Time Bands: A Software Approach for Timing Analysis on Resource Constrained Systems

Giacomo Valente
DISIM
Via Vetoio, 1
L'Aquila, Italy
giacomo.valente@graduate.univaq.it

Marco Rotondi
DISIM
Via Vetoio, 1
L'Aquila, Italy
marco.rotz@gmail.com

Vittoriano Muttillo
DISIM
Via Vetoio, 1
L'Aquila, Italy
vittoriano.muttillo@graduate.univaq.it

ABSTRACT

Timing analysis of embedded systems is an operation performed when there are tasks that have to execute with a well precise deadline, and need to be scheduled, such as those on real-time systems. The diffusion of embedded systems to different kind of application areas is driving platforms toward heterogeneous multi-core architectures, that require a timing analysis done by using measurement based techniques. Measurements collection, when done via an instrumentation of the application, can cause an overhead in the execution time, footprint and necessary space to store data, that can affect the behaviour of the system. In such a scenario, this work proposes a framework that allows a user to quickly perform instrumentation choices, by using a concept named Time Band, and to have a direct feedback about the impact of its choices on some performance parameters. Time Band is then applied to Rapitime, a diffused timing analysis tool, and first tests have been done on IA-32 and PowerPC architectures, showing the advantages of different techniques the can be applied to realize the framework.

Keywords

Timing analysis, Resource constrained systems, Embedded Systems, WCET

1. INTRODUCTION

The analysis of a system from the point of view of timing, named timing analysis, relates to the set of operations performed to collect information about its timing performance. *Worst Case Execution Time* (WCET), *Worst Case Response Time* (WCRT) and *High-Watermark Execution Time* (HWET) are three parameters used in different contexts, for example in real-time systems design and general embedded systems development. Several tools have been developed in order to support timing analysis, each one targeting one or more of the representative times indicated above. Such tools rely on different techniques to make evaluation or estimation, for example *static analysis* or *measurement based* ones. In this context, the latter have gained a strong attention in the last years [5], mainly because the static analysis generally requires a model of the processor that could be not available or too complex, in particular when considering the complex architectures (e.g. multi-core) recently adopted also in the embedded systems domain. On the contrary, the measurement based analysis helps to support the calculation of representative times by keeping into account the real target behaviour. In this context, *Rapitime* [3] represents a tool that performs a measurement based analysis: it makes use of measurements taken from the system at runtime, and joins them with a static description of the application, in order to delineate the worst case path that leads to WCET. The measures are constituted of timestamps related to execution of the code, taken after a source code instrumentation phase and using *Instrumentation Points* (Ipoints) inserted in the application: the level of instrumentation can be selected by the designer. However, since in the embedded systems domain resources can be very limited, the use of Ipoints can easily lead to a meaningful overhead, mainly related to *Timestamp Acquisition* (TA) and their *Storage*, the two operations required by the instrumentation action. So, this overhead can distort the real behaviour of the system and, more in general, can drive to the variation of other features such as *Memory Footprint* (MFP) and *Execution time* (ET), and also provide a certain *Storage Impact* (SI). For such a reason, there is the need for a solution that offers a direct feedback, among the possible instrumentation strategies, about the impact on the parameters that characterize the application (MFP, ET, and SI) and the level of accuracy that can be obtained. Moreover, such a solution should provide, while satisfying typical embedded systems non-functional requirements (e.g. MFP, ET, and SI), a set of timestamps sufficient to provide the required monitoring action (a feature called "monitorability"). In this context, this work provides the preliminary vision of a framework, and related techniques/tools, that, starting from measurement based timing analysis tools, offers this kind of solution.

ICPE'17 April 22-26, 2017, L'Aquila, Italy

ACM ISBN 978-1-4503-4404-3/17/04.
DOI: http://dx.doi.org/10.1145/3030207.3053665

2. PROPOSED APPROACH

The work proposes a framework that supports the designer for timing analysis on embedded systems. Specifically, it offers a direct feedback among instrumentation choices, their impact on the parameters that characterize an application (MFP, ET and SI) and the level of accuracy. The

work is presented basing on *Rapitime* tool. A new concept, named *Time Band*, is proposed: it refers to a source code instrumentation with different detail degrees. For each Time Band, a proper instrumentation will be obtained, that gives some information and causes certain overhead. Basing on this concept, the user will have the possibility to use the framework in two ways: the first by inserting information about platform and application (such as allowed ET overhead, MFP increase and SI), defining the type of measure required, and waiting for an automatic generated configuration suitable for his constraints; the second by acting directly on Time bands selection and code segments in order to drive the instrumentation and, consequently, the accuracy and the overhead. In order to identify a good instrumentation solution, and to address the problem of overhead limitation, a *runtime variable configuration* is proposed, with implementation in four versions using four different techniques reported in the poster.

3. RUNTIME VARIABLE CONFIGURATION

Four techniques are proposed to support the identification of an instrumentation solution able to fit monitorability and non functional requirements. In the following, a timestamp is considered as an instance of an Ipoint, while N is supposed to be the total number of Ipoints that are inserted in the code. The techniques are the following ones: (i) *Conditional*, (ii) *Function Pointer*, (iii) *Assembly Nop* and (iv) *Function Copy*.

In the first one, a *conditional* expression checks if an Ipoint is enabled by using an array of size N called *enabled_Ipoints*: if yes, the timestamp is stored. A considerable overhead has been observed, also when all Ipoints are disabled: this is mainly due to the added instructions for conditional checks.

The second technique is similar to conditional but it uses function pointers. Two functions are considered: *store_timestamp* and *do_nothing*; the latter has the same signature of the former, but it has an empty body. *enabled_Ipoints* is a function pointer array that points to *store_timestamp* if Ipoint is enabled, otherwise it points to *do_nothing*.

In the third technique, the *Nop* assembler instruction is considered: *Nop* is a common word to indicate no operation. After code instrumentation, it is possible to substitute all calls to *store_timestamp* with *Nops* and then restore only the desired ones. This solution shows a problem: on the tested IA-32 architecture, an overhead on ET similar to those obtained with a function call has been noted. This overhead is architecture dependent. Therefore, this way can be investigated only in the case the architecture provides a very low overhead.

In the last technique, named *Function Copy*, every function has a copy. The original one is not instrumented, named *function_name_NO_INSTR* while the copy, *function_name_INSTR*, yes: a function pointer is used to switch between these two ones.

4. VALIDATION

Some validation activities have been performed by inserting the different techniques in the *cjpeg* application, a program to compress an image file in JPEG format, taken from *MiBench* benchmark suite [4]. These first activities are related to the evaluation of the impact of the techniques on the original program: they have been done considering only empty instrumentation. Experiments have been performed on two platforms: the first is a laptop with a Linux 64-bit OS running on Intel Core i7-2630QM processor with 4 GB RAM memory, while the second a PowerPC G4 architecture[1] with 512 MB RAM memory, where execution has been simulated by using QEMU [2]. Results are reported in Table 1. *Original* row is the original application, *Time Full* refers to the application instrumented with the *Time Full* profile of *Rapitime*, that offers the maximum level of accuracy, but also the maximum ET, MFP and SI overhead, and the four following rows represent the four different techniques to perform the *runtime variable configuration* proposed approach. The PPC Nop implementation is scheduled for a future work because the assembly is different than x86.

Table 1: Empty instrumentation measures

	Intel		PowerPC	
Technique	**ET** (%)	**MFP** (%)	**ET** (%)	**MFP** (%)
Original	0	0	0	0
Time full	400	23	763	45
Conditional	41	62	118	90
Function pointer	80	63	407	91
Nop	40	23	-	-
Function copy	6	123	10	109

5. ACKNOWLEDGMENTS

This work has been supported by Rapita Systems LTD.

6. REFERENCES

[1] Powerpc. https://www-01.ibm.com/chips/techlib/techlib.nsf/productfamilies/PowerPC/. Accessed: 2016-12-22.

[2] Qemu homepage. http://wiki.qemu.org/Main_Page. Accessed: 2016-12-22.

[3] Rapitime. https://www.rapitasystems.com/products/rapitime. Accessed: 2016-12-22.

[4] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, and R. B. B. T. Mudge. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14, Dec 2001.

[5] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Paut, P. Puschner, J. Staschulat, P. Stenstrom. The worst-case execution-time problem — overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.