Model Identification for WCET Analysis

Björn Lisper and Marcelo Santos
School of Innovation, Design, and Engineering, Mälardalen University
Box 883, S-721 23 Västerås, Sweden
{bjorn.lisper,marcelo.santos}@mdh.se

Abstract

Worst-Case Execution Time (WCET) analysis derives upper bounds for the execution times of programs. Such bounds are crucial when designing and verifying real-time systems. Static *WCET analysis derives safe upper bounds. For complex hardware architectures the hardware modelling is still a challenge, leading to long analysis times and a risk of large WCET overestimation. Therefore, hybrid WCET analysis methods have appeared, where measurements are used to augment or replace the detailed low-level static WCET analysis. These methods do not in general yield a safe WCET estimate, but can still be appropriate in soft real-time systems where such WCET estimates are not crucial.*

In this paper we make two contributions. First, we develop a hybrid WCET analysis method, which uses regression to identify parameters in the common linear Implicit Path Enumeration Technique (IPET) model for WCET calculation. The method can use timing measurements of different granularity, including end-to-end measurements, which reduces the need for fine-grained timing measurement instrumentation. It uses a novel kind of regression, which guarantees that the identified model does not underestimate any observed execution times. Second, we initiate the development of an IPET-based theory for hybrid WCET analysis test coverage, and we formulate and prove a coverage criterion for the tests needed to identify a safe model.

1. Introduction

The *worst-case execution time* (WCET) is needed to verify hard real-time properties. A *static WCET analysis* finds an upper bound to the WCET of a program from mathematical models of the hard- and software involved. If the models are correct, the analysis will derive a timing estimate that is *safe*, i.e., greater than or equal to the WCET.

To statically derive a timing bound for a program, information on both the *hardware timing characteristics*, such as the execution time of individual instructions, as well as the program's *possible execution flows*, to bound the number of times the instructions can be executed, must be derived. Consequently, static WCET analysis of a program is usually divided into three phases: a *flow analysis* of the program, a *low-level analysis* where the execution times for sequences of instructions in the program are decided from a performance model for the hardware, and a final *calculation* phase where the flow and timing information is combined to yield a WCET estimate [1].

Flow analysis can be done in many ways. Since this analysis is not the focus of this paper, we only mention [2] where abstract interpretation-based methods to detect bounds for loop iterations and infeasible paths are given. There is an extensive literature on low-level analysis: here, we briefly mention relevant work on pipeline analysis [3], cache analysis [4], [5], and branch predictors [6]. Calculation is nowadays mostly done by the *implicit path enumeration method* (IPET) [7], [8], since it offers superior generality and precision as regards program structure, hardware complexity, and handling of program flow constraints.

A problem with static WCET analysis is that these analyses can be hard. In particular, low level analysis is hard for modern processors with features such as caches, branch predictors, and dynamic out-of-order instruction scheduling. For such processors it can take a long time to obtain acceptably precise estimates [5], [1]. Also, the difference between the true WCET and the average execution time will typically increase. This may yield poor hardware utilization if dimensioning the system according to the worst case.

This has lead to an increased interest in *measurement-based* methods, where the hardware timing is estimated from measurements rather than through a formal timing model, and *hybrid* methods where measured data replaces some formally derived timing information in a static analysis framework. An example is the probabilistic method by Bernat et. al [9]. These methods typically avoid the cumbersome hardware modelling: thus, they can be quickly adapted to new hardware architectures. On the downside, the computed WCET estimate is in general not safe anymore. The methods need good sets of test data, which provide enough coverage to expose the worst timing behaviour for different program parts. They often rely on some sort of code instrumentation, which then yields a *probe effect* distorting the measured times: this can be especially disturbing if the instrumentation is fine-grained. However, in many soft real-time applications it is more important to utilize the hardware well than to always meet all deadlines: then, an underestimated WCET can be acceptable as long as the

resulting deadline violations are not too frequent.

We have developed a framework for measurement-based WCET analysis. Information from measurements of different granularity, including end-to-end timing measurements, can be freely combined in this framework. The basic idea is to view the measurement-based low-level analysis as a *model identification problem*, where parameters of a linear model are fit to measured data. Our contributions are:

- We formulate the derivation of a program timing model from measured timing data as a model identification problem. This model can be directly used for WCET analysis using standard methods: in particular, a hybrid method using static flow analysis and WCET calculation is immediate. We evaluate this hybrid WCET analysis method experimentally.
- We define a novel regression technique for identifying linear models, which guarantees that an identified model does not underestimate any output from the modeled system used in the model identification process. This is a necessary property of timing models used for WCET estimation.
- We lay the foundation for a theory of testing coverage for measurement-based WCET analysis, which is based on the IPET model. Within this framework we prove a coverage criterion which, when applicable, guarantees that the estimated WCET is safe.

The coverage criterion is in practice probably mostly applicable to quite simple hardware architectures. For hard real-time systems using such architectures, our hybrid method can be used with the criterion to produce safe WCET estimates. For more complex architectures the method can still be used to produce possibly unsafe WCET estimates: these may still be useful in soft real-time systems.

The rest of this paper is organized as follows: in Section 2 we discuss related work, focusing on measurement-based methods. In Section 3 we first give a short primer to model identification. We then review the widely used linear IPET model for WCET calculation, we give a method to identify the parameters in the model from measured execution times, and we describe a hybrid method for WCET analysis based on this. Section 4 provides our theoretical development of testing coverage criteria for measurement-based WCET analysis, when the analyzed system is described by a linear IPET model. In Section 5 we make an experimental evaluation of our hybrid method. Section 6, finally, wraps up and provides some topics for further investigation.

2. Related Work

Measurement-based WCET analysis methods can be roughly divided into two groups: methods which attempt to *search for the input causing the WCET to appear*, and subsequently runs the program with this data while measuring its execution time, and methods that *mimic the static approach* with flow- and low-level analysis, however using measured data, followed by a calculation. In the first class, we find some methods which are based on evolutionary algorithms [10], [11]. In the *single-path approach* [12], programs are written as to only have a single possible execution path. If the hardware in addition is simple enough, then the execution time is independent of the input and the WCET can be found by a single end-to-end measurement.

In the second class we find the probabilistic WCET analysis method in [9], where a fine-grained instrumentation is used to measure the execution times of small program fragments (like basic blocks). This method also estimates probability distributions. The method was used by Colin and Petters to find which processor features have the largest influence on the WCET [13]. Later, the *instrumentation point graph* [14] was introduced as a means to describe code instrumentation for timing measurements. A second representative is found in [15], where somewhat longer execution paths are timed and the results are combined into a WCET estimate. Model checking is used to generate test data.

The work by Lindgren et. al. [16] also belongs to the second category, and is the one closest to our approach. A linear equation system is set up from a set of measured end-to-end execution times, and is solved for the basic block execution times. Due to program flow constraints, all these execution times cannot always be solved for, but basic blocks can sometimes be grouped to yield a solvable execution system.

Recent work [17] in instruction set simulation is related in that a model identification approach, somewhat similar to ours, is used to determine an approximate timing model for the simulators.

Finally, our criterion for safe WCET estimates is closely related to similar criteria in *basis path testing* [18].

Our work generalises the hybrid approaches above by allowing a more flexible combination of timing measurements for different execution paths, including end-to-end measurements. This enables a more coarse-grained instrumentation, which will decrease probe effects. Since our proposed method is based on the IPET model, it is immediate how to set up the analysis to have good precision also for variable basic block execution times caused by pipelines, caches, etc.: the method in [16] can only handle simple processors where basic blocks have constant execution time, and it is unclear whether the other methods can handle variable execution times in a more than approximate way. The coverage theory presented in Section 4 applies also to the previously published hybrid methods: as far as we know, no such theory has been published before.

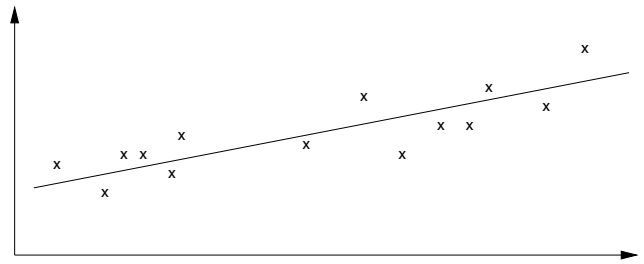

Figure 1. Least-squares fitting of linear function to data.

3. Identification of IPET Models for WCET Calculation

Model identification means to fit the parameters of a mathematical model to observed data. Typically, the model is a numerical function f with unknown coefficients, and the observations are represented as a vector of inputs $\vec{x}$ with a vector of corresponding observed outputs $\vec{y}$. The identification then amounts to finding values for the coefficients which make each function value $f(x_i)$ match the corresponding observed value y_i as closely as possible. Writing $f(\vec{x})$ for $(f(x_1), \ldots, f(x_n))$, this usually amounts to minimizing $\|\vec{y} - f(\vec{x})\|$ for some vector norm $\| \cdot \|$. A very common choice is the Euclidean norm $\| \cdot \|_2$, which yields the well-known "least-squares" approximation [19]. When the function values are linear combinations of the coefficients to be identified, this is called *linear regression*. See Fig. 1 for the canonical example of fitting a linear function $f(x) = ax + b$ to observed data by selecting a and b as to minimize the Euclidean norm.

IPET (Implicit Path Enumeration Technique) is widely used in WCET analysis for calculation of WCET estimates [7], [8]. The basis is a linear timing model, which enables the formulation of the WCET calculation as an integer linear programming (ILP) problem.

We now give an account for the IPET model. We use the following formulation: the underlying program model is a *control flow graph* (CFG) (N, E, s, t), where N is a set of nodes, $E \subseteq N \times N$ is a set of edges, $s \in N$ is the *enter node*, and $t \in N$ is the *exit node*. Nodes typically correspond to basic blocks, and the edges represent possible transfers of control. s and t provide the unique enter and exit points of the program.

For each node or edge $i \in N \cup E$, we assume an *execution time* τ_i, and an *execution count* c_i. Each execution count c_i can be seen as the final value of a counter, which is initialized to zero and then incremented each time entity i (node or edge) is traversed (see Fig. 3). We assume a system of integer linear inequalities $A\vec{c} \leq \vec{b}$ constraining the execution counts, where $\vec{c}$ is the vector of these [8]. This system is given by the flow analysis, and will typically contain bounds on loop iterations, infeasible path constraints, and *structural constraints* which are imposed by the CFG structure (see Section 4). The sum $\vec{c}^T\vec{\tau} = \sum_{i \in N \cup E} c_i \tau_i$ is the *total execution time* of the CFG for the given execution counts. An estimated WCET of the modelled system is then given by the solution to

$$\max \vec{c}^T \vec{\tau} \tag{1}$$

subject to the constraints $A\vec{c} \leq \vec{b}$. This is an ILP problem, which can be solved using standard ILP solvers [20].

The basic IPET model is valid for processors with constant execution time for the instructions. In addition, the execution times for the edges in the CFG can account for the effects of simpler pipelines, typically by setting them to negative *overlap factors* which model the possible effects of pipelining across basic block borders [3]. The effects of more complex architectural features, like advanced pipelines, caches, branch predictors, etc., can be taken into account by making the IPET model more context-dependent through considering different *instances* of CFG entities. For instance, in systems with caches it is often beneficial to treat the first loop iteration(s) separately, since they often have different cache hit ratio than the rest of the iterations [4]. The loop bodies will then be given separate execution counts for these iterations. The resulting IPET model will have more variables, but can be made more precise. IPET models have been used successfully for processors up to the level of PowerPC and Motorola Coldfire [5].

In static WCET analysis, the low-level analysis uses formal hardware architecture models to find a value for $\vec{\tau}$ in (1) which yields a safe timing model. As hardware architectures grow more complex, this becomes increasingly difficult. An alternative is to find $\vec{\tau}$ from observations. The IPET timing model is *linear* in $\vec{\tau}$: this enables the use of regression methods to identify $\vec{\tau}$ from observations.

Assume an IPET model with n *entities* (i.e., instances of basic blocks or CFG edges used by the model). An *observation* of a program run is a pair $(\vec{c}, t)$ where $\vec{c}$ is the vector of execution counts for the run, and t is its measured execution time. Assume a set of observations $(\vec{c}_j, t_j)$, $j = 1, \ldots, m$. The IPET model then gives rise to a linear equation system

$$C\vec{\tau} = \vec{t} \tag{2}$$

where C is the $m \times n$-matrix with rows $\vec{c}_j$ of recorded execution counts, $\vec{\tau}$ is the array of execution times τ_i for IPET entities i, and $\vec{t}$ is the array of observed execution times t_j.

If $m > n$, then (2) is overdetermined: standard linear regression (the "least-squares" method) can then be used to find $\vec{\tau}$ such that $\|\vec{t} - C\vec{\tau}\|_2$ is minimized. However, the resulting IPET model will in general have the undesired property that *for some runs j, it will underestimate the observed execution time t_j*. This makes linear regression less suitable for deriving IPET models to be used in WCET

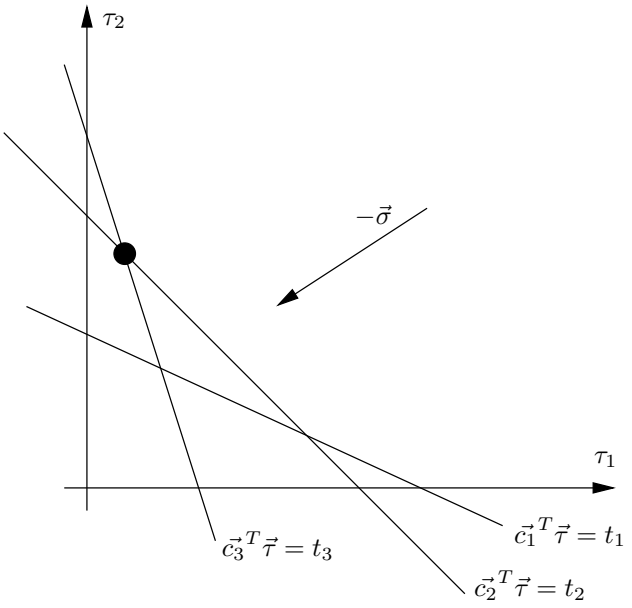

Figure 2. Illustration of max regression.

calculation. Therefore, we propose a novel regression technique, which guarantees that observed execution times are never underestimated by the resulting model:

Definition 1: Max regression analysis is to solve the linear programming (LP) problem $\min \vec{\sigma}^T\vec{\tau}$ for $\vec{\tau}$, for some $\vec{\sigma} \geq \vec{0}$, subject to the systems of linear constraints $C\vec{\tau} \geq \vec{t}$.

Max regression is illustrated in Fig. 2: two dimensions (two IPET entities), with three observations $(\vec{c_i}, t_i)$, $i = 1, \ldots, 3$, and an objective function defined by $\vec{\sigma}$. The black dot indicates the optimal corner, which yields the identified values for τ_1 and τ_2.

By definition, models obtained by max regression analysis can never underestimate observed execution times. Of course, the resulting IPET model may still underestimate the actual WCET unless the observed program runs provide enough coverage: this is true for measurement-based WCET analyses in general. For soft real-time applications, this may still be sufficient given that the likelihood of overruns is small. Also, the coverage criteria in Section 4 can be used to ensure that the estimated WCET is safe for all runs, given that the IPET model is sufficiently exact.

If $C\vec{\tau} \geq \vec{t}$ contains a lower bound $\tau_i \geq e_i$ for each IPET entity i, then, since $\vec{\sigma} \geq \vec{0}$, the LP problem has a bounded solution. Such lower bounds always exist, and can be added to the constraints arising from the observations: for basic blocks a trivial lower bound is zero, but for CFG edges the lower bounds might be negative if pipeline overlaps are modeled.

In the max regression objective function, there is some freedom to choose $\vec{\sigma}$. Ultimately, it should be chosen to give a solution $\vec{\tau}$ which yields a minimal WCET estimate when solving (1). To find such a $\vec{\sigma}$ is nontrivial, and a topic of future research. A possible heuristic is to choose σ as a linear combination of rows in C, where the coefficients for the rows are weights for the corresponding observations (higher weight $\implies$ the resulting model yields a tighter estimate for the corresponding observed execution time). If all program runs are of equal importance, then $\vec{\sigma}$ can be chosen as the mean value $(\sum_{j=1}^{m} \vec{c_j})/m$, where $\vec{c_j}$ is row j of C.

The max regression LP problem is similar to the IPET ILP problem, but they are different: max regression finds minimal execution times for CFG entities given observed running times and execution counts, whereas IPET ILP maximizes a linear timing model, with fixed execution times for CFG entities, subject to linear constraints on execution counts. Note that max regression is formulated as an LP problem rather than ILP: this enables the use of much more efficient methods than for ILP such as the simplex method [21]. If an integral solution is desired (say, in number of cycles), then the elements in the LP solution can be rounded upwards to the next largest integer. This will yield an IPET model which is at least as safe as the original one.

Regression can also accomodate observations of partial program runs, where only parts of program runs are observed and measured. Execution counts for program parts not measured will be zero. In the extreme case, execution times of the program may be measured at basic block level: a single run of a basic block will yield a row in C which is a unit vector.

Max regression followed by IPET calculation yields a WCET analysis method which avoids static low-level analysis. The IPET calculation, however, may well be done with constraints on the execution counts which are derived by a static flow analysis of the program: this yields a hybrid analysis.

Model identification according to the above requires that the execution counts are recorded. This can be done non-intrusively by creating a version of the program with explicit execution counters, which first is run for some indata where the final values of the execution counters are saved, and then the original program is run with the same inputs and its execution time is measured [16]. Fig. 3 shows the original and instrumented versions of a simple loop program, where the first iteration of the loop is treated separately to increase the precision of the IPET model. (For simplicity, only basic blocks are instrumented with execution counts.) Fig. 4 gives pseudocode for a hybrid algorithm, which performs measurements of execution times and recordings of execution counts followed by max regression and final WCET estimation.

What is the complexity of the hybrid algorithm outlined in Fig. 4? Assume an IPET model with n entities and m test runs for different indata. Disregarding the testing phase, the cost comes from solving the LP problem for

Original program, with basic blocks (1), (2), (3)

```
        i = 0;                  (1)
L:      if i > 10 goto exit;    (2)
        i = i + 1;              (3)
        goto L;
exit:
```

Instrumented program, with counters c_1, c_{21}, c_{2*}, c_{31}, c_{3*}

```
        c1, c21, c2*, c31, c3*  =  0;
        c1  =  c1 + 1;
        i = 0;                  (1)
        first  =  true;
L:      if first then c21  =  c21 + 1
                  else c2*  =  c2* + 1;
        if i > 10 goto exit;    (2)
        if first then c31  =  c31 + 1
                  else c3*  =  c3* + 1;
        i = i + 1;              (3)
        first  =  false;
        goto L;
exit:
```

Figure 3. A simple program, and its instrumented version.

Inputs: Program P, instrumented program PI, set D of test input data, function f to compute objective function vector, IPET flow constraint system $A\vec{c} \leq \vec{b}$.
Functions used: *time* end-to-end execution time, *counts* final execution count vector, *max_reg* solution vector to max regression problem.

$WCET\_est(P, PI, D, f, A\vec{c} \leq \vec{b}) =$
local $C, \vec{t}, \vec{\tau}, \vec{\sigma}$
$i = 0$
foreach $d \in D$ *do*
  $t_i = time(P(d))$
  $\vec{c_i} = counts(PI(d))$ /* $\vec{c_i}$ = row i of C */
  $i = 1 + 1$
$\vec{\sigma} = f(C)$
$\vec{\tau} = max\_reg(\vec{\sigma}, C, \vec{t})$
$WCET = \max \vec{c}^T \vec{\tau}$ subject to $A\vec{c} \leq \vec{b}$
return $WCET$

Figure 4. A hybrid algorithm for WCET analysis.

the max regression and the ILP problem for the WCET calculation. ILP is NP-complete, and thus exponential in n (unless $P = NP$). However, WCET calculation by ILP is used by most WCET analysis tools today, and so these tools have the same worst-case complexity. As for the LP solution of the max regression problem, the common simplex method is worst-case exponential but typically much faster, with an empirically expected complexity of $O(m^2 n)$ [21]. There are also interior-point methods for LP with polynomial worst-case complexity [21].

4. Coverage Criteria for Exact Models

In this section we will prove a result regarding testing coverage for WCET analysis. This result holds when the IPET model contains enough detail so it can be made *exact*: then (2) holds, and since this is a linear equation system we can use some basic theory from linear algebra. The result builds on the following: for any program, the possible runs will yield a set of execution count vectors for the chosen IPET model. No matter how many possible runs there are there will only be at most $r \leq n$ linearly independent such vectors, where n is the number of IPET entities. Due to program flow constraints, we may have $r < n$. Now, if the test runs produce r linearly independent execution count vectors, then a vector $\vec{\tau}$ of IPET entity execution times can be solved from the equation system formed from the results of the test runs: this vector will provide an exact IPET timing model. (If $r < n$ the equation system will have many solutions, but for any two solutions $\vec{\tau}$, $\vec{\tau'}$ will hold that $\vec{c}^T\vec{\tau} = \vec{c}^T\vec{\tau'}$ for any execution count vector $\vec{c}$ that can arise from a possible program flow.)

If the possible program flows can be exactly characterized by a linear constraint system, then r can be found from that system. If the constraint system overapproximates the possible program flows, it can still be used to find an upper bound to r. This gives a sufficient coverage criterion: if the number of linearly independent observed execution count vectors equals the value of r estimated from the linear constraint system, then any solution $\vec{\tau}$ yields an exact IPET model which can be used to produce a safe WCET estimate. Another consequence of the result is that it suffices to make r test runs to identify an exact IPET model, provided that they all produce linearly independent execution count vectors.

To show the result we first need to formalize what we mean by system, observations, and model. In the following, $\mathbb{N}$ denotes the set of natural numbers, $\mathbb{Z}$ the set of integers, $\mathbb{R}$ the set of real numbers, and $\mathbb{R}_+$ the set of positive real numbers.

Definition 2: An n-dimensional *IPET system* is a subset of $\mathbb{N}^n \times \mathbb{R}_+$. The IPET system S is *linear* if there exists a $\vec{\tau} \in \mathbb{R}^n$ such that $\vec{c}^T\vec{\tau} = t$ for all $(\vec{c}, t) \in S$. A *set of observations* of the IPET system S is a finite subset of S.

For an IPET system S, we define $C(S) = \{\, \vec{c} \mid (\vec{c}, t) \in S \,\}$, and $t(S) = \{\, t \mid (\vec{c}, t) \in S \,\}$. $E(S) \subseteq S$ is the set of *end-to-end runs* of S. For any $O \subseteq S$ we demand that $E(O) = O \cap E(S)$.

An IPET system is a set of observations as defined in Section 3. It can be seen as an abstract performance model where only the relation between execution counts and running times is kept. For a given program running on a certain hardware, the IPET system is formally obtained by gathering all possible observations, for all possible inputs. It is then assumed that we know from the program which runs are end-to-end runs: they are the interesting ones as regards WCET, and we define $WCET(S) = \max_{t \in T(E(S))} t$. However, we also allow IPET systems S (and sets of observations) to contain *partial runs* $\in S \setminus E(S)$. This is since we want to retain the possibility to incorporate fine-grained

measurements, on parts of programs, in our model.

Definition 3: An n-dimensional *IPET model* is a triple $M = (\vec{\tau}, A, \vec{b})$ where $\vec{\tau} \in \mathbb{R}^n$, A is an $m \times n$ integer matrix, and $\vec{b} \in \mathbb{Z}^n$. Defining $C(M) = \{ \vec{c} \mid A\vec{c} \leq \vec{b} \}$, we say that M is *safe* with respect to the IPET system S if there exists a $\vec{c} \in C(M)$ such that $\vec{c}^T\vec{\tau} \geq WCET(S)$, and it is *tight* with respect to S if in addition $\max_{\vec{c} \in C(M)} \vec{c}^T\vec{\tau} = WCET(S)$. M is a *safe flow model* for S if $C(M) \supseteq C(E(S))$, it is an *exact flow model* for S if $C(M) = C(E(S))$, and it is an *exact timing model* for S if $\vec{c}^T\vec{\tau} = t$ for all $(\vec{c}, t) \in S$.

Thus, an IPET system is linear if and only if it has an exact timing model, and a model which is both an exact timing and flow model is always tight (but tight models need not be exact). Also, a model which is both an exact timing model and a safe flow model is always safe. In practice, IPET models which are exact timing models will be prohibitively big except for programs running on simple hardware architectures, but the concept allows us to develop some theory that can serve as a starting point for a more general theory of WCET testing coverage, which includes also non-exact timing models.

We now proceed to show some sufficient conditions for when a vector $\vec{\tau}$, defining an exact timing model $M = (\vec{\tau}, A, \vec{b})$ for some IPET system S, can be found from a set of observations. This provides a result on coverage, telling how many program paths must be exercised to obtain an exact timing model. If M in addition is a safe flow model for S, then M is safe with respect to S and solving (1) will yield a safe WCET estimate.

For a set of vectors V, we define $rank(V)$ as the maximal number of linearly independent vectors in V, and $rank(A)$ as the maximal number of linearly independent rows for a matrix A. We will use the fact from linear algebra that if an overconstrained linear equation system $A\vec{x} = \vec{b}$ with $m > rank(A)$ equations indeed has solutions, then the set of solutions is the same as for any system formed by selecting $rank(A)$ equations from this system, with linearly independent matrix rows.

By abuse of notation, we write $C(S)$ also for the matrix formed by taking the vectors $\vec{c}$, for every $(\vec{c}, t) \in S$, and $\vec{t}(S)$ as the corresponding vector of times t. With this notation, $rank(C(S))$ retains its meaning also when $C(S)$ is seen as a matrix. In the following S denotes an n-dimensional linear IPET system, and O a set of observations of S.

Lemma 1: If $rank(C(O)) = rank(C(S))$, then the linear equation system $C(O)\vec{\tau} = \vec{t}(O)$ has solutions, and each solution is also a solution to $C(S)\vec{\tau} = \vec{t}(S)$.

Proof: Since S is linear, $C(S)\vec{\tau} = \vec{t}(S)$ has at least one solution according to Definition 2. Thus, $C(O)\vec{\tau} = \vec{t}(O)$ must have at least that solution, since this system of equations is a subset of $C(S)\vec{\tau} = \vec{t}(S)$ (and thus constrains the solutions less). Since $C(S)\vec{\tau} = \vec{t}(S)$ has solutions, then any solution to a system formed by selecting $rank(C(S))$ equations from this system, with linearly independent matrix rows, is a solution to the whole system. Since $rank(C(O)) = rank(C(S))$ we can pick any $rank(C(S))$ such equations from $C(O)\vec{\tau} = \vec{t}(O)$: any solution to these equations will then be a solution to both $C(O)\vec{\tau} = \vec{t}(O)$ and $C(S)\vec{\tau} = \vec{t}(S)$, and all solutions to $C(O)\vec{\tau} = \vec{t}(O)$ can be found in this way. □

Thus, if Lemma 1 holds, any solution to $C(O)\vec{\tau} = \vec{t}(O)$ will always form an exact IPET timing model together with any constraint system $A\vec{c} \leq \vec{b}$.

Corollary 1: If $rank(C(O)) = rank(C(S))$, if $\vec{\tau}$ is a solution to $C(O)\vec{\tau} = \vec{t}(O)$, and if $(\vec{\tau}, A, \vec{b})$ is a safe flow model for S, then $(\vec{\tau}, A, \vec{b})$ is safe with respect to S.

Corollary 1 provides the basic coverage criterion. When the conditions hold, solving (1) will yield a safe WCET estimate. We also make the following simple observation on the number of needed observations:

Corollary 2: There exists a set of $rank(C(S))$ observations O such that the conditions in Corollary 1 hold for it.

We now proceed to show some variations of Lemma 1, which yield coverage criteria that are applicable in different situations.

Proposition 1: If $rank(C(O)) = n$, then $C(O)\vec{\tau} = \vec{t}(O)$ has a unique solution which is also a solution to $C(S)\vec{\tau} = \vec{t}(S)$.

Proof: Since $n = rank(C(O)) \leq rank(C(S)) \leq n$ we have $rank(C(S)) = n$. Existence of solution, and equality of set of solutions, then follows by Lemma 1. Uniqueness follows since $rank(C(O)) = n$: according to the proof of Lemma 1 we can then pick n equations from $C(O)\vec{\tau} = \vec{t}(O)$ to form an equation system with a non-singular $n \times n$ system matrix, whose set of solutions is the same, and by basic linear algebra this system has a unique solution. □

Proposition 1 is applicable mostly to IPET systems which contain partial runs, where the timing of small parts of the program can be observed. If S contains only longer runs stretching several basic blocks, then most likely $rank(C(S)) < n$ due to the flow constraints that the underlying program must obey. See further below.

If $rank(C(O)) < n$, then it can be hard to know whether $rank(C(O)) = rank(C(S))$ since we typically know only O, not S. If indeed $rank(C(O)) < rank(C(S))$, then there is a risk that execution times are underestimated. Consider, for instance, a set of observations where $c_1 = c_2$ for all observations. There are then many possible solutions, all satisfying $\tau_1 + \tau_2 = t$ for some t. For instance, we may have $\tau_1 = 0$, $\tau_2 = t$. If indeed $c_1 = c_2$ for all runs in S, then any selection of τ_1, τ_2 according to this will give the same running times in the IPET model. But if $rank(C(O)) < rank(C(S))$, then there might be unobserved runs in S where c_1 is large and c_2 small: see Fig. 5, where $c_1 = c_2$ for all runs where the condition b remains false, but $c_1 > c_2$ if b sometimes becomes true. If the "real" τ_1 is large, then a WCET estimate based on the τ_1 and τ_2 above may be a big underestimation of the

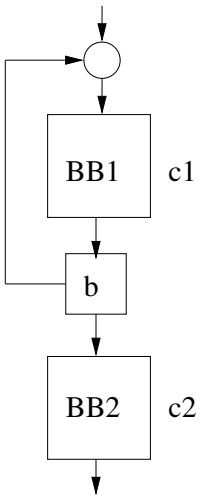

Figure 5. CFG illustrating the coverage problem.

real WCET. Therefore, it is important to know whether $rank(C(O)) = rank(C(S))$ or not. If we have an IPET model which is a safe flow model, then the proposition below can be used:

Proposition 2: If M is a safe flow model of S, and if $rank(C(M)) \leq rank(E(O))$, then $C(E(O))\vec{\tau} = \vec{t}(E(O))$ has solutions, and each solution is also a solution to $C(E(S))\vec{\tau} = \vec{t}(E(S))$.

Proof: Since M is a safe flow model of S we have $rank(C(E(S))) \leq rank(C(M))$. Thus, $rank(C(M)) \leq rank(C(E(O))) \leq rank(C(E(S))) \leq rank(C(M))$, which implies $rank(C(E(O))) = rank(C(E(S)))$. Existence of solution, and equality of set of solutions, then follows by Lemma 1. □

How to decide $rank(C(\vec{\tau}, A, \vec{b}))$? The linear constraint system $A\vec{c} \leq \vec{b}$ will in general contain a number of linear *equations* in additions to the inequalities. Such equations can arise from *structural constraints*, which are imposed by the CFG structure, or *semantic constraints*, which depend on the semantics of the CFG nodes.

Let $in(i)$ and $out(i)$ be the sets of incoming and outgoing edges of node i in the CFG. The structural constraints are then [8]

$$\begin{aligned} c_i &= \sum_{j \in in(i)} c_j = \sum_{j \in out(i)} c_j, \quad \text{internal nodes } i \\ c_x &= 1, \quad x \in \{s, t\} \text{ (enter/exit node)} \end{aligned}$$

Semantic equation constraints can for instance be:

- *exact loop bounds*, $c_i = k$ where k is a constant,
- *mutually exclusive infeasible paths*, $\sum_{i \in N'} c_i = k$, where N' is a set of CFG nodes, and
- *correlated execution*, $c_i = k \cdot c_j$.

The equations present in the linear constraint system $A\vec{c} \leq \vec{b}$ can be written

$$A^{=}\vec{c} = \vec{k} \tag{3}$$

Every linearly independent equation can be used to eliminate an execution count variable, thus reducing the maximal rank of the system. We obtain the following corollary to Proposition 2:

Corollary 3: If M is a safe flow model of S, and if $n - rank(A^{=}) \leq rank(E(O))$, then $C(E(O))\vec{\tau} = \vec{t}(E(O))$ has solutions, and each solution is also a solution to $C(E(S))\vec{\tau} = \vec{t}(E(S))$.

Corollary 3 yields the following test: if, for a set of observations, the rank of the observed execution count matrix $E(O)$ is at least as big as the number of entities in the IPET model minus the rank of the equational constraint matrix $A^{=}$, then the solutions to the set of observations is the same as for the observed system.

5. Evaluation

We have used five programs from the Mälardalen WCET Benchmark suite [22] to evaluate the hybrid method outlined in Fig. 4, which uses max regression to identify an IPET model followed by an IPET calculation using statically derived program flow constraints. The programs were selected according to the following criteria: (1) they should not be single-path programs, since that would yield a trivial model identification (only a single linearly independent parameter), (2) for the sake of comparison, the worst execution path leading to the real WCET should be known, and (3) tight program flow constraints should also be known. (2) is needed to find the real overestimation of the WCET, and (3) is desirable since we wanted to uncover the possible misprediction stemming from the model identification, not from having imprecise program flow constraints.

Each of the four first benchmarks was run with 100 different sets of random input test data (arrays of length 100[1], filled with random data). The random data was generated by a separate program that wrote the corresponding array declarations in the source code files for the benchmarks. The fifth benchmark, which computes an upper triangular matrix by Gaussian elimination with full pivoting, was run on randomly selected 10×10 0/1-matrices with uniformly distributed data. To obtain values for the execution counters, the source codes were instrumented with the counters, compiled with gcc 4.1.2, and run with the respective test input sets. To obtain execution times, we used the SimpleScalar toolset [23]. The code was compiled for the MIPS-like Portable ISA, using `sslittle-na-sstrix-gcc` with optimizations disabled. For each set of input data the code was emulated using the simulator `sim-outorder`, with flags `-bpred nottaken`, and the simulated end-to-end execution time was recorded.

1. For `bs` we used arrays with 1000 elements.

With this flag setting, `sim-outorder` simulates a processor with out-of-order issues of instructions, main memory latency 18 cycles for first access and 2 cycles for next accesses, 8KB L1 data and instruction caches, respectively (1 cycle), 256KB L2 cache (6 cycles), all caches LRU, TLB miss penalty 30 cycles (64 entries for instruction and 128 entries for data), 4 integer ALU's, fetch width 4 instructions, and dispatch queue 16 instructions. Branch prediction is always-not-taken. Our benchmarks only use integers.

We then identified an IPET model for each benchmark with max regression, using the mean value of the execution count vectors as objective function. Finally, a WCET estimate was calculated by solving the IPET ILP problem with a tight set of linear flow constraints. For both calculations, we used `lp_solve` [20].

For each benchmark, we also measured the real WCET by running it with `sim-outorder` using a (known) input data set provoking the worst-case execution path.

The compiler adds some startup code. For this code we had no source code that we could instrument with counters. Rather than instrumenting the binary, we treated the startup code in two different ways. First, we estimated the execution times for the benchmark code, without the startup code, as follows. For each program, we compiled a version with the call to the benchmark function commented out, but global variables, function declarations etc. being left in the code. Then, we ran this version with the same input sets as the full program, and subtracted the obtained execution times from the corresponding times for the full program. Second, we estimated the WCET for the full program using an IPET model for the full program, including startup code, by adding a single counter for the startup code (thus treating it as a single basic block).

All analyses were done on a PC with an Intel Centrino Duo 1.2GHz CPU and 1 GB memory, running Debian 4 Linux. For all benchmarks, the analysis time (calculating objective function, and solving the LP and ILP problems) took less than three seconds.

We used IPET models with one counter per basic block, except for loops. For basic blocks within a loop, the models have two counters: one counting each first entry to the basic block, and one counting subsequent executions of the block in a loop. The models have no counters for the edges in the CFG:s. This means that the models can take loop startup effects into account, but they do not explicitly model pipeline overlaps. However, the model identification should to some degree be able to compensate for this through assigning the basic blocks somewhat shorter execution times. These simple models can surely not be made exact for the simulated processor: thus, the coverage criterion in Section 4 is not applicable and the method could possibly underestimate the WCET.

Table 1 lists the benchmark programs, and gives some characteristics: lines of C code excluding comments and declarations (#LC), and the size of the IPET model in # of counters (#IPET). In Table 2 we give the results: WCET for program without startup code (WCET), corresponding WCET estimate from the model (eWCET), overestimation (%), and ditto for the programs including startup code (WCET+s, eWCET+s, %).

Table 1. Benchmark programs, and IPET model size

Program	Description	#LC	#IPET
`bsort100`	Bubblesort	18	11
`bs`	Binary search	16	7
`insertsort`	Insertion sort	12	5
`select`	Select kth largest number	39	17
`gauss`	Gaussian elimination	40	11

Table 2. Results

Program	WCET	eWCET	%	WCET+s	eWCET+s	%
`bsort100`	4993014	5040890	0.96	5101354	5147990	0.91
`bs`	5416	5449	0.71	113756	113789	0.02
`insertsort`	4395811	4567200	3.90	4504183	4675570	3.80
`select`	278977	307204	10.12	387349	415576	7.29
`gauss`	682699	698457	2.31	790979	806737	1.99

For no benchmark did the method underestimate the WCET. The overestimation lies in the range 0.47-10.12%. Taking into account that the model is quite crude for an out-of-order CPU with caches, we think that some of the figures are surprisingly good. For `select` we are quite certain that we have not found a tight set of flow constraints, since the control structure of this program is quite tricky resulting in complex infeasible path constraints. For the other benchmarks, the maximal overestimation is 3.90%.

For all benchmarks the model identification does a very good job of identifying the execution time for the unknown startup block, despite that it is modeled with only a single execution counter. For fixed input (array) size this code is single-path, executing the same sequence of instructions every time. This explains why a single counter (and single execution time) can provide a good model for this code.

Fig. 6 shows how the WCET estimate for Gaussian elimination varies over the first 100 runs (full line). As seen,

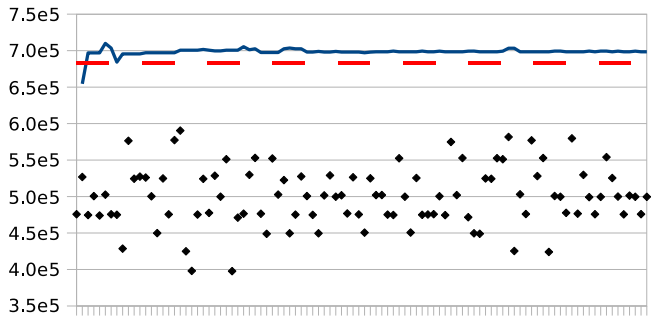

Figure 6. WCET estimation for Gaussian elimination.

it converges quickly towards an estimate that slightly overestimates the real WCET (dashed line). It is also interesting to compare with the actual end-to-end running times for the runs (dots): They are nowhere close to the WCET. It is evident that the method produces a reasonable WCET estimate with much fewer runs than if estimating from just by high-water-marking the end-to-end running times, which is still current industrial practice. The other benchmarks show a similar behavior.

For `select`, the estimated WCET (without startup code) oscillates between the two values 307204 and 338825, and similarly with startup code, as the number of input test data sets increases. This is due to that the objective function for the max regression, which is formed by averaging over the observed execution count vectors, varies with different sets of test inputs in such a way that the optimal corner for the LP problem changes. Since the coordinates of this corner provide the objective function for the IPET ILP maximization problem, this will affect the resulting WCET estimate.

6. Conclusions and Future Work

We have presented a systematic approach to measurement-based WCET analysis. This approach is based on model identification, and it uses the fact that IPET models are linear. We defined *max regression*, which is based on linear programming rather than the least-squares method and has the property that the model will not underestimate any observed execution times. We then outlined a hybrid method for WCET analysis, which uses max regression to identify the parameters in an IPET model and then calculates a WCET estimate by solving the IPET ILP problem. We proved a simple, sufficient criterion for coverage which ensures that the identified IPET model is an exact timing model when the underlying "IPET system" is linear (i.e., the timing behaviour of the real underlying system can be exactly described by an IPET model with the selected counters). Finally, we evaluated the precision of the method by identifying a simple IPET model for some benchmark codes and computing WCET estimates using the outlined hybrid method. The precision ranged from excellent to fair.

An obvious topic for further research is to try out the method on a larger set of benchmarks. Another issue is how to set up IPET models for different classes of architectures. How detailed do they need to be to obtain good enough WCET estimates in the end? Also, are there systematic ways to find an objective function for the max regression LP problem that in the end yields a WCET estimate that is as tight as possible?

The issue of coverage is crucial. We proved some results for linear IPET systems, but it is not certain that other than quite simple processor architectures can have reasonably sized linear IPET systems. Can we, for instance, find coverage criteria that guarantee the absence of underestimations also for nonlinear IPET systems (for which there are no exact linear models using the same execution counters)? Any such criteria will likely be architecture-dependent.

The method allows to include timing bounds from a static low-level analysis by adding constraints from such an analysis to the max regression problem. For instance, a static low-level analysis might find both lower and upper bounds on the execution times of basic blocks, which then can be added as constraints to yield more exact estimates of the execution time vector $\vec{\tau}$ of IPET entities. This approach to hybrid analysis remains to be explored.

Finally, an interesting observation is that for our benchmarks, modeling the startup code as a single basic block gave very good results. Although this code clearly was particularly suitable for this simple model, this hints at the possibility to identify simple linear models for unknown code, like library code, which is used by the analyzed program. Such code often presents a problem for WCET analysis, and it is not always so easy to instrument. But by no means are we restricted to use execution counters in the linear models: we could select any other numerical parameters, like numerical arguments to library functions, or argument sizes, as long as the calling user code can be instrumented to record them. In this way we can treat the unknown code as a black box. The resulting models may be less precise, but can still be helpful in soft real-time contexts where ease of modeling is more important than precision and absolute safety of the WCET estimate.

Acknowledgment

This research was supported by the KK-foundation through grant 2005/0271, the ALL-TIMES FP7 project, grant agreement no. 215068, and the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS.

References

[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution time problem — overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 1–53, 2008.

[2] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper, "Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution," in *Proc. 27^{th} IEEE Real-Time Systems Symposium (RTSS'06)*, Dec. 2006.

[3] J. Engblom, "Processor pipelines and static worst-case execution time analysis," Ph.D. dissertation, Uppsala University, Dept. of Information Technology, Uppsala, Sweden, Apr. 2002, ISBN 91-554-5228-0.

[4] C. Ferdinand and R. Wilhelm, "Efficient and precise cache behavior prediction for real-time systems," *Real-Time Systems*, vol. 17, pp. 131–181, 1999.

[5] S. Thesing, "Safe and precise wcet determination by abstract interpretation of pipeline models," Ph.D. dissertation, Saarland University, 2004.

[6] F. Bodin and I. Puaut, "A WCET-oriented static branch prediction scheme for real time systems," in *Proc. 17^{th} Euromicro Conference of Real-Time Systems, (ECRTS'05)*, Jul. 2005, pp. 33–40.

[7] Y.-T. S. Li and S. Malik, "Performance analysis of embedded software using implicit path enumeration," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 12, pp. 1477–1487, Dec. 1997.

[8] P. P. Puschner and A. V. Schedl, "Computing maximum task execution times – a graph-based approach," *Journal of Real-Time Systems*, vol. 13, no. 1, pp. 67–91, Jul. 1997.

[9] G. Bernat, A. Colin, and S. M. Petters, "WCET analysis of probabilistic hard real-time systems," in *Proc. 23^{rd} IEEE Real-Time Systems Symposium (RTSS'02)*, Austin, TX, Dec. 2002.

[10] R. Kirner, I. Wenzel, B. Rieder, and P. Puschner, "Using measurements as a complement to static worst-case execution time analysis," in *Intelligent Systems at the Service of Mankind*. UBooks Verlag, Dec. 2005, vol. 2.

[11] J. Wegener and M. Grochtmann, "Verifying timing constraints of real-time systems by means of evolutionary testing," *Real-Time Systems*, vol. 15, no. 3, pp. 275–298, Nov. 1998.

[12] P. Puschner, "The single-path approach towards WCET-analysable software," in *Proc. IEEE International Conference on Industrial Technology*, Dec. 2003, pp. 699–704.

[13] A. Colin and S. Petters, "Experimental evaluation of code properties for WCET analysis," in *Proc. 24^{th} IEEE Real-Time Systems Symposium (RTSS'03)*, 2003.

[14] A. Betts and G. Bernat, "Tree-based WCET analysis on instrumentation point graphs," in *Proc. Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006)*. IEEE Computer Society, 2006, pp. 558–565. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ISORC.2006.75

[15] I. Wenzel, B. Rieder, R. Kirner, and P. P. Puschner, "Automatic timing model generation by CFG partitioning and model checking," in *Proc. Design, Automation and Test in Europe (DATE'05)*, vol. 1, Mar. 2005, pp. 606–611. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/DATE.2005.76

[16] M. Lindgren, H. Hansson, and H. Thane, "Using Measurements to derive the Worst-case Execution Time," in *Proc. 7^{th} International Conference on Real-Time Computing Systems and Applications (RTCSA'00)*. Cheju Island, South Korea: IEEE Computer Society, Dec. 2000, pp. 15–22.

[17] B. Franke, "Fast cycle-approximate instruction set simulation," in *Proc. 11^{th} International Workshop on Software and Compilers for Embedded Systems (SCOPES 2008)*, H. Falk, Ed., Munich, Mar. 2008, pp. 69–78.

[18] J. Yan and J. Zhang, "An efficient method to generate feasible paths for basis path testing," *Information Processing Letters*, vol. 107, no. 3–4, pp. 87–92, 2008.

[19] Å. Björck, *Numerical methods for least squares problems*. Philadelphia: SIAM, 1996, iSBN 0-89871-360-9.

[20] M. Berkelaar, *lp_solve: (Mixed Integer) Linear Programming Problem Solver*, 2004, `ftp://ftp.es.ele.tue.nl/pub/lp_solve`.

[21] Y. Shi, S. Yong, Y. Peng, and P. Yi, *Multiple criteria and multiple constraint levels linear programming: concepts, techniques and applications*. World Scientific, 2001, iSBN 9810237383.

[22] Mälardalen University, "WCET project homepage," 2008, `www.mrtc.mdh.se/projects/wcet`.

[23] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *SIGARCH Comput. Archit. News*, vol. 25, no. 3, 1997.