WCET Estimation from Object Code implemented in the PERF Environment

Douglas Renaux, João Góes, Robson Linhares
Laboratory of Embedded Systems Innovation and Technology – LIT
CEFET-PR
Brasil

Abstract

"*The estimation of the Worst Case Execution Time of a function produces results that are **safe** and that have a **low error**, even in architectures using pipelines and caches.*" This is our thesis; in this paper we present results that indicate that this thesis is correct.

The two basic approaches to obtain WCET of a piece of code are estimation and measurement. At LIT, a tool called PERF is under development. This tool uses both approaches so to obtain the best of both worlds. Measurement provides precise results, but requires the target to be built and running the worst possible scenario, which is often hard to determine. On the other hand, the precision of estimation methods is highly dependent on the complexity of the estimation model. PERF is a design and evaluation environment: a project can be defined, files can be edited, compiled, linked; the resulting code can be analyzed from a timing perspective both via estimation and via measurement. In this way, we intend to encourage the developer to perform time estimations as early as possible in the design cycle. Any tool (commercial or academic) can be inserted in PERF via *plug-ins*. This was the case of the text editor, the compiler and the linker. Hence, PERF is actually a framework to which many tools can be added. PERF works with the object code generated by the integrated tools, in order to obtain execution time limit estimations for functions that compose a real-time systems' software project.

In this paper we present the PERF environment's architecture, with emphasis on the integrated time estimation model and the results obtained using this model.

1. Introduction

Any tool which intends to estimate execution times should take two different domains into consideration: the source code, which the developer usually uses to develop his software project, and the executable code, on which the time estimations are actually performed.

The problem of execution time estimation of a program is usually divided into 3 sub-problems: execution path analysis on source level; source code and machine code correlation; and execution time analysis for each individual machine instruction at each path of the object code.

A strategy based on these 3 sub-problems would search for existing paths in the source code and tries to relate them with the corresponding paths in the object code. Meanwhile, this relationship may not be easily determined, especially in cases where the code optimizations are enabled. The correlation between source-code and object-code may not be trivial, specially for a tool which intends to do this automatically; so, it is possible to concentrate the estimation tool's work on the object-code path's determination and on the estimation of each of its execution times, leaving to the compiler the responsibility of doing the correlations and interpreting the results obtained through the analysis of object code. Another advantage of this strategy comes from the possibility of analyzing object-code present in code libraries, to which the source code is usually not available.

The resolution of the third sub-problem, concerning to the individual execution times of any machine instruction, is affected by the quality of the model that expresses the hardware platform on which the code is to be executed. In order to minimize the estimation error, when compared to the measured time values for a given execution path, this model should consider internal architectural features that have any influence over the execution times of the instructions, such as cache memories and pipelines (if available). Moreover, it is useful for the model to be reconfigurable, so that the estimation tool is able to address several target architectures of real-time systems. The reconfigurability feature can be available to the tool's user, in a way that allows him to adapt the estimation process to the architectures of his interest, using configuration parameters; that requires the estimation algorithms to be generic and able to use the configuration parameters of any architecture in a similar way.

2. The PERF environment

PERF is a tool, under development at LIT, which intends to be a complete design and evaluation environment. At PERF a software project can be defined, edited, each of its modules can be compiled and linked; moreover, the resulting object code can be analyzed from a timing perspective both via estimation and via measurement, which makes PERF suitable for development of software for real-time systems. In this way, the use of PERF intends to encourage the developer to perform time estimations as early as possible in the design cycle.

PERF's architecture is composed of a central core, controlled through a graphical interface. This core is intended to manage a set of integrated tools, each one performing a determined task in the development process (editors, compilers, linkers, time analyzers, etc.) and configured via a *plug-in*. Any tool (commercial or academic) can be inserted in PERF via *plug-ins*, allowing the developer to use only the functionalities which are strictly needed for the target platform and the type of design /evaluation process of interest.

2.1 Time estimation tool

PERF aggregates an execution time estimation tool, whose estimation process is shown in Figure 1. This figure shows that an important design decision of PERF's estimation tool was to analyze object code, instead of source code or executable code. Object code has all the information that is relevant for timing analysis and it may include debug information that relates the executable code to the source code. Also, by analyzing object code, all the libraries can have their timing information extracted, even the commercial libraries.