

# Computation of WCET for Pipelined Processors with Caches

Franck Cassez\*

CNRS/IRCCyN

email:franck.cassez@cnrs.irccyn.fr

http://www.irccyn.fr/franck

## ***Research Context.***

Embedded real-time systems are composed of a set of tasks (software) that run on a given architecture (hardware). These systems are subject to strict timing constraints (they are often called *critical* systems) and these constraints must be enforced by a scheduler. Designing an effective scheduler is possible only if some bounds are known about the execution times of each task. Determining safe upper bounds amounts to computing the so-called *worst-case execution-time* (WCET) of a program on a given hardware. Performance wise, determining tight bounds for WCET is crucial as using rough over-estimates might either result in a set of tasks being wrongly declared non schedulable, or a lot of computation time might be wasted in idling cycles and loss of energy/power.

## ***Main Features of WCET Problem.***

Computing the WCET of a program is usually a very hard problem and the difficulties stem from the fact that:

- the hardware on which a program runs usually features a multi-stage *pipelined* processor and some fast memory components called *caches*; executing a sequential program is then a concurrent process where the different stages of the pipeline and the caches and the main memory run in parallel;
- the WCET must be computed on the binary code (or an assembly language equivalent version): this is because the complex hardware executes binary code;
- modern compilers produce smart or optimized compiled programs, it is very difficult to use the knowledge (e.g., a loop invariant) that might exist on a source, say C program;
- computing a WCET implicitly means considering all possible inputs of the program to ensure a real safe upper bound.

## ***Techniques and Tools for Computing WCET.***

There are two main classes of techniques to obtain WCET: (*i*) testing-based methods: they consist in running the program with some input data usually on a real platform; these methods are not suitable for safety critical embedded systems as they do not guarantee a full coverage of input data; (*ii*) verification-based methods: these methods often

---

\* Author supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme.

rely on the computation of an *abstract* graph, the control flow graph (CFG), and an abstract model of the hardware.

As we want to compute safe WCET, we can only use verification-based techniques. All the currently used verification-based techniques and tools for computing WCET rely on *annotations* on the binary program to analyse. These annotations are often manually asserted and this is error-prone. Moreover, the algorithms and tools that implement these techniques are rather monolithic and difficult to adjust to a new hardware.

### ***Our Approach.***

In the project *Theory Applied to Real Embedded Systems (TARES)* we want to overcome the previously mentioned limitations of the WCET techniques and design algorithms and tools that:

- are fully automatic: no need for user/programmer annotations;
- enables one to describe the hardware easily;
- compute safe WCET (considering all input data).

We model the WCET problem as a *two-player timed game*. Intuitively Player 1 is the program, and Player 2 is in charge of deciding the outcome of the *comparison* instructions (e.g., `cmp`, `tst` that set the conditional branching conditions) that depend on the input data. As the choice of the input data is not controllable by Player 1, we obtain a two-player (untimed) game. Timing constraints come from the pipeline and caches timing specifications. The pair  $\langle \text{program}, \text{hardware} \rangle$  is thus modelled by a two-player *timed game*. The problem we solve on this timed game is an *optimal time reachability problem*:

“What is the optimal time for Player 1 to reach the end of the program ?”

To build the untimed game, we need to infer some knowledge on the program to be analysed and this is done using the technique of *program slicing*. Modeling of the hardware (pipeline and caches) is done using *timed automata*. To solve timed games we use the model-checker UPPAAL-TiGA.

A prototype tool chain has already been designed for the ARM9 processor; details are available at <http://www.ircyn.fr/franck/wcet>.

### ***Internship Proposals.***

There are quite a lot of areas to be further investigated during an internship on this project, depending on the Intern abilities and expectations:

- refine the program slicing technique;
- develop a technique and theory to reduce the number of paths to be explored;
- design more accurate (timed automata) models for the hardware (pipelines and caches) and for new hardware (ARM11 etc).

All of these subjects can be addressed from a theoretical angle and then the solution can be implemented in a prototype tool.