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.

![Figure 1 - Time estimation process used in Perf](image)

Figure 2 shows how PERF presents the different views of a function to the developer. It is worth noticing that the correlation between source and machine code can also be obtained through debug information (as it is shown by the gray shaded areas on the source code, machine code and control flow graph views); nevertheless, the developer can still be asked to provide data flow information necessary for the estimation, such as number of loop iterations, so that the several execution paths found in the control flow graph (CFG) are correctly estimated and the correct values for BCET, TCET and WCET are shown in their respective view.

How does PERF obtain execution time estimations (Figure 1):

1) **Analyze the object code and extract the control flow graph.**

Each node of the graph represents a code segment, i.e., a sequence of machine instructions at which only the last instruction can be a branch (absolute jump, conditional jump, procedure call, software interrupt,...) and only the first instruction is the target of a branch. The control flow graph is obtained in a
function-by-function fashion, even for the commercial libraries, and each function can have its CFG individually shown, as well as the possible correlation with the source code (if available).

Figure 2 - The PERF tool presenting several views of a function: source, machine code, control graph and timing information

2) All possible paths in a procedure call are analyzed.

Loop structures require information from the user specifying the minimum, typical, and maximum number of iterations. This information can be given in the form of comments in the source code or requested interactively during the analysis (Figure 2).

3) For each path, a time estimation (BCET, TCET, and WCET) is obtained based on the configured hardware model for the target platform.

The hardware model is the main functionality of the execution time estimation tool. This model intends to address the third sub-problem of the execution time estimation problem (the analysis of the individual execution times for each machine instruction) by considering internal architectural features of the target platform’s hardware.

The computation model implemented at PERF is divided in two basic parts: the estimation algorithms, which are generic and should work in a similar view, independent of the target platform; and the architecture configuration parameters, obtained from an external plugin.

The generic estimation algorithms consider the influence, on machine instruction execution times, of multi-stage pipelines, instruction caches of several sizes and associativity degrees and prefetch queues. It is possible, considering the influence of these features in a correct way, to obtain time estimation values with estimation errors lower than 10%, comparing to measured time values; this is valid not only for RISC architectures but for CISC architectures too. The efficiency of the model’s work depends on a correct configuration for the architectural parameters via the configuration plugin; it is conceived as a series of data structures in C++ programming language, which can be configured by a developer using the manufacturer information about the target platform.

The model differentiates the values of BCET and WCET, for each analyzed path, by a pessimistic factor introduced in the algorithms. For this factor to be correctly addressed, the best and worst case execution time values for each machine instruction should be modeled during the configuration of the model; moreover, pessimistic and non-pessimistic considerations are also made concerning to the removal of instructions from the prefetch queue and the existence of bus conflicts between instruction and data accesses.
TCET values, by their way, are obtained through typical-case annotation, provided by the user for the number of loop iterations; in functions where no loop structure is present, the TCET value is the same as the WCET value.

The loop processing, for any analyzed path, is done using a strategy that eliminates loop redundancies. This decreases the processing times for loops with a great number of iterations.

4) A report is generated to inform BCET, TCET and WCET of each function.

The execution time of called functions is included in the execution time of the caller, as long as the execution time of the callee is known. Several iterations may be required to evaluate all these dependencies.

3. Results

Two processor architectures were initially considered for the time estimation process and modeled, using the proposed hardware model. The first one was a Intel 80C186EC processor, a CISC core with two pseudo-pipelined stages, a 6-byte prefetch queue and no instruction cache memory; the second one was a Motorola PowerPC MPC860 processor, a RISC core with an 8-stage pipeline and a 2-set associative instruction cache memory of 4 Kb.

The estimation tool was used to estimate the execution time of about a hundred functions, including all the functions of a real-time kernel. The results obtained from the estimation tool are all safe (the estimated WCET is never lower than the actual) and the maximum error was of 10% for the 80C186EC architecture and 15% for the PowerPC architecture. This error, above the initial requirement of 10%, is due to the inadequate equipment used to validate the model by comparison with measured values; this is one of the difficulties that can be faced by the developer which intends to configure his own processor model, provided that the manufacturer information for each instruction’s execution time of a certain processor are not always accurate.

Some weaknesses are still detected on PERF’s execution time estimation tool:
- The inefficiency of solving all kinds of jump tables, used for indirect branch operations (for example, the compilation of most ‘switch’ structures). PERF is only capable at this moment to find jump tables in architectures for which the base address is encoded in the instruction code (for example, the Intel 80C186EC architecture).
- For optimized code and code for which the source is not available (for example, libraries), it can be difficult for the developer to determine the exact number of some loops’ iterations, when asked by the tool to provide those values.

4. Conclusions and future work

On going work in the PERF environment include: improving the computational models for the PowerPC architecture to reduce the estimation error, development of computational models of data caches (currently only instruction caches are modeled) and a tool for scheduling analysis is also under development.

The results obtained so far with PERF are very encouraging. Our previous work on timing analysis included the analysis of source code, of the assembly listing produced by the compiler and of binary code. Each one has strong aspects, but from our perspective, object code analysis is the best option.

References

LINHARES, R. Modelamento de Hardware Visando À Estimação do Tempo de Execução de Programas. Dissertação de Mestrado, CPGEI, CEFET-PR. Curitiba, PR, Dezembro 2001