Regression-based Statistical Bounds on Software Execution Time *

Peter Poplavko[1], Ayoub Nouri[1], Lefteris Angelis[2,3], Alexandros Zerzelidis[2], Saddek Bensalem[1], and Panagiotis Katsaros[2,3]

[1] Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
[2] Information Technologies Institute, Centre of Research & Technology - Hellas
6th km Xarilaou - Thermi, 57001, Thessaloniki, Greece
[3] Department of Informatics, Aristotle University of Thessaloniki, Greece
ayoub.nouri@univ-grenoble-alpes.fr

Abstract. Our work aims at facilitating the schedulability analysis of non-critical systems, in particular those that have soft real-time constraints, where WCETs can be replaced by less stringent probabilistic bounds, which we call Maximal Execution Times (METs). In our approach, we can obtain adequate probabilistic execution time models by separating the non-random input data dependency from a modeling error that is purely random. To achieve this, we propose to take advantage of the rich set of available statistical model-fitting techniques, in particular linear regression. Although certainly the proposed technique cannot directly achieve extreme probability levels that are usually expected for WCETs, it is an attractive alternative for MET analysis, since it can arguably guarantee safe probabilistic bounds. We demonstrate our method on a JPEG decoder running on an industrial SPARC V8 processor.

1 Introduction

We propose a new statistical measurement-based method, for the timing analysis of software programs. Such methods aim at highly-probable execution time overestimations, as opposed to the 100% certain upper bounds given by common worst-case execution times (WCET) techniques. This option can be justified in many practical situations. For systems that do not have safety requirements (*e.g.,* car infotainment) that are characterized by weak, soft or firm real-time constraints, we can rely on statistical (over-)estimations based on extensive measurements that we call probabilistic *maximal* execution times (MET).

The methods used to estimate arguably reliable METs are referred to as *measurement-based timing analysis* (MBTA) techniques. In the recent research literature, the reliability of MBTA techniques has been improved, even to the level of considering them eligible for WCET estimates for hard real-time systems, under some restrictive hardware assumptions (*e.g.,* cache randomization).

* The research leading to these results has received funding from the European Space Agency project MoSaTT-CMP, Contract No. 4000111814/14/NL/MH

Such estimates are the so-called probabilistic WCETs, *i.e.,* METs that hold at an extremely high probability $(1 - \alpha)$ with $\alpha = 10^{-15}$ per program execution [CSH+12] or 10^{-9} per hour, which corresponds to the most stringent requirements in safety-critical standards.

Analyses aiming to 'true' WCET (with $\alpha = 0$) are costly to adapt to new application domains and processor architectures, as they require the construction of complex exact models that have to be verified. The techniques based on extreme value theory (EVT) can ensure the levels of probability that render them suitable for WCET. However, these techniques assume that the execution times are random and identically distributed, a strong assumption that does not generally hold in practice. Execution times typically show significant autocorrelations and their probability distribution varies due to the input data dependencies.

For non safety-critical systems, one can settle for METs characterized by α a few orders of magnitude larger than that claimed by EVT methods (10^{-15}). In this case, it is possible to rely on a rich set of mature statistical model fitting tools, such as *linear regression*, which can handle the input data dependencies. In this paper, we propose a novel probabilistic MET analysis technique that builds upon linear regression and the associated statistical analyses.

The contributions of the paper are the following. In Section 2, we discuss MBTA and recall linear regression basics. In Section 3, we introduce a regression model, called Maximal Regression Model, that yields probabilistic upper bounds for METs estimation, using confidence intervals. A great challenge for building a regression model is to come up with the most influential explanatory variables of the execution time. For this, we propose, in the same section, stepwise regression, an iterative method for building a compact model including the most relevant variables. Since the proposed method is measurement-based, we also propose a statistical technique for assessing the input data in order to obtain pertinent measurements. In Section 4, we rely on all these techniques to propose a complete design flow for METs estimation. In Section 5, we demonstrate our flow using a JPEG decoder case study with a significant input data dependency, which runs on a state-of the art industrial SPARC V8 architecture with caches, reset at every execution start. The related work is further discussed along with the conclusions, in Section 6.

In [LS09], it was proposed to use linear regression for conservative execution time analysis, but without profiting from the rich statistics associated with it. More specifically, that work aims at 100% conservative estimates (without probabilities) and for this reason it focuses on non-statistical linear model fitting techniques. However, targeting 100% conservative estimates may result in a costly analysis, losing the advantage of regression. Moreover, their technique for calculating the regression parameters is rather *ad hoc* and is not described in detail. On the other hand, in [LS09], an important connection is established between linear regression and WCET analysis methodologies, which is based on implicit path enumeration.

In that work, some interesting possibilities are also shown for the explicit modeling of hardware effects, *e.g.,* pipelining, which could be used in our work.

However, for simplicity, in the present paper we do not address the hardware modeling issue directly, but undoubtedly this is an important future work matter. Nevertheless, since our analysis is based on measurements on real hardware and since the variability attributed to hardware is consequence of the variability of input data, we believe that hardware effects are covered indirectly up to a level of accuracy that may be appropriate for non safety-critical applications.

2 Common Probabilistic Techniques

In this section, we first review the general MBTA setting, and we recall the basics of linear regression while providing an interpretation in the context of MBTA.

2.1 Probabilistic Measurement-based Timing Analysis

MBTA consists of initially performing multiple measurements of the program execution times and/or the execution times for its blocks of code, and a subsequent analysis to combine the results and thereafter to calculate the MET bound. The probabilistic variant of MBTA utilizes statistical methods [CSH+12] for the analysis phase.

We denote by Y the execution time, which in general depends on some other variables, X_i. An MET bound with probability $(1-\alpha)$ can be obtained by finding the minimal y such that $\mathcal{P}r\{Y < y\} \geq (1-\alpha)$. Suppose that Y is random with a known continuous distribution f, denoted $Y \sim f$. A possible solution is given through the *quantile function* of that distribution: $y = Q_f(1-\alpha)$, such that, by definition, $\mathcal{P}r\{Y < y\} = (1-\alpha)$.

In the case when Y is normally distributed, *i.e.,* $Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$, we have $y = \mu_Y + \sigma_Y \Phi^{-1}(1-\alpha)$, where Φ^{-1} is the quantile function for $\mathcal{N}(0,1)$. In order to calculate METs using this formula, the 'mean' μ_Y and the 'standard deviation' σ_Y have to be estimated from measurements with enough precision, which requires a large enough number of measured Y samples. The normal distribution can describe many random physical variables, especially noise and measurement errors in model parameters. Furthermore, it provides access to a rich set of mature statistical tools for reliably estimating parameters from measurements.

Unfortunately, neither normal nor any other distribution law can be justified to describe execution times *directly*. Therefore probabilistic MBTA techniques do not consider execution time itself as a random variable, but only some of its characteristics. For example, the normal distribution can be adequate if we suppose that we dispose of an 'oracle model' that for each program run can predict its execution time Y almost perfectly, but still makes a small error due to various independent factors ignored by the 'oracle'. Then it is reasonable to apply the normal distribution law to characterize the error of the 'oracle'. This is, in fact, the underlying idea of our method. It should be mentioned, though, that normal distribution is only adequate for the values of α that are not too small, and thus this idea can be applied only for soft real-time systems.

To sustain very small α, MBTA analyses use EVT [CSH+12]. They apply EVT probability distribution laws, again, not to the execution times directly but to their upper bounds. However, as noted in [CSH+12], to justify the EVT-based techniques an important requirement is that the execution times should be independent and identically distributed (*iid*) random variables. However, this requirement is typically violated due to the dependency on input data via multiple conditional branches and loops in the program. The input data parameters are not *iid* and in a certain sense they are even 'non-random' (no practically adequate distribution law can characterize them). Therefore for programs with complex control flow the applicability of EVT-based techniques is difficult to justify. By contrast, using linear regression, our method separates the non-random factors from the modeling error. The regression is our 'oracle model'.

2.2 Linear Regression in the Nutshell

Linear regression is mostly used to predict *average* execution times [EFH04,HJY+10]. Though our goal is to produce *upper bounds*, we use the same approach as the starting point. The main goal of linear regression is to model a variable of interest Y, called dependent variable, with explanatory variables (or predictors) X_i. The fundamental requirements for the validity of such an analysis is that (i) X_i should have approximately linear contribution to Y and (ii) the approximation error should be normally distributed. The first requirement is realistic since one can always decompose execution time as a linear combination of code block contributions. The second requirement validity is motivated in the previous subsection and is further confirmed by experiments (see Section 5).

The concrete values of X_i represent the possibility to 'explain' (or 'predict'), with some precision, the concrete value of Y. For MET, an important implication is that if we can obtain bounds for X_i this helps us to derive a bound on Y as well. In linear regression [DS81], the dependence of Y on X_i is given by

$$Y(n) = \beta_0 + \beta_1 X_1(n) + \ldots + \beta_{p-1} X_{p-1}(n) + \epsilon(n) \tag{1}$$

In the context of MBTA, the dependent variable Y is the program execution time, and $Y(n)$ is its n^{th} observation in a series of measurements. Coefficients β_i are *parameters* that have to be *fitted* to measurements $Y(n)$ to minimize the *regression error* $\epsilon(n)$. The dependent variable Y, the error ϵ, and the parameters β_i are components of the execution time and therefore they can be modeled as real numbers. Their probability distributions are assumed to be continuous, as it is usually the case for timing metrics in statistical MET methods [CSH+12,BCP02]. On the contrary, the predictors X_i are discrete; they are in fact non-negative integers that count the number of times that some important branch or loop iteration in the program is taken or skipped. The corresponding parameter β_i can be either positive, to reflect the processor time spent per unit of X_i, or negative, to reflect the economized time.

From a probabilistic MBTA perspective, equation (1) has a concrete meaning. It captures the 'non-randomness' of Y by building a model $\sum_i \beta_i X_i(n)$ which

'explains' its dependence on some factors X_i, with different weights β_i, reflecting the complexity of the program. Ideally, the remaining 'non-explained' part is a random variable $\beta_0 + \epsilon(n)$ with β_0 representing the mean value and $\epsilon(n)$ the random deviation, whereby $\epsilon(n)$ are hopefully independent and normally distributed, by $\mathcal{N}(0, \sigma_\epsilon)$.

The probability bounds proposed in this work are accurate only if this assumption is valid. However, they are generally believed to be robust with respect to deviations from the normal distribution. We can justify the 'randomness' of ϵ by the hypotheses that all non-random factors have been captured by X_i. Also, the normality of ϵ can be justified using the central limit theorem based on the intuitive observation that the sources of execution time variation, *e.g.,* non-linearity of X_i, are additive in nature and independent.

The parameters β_i are 'ideal' abstractions whose exact values are unknown. They can only be estimated based on measurement data, *e.g.,* with the *least-squares method*. We denote by b_i the estimate of β_i and by $\widehat{Y}$ the estimate of Y. Hence, when ϵ is 0, we get an *unbiased regression model*

$$\widehat{Y}(n) = b_0 + b_1 X_1(n) + \ldots + b_{p-1} X_{p-1}(n) \tag{2}$$

whereas the difference $e_{\text{res}}(n) = Y(n) - \widehat{Y}(n)$, called *residual*, serves as an estimation of the error $\epsilon(n)$: $\epsilon(n) \approx e_{\text{res}}(n)$.

For more convenience, we use a vector notation. Let $\mathbf{x} = (X_i \mid i = 0 \ldots (p-1))$, where $X_0 = 1$ is an artificial constant predictor that corresponds to b_0, and $\mathbf{b}$ the vector of parameters estimators. The regression model can thus be seen as the product of $\mathbf{b}$ and $\mathbf{x}$.

The model parameters are obtained from a set of measurements - the so-called training set - through a process known as *model training* (or *fitting*). In our case, the training set consists of N measurements of $Y(n)$ and $\mathbf{x}(n)$, with N recommended in practice to be $N \gg p$, i.e. at least $N > 5p$ [LS09]. We consider a training-set with predictors measurements organized into a $p \times N$ matrix $\mathbf{X}^{\text{train}} = [\mathbf{x}(1) \ldots \mathbf{x}(N)]$, and the corresponding N-dimensional vector of execution time measurements $\mathbf{y}^{\text{train}} = (Y(n) \mid n = 1 \ldots N)$.

3 Linear Regression for MET

In this section, we introduce the maximal regression model for conservative estimation of MET. Then, we propose a technique to identify the most relevant predictors for the model. Since we rely on measurements, we also present a technique for collecting enough input data to ensure a good coverage.

3.1 The Maximal Regression Model

The least-squares method provides a closed form formula to compute the vector $\mathbf{b}$ from $\mathbf{X}^{\text{train}}$ and $\mathbf{y}^{\text{train}}$ (see [DS81]). However, each least-square model parameter b_i is itself a random variable, because it is obtained from a training-set $\mathbf{y}^{\text{train}}$

'perturbed' with a random error ϵ. It turns out from theoretical studies that each estimate b_i can be seen itself as a sample from a normal distribution, since different training sets would lead to distinct samples b_i from the distribution shown in Fig. 1. This distribution has as mean value the unknown parameter β_i and therefore, the estimator samples b_i are likely to be close to β_i.

For the estimation of METs, the model parameters b that are simply '*close*' to β are not adequate. We prefer a conservative model consisting of parameters b^+ that are likely to be *larger* than β. Such parameters can be obtained using the notion of confidence interval, which is an interval $\Delta b = [b^-, b^+]$ that likely contains β (see Fig. 1), such that

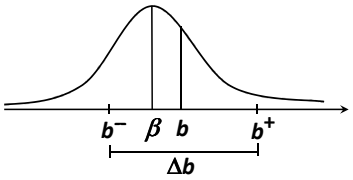


Fig. 1: Parameter Confidence Interval

$$\mathcal{P}r\{\beta \in \Delta b\} = (1 - \alpha) \tag{3}$$

where α is some small value, usually specified in percents, *e.g.,* $\alpha = 5$ %. By symmetry with the distribution of b, if we use b^+, the upper bound of Δb, as coefficient estimator, then our model in the above example is conservative, *i.e.,* with probability $(1 - \alpha/2)$. Therefore, our *maximal regression model* is not the usual unbiased regression model of equation (2), but

$$\widehat{Y}^+(n) = b_0^+ + b_1^+ X_1(n) + \ldots + b_{p-1}^+ X_{p-1}(n) + \epsilon^+ \tag{4}$$

where we assume that ϵ^+ is the (probabilistic) maximal error. By analogy to b^+, we set it to a value, such that $\mathcal{P}r\{\epsilon(n) < \epsilon^+\} \geq (1 - \alpha/2)$. Because $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$ we could use $\sigma_\epsilon \cdot \Phi^{-1}(1 - \alpha/2)$. However, just as the case where we did not know the exact value of β_i and had to obtain an estimate b_i instead, we do not know the value of σ_ϵ and have to use $\epsilon^+ = \widehat{\sigma}_\epsilon^+ \cdot \Phi^{-1}(1 - \alpha)$. The estimate $\widehat{\sigma}_\epsilon^+$ should be pessimistic, *i.e.,* it should be biased to be larger than the value of σ_ϵ with a high probability. When obtaining its unbiased estimate, $\widehat{\sigma}_\epsilon$, the sum of squares of regression 'residual' is involved, $e^2_{\text{res}}(n) = (Y(n) - \widehat{Y}(n))^2$, which is calculated from the training set. Based on the properties of the residual [DS81], we can show that for $\widehat{\sigma}_\epsilon^+ = \sqrt{\sum_{n=1}^{N} (Y(n) - \widehat{Y}(n))^2 / Q_{\chi^2(N-p)}(\alpha/2)}$, we have $\mathcal{P}r\{\sigma_\epsilon < \widehat{\sigma}_\epsilon^+\} = (1 - \alpha/2)$, where $Q_{\chi^2(N-p)}$ is the quantile function of a χ^2 distribution with $(N - p)$ degrees of freedom.

By comparison of equations (1) and (4) we can see that all the terms of the first are likely to be inferior to the corresponding terms of the second, and therefore $\widehat{Y}^+(n)$ is a probabilistic bound of $Y(n)$. Moreover, we have

$$\mathcal{P}r\{Y(n) < \widehat{Y}^+(n)\} \geq \left(1 - \frac{(p+2)\alpha}{2}\right) \tag{5}$$

since we have $(p + 2)$ parameter estimates.

3.2 Identifying the Predictors: Stepwise Regression

When modeling execution times using a regression model, the simplest way to construct the set of predictors is to create a predictor for every block of code of the program that counts the number of block's executions, see *e.g.,* [LS09]. In this case, every program operator that introduces branching, *e.g.,* loop, 'if' operator, would contribute at least with one predictor. This results in a relatively large set of predictors, that we denote P and we call it the set of *potential* predictors. In our method, we would like to *identify* only a small sufficient subset of P for our regression model. By abuse of notation, we call it p (*i.e.,* the same notation for the set and the number of predictors).

By simple rule of thumb $N > 5p$, we see that by ignoring one predictor we can save 5 measurements. However, the rationale is not merely a less costly model, but also the so-called *principle of parsimony*: a model should not contain redundant variables. Many predictors are interdependent, as, for example, in nested loops, where the (total) number of inner-loop iterations is likely to have a strong dependence on the number of outer-loop iterations. From a pair of dependent variables we can try to keep only one, while attributing the small additional effect of the other variable to random error ϵ. If we keep too many variables in P, we will have overfitting, which means that our model will perfectly fit the training set, but it will not be able to reliably predict any program execution outside this set. The reason for this is that an overfitted model would exactly fit not only the 'true' linear dependence $\beta_i X_i$, but also the particular sample of non-linear random noise 'ϵ' encountered in the training set, but not in other samples.

In most of the previous literature on execution time modeling, the identification of predictors is either manual or *ad hoc*. Here we point to a practical and mathematically sound algorithm for identifying the subset p of P. In applied mathematical studies, the identification of a subset of useful predictors in a set of candidates is an important problem to solve (see Ch. 15 of [DS81]).

An overall strategy of most such methods is based on starting with one predictor and observing the reduction of the model error when adding new predictors. It is thus expected that at a certain number of predictors, the error reaches saturation and new variables do not reduce it significantly anymore. At this point, we stop by adopting the hypothesis that the remaining error represents a 'random noise'. One of the most well-established methods is *stepwise regression*, which we propose for use in the MET analysis. This algorithm is outlined here by the following simple procedure (see [DS81] for details). A tentative set p of identified predictors is maintained, containing initially (for $p = 1$) only the constant predictor $X_0 = 1$, which is always kept in the set. The algorithm first tries to add a variable that is 'worthwhile' to add and then to remove a variable that is not worthwhile to keep; the same step is repeated until no progress can be made. A variable is added when it is moved from P to p and it is removed when moving it backwards. When there is no variable that can be added or removed, the algorithm stops.

The criterion for considering a predictor 'worth' to be included depends on the other variables that are already in p; the decision is based on evaluating the least-squares regression $\widehat{Y}$ with and without the candidate predictor. Intuitively, a predictor is 'worth' if its 'signal to noise ratio' is significantly large. The 'noise' here is the total model error, which is evaluated based on the residual sum of squares and the 'signal' is the contribution of the variable to the variance of $\widehat{Y}$. If the variance does not change significantly (compared to the total error) when the predictor is kept, then the predictor is not 'worth'. The whole procedure is controlled by a parameter α^{sw} that sets a threshold for variable acceptance and rejection, and is based on statistical hypothesis-testing procedures under the assumption that modeling error is normally distributed.

3.3 Quality of Input Data: Cook's Distance

The set of measurements should represent all important scenarios that may occur at runtime. To ensure this, the engineer should discover the most influential algorithmic complexity parameters of the program that may vary at run time. Then, it is essential to obtain an input data set, where every *combination* of these factors is represented *fairly*.

For the linear regression, a useful mathematical metric of input-data quality is *Cook's distance*. Given a set of measurements, this metric ranks every measurement n by a numeric 'distance' value $D(n)$ that indicates the amount to which the measurement influences the whole regression model. The regression model should not be dominated by 'odd' measurements; it is generally recommended to have $D(n) < 1$ or even $D(n) < 4/N$. For convenience, let us refer to the measurements with $D(n) > \theta$ as the *bad samples*, for some threshold θ. These samples should be examined, and one should either add more similar samples (so that they are not exceptional anymore) or remove them from the training set (keep them for testing).

4 A Design Flow for MET

We build upon the techniques presented in the previous section to present a complete design flow for MET estimation[4]. In this section, we give a simplified view of the flow and we discuss its steps (see Figure 2). The first phase of the flow is the instrumentation of the input program in order to obtain measurements. Then, the most relevant predictors p are identified. Finally, the model to estimate MET is produced in the model construction phase.

4.1 Instrumentation and Measurements

In some MBTA approaches, multiple blocks of code may have to be instrumented and measured [BCP02]. Such instrumentation can be intrusive, whereas it is

[4] Sources (Octave) can be found at `www-verimag.imag.fr/~nouri/exec-time-lra`

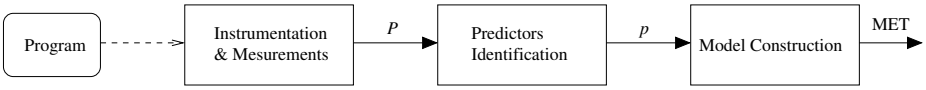


Fig. 2: A simplified view of the MET design flow

likely to obtain inaccurate results when adding the block contributions, due to various hardware effects (e.g. pipelining).

However, the instrumentation is not intrusive in a regression-based approach, where measurements are end-to-end, *i.e.,* they include the entire program. For the end-to-end measurements, $Y(n)$, the program has to be instrumented only at the start and the end. As for the measurements needed to construct the set of potential predictors P and to obtain their values $X_i(n), i \in P$ the instrumented program does not have to run on the target platform; a workstation can be used instead, but it is essential to run the program with the same input data, as those used for the $Y(n)$ measurements. We refer to these measurements as functional simulations[5].

The instrumentation for functional simulations consists of inserting *instrumentation points (i-points)* into the source code of the program. The i-points are inserted at every point, where the control flow diverges or converges, *e.g.,* at the start/end of the conditional and loop blocks, at the branches of the conditional statements *et cetera*. An i-point is a subroutine call passing the i-point identifier 'q', *e.g.,* in Fig. 3 we have points with $q = 1, 2, \ldots, 5$. The goal is to get a measurement record about the path followed in an simulation run. This information consists of the sequence of i-points visited during the simulation run, which is called *i-point trace* and it is denoted as $\mathcal{T}r(n) = (q_1, q_2, q_3, \ldots)$. Examples of traces for Fig. 3 are (1,2,4), (1,2,3,4,5), and (1,2,4,5,5,5).

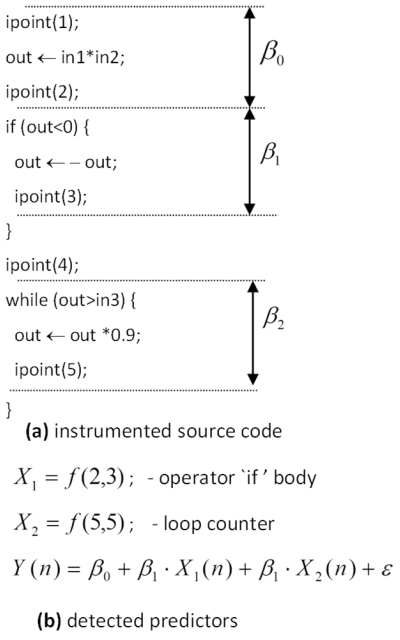


Fig. 3: Instrumentation and predictors

From the traces we automatically detect the set of basic blocks, which correspond one-to-one to predictors in P. We count the number of their occurrences in the trace, denoted $f(q_1, q_2)$, where q_1 and q_2 are the i-point boundaries

[5] For a higher precision, instead of instrumenting the source code one could instrument the binary code for the target platform and run it on an ISS simulator for construction of P, while still using non-instrumented version for the end-to-end execution time measurements on the target platform.

of the block. We have $\forall k, X_k = f(q_i, q_j)$ for some i, j. For the example in Fig. 3 we would detect a predictor $f(2,3)$, which corresponds to the 'if' operator body, and predictor $f(5,5)$, which corresponds to one loop iteration. For a general procedure, we refer the reader to [PAN+16].

4.2 Final Flow Steps

As sketched in Fig. 2, having done the measurements and detected the set of potential predictors P, we identify the final set of predictors $p \subseteq P$, as described in Section 3.2. It is worth mentioning that measurements are separated into two sets, a training set $(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}})$ and a test set $(\mathbf{X}^{\text{test}}, \mathbf{y}^{\text{test}})$[6], and that only the former is used to construct the model, whereas the latter is used to evaluate its quality. The next step of the flow is the construction of the maximal regression model, as described in Section 3.1. This phase will instantiate the model, *i.e.,* given the set of predictors and their associated measurements $\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}}$, it will estimate their coefficients, *et cetera*. In this phase, we also evaluate the quality of input data as described in Section 3.3. Finally, we calculate a MET bound.

Pragmatic MET. Our maximal regression model could be used within the context of the *implicit path enumeration technique* (IPET) [LS09]. In this case, the MET would be computed by $\epsilon^+ + \max_{\mathbf{x}\in\mathbb{X}} \left(\sum_{i=0}^{p-1} b_i^+ X_i\right)$, with $\mathbb{X}$ the set of all vectors $\mathbf{x}$ that can result from feasible program paths. This is achieved by solving the integer linear programming (ILP) problem with a set of constraints on the variables X_i. The constraints are derived from a *static program analysis*, which requires sophisticated tools, as well as from user provided hints, such as loop bounds. We have not yet implemented the IPET method in our flow. Currently, we assume that for each predictor we have (either from measurements or user hint) its minimal and maximal bound, X^- and X^+ and we calculate the pessimistic estimate: $\epsilon^+ + b_0^+ + \sum_{i=1}^{p-1} (bX)_i^+$, where $(bX)^+$ is b^+X^+ if $b^+ > 0$ or b^+X^- otherwise. We refer to this estimate as the *pragmatic MET*.

It is true that the pragmatic MET can be very pessimistic; for example, in switch-case branching it may associate with every case a separate predictor and then assume that they all take the maximal value simultaneously. Nevertheless, the pragmatic MET is safe with the probability bound (5) if the regression model itself is safe with this bound.

5 A JPEG Decoder on a SPARC Platform

We use a JPEG decoder program written in C[7] to illustrate our method. The JPEG decoder processes the header and the main body of a JPEG file. Basically,

[6] A common practice is to consider 70% for the training set and 30% for the test set.
[7] Downloaded from Internet, presumably authored by P. Guerrier and G. Janssen 1998

the main body consists of a sequence of compressed MCUs (Minimum Coded Units) of 16×16 or 8×8 pixels. An MCU contains pixel blocks also referred to as 'color components', as they encode different color ingredients. In the color format '4:1:1' an MCU contains six blocks. For monochromatic images, the MCU contains only one pixel block. The pixel blocks are represented by a matrix of Discrete Cosine Transform (DCT) coefficients, which are encoded efficiently over few bits, so that a whole pixel block can fit in only a few bytes.

The hardware for the execution time measurements was an FPGA board featuring a SPARC V8 processor with a 7-stage pipeline, a double-precision FPU, a 4 KB instruction cache, a 4 KB data cache, a 256 KB Level-2 cache, and an SDRAM. The data caches were reset at every new program run (*i.e.,* after loading a new JPEG image), so that the data caches are always empty at the beginning.

5.1 Instrumentation and Measurements

We used 99 different JPEG images of different sizes and color formats, which yields 99 execution traces including the predictors X_i and the execution time Y.[8] From the generated traces we detected 103 potential predictors. We then randomly split the complete set of 99 measurements into $N = 70$ for the training set, and 29 for the test set used to verify the regression model. In the training set, 8 predictors showed up as constants and they were therefore eliminated, thus ending up with $P = 95$ potential predictors, plus one constant $X_0 = 1$ added by default. Since we had a training set size $N = 70$, by the rule of thumb, we should not exceed $N/5 = 14$ variables, to avoid overfitting.

It is worth mentioning that the maximal observed execution time (over the whole set of measurements) corresponds to an image of a particularly large size, yielding *maximal measured time* of 23643 Mcycles, while the *mean time* was only 1000 MCycles. In the remaining discussion, all timing values (*e.g.,* errors) are reported in Megacycle units. We use $\alpha = 0.05$ for the maximal regression parameters and MET, but we also present the final estimate for $\alpha = 0.00005$.

5.2 Predictors Identification and Model Construction

Basic Model. The simplest model to build is when $p = 1$, *i.e.,* when the execution time is modeled as a purely random variable $\beta_0 + \epsilon(n)$ without non-random contributors. This case corresponds to a naïve measurement-based method where the execution time does not capture the non-random factors. In this case, we cannot expect good results with such a strong input data dependency as in JPEG decoders. Indeed, we carried out a normality test for Y using the Kolmogorov-Smirnov test that reported only a mere 2% likelihood, which was not surprising as the histogram for Y was considerably skewed and had a few extreme values due to images of exceptionally large size.

[8] We could not obtain more measurements because the FPGA card was available for a limited period of time, and loading data into it required some manual work.

The obtained error in this case is large (compared to the mean) $\epsilon^+ = 6650$ and the *pragmatic MET* is ≈ 8000, which underestimates the maximal measured time; this adversity is the consequence of the relatively large model error whose distribution was essentially not normal and which was actually not random (it could be easily controlled, *e.g.,* made large by using large JPEG images).

In line with our methodology, these observations point to a need of adding more predictors into the model (*e.g.,* those characterizing the image size) in order to ensure a smaller, random and normally distributed error, so that the computation of MET is more accurate.

Our Method. We first tune the α^{sw} ($\approx 20\%$) to obtain $p = 6$, *i.e.,* to have 5 predictors. Table 1 shows the identified variables – in the order of their identification – and the corresponding MET calculation on the training set. The meaning of the identified variables is the following. The first predictor $f(271, 244)$ corresponds to the *byte count* in the 'main body' of JPEG. The second basic-block counter $f(90, 30)$ gives the *pixel block count* specifically for those blocks that had correct prediction of the 0-th DCT coefficient. Typically, such blocks are not costly in terms of needed bytes for encoding. At the same time, the contribution of the costly blocks can be captured by the first predictor. Hence, the $f(90, 30)$ as second predictor can account for the additional computations that were not accounted for by the first predictor; a similar variable in P, the *total* pixel block count, $f(406, 26)$, would give less additional information and hence was not identified by our method.

The remaining predictors have less impact on the execution time. The third predictor, $f(101, 101)$, corresponds to the number of *elements in the color format* minus one, *e.g.,* $5 = 6 - 1$ for the 4:1:1 format and 0 for monochromatic images. Equivalently, it gives the number of pixel blocks per MCU block minus one. We note that this predictor has a negative regression coefficient. The JPEG decoding is characterized by two related cost components: a cost per pixel block (reflected by the first two predictors) and a highly correlated cost per MCU block. The more pixel blocks fit into one MCU, the less overhead per pixel block has the MCU processing and this presumably explains why the found coefficient is negative. The fourth identified predictor, $f(80, 81)$, counts the number of *'padded' image dimensions*, X and Y, *i.e.,* the dimensions which are not exactly proportional to the MCU size (16 or 8 pixels). When an image has such dimensions, less processing is required and less data copying for 'partial' MCU blocks, which presumably explains the negative coefficient for this predictor. Finally, the predictor $f(409, 410)$ is zero for colored images. This predictor counts the total number of MCUs in monochromatic images and its impact is presumably complementary to that of $f(101, 101)$.

The obtained *pragmatic MET* is 26696, which, as we expected, exceeds 23643, the *observed maximal time*. For the MET, we used the X^+ and X^- observed in the measurements. We recall that the pragmatic MET is likely to incur extra overestimation by including unfeasible paths. In fact, this is presumably the case for the presented model, as the calculation in Table 1 may combine a relatively

p	b^-	b^+	X^-	X^+	$(bX)^+$
(Constant)	409.660	637.29	1	1	637
$f(271, 244)$	0.010	0.011	3688	1818500	19752
$f(90, 30)$	0.055	0.070	28	27215	1917
$f(101, 101)$	−49.506	−11.530	0	5	0
$f(80, 81)$	−113.010	−26.009	0	2	0
$f(409, 410)$	0.013	0.022	0	192280	4150
ϵ^+	–	–	–	–	240
Pragmatic MET	–	–	–	–	26696

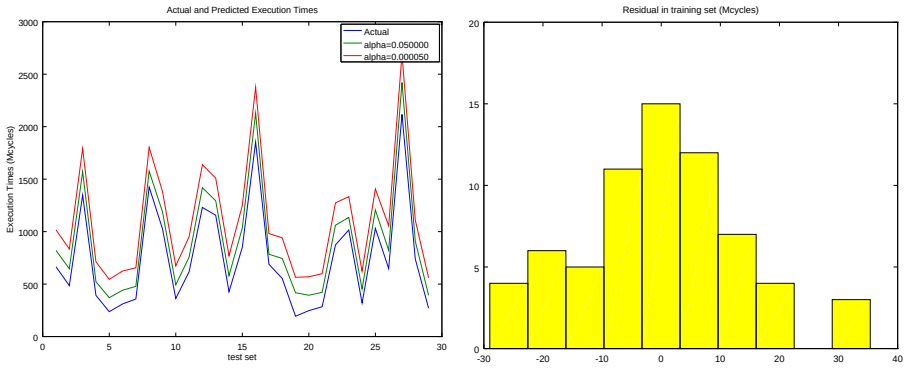
Table 1: Stepwise Regression Results in the Training Set

large byte and block count that is typically required for colored images with pessimistic contributions of the predictors representing monochromatic images. With the IPET approach this possibility would be excluded and a more realistic worst-case vector $\mathbf{x}$ would have been obtained. A lower bound on hypothetical IPET results with the given model is 25764, which is calculated as the observed maximum value of $\widehat{Y}^+(n)$. Compared to $p = 1$, we see a significantly smaller error $\epsilon^+ = 240$. In the test set, we saw reasonably tight overestimations from $\widehat{Y}^+(n)$, however, two underestimations were detected. Analyzing these two samples, we saw that they had Cook's distance significantly larger than all other samples.

Our quality of input data assurance procedure has moved the two samples from the training to the test set and we re-constructed the model for $p = 6$. The obtained error was $\epsilon^+ = 52$ and we observed a tight overestimation for all samples. The normality test of the residual returned 26% likelihood on the training set. The MET has become less accurate, reaching 28048. This is presumably explained by the degraded stability of regression accuracy for the bad samples; the sample that provided X^+ and maximal Y was among such samples. This corresponded to a monochromatic image of exceptionally large size, whereas a vast majority of other samples were color images of much smaller size. In practice, such a situation should be avoided by well prepared measurement data. For technical reasons we could not repair the situation by adding more measurements but we decided to keep the bad samples for illustrative purposes. An observation that should be made, though, is that the instability did not result in unsafe underestimation, but instead in a safe overestimation.

By experimenting with larger values of p, we found that the model with $p = 8$ was optimal. The error ϵ^+ was reduced to 35 and stopped improving, thus showing saturation. With more variables, a degradation of model tightness was observed, probably because the new parameters b started getting 'blurred', showing a Δb much larger than b. The optimal $p = 8$ yielded 97% error normality likelihood, with tight overestimations for all measured samples except for the bad ones; the resulting MET was 56538, not particularly tight due to bad samples, but safe. By (5) this estimate corresponded to $\mathcal{P}r > 0.725$ – for $\alpha = 0.05$. The MET estimations using the same model at $\mathcal{P}r > 0.999725$ amounts to 58859. As it is shown in Figure 4a, the corresponding maximal regression model showed tight overestimations over the measurements not only for $\alpha = 0.05$ but also for

$\alpha = 0.00005$. In Figure 4b, the histogram of residual error is shown that is close to the normal distribution. This is in line with the 97% estimate of normality test and it justifies the use of statistical formulas associated with linear regression.



(a) Obtained execution times (on test set) (b) residuals (on training set)

Fig. 4: Maximal regression model results for $p = 8$

6 Related Work

Historically, linear regression and other model fitting techniques have been mostly used to predict *average*, not conservative, software performance in terms of execution time, *e.g.,* [EFH04], and energy consumption. A regression for *maximal* execution time was proposed in [LS09], but, unlike our work, their regression model is not based on statistical techniques. Instead, the authors sketch an *ad hoc* linear programming based approach and they admit that additional future work is still required. In contrast to our work, *all* potential predictors are included in the model, instead of a small subset of the significant ones, and therefore their techniques presumably require many more measurements to avoid overfitting, and more costly calculations to estimate all parameters. The coverage criteria are based on existence of an hypothetical exact model with a large enough number of variables, which should be known, whereas we tolerate presence of error and estimate the coverage probabilistically. On the other hand, they have showed how a maximal regression model, such as ours, could be combined with existing complementary WCET techniques for calculating tighter execution time bounds than our pragmatic MET formula.

In [HJY+10], regression analysis is used in the context execution time prediction. The proposed method, called SPORE, considers polynomial regression models, as opposed to our work. Although it fundamentally differs from our work, the SPORE method is faced with similar challenges, namely, identifying a relevant compact set of predictors. Two ways are proposed in [HJY+10], which are both variants of the LASSO (least absolute shrinkage and selection operator)

[HTF09] statistical technique. However, since used for prediction, the selection method seems to give an important weight to the cost of computing each predictor. This potentially results in eliminating relevant predictors. Furthermore, no clear indication is given regarding the choice of the input data sample and its impact on the accuracy of the obtained model.

Among the works on statistical WCET analysis, we only consider those that take into account non-random input data parameters. One of the methods proposed in [CSH+12] is to enumerate execution paths of the program and treat them separately, however this approach is appropriate only for programs with simple control flow structure. Another approach is proposed in [BCP02]. In that work, program paths are modeled using 'timing schema', which split the program into code blocks. The WCET distributions of each block are measured separately and then the results for the different blocks are combined. However, this approach requires executing instrumentation points together with timing measurements, which introduces the unwanted probe effect.

7 Conclusions

In this paper, we have presented a new regression-based technique for the estimation of probabilistic execution time bounds. Unlike WCET analysis techniques, it cannot ensure safe estimates at very high probability levels, but it can be utilized for preliminary WCET estimates and in the context of non safety-critical systems. We have described a complete methodology for model construction, which includes an algorithm for identifying the proper model variables and an algorithm for finding conservative model parameters. So far, this technique was tested with only one program, a JPEG decoder, through a limited set of measurements. Nevertheless, it has shown promising results, by giving tight overestimations in the tests.

In future work, it would be interesting to combine the presented regression technique with a complete WCET analysis flow using implicit path enumeration techniques and to study how to model hardware effects using specially defined predictors, similarly to [LS09]. An investigation of possible connections between regression and extreme value theory is also needed, in order to produce high-probability bounds, as in [CSH+12]. Finally, we observed that by putting too many variables into the multi-variate regression analysis the estimation of model parameters is weakened, which manifests in 'blurred' parameter confidence intervals. Therefore, it is interesting to investigate splitting the program into blocks characterized by a smaller set of variables and combining the results by their joint distributions, as in [BCP02].

References

BCP02. G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real-time system. In *Proc. RTSS'02*, pages 279–288. IEEE, 2002.

CSH+12. L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *Proc. ECRTS'12*, pages 91–101. IEEE, 2012.

DS81. Norman R. Draper and Harry Smith. *Applied regression analysis (2nd edition)*. Wiley, 1981.

EFH04. E. M. Eskenazi, A. V. Fioukov, and D. K. Hammer. Performance prediction for component compositions. In *CBSE'04*, pages 280–293. Springer, 2004.

HJY+10. L. Huang, J. Jia, B. Yu, B-G. Chun, P. Maniatis, and M. Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Proc. NIPS'10*, pages 883–891, USA, 2010. Curran Associates Inc.

HTF09. T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction (2nd edition)*. Springer, 2009.

LS09. Björn Lisper and Marcelo Santos. Model identification for WCET analysis. In *Proc. RTAS'09*, pages 55–64. IEEE, 2009.

PAN+16. P. Poplavko, L. Angelis, A. Nouri, A. Zerzelidis, S. Bensalem, and P. Katsaros. Regression-based statistical bounds on software execution time. Technical Report TR-2016-7, Verimag Research Report, 2016.