number 216008 (*Predator*).

G. Barthe and M. Hermenegildo (Eds.): VMCAI 2010, LNCS 5944, pp. 3–22, 2010.

While using similar methods in the combination of execution times of program fragments, the two methods take fundamentally different approaches to compute these estimates:

- Static analyses based on abstract models of the underlying hardware compute invariants about the set of all execution states at each program point under *all* possible initial states and inputs and derive upper bounds on the execution time of program fragments based on these invariants.
- Measurement executes each program fragment with a subset of the possible initial states and inputs. The maximum of the measured execution times is in general an underestimation of the worst-case execution time.

If the abstract hardware models are correct, static analysis computes safe upper bounds on the WCETs of program fragments and thus also of tasks. However, creating abstract hardware models is an error-prone and laborious process, especially if no precise specification of the hardware is available.

The advantage of measurement over static analysis is that it is more easily portable to new architectures, as it does not rely on such abstract models of the architecture. On the other hand, soundness of measurement-based approaches is hard to guarantee. Measurement would trivially be sound if all initial states and inputs would be covered. Due to their huge number this is usually not feasible. Instead, only a subset of the initial states and inputs can be considered in the measurements.

This paper provides an overview of our state-of-the-art timing-analysis approach. In Section 2, we describe the architecture and the component functionalities of our framework for static timing analysis.

Section 3 is devoted to several aspects of the memory hierarchy, in particular caches. Memory-system performance often dominates overall system performance. This makes cache analysis so important for timing analysis. The most relevant property is *predictability* [6,7]. This notion—a hot topic of current research—is fully clarified for caches [8,9]. A second notion, exemplified for caches, is the *relative competitiveness* of different cache architectures. They allow one to use the cache-analysis results of one cache architecture to predict cache performance for another one. A third property of cache architectures is their *sensitivity to the initial state*. Results show that some frequently used cache replacement-strategies are highly sensitive. This has severe consequences for measurement-based approaches to timing analysis. Missing one initial cache state in a non-exhaustive set of measurements may lead to dramatically wrong results.

Data-cache analysis would fail for programs allocating data in the heap since the addresses and therefore the mapping to cache sets would be statically unknown. We approach this problem in two different ways, firstly, by converting dynamic to static allocation and using a parametric timing analysis, and secondly, by allocating in a cache-aware way.

Pipelines are much more complex than caches and therefore more difficult to model. The analysis of their behavior needs much more effort than cache analysis since a huge search space has to be explored and no abstract domain with a