Automatic Scenario Detection for Improved WCET Estimation

Stefan Valentin Gheorghita, Sander Stuijk, Twan Basten and Henk Corporaal
Eindhoven University of Technology, EE Department, Electronic System Group
{s.v.gheorghita, s.stuijk, a.a.basten, h.corporaal}@tue.nl

ABSTRACT

Modern embedded applications usually have real-time constraints and they are implemented using heterogeneous multiprocessor systems-on-chip. Dimensioning a system requires accurate estimations of the worst-case execution time (WCET). Overestimation leads to over-dimensioning. This paper introduces a method for automatic discovery of scenarios that incorporate correlations between different parts of applications. It is based on the application parameters with a large impact on the execution time. We show on a benchmark that, using scenarios, the estimated WCET may be reduced with 16%.

Categories and Subject Descriptors: C.3 [Special-Purpose and Application-Based Systems]: Real-time and embedded systems

General Terms: Algorithms, Performance, Design

Keywords: WCET, Real-Time, Scenarios

1. INTRODUCTION

Embedded systems usually consist of processors that execute domain-specific programs. Many of their functionalities (tasks) are implemented in software, which is running on one or multiple generic processors, leaving only the high performance functions implemented in hardware. Typical examples of embedded systems include TV sets, cellular phones and printers. As many of these systems have real-time constraints, to dimension them, accurate estimations of the worst-case and best-case execution time (WCET and BCET) of their tasks are required. More precisely, it is required to tightly bound the execution times of all feasible paths of the program. If the minimum and maximum duration of all these executions are denoted by T_{min} and T_{max}, the *actual bounds* of a program execution time are given by the interval $[T_{min}, T_{max}]$. The goal of the estimation is to find an interval $[t_{min}, t_{max}]$ that tightly encloses the actual bounds (see fig. 1) [9]. This interval represents the estimated bounds of the program execution time, and respectively, t_{min} and t_{max} are the estimated BCET and WCET of the program. Since estimation of WCET and of BCET are very similar to each other and the techniques developed for one can be easily adapted for the other, we focus only on WCET.

DAC 2005, June 13–17, 2005, Anaheim, California, USA.

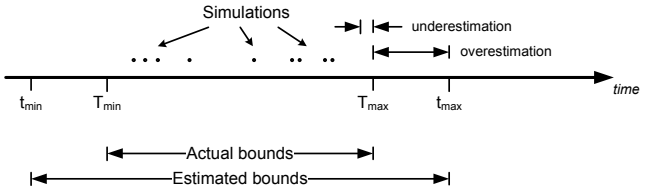


Figure 1: Estimated vs. actual bounds.

To determine the estimated WCET of a program, all the factors that affect the program execution time must be considered: the feasible execution paths and the execution time of each instruction in each path. In this paper, we discuss the first factor, which is platform independent. The second one depends on architecture parameters, like number of cycles per instruction type, memory hierarchy and pipelining. A micro-architecture model is needed to analyze it.

One of the problems in finding the estimated WCET of a program is that the longest execution path is unknown in many cases. If it can be determined, the problem is trivial to solve. Simulation of all execution paths is clearly impractical as their number is usually exponential in the program size. The results from the simulation of a subset of feasible execution paths are very likely to fall strictly within the actual bounds of the program, even if the subset was very carefully selected ([4]). This leads to an underestimation of the bounds (see fig. 1). With some extensions, simulation-based analysis can be used for designing soft real-time systems, but it cannot be tolerated in analysis of hard real-time systems.

To avoid the explosion in the number of execution paths, many approaches use a timing schema as the basis for estimating the WCET. Such a timing schema is attributed to certain high-level language constructs, and it is essentially a set of formulas for computing an upper bound on their execution time [17]. Nevertheless, the timing schema cannot be directly applied to programs because not all the needed information is contained in their source code. One of the reasons is that programs contain non-manifest loops. In many cases, the number of iterations of these loops cannot be determined automatically as they may depend on input parameters. With only a few exceptions (e.g. [16, 5]), all the existing techniques rely only on the programmer to provide loop bounds.

Although by using a timing schema the explosion in the number of paths is avoided, often a large number of infeasible paths is considered in WCET estimation, potentially introducing a big overestimation (see fig. 1). This is because a timing schema does not differentiate between infeasible and feasible paths, and the estimated WCET may be due to one of the infeasible paths.

In this paper, we propose an automatic method for reducing the number of infeasible paths considered in a timing schema based

WCET estimation. We use static analysis to discover the correlations between parts of an application. These correlations are used to partition the application in different, so-called, *scenarios*. The application estimated WCET is computed as the maximum estimated WCET of these scenarios. Our method is platform independent and can be applied on top of all existing WCET estimation methods based on timing schema.

The paper is organized as follows. Section 2 compares our work with related approaches. Section 3 describes how a timing schema works. Section 4 shows how scenarios can be used to estimate more accurately the WCET for an application. In section 5, we introduce an algorithm suitable for scenario discovery. The evaluation of our approach on two test cases is presented in section 6. Our conclusions and future plans are presented in section 7.

2. RELATED WORK

In this section, we compare our work with different approaches for WCET overestimation reduction. Other areas where scenarios are already used are also presented.

Many approaches to reduce the overestimation of WCET have been studied. Some of them use C [14] or assembly [11] level user annotations to describe the maximum number of executions for different statements. On top of these approaches, in [13], a mechanism that allows a user to specify the correlations between parts of the application is added. However, all of these approaches require correlation information added into the source code, which is what we avoid in our work.

Another way to control the WCET overestimation is parametric WCET. There are different methods to compute it, based on timing schema [6] and path enumeration [3]. Manual annotations for constraints on loop counters and infeasible paths are needed. As an extension, in [18], an iterative method to compute parametric WCET bounds for simple loops has also been suggested. However, even for a fully automatic approach, which can find both loop bounds and infeasible paths [10], there is a huge explosion in the number of parameters. It is very hard to identify the most important parameters only by the variables' name. In our approach, we introduce a method which helps in identifying the parameters that influence the estimated WCET the most.

The scenario concept was first used [19] to capture the data-dependent dynamic behavior inside a thread, in order to better schedule a multi-thread application on a heterogenous multi-processor architecture. In [12], the authors try to apply different source-to-source transformations to each discovered scenario to improve the overall application performance. To the best of our knowledge, we are the first ones to present a technique for automatically detecting scenarios and the first ones to use scenarios to reduce WCET overestimation.

3. A SIMPLE TIMING SCHEMA

Before going into the depth of our method, we first present how a timing schema works. All existing timing schema are based on Shaw's schema [17]. It estimates the program WCET in a bottom-up manner, using the following rules, in which B, B_1, B_2 are blocks of statements and n is the number of loop iterations:

$$\text{WCET}(B_1; B_2) = \text{WCET}(B_1) + \text{WCET}(B_2) \tag{1}$$

$$\begin{aligned}&\text{WCET}(\textit{if } B \textit{ then } B_1 \textit{ else } B_2) = \\ &\text{WCET}(B) + \textit{max}(\text{WCET}(B_1), \text{WCET}(B_2))\end{aligned} \tag{2}$$

$$\begin{aligned}&\text{WCET}(\textit{while } B \textit{ do } B_1) = \\ &(n+1) * \text{WCET}(B) + n * \text{WCET}(B_1)\end{aligned} \tag{3}$$

Informally, the WCET of a sequence of two blocks of statements is the sum of their WCETs (eq. 1). For an `if-then-else` statement, the WCETs of `then` and `else` branches are compared and the maximum is added to the WCET of the `if` condition (eq. 2). For a `while` loop, the WCETs of the loop body and condition are multiplied by the number of iterations, and the condition WCET is added one more time because of the loop exit test (eq. 3). These equations cover the entire ANSI C grammar (which is the most used programming language for the embedded systems area), as all other control constructs can be rewritten using these simple ones.

4. SHARPER UPPER BOUNDS USING SCENARIOS

In order to refine the estimation of the WCET, we divide the application in a set of scenarios. *A scenario is defined as the application behavior for a specific type of input data.* The set of scenarios must cover all possible input data. An example of a scenario for the H.263 decoder [15] is the application behavior for any frame of type *P*. Together with scenarios for frame types *I* and *B*, they cover all possible behaviors.

For each scenario, those parts of the application source that are never executed, are identified and removed, and the WCET is estimated using, for example, Shaw's schema. Preserving the conservativeness of estimation, the WCET of the entire application is then defined via the following equation:

$$WCET(app) = \max_{S \in Scenarios}(WCET(S)) \tag{4}$$

To emphasize the possible benefit of scenarios in WCET computation, fig. 2 presents an educational example. Note that only the order in which functions f and g are executed differs, based on the value of ct. Using only a timing schema, the estimated WCET is

$$2 \cdot 8 \cdot \max(WCET(f), WCET(g)) + const.$$

where $const$ represents the test and loop overhead. Considering two scenarios defined on different values of variable ct (the first scenario for $ct = 1$, and the second one for $ct \neq 1$), the WCET is

$$8 \cdot (WCET(f) + WCET(g)) + const$$

If the WCETs of f and g are very different, then the use of scenarios seriously reduces the overestimation compared to the approach based only on timing schema.

Besides correlations between different parts of the code, as illustrated above, scenarios may also incorporate a different number of loop iterations. For example, in one scenario, a loop iterates for maximally 10 times, and in another scenario the same loop iterates for maximally 5 times only. If the WCET for this code is computed without considering scenarios, the maximum number of iterations must be considered 10. In [7], a detailed example is presented.

Both the correlations between different parts of the source code and the number of loop iterations are considered in our algorithm for detecting scenarios.

5. AUTOMATIC SCENARIO DETECTION

Our approach is based on static analysis of the application source code and it consists of five steps: *(1)* identify the parameters that could potentially have an impact on the application execution time, *(2)* compute the maximum possible impact of these parameters on the WCET, *(3)* partition the application in scenarios considering these parameters together with their impact, *(4)* generate source code for each scenario and estimate their WCET using a timing schema and *(5)* compute the application WCET using equation 4.

	source code	Influence coefficient for variable ct
1	`if (ct == 1)`	$IC(ct) = 2 \cdot [max(8 \cdot IC_f(ct), 8 \cdot IC_g(ct)) + abs(8 \cdot WCET(f) - 8 \cdot WCET(g))]$
2	`  for (y=0; y<8; y++)`	$IC(ct) = 8 \cdot IC_f(ct)$
3	`    f(&b[x*8+y]);`	$IC(ct) = IC_f(ct)$
4	`else /* ct!=1 */`	
5	`  for (y=7; y>-1; y--)`	$IC(ct) = 8 \cdot IC_g(ct)$
6	`    g(&b[x*8+y]);`	$IC(ct) = IC_g(ct)$
7	`if (ct != 1)`	$IC(ct) = max(8 \cdot IC_f(ct), 8 \cdot IC_g(ct)) + abs(8 \cdot WCET(f) - 8 \cdot WCET(g))$
8	`  for (y=0; y<8; y++)`	$IC(ct) = 8 \cdot IC_f(ct)$
9	`    f(&b[x*8+y]);`	$IC(ct) = IC_f(ct)$
10	`else /* ct=1 */`	
11	`  for (y=7; y>-1; y--)`	$IC(ct) = 8 \cdot IC_g(ct)$
12	`    g(&b[x*8+y]);`	$IC(ct) = IC_g(ct)$

Figure 2: Educational example

1: The first step is based on the fact that usually a few parameters have a significant impact on the application execution time (e.g. in a video decoder: image size and type). Many of these parameters are read at the beginning of the execution and remain constant for the rest of it. Moreover, usually, there is only a small set of possible values for them (e.g. for an H.263 decoder, there is one variable which specifies the image type, with three possible values: *I*, *B* or *P*). In a C source code, these parameters usually appear as variables or fields of structures of integer or enumeration type. There is only one statement for each parameter in the program that changes its value (often it is set based on the program input data).

2: To identify from these parameters the ones that might influence the WCET the most, we first compute the application WCET using a timing schema. Then, the possible impact on the WCET of each variable or structure field (denoted by v) that respects the above observations, is computed in the form of its so-called *influence coefficient* ($IC(v)$). The $IC(v)$ represents the maximum possible variation (in cycles) caused by the different values of v on the WCET of the application.

To this end, the abstract syntax tree (AST) of the program is traversed in a post-order manner and $IC(v)$ is computed in each statement s as a sum of its contribution and the maximum of $IC(v)$ computed for all its successors in the program (eq. 5).

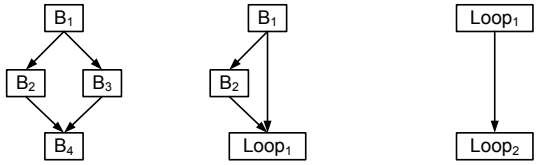
$$IC_s(v) = contribution(s) + \max_{x \in successors(s)}(IC_x(v)). \quad (5)$$

A statement may have zero, one or multiple successors. The last statement of a function or a loop body has no successors (e.g. fig. 2, line 12). Control statements may have multiple possible next statements (e.g. for if, the first statement of the $then$ and the $else$ branches, as appears in the max factor of the $IC(ct)$ from fig. 2, line 7). The rest of the statements has only one successor. For each statement, its successors are always already processed due to the post-order traversal of the AST.

The contribution of a statement to $IC(v)$ quantifies the maximum variation in execution time of the statement as caused by different values of v. Depending on the statement type, the contribution is:

- **function call**: the $IC(v)$ computed in the first statement of the called function f (e.g. fig. 2, line 12). If v is a parameter of the function call, a renaming is done for computing the $IC(v)$ inside the function.
- **loop**: the $IC(v)$ computed in the first statement of the loop body multiplied by the maximum number of iterations (e.g. fig. 2, line 11).
- **if**: if v appears in the if condition, and it is compared to a constant,

$$abs(WCET(then) - WCET(else)), \quad (6)$$



$S_1 : B_1, B_2, B_4$ $S_2 : B_1, B_3, B_4$ (a)

$S_1 : B_1, B_2, Loop_1(x)$ $S_2 : B_1, Loop_1(y)$ (b)

$S_1 : Loop_1(s), Loop_2(t)$ $S_2 : Loop_1(u), Loop_2(v)$ (c)

Figure 3: Examples of good scenario selection ($x < y$, $s < u$, $t > v$ are the number of iterations for loops).

or else 0 (e.g. abs factor of $IC(ct)$ from fig. 2, line 7).

- **switch**: if v appears in the $switch$ condition,

$$\max_{B \in Branches}(WCET(B)) - \min_{B \in Branches}(WCET(B)), \quad (7)$$

or else 0.

- **the rest of statements**: 0.

Equations 6 and 7 represent the only points where values different from zero are injected in our algorithm. Scenarios can be used for other purposes as well, such as memory usage estimation. The same method can be applied but with different formulas for equations 6 and 7.

3: The first statement of the program will yield the ICs computed for each possible parameter. To avoid an explosion in the number of scenarios, different criteria to select which parameters are used to define scenarios might be used. The selection may incorporate knowledge about the application combined with heuristics based on the computed ICs. An example heuristic is to select only those parameters with very big IC values. The best heuristic is an open point for further research. The algorithm used in the selection stage can almost always be automated and depends on what the scenarios are used for.

For each *selected* parameter, the constants it is compared to in the source code are collected. These constants, together with the comparison operators, are used to split the set of possible values of the parameter in subsets. A scenario is characterized in the end by the possible values of the selected parameters.

Figure 2 shows how the IC for variable ct is computed for the example presented in section 4. As it could already be seen in the source code, we can automatically detect two scenarios based on ct: $ct = 1$ and $ct \neq 1$.

At this point, we can refine our notion of a scenario to be a *part* of the application source code with a *specified maximum number* of loops iterations. The scenario's set of execution paths consists of all possible execution paths through it. In order to potentially obtain a reduction for the estimated WCET using scenarios, a scenario

should not include all application execution paths. To avoid an explosion in the number of generated and evaluated scenarios in step 4 of our algorithm, scenarios that have the set of execution paths included in another scenario's set may be ignored. To fulfill these two conditions, each pair of selected scenarios must fall in one of the following cases:

- there must be at least one part of the source code which is included in the first one and not in the second one, and vice versa (e.g. scenarios S_1 and S_2 from fig. 3(a)).
- one of the scenarios includes a part of the code which is not included in the other one and it executes a loop for a smaller number of iterations (e.g. scenarios from fig. 3(b)).
- they have different maximum numbers of iterations for two loops and for one loop the first scenario must iterate more than the second scenario, and vice versa for the second loop (e.g. scenarios from fig. 3(c)).

4: For each scenario, a modified version of the *unreachable code elimination compiler phase* is used to remove code never executed because of specific parameters values. The estimated WCET is computed based on a timing schema, like Shaw's one.

5: In the end, equation 4 is used to obtain the application WCET.

6. EVALUATION

All the presented steps were implemented on top of SUIF [2]. For our experiments, we used a micro-architecture model similar to ARM7TDMI [1]. This processor does not have caches and the pipeline effects were considered only inside basic blocks. For computing the WCET of scenarios we use Shaw's timing schema [17]. To determine the loop bounds, we use Rustagi's approach [16]. In the case when they cannot be automatically derived, we provided them ourselves.

We tested our method on two multimedia applications, an MP3 decoder [8] and a restricted H.263 decoder [15] that supports only *I* and *P* frames. The benchmarks structure and the steps of our experiments are detailed in [7].

For the MP3 decoder, we first estimated its WCET based only on the timing schema and computed the influence coefficient (IC) for all possible parameters. The ones with relevant IC (bigger than 100 cycles) were used to define scenarios. By splitting the application based on them, we obtain three possible scenarios (see table 1). The estimated WCET for each discovered scenario is smaller than the application WCET previously estimated. At the end, by using equation 4, the application WCET is reduced with 16%. If the rules from step 3 are relaxed, our tool generates 12 scenarios. However, as we expected, the application WCET is not reduced furthermore.

For the H.263 decoder, we did not reduce the overall WCET estimation. This is because the processing performed for an I-frame is a true subset of the processing done for a P-frame. However, relaxing the rules from step 3, two scenarios were discovered and an improvement of 39% for one of them was obtained (see table 2). Although based on scenarios the application estimated WCET is not reduced, the fact that there are scenarios with different WCET may be exploited in real-time systems design (e.g. use the spare time for I-frames to save energy by voltage/frequency scaling).

Table 1: MP3 Decoder scenarios ($WCET = 2.405.968$)

scenario number	block type	mixed flag	WCET (cycles)	reduction
1	2 (short)	0	2.079.124	16%
2	2 (short)	1	1.984.816	18%
3	$\neq$ 2 (long)	*	1.666.804	31%

Table 2: H.263 Decoder ($WCET = 1.308.538$)

Scenario	WCET (cycles)	Reduction
$pict\_type = 1 (Pframe)$	1.308.538	0%
$pict\_type = 0 (Iframe)$	794.350	39%

7. CONCLUSIONS AND FUTURE WORK

We have presented an automatic method for detecting sharper upper bounds on the estimated WCET of an application. It can be applied on top of all WCET estimation approaches based on timing schema. It is based on scenarios that incorporate both the correlations between different parts of the application source code and different numbers of iterations for the same loop. To discover scenarios, we propose an algorithm based on static analysis of the source code. A solution for preventing an explosion in the number of scenarios was also proposed. Our method was tested on two applications and, in one case, the estimated WCET was reduced with 16%.

In the future, we plan to develop methods to dimension real-time systems based on scenarios. We will consider that there are different WCETs per scenario together with knowledge about possible sequences of scenarios. We also want to extend our work in detecting scenarios to multi-tasking applications. Furthermore, we are investigating ways of detecting scenarios based on profiling.

8. REFERENCES

[1] ARM7TDMI datasheet. `http://www.arm.com/products/CPUs/ARM7TDMI.html`.

[2] S. Amarasinghe, J. Anderson, M. Lam, and C.-W. Tseng. An overview of the SUIF compiler for scalable parallel machines. In *Proc. of the 7th Conference on Parallel Processing for Scientific Computing*, San Francisco, CA, 1995.

[3] G. Bernat and A. Burns. An approach to symbolic worst-case execution time analysis. In *Proc. of 25th Workshop on Real-Time Programming*, May 2000.

[4] G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proc. of 23rd IEEE RTSS*, pages 269–278, Dec. 2002.

[5] J. Blieberger. Discrete loops and worst case performance. *Computer Languages*, 20(3):193–212, Aug 1994.

[6] A. Colin and G. Bernat. Scope-tree: A program representation for symbolic worst-case execution time analysis. In *Proc. of 14th IEEE ECRTS*, pages 50–63, Vienna, Austria, June 2002. IEEE.

[7] S. V. Gheorghita, et al. Sharper WCET upper bounds using automatically detected scenarios. Technical Report ESR-2005-04, TU/e, EE Dept., Electronic Systems Group, Eindhoven, Netherlands, Mar. 2005.

[8] K. Lagerström. Design and implementation of an MP3 decoder, May 2001. M.Sc. thesis, Chalmers University of Technology, Sweden.

[9] Y.-T. S. Li and S. Malik. *Performance Analysis of Real-Time Embedded Software*. Kluwer Academic Publishers, 1998.

[10] B. Lisper. Fully automatic, parametric worst-case execution time analysis. In *Proc. of 3rd Int. Workshop on WCET Analysis*, pages 99–102, July 2003.

[11] A. K. Mok, et al. Evaluating tight execution time bounds of programs by annotations. In *Proc. of the 6th IEEE RTOSS*, pages 74–80, May 1989.

[12] M. Palkovic, et al. Augmenting the exploration space for global loop transformations by systematic preprocessing of data dependent constructs. In *Proc. of PA3CT Symposium*, pages 33–36, Edegem, Belgium, Sep. 2004.

[13] C. Y. Park. *Predicting Deterministic Execution Times of Real-Time Programs*. PhD thesis, University of Washington, Seattle, Aug. 1992.

[14] P. Puschner and C. Koza. Calculating the maximum execution time of real-time programs. *Journal of Real-Time Systems*, 1(2):159–176, Sep. 1989.

[15] K. Rijkse. Video coding for narrow telecommunication channels at <64kbits/s. Technical report, Telenor R&D, 1995.

[16] V. Rustagi and D. B. Whalley. Calculating minimum and maximum loop iterations. Technical report, CS Dept., Florida State Univ., May 1994.

[17] A. C. Shaw. Reasoning about time in higher-level language software. *IEEE Transactions on Software Engineering*, 15(7):875–889, July 1989.

[18] E. Vivancos, et al. Parametric timing analysis. In *Proc. of the ACM LCTES'2001*, pages 88–93, Utah, June 2001.

[19] P. Yang, et al. *Multi-Processor Systems on Chip*, chapter Cost-efficient mapping of dynamic concurrent tasks in embedded real-time multimedia systems. Morgan Kaufmann, 2003.