WCET Analysis of Probabilistic Hard Real-Time Systems *

Guillem Bernat Antoine Colin Stefan M. Petters
Real-Time Systems Research Group
Department of Computer Science
University of York, UK
{bernat,acolin,petters}@cs.york.ac.uk

Abstract

Traditional approaches for worst case execution time (WCET) analysis produce values which are very pessimistic if applied to modern processors. In addition, end to end measurements as used in industry produce estimates of the execution time that potentially underestimate the real worst case execution time. We introduce the notion of probabilistic hard real-time system as a system which has to meet all the deadlines but for which a (high) probabilistic guarantee suffices. We combine both measurement and analytical approaches into a model for computing probabilistically bounds on the execution time of the worst case path of sections of code. The idea of the technique presented is based on combining (probabilistically) the worst case effects seen in individual blocks to build the execution time model of the worst case path of the program (such case may have not been observed in the measurements). We provide three alternative operators for the combination based on whether the information of their dependency is known. Experimental evaluation of a two case study shows extremely low probabilities of the values obtained by traditional analysis.
Keywords: *probabilistic analysis, hard real-time, worst case execution time, execution profiles*

1 Introduction

The use of embedded programmable units in everyday life is constantly increasing. An obvious example for this are modern cars. Wiring harnesses are replaced by bus systems, switches by smart switches and engine controllers by powerful CPUs. This allows for easy integration of additional sensors and a more effective fault analysis in case of a malfunction. The amount of electronics (and therefore software) is expected to reach 25% of the total cost of production of mid sized cars by 2005 (cf. [3]). Since the units in a car have to work under extreme conditions (as regards vibration and temperature) with virtually no preventive maintenance, the requirements on dependability are raised considerably. This holds true not only for the hardware, but also the increasing software share in such systems. Another constraint of industry is the price. Since the number of cars of a certain make is usually very large, saving a couple of pennies by using a slightly less powerful processor or concentrating several tasks in one processor instead of using a small 8 bit processor for each is very desirable. This leads to a major tradeoff between reduced cost per delivered piece and the risk of having to recall delivered products, the loss of customer satisfaction or even the liability for a resulting accident. As these systems are also real-time systems, the correctness of software in such systems relies not only on functional correctness but also on the timely delivery of the computed results. The timing analysis (by schedulability analysis) as part of a certification procedure relies on adequate knowledge of the *worst case execution time (WCET).*

A major problem with modern and most probably future processors as regards WCET estimation, is that the execution time of instructions is no longer constant. The sources of the execution time deviation can be classified in two categories: data dependent or history dependent.

Classical examples for a data dependent execution time are multiplication and division. Depending on the implementation in the hardware architecture, the execution of such an instruction takes a fixed or data dependent time for completion. History dependent execution times are produced for example by caches, pipelines and branch prediction algorithms. Often the effects cannot be exactly distinguished. Typical sources of such a blending effect are out-of-order execution of processors which are clocked internally higher than the peripheral units. In the latter case a tiny deviation in the code determines whether a load instruction is executed on one or the other external cycle.

While the effects of these sources of variable execution time can be observed in the real execution of a program, an exact prediction is only partially possible. Additionally the

*The work presented in this paper is supported by the *European Union* under Grant Next TTA "IST-2001-32111" and the French Department of Defence.

processor description published by the processor vendors is often inaccurate or coarse to hide the intellectual property.

Current techniques for WCET analysis aim at finding the absolute upper bound on the execution time. For modern high performance processors with, for example, out-of-order execution, these technique may produce estimates for the WCET which are pessimistic due to the simplifications that may need to be made and due to the inherent variability of the execution time. As pure end-to-end measurements of program execution times under a limited amount of test cases are risky as regards the reliability of having observed the worst case, this method is only partially trusted. In industry most engineers work with safety margins to cope with the problem of uncertainty that the worst case is covered by their experiments. However, the safety margins are not the result of analytical reasoning, but of experiences in similar type of applications. In contrast to this, the conservative and analytical correct reasoning to bound the WCET is usually extremely pessimistic.

To cope with the variable execution times of instructions and the only coarse knowledge of its sources while avoiding excessive overestimation, we base our analysis on the notion of a *grey box* approach i.e.; the effects are understood in principle, but can not be captured analytically without very pessimistic simplifications and a cycle true simulation of all possible program paths with all potential input data combinations is inhibited by the complexity of the problem. We address the complexity issue by describing smaller units of the program in statistical terms utilising *execution profiles (EP)* (cf. § 3) and reason on the WCET of the whole program by combining probabilistically the worst case effects of the individual units. Thus effectively providing an estimate of a combination of effects possibly not seen in the end-to-end measurements providing the profiles for the individual units.

The concept of execution profile is not only applicable for describing the worst case behaviour for analysing tasks running in isolation, but also, for example, for the description of the interaction between concurrently running tasks in a preemptive system. However, in this paper we limit ourselves on the application for the WCET estimation and investigate the further applicability in future work (cf. § 9).

This paper presents the following contributions:

Probabilistic hard real-time systems We introduce the concept of *probabilistic real-time systems* for systems where all deadlines must be met for which a probabilistic argument of how likely a deadline missed is acceptable. Typical target probabilities are 10^{-6}, 10^{-12} etc.

Concept of execution profiles as a mechanism to capture the variability of the execution time of paths of a section of code. § 3 provides a detailed mathematical analysis of the properties of EPs.

Combination of profiles The combination of EPs of individual blocks to the EP of a path utilising a probabilistic timing schema is described in § 4.

Model of dependencies We distinguish three cases: Either two EPs are independent (cf. § 4), or the dependencies are known (cf. § 5) or no dependency information is known (cf. § 6). For each of these cases a separate operation to combine the EPs are provided that enable the production of an integrated timing schema.

Experimental evaluation We have implemented a prototyping tool providing the necessary operations. In § 8 two case studies show the deployment of the method and the effects of the different proposed mechanisms.

2 Related Work

Beside path based and implicit path enumeration based approaches, the use of timing schemas are a main theme in the area of WCET estimation research. As our method can be used as an extension of the timing schema based approach (cf. [12]), we provide a short introduction into this method.

A simple timing schema is based on a syntax tree representation of the code. For each node of the tree, it computes $W(\mathsf{X})$ an integer that represents the worst case execution time of X as a function of the execution time of its parts. The leaves of the syntax tree correspond to basic blocks. The execution time of these basic blocks is obtained in several ways for instance by counting the cycles in each block. The basic timing schema is therefore given by:

- $W(\mathsf{X}) =$ integer, when X is a basic block.
- $W(\mathsf{X;Y}) = W(\mathsf{X}) + W(\mathsf{Y})$
- $W(\textsf{if Z then X else Y}) =$
  $$W(\mathsf{Z}) + \max\{W(\mathsf{X}), W(\mathsf{Y})\}$$
- $W(\textsf{for Z loop X}) = (n+1)W(\mathsf{Z}) + nW(\mathsf{X})$

Where we assume that the loop iterates at most n times. By recursively applying these rules an estimate of the WCET of an arbitrary section of code can be obtained. There are other more sophisticated schemas for capturing peculiar features of the programs or hardware architecture (see for instance [6]). However, to illustrate the concepts this simple schema suffices.

In the field of statistical real-time analysis we have to distinguish between statistical methods to analyse the interactions of tasks in a system and those deployed to provide the WCET. There has been some work on the area of probabilistic methods for schedulability analysis, Gardener and Liu focus in [11] on the schedulability analysis of soft real-time systems. The target is to provide a statistical measure of the amount of missed deadlines to be expected. The work by [9] extends these results further to compute a profile of the response times of tasks.

The work from Burns and Edgar in [5] applies extreme value statistical analysis on end to end measurements of a task to reason about the probability of WCET being greater than the largest execution time observed during any of the tests of the program. Our approach uses a different strategy, we also obtain data from measurement but we analyse small sections of code and provide a mechanism to combine together the worst cases of what has been observed for each of the units.

3 Execution Profiles

An execution profile (or *EP*) associated with a piece of code is a representation of the relative frequencies with which some particular events happen. These events can be, for example, reaching a number of instruction cache misses during the execution of the piece of code. In this case the *cache miss EP* represents the relative frequencies of obtaining n cache misses for one execution this would allow for a worst case cache miss analysis. Another possible "event" is executing the piece of code within n cycles, which leads to the EP of execution time.

In this paper, we focus on a particular use of EPs in the domain of WCET analysis. The "events" whose frequencies are represented by the execution profiles are the different execution times that a piece of code may require to execute. Such an execution profiles representing the relative frequencies of execution times is an *execution time profile* (ETP for short).

3.1 Obtaining execution profiles

There are several possible means to obtain the execution profile of a piece of code.

- Measuring real executions using some probing system on a real processor.
- Using a processor simulator which will execute the piece of code and provide all the required information.
- Or using some analytical method.

Using the real processor to obtain execution profiles ensures that the measurements will capture all the effects of the hardware. The drawback is that depending on the chosen architecture, it may be difficult, if not impossible, to obtain accurate measurements. It may also be necessary to add some probing code into the program which will affect the measurements (cf. [13]). Another encountered problem is the variable execution time of some instructions depending on the manipulated data (e.g. floating point multiplication).

On a simulator the probing problem does not exist as it provides an extensive execution trace. The drawback of this method is the possible difference between the processor behaviour and its simulated one. This may be due to unsimulated features or errors in the processor documentation used to build the simulator. Some work has been done on validating processor simulators against real hardware [8, 10].

The EPs could also be provided by analytical methods as the ones used in static WCET analysis. The methods exposed in [2, 7], which respectively provide potentially different worst case number of cache misses and WCET for the different loops in which a piece of code is included, could be adapted to provide EPs. As these methods require an accurate description of the hardware, they lead to the same issues as the use of simulators.

3.2 Granularity

Up to this point the units of the program described have not been defined. In general any unit whose execution profile can be provided with sufficient confidence is a suitable unit. While this condition is usually not met for end to end measurements and simulations, describing individual assembler instructions with execution profiles lead to unnecessary pessimism in the general case. So, basic blocks have been chosen as the building blocks and the terms "unit" and "basic block" are used interchangeably. However, still larger units may be possible, depending on the combination of hardware and software to be analysed.

Finally a definition of the execution profile as regards to start and end points is needed. the Standard WCET approaches define the execution time of a basic block from the point the first instruction of this block enters the pipeline until the last instruction of the block leaves the pipeline. This approach introduces either additional pessimism or additional complexity as it needs a model to handle the overlay due to pipelines. Instead we propose to pick a certain stage of a pipeline and assign the execution profile for the time needed between the first instruction of a unit entering this stage of the pipeline (e.g. the execution stage) until the first instruction of the next unit enters this stage of the pipeline. All effects of overlapping pipeline stages are therefore covered inside the two execution profiles describing the consecutive units.

4 WCET Analysis of Independent Execution Time Profiles

In § 3 we have introduced informally the concept of execution time profiles. We now present a more formal definition together with the analysis of their properties and operations. The objective of § 4, § 5 and § 6 is to provide an algebra of execution time profiles that allows to derive a probabilistic timing schema.

This section concentrates on the model that assumes that all ETPs are independent. For such systems the main operation to combine profiles is the convolution. Please note that this is not adequate if the execution time are not independent. In the case of dependent execution times we propose in § 5 the joint execution profile which captures such dependencies. We also provide and an equivalent operator for the convolution for such joint execution profiles. Finally, if neither independence can be proven, nor detailed dependence information is known, then a safe (pessimistic) operator for combining ETPs is provided in § 6. A final timing schema that wraps up the different calculation methods is then discussed.

4.1 Basic Definitions

The execution time, X, of a section of code X, is a discrete random variable [1]. It represents "the execution time of the paths in X". The question we want to ask about X is what is the probability of ever observing a run with execution time greater than a given time t. Moreover, we want to provide a mechanism to combine two or more of such random variables to produce an estimate of the longest execution time of sequences of sections of code.

Standard statistical techniques make a series of assumptions about the properties of such random variables. The main one is to assume that the random variables are *iid* (independent and identically distributed) if this is the case there is a massive body of work on properties and operations that can be performed. However, we argue that execution times of small sections of code are not iid. It is not independent because there may be a dependency between two consecutive observations of the same block (for instance, the difference between first and additional iterations of the loops may depend on whether memory references are on cache). Also, the random variable is not identically distributed as the distribution may change over time, the execution path depends on the state of the program, which changes over time therefore changing the execution time (consider for example two nested triangular loops, the execution time of the inner loop is a function of the induction variable of the outer loop, and therefore its execution time distribution varies over time). In addition, even if the random variable was identically distributed we don't know its distribution.

Our aim is to characterise the longest execution time of a program by combining together the observed execution time of its parts. This combination should be biased towards the worst case and even pessimistic. Standard statistical methods model the central part of the distribution and therefore are not suitable for this purpose. Extreme value statistics address this particular issue by modelling the tails of the distribution, however, the manipulation of such distributions is very complex and limited. For instance, in order to combine two random variables it is generally assumed that they are mutually independent. One of the main hypothesis of our work is that such assumption can not be made in the general case as the execution times may be dependent.

For all these reasons we have provided in the rest of the paper a mechanism to manipulate directly (and numerically) the probability mass distribution function $x(t) = P(X = t)$ of random variable X, which is what we have called an ETP[1]. An ETP is therefore a function with an integer domain and the operations on ETPs are operations on functions. We use the accumulated probability mass function $\bar{x}(t) = P(X \geq t)$. $x(t) = p$ means that the probability of X taking t time units is p, whereas $\bar{x}(t) = q$ means that the probability of X taking *at least* t units is q. Note that $x(t)$ may be defined for negative values of t, meaning that there is a non-zero probability of "gaining" some time.

We consider systems that have bounded execution times, therefore we can define the two extremes of x, $x^+ = \max\{t|x(t) > 0\}$, $x^- = \min\{t|x(t) > 0\}$.

Let k be a real number, and x, y two ETPs. Then a *scaling* of x by a factor k is given by $(k \cdot x)(t) = k \cdot x(t)$. We can also add and subtract ETPs which corresponds to addition and subtraction of functions. For instance, $(x + y)(t) = x(t) + y(t)$ and $(x - y)(t) = x(t) - y(t)$ have the usual meaning. Based on this for example, a linear combination of ETPs can be defined. One has to be careful when using these operations as they may result in an invalid profile (not a valid probability mass function, and therefore normalisation may be required). We define the weight of x as $|x| = \sum_{\forall t} x(t)$. If $|x| = 1$ we will say that the ETP is in *normal form*. The normalised ETP $||x||$ of x is therefore given by $||x|| = \frac{x}{|x|}$.

The p-point of an ETP is the smallest t such that $\bar{x}(t) \geq p$. This is denoted by $\pi(x, p)$. This is the rightmost point in time where the weight to the right of t is at least p. Given an ETP x the right cut-off of x at weight p, denoted by $\lambda_{<p>}(x)$ is a new ETP obtained from x by only selecting the rightmost points in x that add up to p. Formally:

$$\lambda_{<p>}(x)(t) = \begin{cases} x(t) & \text{if } t > \pi(x,p) \\ p - \bar{x}(t+1) & \text{if } t = \pi(x,p) \\ 0 & \text{otherwise.} \end{cases}$$

A shift of d time units on $x(t)$ corresponds to $x(t - d)$. The folded version of $x(t)$ from the origin is the ETP $x(-t)$.

4.2 Combining ETPs

The key component for deriving a timing schema is to provide a mechanism for combining ETP. We first analyse the problem from the perspective that no ETPs are dependent. The formulation of the problem is as follows: given two ETPs x, y that correspond to two pieces of code X and Y, we are interested in defining the random variable $Z = X + Y$ which represents the execution profile of the execution of X;Y;.

[1]We will use capital italics (X) to denote a random variable, sans serif capitals (X) for its associated code and lower case italics (x) for its ETP.

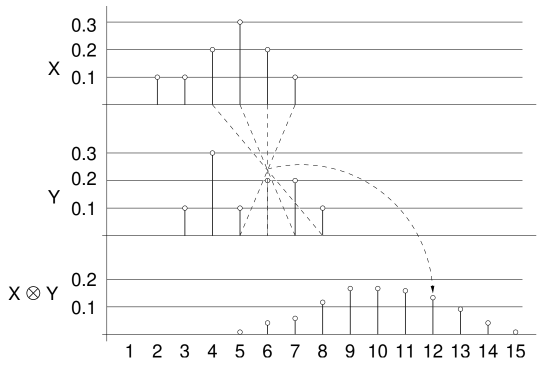


Figure 1. Convolution operation example.

4.2.1 Convolution

If X and Y are mutually *independent*, the execution profile of Z, z is the convolution ($z = x \otimes y$) of the probability mass functions of X and Y [1]. Where the discrete convolution is defined as:

$$(x \otimes y)(t) = \sum_{\forall s} x(s)y(t-s) \tag{1}$$

A graphical interpretation helps to understand the way the convolution is performed. Figure 1 shows the calculation of $(x \otimes y)(t)$. Take for example $(x \otimes y)(12)$. It is made up of the sum of all $x(s)y(t)$ such that $s + t = 12$. If x and y are independent then the probability of $x(s)$ occurring as well as $y(t)$ is equal to $x(s)y(t)$.

The convolution has some useful properties. It is (a) commutative, $x \otimes y = y \otimes x$, (b) associative $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ and (c) distributive $(x \otimes y) + z = (x \otimes z) + (y \otimes z)$. The same figure helps to visualise one very important property that relates to the extreme points. If $z = x \otimes y$, then $z^- = x^- + y^-$ and $z^+ = x^+ + y^+$. The smallest non-zero element of z is at index $x^- + y^-$ and the largest non-zero element is at index $x^+ + y^+$, the worst case of the convolved ETP corresponds to the addition of the worst cases of the individual ETP. Note that for practical implementations, then the summation in (1) ranges from $s \in [x^- + y^-, x^+ + y^+]$. Also note that if $|x| = |y| = 1$ then $|x \otimes y| = 1$.

There is one special profile of interest: the profile zero, denoted by δ given by:

$$\delta(t) = \begin{cases} 1 & \text{if } t = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

It has the property that for any profile x, $x \otimes \delta = x$. By definition, we denote $x^0 = \delta$.

Note that due to the multiplicative effect of the the convolution extremely low values of probabilities may be computed. Values in the order of 10^{-100} appear frequently. It is arguable whether these values really have a physical meaning, however low they are they should be at least marked as being different than zero as there is a conceptual distinction between non-zero and zero valued values of $x(t)$.

4.2.2 Maximum

Given two profiles, x and y it is interesting to define the profile of the maximum $z = \max\{x, y\}$. This will be needed in the analysis of if-then-else structures. It is given by the addition of the profiles of x and y so that the right end adds-up to probability one. Formally:

$$\max\{x, y\} = \lambda_{<1>}(x + y) \tag{3}$$

Note that $\max\{x, y\}$ captures the longest execution paths that are either in x or in y.

4.2.3 Power

We are also interested in modelling the profile of the repetitive execution of a piece of code (loops). Given a profile x, then *exactly* n convolutions of x, denoted by x^n, is given by:

$$x^n = \overbrace{x \otimes x \otimes \cdots \otimes x}^{n} \tag{4}$$

There is an essential difference if the loop does not iterate exactly n times but n is only the *maximum* number of iterations. In this case we have to consider the maximum of the profiles that correspond to 1,2, ... n iterations. We denote this case by $x^{<n>}$ and it is given by

$$x^{<n>} = \max\left\{x^1, x^2, \cdots, x^n\right\} \tag{5}$$

4.3 Timing Schema for Independent ETPs

We can now introduce a timing schema for mutually independent ETP. Let $\mathcal{W}(\mathsf{X})$ denote the execution profile of section of code X (it can represent a single instruction, basic block or a full subtree of the syntax tree). Using the concepts just defined the timing schema is then:

- $\mathcal{W}(\mathsf{X}) =$ Execution Time Profile, $when \mathsf{X} is a basic block$
- $\mathcal{W}(\mathsf{X;Y}) = \mathcal{W}(\mathsf{X}) \otimes \mathcal{W}(\mathsf{Y})$
- $\mathcal{W}(\textsf{if Z then X else Y}) =$ $\mathcal{W}(\mathsf{Z}) \otimes \max\{\mathcal{W}(\mathsf{X}), \mathcal{W}(\mathsf{Y})\}$
- Iterates *exactly* n times: $\mathcal{W}(\textsf{for Z loop X}) = \mathcal{W}(\mathsf{Z}) \otimes (\mathcal{W}(\mathsf{X}) \otimes \mathcal{W}(\mathsf{Z}))^n$
- Iterates *at most* n times: $\mathcal{W}(\textsf{for Z loop X}) =$ $\max\left\{\mathcal{W}(\mathsf{Z}), (\mathcal{W}(\mathsf{X}) \otimes \mathcal{W}(\mathsf{Z}))^{<n>}\right\}$

As with the traditional timing schema, the rules are applied in postorder on the syntax tree. When there are multiple alternatives, the order is not important.

With the notions developed here it is then possible to deduce alternative formulations for these same constructs or to define more specialised rules, for instance, to define a schema for the switch statement, function calls, exception handlers, etc.

5 Dependent ETPs

The description up to now has assumed that the ETP are independent. The mathematical solution is very nice as we can exploit the properties of the convolution. However, the resulting computed execution profile is only an approximation of the *real* profile[2]. There are effects that are (possibly highly) correlated and such correlation is ignored in the previous model. This section addresses the issue of providing an alternative operator for the combination of profiles when the precise information of the dependency between profiles is known. The problems to address are:

- Obtaining information of the dependencies of the code (what information needs to be extracted to be able to determine that there is a dependency of some form). For example, computing correlation indexes from measured execution times or determining dependencies from static code analysis, as well as standard statistical non-parametric hypothesis test for independence [1].

- Representation mechanism. How to represent this information in a form that can be manipulated. For example, as an array of correlations, array of individual probabilities of combined scenarios, code annotations to mark dependent paths, etc. There is a clear trade-off between the amount of information captured, the accuracy of the final result and the complexity of the method.

- Calculation mechanism. Within the context of a timing schema, the issue is to provide an alternative set of rules to compute the profiles with these dependencies.

The two main sources of dependencies result from low-level hardware optimisation features, and high level path dependencies. Low level features result for example from effects of caches and pipelines. These dependencies have an impact on close neighbouring blocks and in the general case the dependency decreases for more distant blocks. High level features result for instance from data dependent paths and mutually exclusive paths. The approach presented in this section addresses the low level issues by exploiting known dependencies in neighbouring blocks, the high-level dependencies are addressed more satisfactorily through adequate timing schema, for instance by defining timing schema for mutually exclusive paths [6].

Using two examples we first analyse the reason why the hypothesis of independence can result either in pessimistic or optimistic estimates. We then discuss the three problems in turn for the case of a sequences of blocks.

[2]We do not know, in the general case how the *real* execution profile looks like (otherwise we would not need to do this analysis!). However, we have build small test cases for which an exhaustive measurement approach is able to produce the *real* ETP. We can then compare them against the computed profiles as well as less exhaustive measurement approaches. One of the main hypothesis of this paper is that in the general case end to end measurements do not capture the worst case.

5.1 Optimism and Pessimism of the Hypothesis of Independence

Assuming independent execution of sections of code may be pessimistic or optimistic. To illustrate this cases assume a simple example of a sequence of blocks X;Y; with execution profiles x and y.

An example of an optimistic estimate is the case when there may be a strong positive correlation between the execution times of certain pairs of execution blocks. It may be the case that when X runs for, say t time units, Y always runs for s time units. Then $P(X = t \wedge Y = s) \neq P(X = t)P(Y = s)$, which is the hypothesis made by the convolution. The stronger the dependency the larger the error. We have observed that each basic block has most of the times two peaks in its ETP, this is usually because the first time it runs it may generate some cache misses. In a sequence of blocks X;Y; where X and Y may generate a cache miss each, it is very likely that when X suffers its cache miss, Y also suffers a cache miss too. In some cases Y only gets a cache miss after X has suffered it. An approach that is not biased towards finding such dependency will be optimistic and could lead to an underestimation of the case when both X and Y suffer from cache misses. Taking into consideration that a cache miss has a large impact in the processor (even 100 cycles) this impact can be significant.

An example of pessimism is the case when (in the same code X;) X and Y share a variable and therefore it is not possible for X generating a cache miss and then also Y generating a cache miss too (as it has just been loaded). If we assume independence, the resulting computed ETP considers the case when both generate a cache miss which is unnecessarily pessimistic.

5.2 Capturing and modelling dependencies: Joint Execution Profiles (JEP)

To help this discussion we assume a measurement based approach in which timing information of the execution time of each basic block in each run is available (see evaluation section for more details). The ETPs are taken from the analysis of a cycle accurate trace obtained by a processor simulator. In this case dependencies can be discovered by looking at the dependencies in each trace.

Consider a sequence of n basic blocks $\mathsf{X}_1; \mathsf{X}_2; \ldots; \mathsf{X}_n;$, with ETPs x_i. The measurement based approach generates a set of runs r_j where each run has the execution time of each block in the run $r_j = \{r_{j,i}\}_{i=1..n}$, where $r_{j,i}$ is the execution time of block X_i in run j. The total execution time of run r_j is simply obtained by $\sum_{i=1}^{n} r_{j,i}$. The ETP x_i

is obtained by building the probability mass function of the set of samples $\{r_{j,i}\}$ for all j.

This process of building the profiles discards the potential information available in the $r_{j,i}$ matrix. We define the *join execution profile* (JEP) as a bidimensional profile w between two blocks X and Y where $w(t,s) = P(X = t \wedge Y = s)$. If x is the ETP of X and y the ETP of Y, w defines explicitly the probability of the case that X runs for t time units *and* Y runs for s time units. The range of w is $(x^- \dots x^+) * (y^- \dots y^+)$.

The sum of all the rows (resp. columns) of w, is given by $w(t,\cdot) = \sum_{\forall s} w(t,s)$ (resp. $w(\cdot,s) = \sum_{\forall t} w(t,s)$). Given a JEP w and two ETP x, y we will say that w is *consistent with x and y* if $w(t,\cdot) = x$ and $w(\cdot,t) = y$ (this also implies that the dimensions of w are adequate).

Operating with JEPs is very similar to operating with profiles. The important realisation is that we can now provide an alternative to the convolution based on the JEP. Given a JEP w by extension of the convolution, we define the convolution of w, $\sigma(w)$ as the ETP where $\sigma(w)(t)$ is the sum of the probabilities of the elements of the SW-NE diagonal of w:

$$\sigma(w)(t) = \sum \{w(s,u) | s + u = t\} \tag{6}$$

Note that this is effectively the same operation as the convolution of equation (1) replacing the product $x(t)y(s)$ by $w(t,s)$. If x and y are independent, then its JEP w has the property $\sigma(w) = x \otimes y$.

The main problem with the JEP approach is that it does not have the same properties as the convolution in the general case. The convolution of JEP is not closed, as $\sigma(w)$ results in an ETP. However, it provides the foundation for replacing some convolutions between ETP by its JEP counterpart.

5.2.1 Dependency tests

We have identified two problems related to the dependency between profiles. The first one is to determine whether two ETPs are independent (is $P(X = t \wedge Y = s) = P(X = t)P(Y = s)$?). The second problem is, independently of the previous result, can we order JEPs according to their degree of dependency; i.e. provide a relational operator $w \geq v$ meaning that w captures more dependency information than v for two arbitrary JEPs w, v (possibly referring to different pairs of ETP).

The first problem is a statistical test of independence. As we do not make any assumption of the underlying distribution we need to use a non-parametric test, for example a chi-squared test [1]. The test hypothesis $H0$ is $P(X = t \wedge Y = s) = P(X = t)P(Y = s)$ and the alternative hypothesis is $P(X = t \wedge Y = s) \neq P(X = t)P(Y = s)$. With some confidence level α the test accepts or rejects the hypothesis $H0$. If so, then for the combination of X and Y the standard discrete convolution is adequate. Otherwise a convolution based on the JEP (see below) or a pessimistic combination (next section) should be used.

We need a simple approach for sorting JEP according to their degree of independence, a simple indicator based on the chi-squared test suffices for our needs. We introduce the dependency index of a JEP, denoted by $\kappa(w)$, as:

$$\kappa(w) = \sum_{\forall t,s} \frac{(w(t,s) - w(t,\cdot)w(\cdot,s))^2}{w(t,\cdot)w(\cdot,s)} \tag{7}$$

Given two JEPs (possibly not independent) w, v we say that w captures more dependencies than v if $\kappa(w) > \kappa(v)$.

In our experiments we compute the JEP from measured data, however there is no reason why these JEP can not be obtained or determined by different means, for instance by static code analysis, or by a combination of both.

5.3 Revised Timing Schema for Dependent Blocks

We can now introduce an specialised rule in our set of timing schema that captures the special case of a sequence of basic blocks with known dependencies: $\mathsf{X}_1; \mathsf{X}_2; \dots; \mathsf{X}_p$. Let $w_{i,j}$ be the JEP of the blocks X_i and X_j in the sequence. The problem is to provide a new timing schema of $\mathcal{W}(\mathsf{X}_1; \mathsf{X}_2; ...; \mathsf{X}_p)$.

If the blocks were independent, then the ETP of the sequence would be $v = x_1 \otimes x_2 \otimes \dots \otimes x_p$ where x_i is the ETP of X_i. The idea behind this method is to replace pairs of convolutions $x_j \otimes x_k$ by the convolution of its JEP $\sigma(w_{j,k})$. There are $q = \lfloor p/2 \rfloor$ of such pairs. The issue is to decide which q pairs of $w_{i,j}$ to replace. It is reasonable to replace the q pairs that account for the maximum dependency information which is captured by the $\kappa(w)$ function. The method can then be expressed in algorithmic form as follows:

1. let $x_1, x_2, \dots, x_p$ be the ETPs of $\mathsf{X}_1; \mathsf{X}_2; \dots; \mathsf{X}_p$

2. Define a permutation ϕ to reorder the x_i : $x_{\phi(1)}, x_{\phi(2)}, \dots, x_{\phi(p)}$ according to the degree of independence so that $\kappa(w_{\phi(2i-1),\phi(2i)}) \geq \kappa(w_{\phi(2i+1),\phi(2i+2)})$ for $i = 1..\lfloor p/2 \rfloor$. The elements x_i are now grouped in pairs and each pair sorted in order of dependency. We can now provide an alternative combination of the sequence, by replacing $\lfloor p/2 \rfloor$ pairs of convolutions between basic blocks by their JEP counterpart:

3. if p is even, then $\mathcal{W}(\mathsf{X}_1; \dots; \mathsf{X}_p) =$

   $$\sigma(w_{\phi(1),\phi(2)}) \otimes \dots \otimes \sigma(w_{\phi(p-1),\phi(p)}) \tag{8}$$

   If p is odd, then $\mathcal{W}(\mathsf{X}_1; \dots; \mathsf{X}_p) =$

   $$\sigma(w_{\phi(1),\phi(2)}) \otimes \dots \otimes \sigma(w_{\phi(p-2),\phi(p-1)}) \otimes x_{\phi(p)} \tag{9}$$

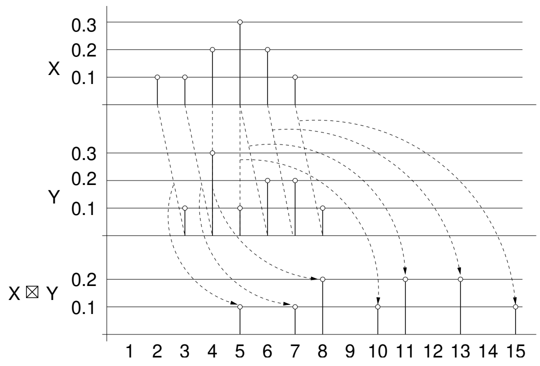


Figure 2. Biased convolution example.

6 Unknown Dependencies between ETPs

The convolution is an adequate operator when we can make the assumption that the ETP are *independent*. However, it has been shown earlier that this can result in optimistic estimates. If information is known about the specific dependencies between blocks (i.e JEP) then the JEP convolution can be used. However, if neither the dependency, nor the independence can be asserted, an operator to combine ETPs is required such that it ensures that the combination is safe (it does not result in an underestimate). For this reason we now present a particular JEP between two profiles x and y, called the *worst JEP*, and denoted by $\hat{w}$ that is consistent with x and y and that is safe.

The way to determine whether a JEP is the worst one, is to define a metric on JEPs and take the JEP that maximises such metric. We define the bias function $\beta(w)$ as the measure of how biased is a JEP towards the worst case. It is defined as $\beta(w) = \sum_{\forall t} t^2\sigma(w)(t)$. The meaning of $\beta(w)$ is to take the contribution of $\sigma(w)(t)$ and weight it quadratically. The more the weight of $\sigma(w)$ is to the right the larger $\beta(w)$. The worst JEP for a pair of ETP x, y is the JEP that is consistent with x and y and that has maximum β among all the possible JEPs consistent with x and y

For notation purposes, we will call the convolution of the worst JEP of two profiles x and y a *biased convolution*, it is denoted by $x \boxtimes y = \sigma(\hat{w})$. It is easy to show that the biased convolution operator is also commutative, associative and distributive.

The worst JEP corresponds to a JEP with a strong positive correlation and where the probability of the rightmost elements of x and rightmost of y is as high as possible. An algorithm to compute such JEP is described below. It is based on taking the two rightmost elements in x, s, and y, t, and determining the maximum probability that s and t may happen which is $p = \min(x(s), y(t))$. This is the probability assigned to the case $w(s,t) = p$. The process is repeated with the remaining ETP. The procedure is illustrated with an example in figure 2. The probability $P(x = 7 \wedge y = 8) = 0.1$. That would assume a total correlation in the worst case. Then, $P(x = 6 \wedge y = 7) = 0.2$, $P(x = 5 \wedge y = 6) = 0.2$ because is the only assignment consistent with the fact that $y(6) = 0.2$. We have now 0.1 units of probability left from $x(5)$ that are used to combine with $y(5)$, etc.

The algorithm to compute the worst JEP is as follows:

```
Computation of ŵ of x and y
ŵ(i,j) := 0  ∀i,j;
i := x⁺;  j := y⁺;
px := x(i);  py := y(j);
while (px > 0  ∨  py > 0)
{ p := min{px, py};
   w(i,j) := p;
   px := px − p;  py := py − p;
   while (px = 0  ∧  i > x⁻)
      { i := i − 1; px := x(i); }
   while (py = 0  ∧  j > y⁻)
      { j := j − 1; py := y(j); }
}
```

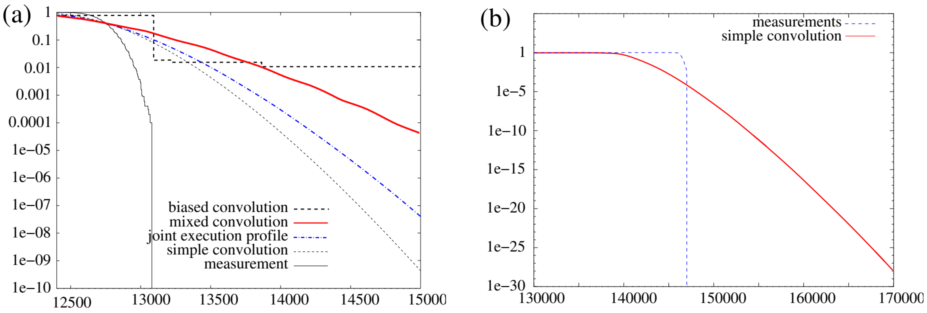
To prove that this algorithm produces the worst JEP we first need to show that such JEP exists. This is based on a constructive proof. Given any JEP w which is consistent with x and y, consider the following transformation of w into w'. Select two rows, r and s, and two columns c and d, such that $r < s, c < d$ and $w(s,c) > 0$, $w(r,d) > 0$. Let $e = \min(w(s,c), w(r,d))$. Then the new JEP w' which identical to w except that $w'(r,c) = w(r,c) + e$, $w'(s,d) = w(s,d)+e$, $w'(s,c) = w(s,c)-e$ and $w'(r,d) = w(r,d)-e$ is also consistent with x and y and $\beta(w') > \beta(w)$ (it is left as an exercise to the reader to show that $\beta(w') = \beta(w) + 2e$). By repeatedly applying such transformation a JEP with maximum β is found. This JEP is unique, and it is easy to show that it is not possible to apply the transformation to the JEP produced by the algorithm, therefore the algorithm produces the worst JEP.

The same timing schema can be used replacing the standard convolution or JEP convolution by the biased convolution. The following section describes in more detail how to select the right operator and how to combine them.

7 Calculation Procedure

The development done up to now allows us to provide the final algorithm for applying the rules of the timing schema that considers programs where for some sections we do not have information of their dependency, for other sections we know (or assume) that they are independent whereas for other blocks detailed information of their dependence is known. We assume that we have the ETP of each basic block x_i, and may have the JEP for a subset of the pairs of blocks ($w_{i,j}$).

The objective is to determine in sequences of operations where the convolution is used, which type of convolution



(c) Source code of (b)

```
for (i = 0;i < VSIZE;i++) {
 for (j = 0;j < HSIZE;j++) {
 if (out[i][j] > 255)
  {out[i][j] = 255;}
 if (out[i][j] < 0)
  {out[i][j] = 0;}
  in[i][j] =
     (unsigned char)out[i][j];
 }
}
```

Figure 3. ETPs of two sample programs

operator to use. The whole procedure of applying the timing schema is summarised as follows:

1. Compute the ETP of sequences of basic blocks $\mathsf{X}_1; \ldots; \mathsf{X}_p$. According to the dependency information available the method to calculate $\mathcal{W}(\mathsf{X}_1; \ldots; \mathsf{X}_p)$ is done in one of three ways:

   (a) if dependency information is known then the convolution of JEPs shall be used as described in equation (8).

   (b) else, if the blocks are independent (test of independence, or assumed explicitly by the analysis) then standard convolution of equation (1) is used.

   (c) Otherwise, no independence assumption can be made and a pessimistic approach shall be used with the biased convolution ($\boxtimes$).

2. For the combination of blocks which are not leafs of the syntax tree:

   (a) For loops: an independence test across loop iterations is applied. If executions times are independent then use the standard convolution operator in the calculation of x^n or $x^{<n>}$, otherwise use the biased convolution in the the calculation of the power.

   (b) Conditional constructs: no dependency information is used and the max of ETP of equation (3) is used.

   (c) For the combinations of sequence of higher level structures (e.g. successive execution of two loops) also an independence test is applied. As before, if they are independent, then the standard convolution can be used, otherwise the biased convolution shall be used.

Alternative or complementary timing schema rules can be defined using the same principles, for instance to model more accurately the difference between the first instance of the execution of a block compared with the rest of the executions, occurrence of cache misses, etc.

8 Case Studies

The examples presented in this section show how WCET analysis can be conducted using ETPs, and the different kind of results obtained by combining ETPs using the methods presented in sections 4, 5 and 6.

The first example (cf. figure 3.a) is a single loop in which pairs of element in an array are accessed randomly. This program has a particular behaviour regarding the data cache. When the access to the first element of a pair is a miss, then accessing the other element will cause a miss. The ETPs of basic blocks used in this experiment, as well as the end-to-end ETPs of the program have been obtained by running the program 10000 times on the *simplescalar* simulator [4]. As the execution path is fixed in this example, the only sources of variability are architectural features (mainly the data cache).

The graphs presented in 3.a show the different end-to-end ETPs (measured and computed). All ETPs are plotted in a negative cumulative way (starting with 1 and ending at 0). A logarithmic scale is used on the vertical axis. The first graph is the measured ETP. It shows that within 10000 runs the worst observed execution time for this program is 13081 cycles and have been observed once. Because of the logarithmic scale ranging from 1 to 10^{-10} the graph of the measured ETP appears at this point as a vertical line. The second graph is the ETP computed using the simple convolution technique presented in § 4, assuming independent ETPs for the basic blocks. As we know that the two memory accesses are positively correlated, the assumption of independence is wrong. Using the biased convolution as presented in § 6 leads to very pessimistic results. Induced by the minimum granularity of $1/10000$, the biased convolution retains this probability up to the theoretical WCET bound at 61015 cycles. A way to describe it more accurately, but without having exact correlation information, would be to consider the blocks inside the loop with the biased convolution, but the loop iterations itself as independent. This provide the curve labelled “mixed convolution”.

The joint execution profile reduces considerably the overestimation. If the blocks were negatively correlated, the joint execution profile curve would be even below that of the normal convolution. Beside the biased convolution, all other techniques provide quickly a probability of 10^{-300}.

The second example is an image processing algorithm. The non deterministic execution time of this program is mainly due to the piece of code presented in figure 3.c. As opposed to the first example, where the executed sequence of basic blocks is always the same, this is not the case in this code. As this image processing program contains some conditional structures, the executed sequence of basic blocks may differ from one run to an other.

The computed end-to-end ETPs take into account the fact that the two conditional statements shown in figure 3.c are mutually exclusive. As in the previous example, the measured and computed ETPs are shown in the graph directly below the code. The WCET bound using conservative WCET estimation approaches would be in this case 809649 cycles. It becomes obvious in this example that the conventional convolution underestimates the execution time to a serious degree compared to the end to end measurements and should therefore be replaced by one of the other operators.

9 Conclusions and Future Work

The use of probabilistic methods in real-time and especially worst case execution time analysis allows to strongly reduce the overestimation produced by traditional approaches. A precondition for their deployment is the notion of probabilistic hard real-time systems. For such systems, it is not mandatory to meet all its deadlines, but a probabilistic guarantee close to 100% suffices.

We have introduced the concept of execution profiles to describe the statistical properties of a section of code. Throughout this paper we have concentrated on the use of execution profiles to describe the WCET of a program. While the profiles of a section of code may be gathered using a variety of existing methods, the presented enhanced timing schema allows for the combination of these sections to provide a model for the longest execution time of a program as a whole. Depending on the knowledge available as regards the dependencies between the EPs of different sections of the code, different join operators are defined for independent sections, sections with known dependency and sections lacking this dependency information at all.

While the first two join operators allow for a close modelling of the execution time, the operator for unknown dependencies uses a pessimistic worst case scenario. The results in the case studies show, that a partially known dependencies between sections of code, enhance the properties of the resulting execution profile of a program considerably.

The future work will focus on the use of execution profiles to describe the effects in the acceleration units like branch prediction and caches of the program. Finally the consequent extension of the model towards schedulability analysis is intended.

Acknowledgements

The authors would like to thank Martin Newby and Veronica Lima for helpfull discussions on early versions of the paper. Additionally the reviewers of RTSS deserve a large "Thank You" for the quality of their comments.

References

[1] A. O. Allen. *Probability, Statistics, and Queueing Theory*. Academic Press, Inc., 111 Fifth Avenue, New York, 1978.

[2] R. Arnold, F. Müller, D. Whalley, and M. Harmon. Bounding worst–case instruction cache performance. In *Proc. of the IEEE Real–Time Systems Symposium (RTSS'94)*. IEEE Computer Society Press, Dec. 1994.

[3] I. Berger. Can you trust your car? *IEEE Spectrum*, 39:40–45, Apr. 2002.

[4] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *Computer Architecture News*, 25, June 1997.

[5] A. Burns and S. Edgar. Statistical analysis of WCET for scheduling. In *Proc. of the IEEE Real–Time Systems Symposium (RTSS'01)*, London, United Kingdom, Dec. 4–6 2001.

[6] A. Colin and G. Bernat. Scope-tree: a program representation for symbolic worst-case execution time analysis. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, Vienna, Austria, June 19–21 2002.

[7] A. Colin and I. Puaut. A modular and retargetable framework for tree-based wcet analysis. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, pages 37–44, Delft, Netherlands, June 13–15 2001.

[8] R. Desikan, D. Burger, and S. W. Keckler. Measuring experimental error in microprocessor simulation. In *Proceedings of the 28th International Symposium on Computer Architecture*, July 2001.

[9] J. L. Diaz, D. F. Garcia, K. Kim, C. Lee, L. Lo Bello, J. M. Lopez, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002*, Austin, Texas, USA, Dec. 3–5 2002.

[10] J. Engblom. On hardware and hardware models for embedded real–time systems. In *Proceedings of the 1st Real–Time Embedded Systems Workshop (RTES'01)*, London, UK, Dec. 3 2001. IEEE.

[11] M. K. Gardner and W. Liu. Analyzing stochastic fixed–priority real–time systems. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, Amsterdam, Netherlands, March 22–26 1999. Springer–Verlag.

[12] C. Park and A. Shaw. Experiments with a program timing tool based on source–level timing schema. *IEEE Transactions on Computers*, 24(5):48–57, May 1991.

[13] S. M. Petters. *Worst Case Execution Time Estimation for Advanced Processor Architectures*. PhD thesis, Institute for Real–Time Computer Systems, Technische Universität München, Munich, Germany, Sept. 2002.