Performance Estimation of Embedded Software with Confidence Levels

Marco Lattuada, Fabrizio Ferrandi

Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Ponzio, 34/5
20133 – Milan (Italy)
{lattuada,ferrandi}@elet.polimi.it

Abstract — Since time constraints are a very critical aspect of an embedded system, performance evaluation can not be postponed to the end of the design flow, but it has to be introduced since its early stages. Estimation techniques based on mathematical models are usually preferred during this phase since they provide quite accurate estimation of the application performance in a fast way. However, the estimation error has to be considered during design space exploration to evaluate if a solution can be accepted (e.g., by discarding solutions whose estimated time is too close to constraint). Evaluate if the possible error can be significant analyzing a punctual estimation is not a trivial task. In this paper we propose a methodology, based on statistical analysis, that provides a prediction interval on the estimation and a confidence level on meeting a time constraint. This information can drive design space exploration reducing the number of solutions to be validated. The results show how the produced intervals effectively capture the estimation error introduced by a linear model.

I. Introduction

Performance estimation is a key aspect of the design of embedded systems: the designer must guarantee that the produced design meets the required performance constraints. Evaluation of the performance only on the latest stages of a design flow is not possible: a late detection of a not met time constraint would require to rerun a large part of the design flow delaying the release of the final system. Several techniques allow to evaluate the performance of a system. Techniques based on running on real systems or on simulators are usually not suitable to be integrated in an embedded system design flow since they are too slow to be adopted in design space exploration [1]. Also Worst Case Execution Time (WCET) techniques [2] can not be easily integrated in a design space exploration automatic flow: even if they can be implemented in automatic analysis tools, they still require designer interaction (e.g., setting the bounds of loop iterations). For these reasons, performance estimation techniques based on mathematical models are usually preferred. The error introduced by performance estimation techniques on some applications can be significant: the produced solutions have still to be validated with exact techniques. The computed predictions can however be useful to select good candidate solutions, reducing the number of validations to be performed. Solutions whose predicted time does not meet time constraints have to be rejected since will likely not meet the constraints. Solutions whose predicted time meets the constraints have instead to be further analyzed by examining the difference between the predicted time and the constraint time. Evaluating how big this difference should be is critical: if we accept design solutions whose predicted time is too close to the constraint, there is a significant probability that they do not actually meet the constraint requiring much more validations and reruns of the design flow in case of violating solutions. On the contrary, a too conservative choice (e.g., discarding also solutions not so close to the time constraint) can increase the requirements in terms of power and resources of the final solution.

Performance estimation techniques usually provide only the punctual prediction of the time of an application: deciding if the application likely meets a time constraint is delegated to the designer. In this paper we propose a performance estimation methodology for single processing element (e.g., GPP, DSP), based on linear regression, aimed at estimating the execution time of a sequential application or of a single task. The methodology does not compute only the punctual prediction of the execution time, but provides for each new analyzed application whose unknown execution time is Y and whose predicted execution time is $\widehat{y}$:

- a prediction interval $PI = [l, u]$ of given confidence level $1-\alpha$, i.e., an interval such that the probability $P[l < Y < u] = 1 - \alpha$ and $\widehat{y} = \frac{l+u}{2}$;
- the confidence level CL on meeting a time constraint t, i.e., the probability $P[Y < t]$.

According to the confidence level, the solution can be accepted, can be redesigned if it does not sufficiently guarantee to meet the constraints or, on the contrary, can be redesigned reducing power and resources usage.

Prediction intervals and constraints checks provide only a confidence level which identifies the probability that the actual execution time is in that interval or that the time constraint is met: to assure to meet the constraints exact techniques have still to be used. Confidence levels however provide the designer a mechanism to control the trade-off between design time and quality of design solutions: a larger confidence level will produce more conservative solutions in less time, a smaller confidence level will produce better solutions in terms of power and resources, but potentially requiring more runs of the design flows.

This paper is organized as follows. Section II presents the related work while Section III describes the proposed methodology. Section IV discusses the experimental results and finally Section V concludes this paper.

II. Related Work

Linear regression is a well known technique exploited for performance estimation in embedded systems; the methodologies based on it mainly differ for the abstraction level of the application analysis.

Suzuki et *al.* [3] for example propose a methodology which predicts the execution time of an application starting from high-level C language statements. Detailed characteristics of the target architecture and effects of the compiler optimizations can not been taken into account, so estimation can be applied only to a limited set of applications. Moving down in abstraction levels or in the compilation flow improves the accuracy of the proposed models since it allows to take into account more details of compiler or target architecture: Lavagno et *al.* [4] for example use virtual instructions instead of source instructions for estimation. Virtual instructions are produced compiling the application source code to a generic intermediate representation with all the significant operations of a RISC processor. The source code is then regenerated from this intermediate representation annotated with timing information of each virtual operation obtained with linear regression. In this way, some compiler optimizations are explicitly taken into account, but details of the target architecture are still ignored.

To take into account most of the aspects of the compiler and of the target architecture, techniques based on analysis of assembly code have been proposed. Wang et *al.* [5] propose a technique that, analyzing the assembly code, takes into account the compiler optimizations applied on the structure of the Control Flow Graph. This type of techniques allows to consider details of both compiler optimizations and target architecture producing very accurate models, but it requires detailed information about the target architecture assembly.

All these methodologies produce punctual prediction: Bjureus et *al.* [6] propose instead a technique for the construction of prediction intervals, named confidence intervals in their work. The execution time of the application is computed as the sum of the contributions of each operation composing it. Since execution time of each operation is expressed by a stochastic variable, the overall result is a stochastic variable too whose mean is the sum of the means of the operations and whose variance is the sum of the variances of the operations. Probability distributions of the single operations have to be built by hand by the designer, so this technique can be applied only when the designer has a very deep knowledge of the target architecture. On the contrary, the methodology proposed in this paper requires the same knowledge required to build the linear model, so it can be applied also without knowledge of the target architecture exploiting source code analysis.

Probability distributions have been introduced also in WCET analysis: Bernat et *al.* [7] introduce the notion of probabilistic hard real time system. These systems still have to meet in principle all the deadlines, but an high probabilistic guarantee suffices. Worst case execution time of the single basic blocks is described as a probabilistic variable: final execution time is produced combining these probability distribution. Even if it relaxes some constraints of the Worst Case Execution Time analysis, this methodology still requires accurate static analysis to build the execution time profiles of the single basic block. For this reason, these analyses are still time consuming and require direct interactions with the designer. The methodology we propose aims at building a prediction interval or computing the confidence level on meeting of a constraints considering average execution time of the application and not the worst case execution time. These results can be however useful to reduce the set of possible candidate solutions to be examined with a Worst Case Execution Time technique.

III. Proposed methodology

The proposed methodology computes prediction intervals and confidence levels on meeting time constraints of new analyzed applications exploiting models built with linear regression technique. For this reason, it requires that the execution time of an application can be approximated as a linear combination of some numerical features describing the application itself. There are not restrictions on the numerical features nor on the target processing element: analysis can be performed at whatever abstraction level and on whatever processing element.

The methodology is mainly composed of two phases:

1. *Model Building*: the model is built exploiting linear regression techniques and a set of applications (training set) whose execution time is known;
2. *Performance Prediction*: a new application *a* is analyzed in order to compute:
   - punctual prediction $\widehat{y}_{\overline{x}_a}$ of execution time (Equation 3),
   - prediction interval PI with a given confidence level (Equation 13) and
   - confidence level CL on meeting a time constraint (Equation 16).

In the following, the two different phases of the methodology are detailed. Note that in the rest of the paper capital letters will be used to identify stochastic variables while small letters will be used to identify values assumed by them; overlined letters will be used to identify vectors.

A. Model Building

The starting point of the proposed methodology is the building of the performance model with regression analysis. Regression analysis refers to mathematical techniques for the analysis of the relationships among data consisting of a dependent variable Y and one or more independent variables X_i. The dependent variable Y is modeled as a function of the k independent variables X_i, the parameters β_i and a normally distributed error term $E \sim \mathcal{N}(0, \delta^2)$ with the standard deviation δ unknown:

$$Y = f(\overline{X}, \overline{\beta}, E) \tag{1}$$

Section III B shows as the probability distribution of the estimation error $E_{\overline{x_a}}$ on a single application a with numerical features vector $\overline{x_a}$ can be used to compute PI and to evaluate the confidence levels on meeting time constraints.

In the performance estimation problem, the dependent variable Y corresponds to the execution time of the application while the independent variables X_i are a set of numeric features which well describe the performance characteristics of the application. Since we are considering a linear model, we have:

$$Y = \beta_0 + \sum_{i \in 1:k} \beta_i \cdot X_i + E \tag{2}$$

Selection of suitable X_i can be a critical aspect of building a performance model: not only the accuracy of the produced model depends on them, but also the size of the prediction intervals. In particular, their width depends on the standard deviation of the model error as Equation 14 will show. For this reason, the more the features well describe the performance of an application, the smaller the prediction intervals will be. Some performance effects, which are due to particular architectural aspects such as presence of caches, can not be easily described by a linear model, so even with the most significant features a prediction error occurs. In the same way, also the choice of the training benchmarks (e.g., applications whose execution time is known and that are used to build the model) can be a critical element. In particular, they must be representative of the new possible applications: the more the new applications differ from the training ones, the larger their prediction intervals will be.

Given the numerical features extracted from the training benchmarks and their execution times, least squares method is applied to compute the parameters β_i of the model. These values are the only output of this phase in methodologies based on punctual predictions. On the contrary, to compute the prediction intervals, we need to save further data of the particular chosen training set:

- $\overline{\overline{x}} = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,k} \\ \vdots & \vdots & . & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,k} \end{pmatrix}$: the matrix of the values of the independent variables X_i (numerical features); n is the number of benchmarks in the training set;
- $\overline{y} = \begin{pmatrix} y_1 \\ \cdots \\ y_n \end{pmatrix}$: the vector of the values of the dependent variable Y (execution times);
- $RSS = \overline{y}'(I_n - (\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{\overline{x}}')\overline{y}$: the residual sum of squares (I_n is the identity matrix of size n).

B. Performance Prediction

The built performance model and the computed data are used to predict the performance of new applications.

Punctual Prediction of Execution Time

Given the model $Y = \beta_0 + \sum_{i \in 1:k} \beta_i \cdot X_i + E$, the prediction $\widehat{Y} = \beta_0 + \sum_{i \in 1:k} \beta_i \cdot X_i$ (i.e., the model without the error) and a new application a whose features vector is $\overline{x_a}$, we can compute the punctual prediction $\widehat{y}_{\overline{x_a}}$:

$$\widehat{y}_{\overline{x_a}} = \beta_0 + \sum_{i \in 1:k} \beta_i \cdot x_{a,i} \tag{3}$$

Prediction Interval with a Given Confidence Level

Given the model $Y = \beta_0 + \sum_{i \in 1:k} \beta_i \cdot X_i + E$, a new application a whose numerical features vector is $\overline{x_a}$, the interval PI, centered in the punctual estimation $\widehat{y}_{\overline{x_a}}$ and in which the actual execution time of the application $Y_{\overline{x_a}}$ will fall with a probability $1-\alpha$, is called $1-\alpha$ *Prediction Interval*. To compute the bounds of PI, we need to compute the probability distribution of the error $E_{\overline{x_a}}$ introduced by the prediction model on the application a:

$$E_{\overline{x_a}} = Y_{\overline{x_a}} - \widehat{Y}_{\overline{x_a}} \tag{4}$$

Note that $E \neq E_{\overline{x_a}}$: the probability distribution of the generic error of the model is different from the probability distribution of the error on a particular application a.

Since $Y_{\overline{x_a}}$ and $\widehat{Y}_{\overline{x_a}}$ are independent variable with normal distribution [8]:

$$Y_{\overline{x_a}} \sim \mathcal{N}\left(\beta_0 + \sum_{i \in 1:k} \beta_i \cdot x_{a,i}, \delta^2\right) \tag{5}$$

$$\widehat{Y}_{\overline{x_a}} \sim \mathcal{N}\left(\beta_0 + \sum_{i \in 1:k} \beta_i \cdot x_{a,i}, \delta^2 \cdot \overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}\right) \tag{6}$$

their difference, which is the error $E_{\overline{x_a}}$, is a normal distributed variable with mean 0 and variance equal to the sum of the variances:

$$E_{\overline{x_a}} = Y_{\overline{x_a}} - \widehat{Y}_{\overline{x_a}} \sim \mathcal{N}\left(0, \delta^2 \cdot \left[1 + \overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}\right]\right) \tag{7}$$

$$\frac{E_{\overline{x_a}}}{\delta \cdot \sqrt{1 + \overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}}} \sim \mathcal{N}(0,1) \tag{8}$$

δ is unknown, but we can create a new stochastic variable using its estimator RSS. Dividing variable of Equation 8 by $\sqrt{\frac{RSS}{\delta^2(n-k)}}$ and since:

$$\frac{RSS}{\delta^2} \sim \chi^2_{n-k} \tag{9}$$

where χ^2_{n-k} is the chi-square distribution with $n-k$ degrees of freedom, we obtain:

$$\frac{E_{\overline{x_a}}}{\sqrt{\frac{RSS}{n-k}} \cdot \sqrt{1 + \overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}}} \sim t_{n-k} \tag{10}$$

where t_{n-k} is the Student's t-distribution with $n-k$ degrees of freedom.

Fixed the numerical features vector $\overline{x_a}$ of the application a,

$$s = \sqrt{\frac{RSS}{n-k}} \cdot \sqrt{1 + \overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}} \tag{11}$$

is a constant, so we can replace it and the error $E_{\overline{x_a}}$ in Equation 10 :

$$\frac{E_{\overline{x_a}}}{\sqrt{\frac{RSS}{n-k}} \cdot \sqrt{1+\overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}}} = \frac{Y_{\overline{x_a}} - \widehat{Y}_{\overline{x_a}}}{s} \sim t_{n-k} \quad (12)$$

We have obtained a relationship between the actual execution time $Y_{\overline{x_a}}$ and the predicted execution time $\widehat{Y}_{\overline{x_a}}$. Given the percentile $t^* = t_{1-\frac{1}{\alpha},n-k}$ (i.e., the $1-\frac{1}{\alpha}$ percentile of the t-student with $n-k$ degrees of freedom), we have:

$$P\left[-t^* < \frac{Y_{\overline{x_a}} - \widehat{Y}_{\overline{x_a}}}{s} < +t^*\right] = 1-\alpha$$

$$P\left[\widehat{Y}_{\overline{x_a}} - t^* \cdot s < Y_{\overline{x_a}} < \widehat{Y}_{\overline{x_a}} + t^* \cdot s\right] = 1-\alpha$$

so, the prediction interval PI for $Y_{\overline{x_a}}$ of $1-\alpha$ confidence level is:

$$PI = [\widehat{y}_{\overline{x_a}} - t^* \cdot s, \widehat{y}_{\overline{x_a}} + t^* \cdot s] \quad (13)$$

The size of the prediction interval is:

$$2 \cdot t^* \cdot \sqrt{\frac{RSS}{n-k}} \cdot \sqrt{1+\overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}} \quad (14)$$

This size depends on three factors:

- the confidence level (i.e., $1-\alpha$) expressed by the percentile t^*: the larger the confidence level, the larger the interval size;
- the error of the prediction model (i.e., $\frac{RSS}{n-k}$): the larger the error, the larger the interval size;
- the distance between the features of the analyzed application a and the features of the training set applications (expressed through $\overline{x_a}'(\overline{\overline{x}}'\overline{\overline{x}})^{-1}\overline{x_a}$): the less is this difference, the smaller the interval size.

Confidence level on meeting time constraint

Starting from the stochastic variable of Equation 12, the confidence level CL on meeting a time constraint can be computed. In particular given a time deadline t, we want to measure the probability that actual ending time of the application *a* is before the deadline:

$$CL = P\left[Y_{\overline{x_a}} < t\right] = P\left[\frac{Y_{\overline{x_a}} - \widehat{Y}_{\overline{x_a}}}{s} < \frac{t - \widehat{Y}_{\overline{x_a}}}{s}\right] \quad (15)$$

$Z = \frac{Y_{\overline{x_a}} - \widehat{Y}_{\overline{x_a}}}{s}$ is a t_{n-k}, so

$$CL = P\left[Y_{\overline{x_a}} < t\right] = F_Z\left(\frac{t - \widehat{y}_{\overline{x_a}}}{s}\right) \quad (16)$$

F_Z is the cumulative distribution function of $Z \sim t_{n-k}$.

IV. Experimental Evaluation

To evaluate the proposed methodology, we apply it in the performance estimation of the LEON3 processor. LEON3 processor[9] is a softcore 32-bit microprocessor compliant with the SPARC V8 ISA licensed under GNU GPL and currently developed by Gaisler Research.

Section IV A describes the experimental setup, then Section IV B presents the experimental results about prediction intervals. Section IV C finally presents a case study of analysis of meeting time constraints.

A. Experimental Setup

We implement the proposed methodology in PandA [10], a framework for the HW/SW codesign based on GNU GCC [11]. Performance estimation models are built combining host profiling information with GCC RTL internal representations as described in [12]: we retrieve for each type of RTL instruction, how many times instructions of that type have been executed. These counters are used as input of the performance models (i.e., as the X_i variables described in Section III). Real execution times of the analyzed applications (i.e., the Y variable) have been obtained with TSIM [13], a cycle accurate simulator of the LEON3 processor.

The models that we build from the RTL representation can not explicitly describe the effects of all the compiler optimizations since we extract them before the end of the compilation flow. To mitigate this issue, different performance models have been built to model the effects on application performance of different optimizations sets. In particular, we build performance models for applications compiled without optimizations (i.e., `-O0`), and with two level of optimizations (`-O1`, `-O2`).

We have chosen two sets of RTL based features for building the performance models: *All* and *Sel*. *All* is composed of all the RTL operation counters; *Sel* is a subset of them obtained by removing by hand counters of operations not suitable for the building of performance models. Examples of RTL operations not suitable for performance model building are operations present in a limited set of benchmarks (e.g., divisions) and operations whose counters can be derived from other counters (e.g., register writings). We build six different models (3 optimization sets $\times$ 2 features sets) as shown in left part of Table I. To build the models and to compute the predictions described in Section III, we use R [14], a free software environment for statistical computing and graphics. The analyzed benchmarks (more than 600) have been extracted from the following benchmark suites: DSP Stone [15], NAS Parallel Benchmark [16], OmpSCR [17], Powerstone [18], Splash 2[19] and the GNU GCC testsuite [11].

To evaluate the accuracy of the produced models, we apply the *K-fold cross-validation technique* with $K = 5$. This technique, aimed at proving that the prediction error of a model does not depend on the particular choice of the training and testing sets, consists of randomly splitting the dataset in K subsets; the model building process is then repeated K times: at each iteration i, all the subsets but the i-th are used as training set and i-th subset is used as testing set to evaluate the prediction error. At the end the overall error is computed as the average error obtained during the K iterations. Cross-validation

TABLE I
CHARACTERISTICS OF THE ANALYZED MODELS. *Features Set* IS THE SET OF THE FEATURES USED TO BUILD THE MODEL, OL IS THE OPTIMIZATION LEVEL CONSIDERED, $Error$ IS THE PREDICTION ERROR ON THE WHOLE SET OF BENCHMARKS WHEN IT IS USED AS TRAINING SET, $CV\,Error$ IS THE CROSS-VALIDATION ERROR.

Model	Features Set	OL	Error	CV Error
All-0	*All*	O0	7.61%	34.51%
All-1	*All*	O1	10.04%	37.85%
All-2	*All*	O2	15.24%	45.55%
Sel-0	*Sel*	O0	8.81%	9.28%
Sel-1	*Sel*	O1	10.53%	10.72%
Sel-2	*Sel*	O2	16.46%	17.10%

error obtained when *All* features are used is larger because they tend to overfit the training benchmark datasets. On the contrary, cross validation error for models which consider only significant features (*Sel*) is quite close to the error obtained exploiting the whole features set.

B. Experimental Results

We evaluate the proposed methodology by checking for each model if it actually forecasts the interval where execution time of unseen applications will fall. For each model, we adopt an approach similar to the K-fold cross-validation method previously described: we randomly divide the analyzed benchmarks in 5 subsets and we run the methodology five times. During each run, we select a different benchmark subset as validation set: all the other benchmarks are used as training set to build the linear model.

For each benchmark of the validation set, we build the prediction intervals of level 90%, 95%, 99%, then we check if the prediction intervals include the real execution time of the application. For each validation set and for each prediction interval level, we count how many benchmarks have the execution time in the corresponding prediction interval. Finally, combining the information obtained on each validation set, we compute the percentage of these benchmarks with respect to the overall data set: the results are reported in columns *BPI* of Table II.

The results show how the obtained percentages are quite close to the expected ones; moreover, there are not any significant differences between models obtained with different features sets nor considering different optimizations. Nevertheless, the significance of the information provided by the prediction intervals is not the same for all the models. Compare for example the *All-0* model and the *Sel-0* model: even if they have almost the same accuracy in modelling the performance of the applications of the whole dataset, they have a significantly different cross-validation error. The first indeed, having too many parameters, tends to model also the rumor introduced by non performance significant features. The average prediction intervals built on them (reported in columns *Avg.* of Table II) are significantly larger to implicitly describe this difference. For example, considering the 99% prediction intervals, the average size of the second model (36.09%) is much smaller than the first (84.97%): in the first case we are saying that given the predicted time $\widehat{y}$ of a benchmark, the actual execution time will be averagely in the interval $[0.82 \cdot \widehat{y}, 1.18 \cdot \widehat{y}]$ with 99% probability, in the second we have to consider an interval $[0.58 \cdot \widehat{y}, 1.42 \cdot \widehat{y}]$ to get the same probability. In this way, the prediction intervals are both correct, but smaller prediction intervals have to be preferred since allow to accept also candidate solutions whose execution time is closer to time constraints. Moreover, if the prediction interval is too large, it can become quite useless for the performance analysis. In columns *Max.* of Table II, the maximum size of prediction interval of the benchmarks is reported: a prediction interval of 524.15% is equivalent to say that the actual execution time of the application is in the interval $[0, 2.62 \cdot \widehat{y}]$. Such a big interval does not provide any useful information to designer which will have to accept only very conservative solutions or have to lower the desired confidence level.

Prediction models built for higher optimization levels, since the analyzed features are not able to describe all the effects of the optimizations performed by the GNU GCC on the final code, present higher estimation error. For this reason, their prediction intervals are larger.

The presented results show how the size of the prediction intervals depend on the accuracy of the performance model on which they are built. Moreover, they provide also some cross-validation information: the smallest prediction intervals are indeed obtained on those models that not only well model the performance of the benchmarks of the training set, but that are also able to well predict the execution time of new applications.

C. A Case Study of Analysis of Time Constraint

In this Section we show how the confidence level CL on meeting a time constraint can help the designer in limiting the number of executions of WCET analysis. In particular, we consider an implementation of the jpeg encoder customized for encoding of 800x600 pictures and we check if its execution requires less than $100 \cdot 10^6$ cycles. Before applying WCET analysis, we check if the current implementation is a good candidate solution or if we can expect in advance that it will not meet the constraint.

The punctual estimation $\widehat{y}_{\overline{x_{jpeg}}}$ obtained with model *Sel0* is $76.491 \cdot 10^6$ cycles: considering its distance from the constraint and the low cross prediction error of the model (9.28%), we would expect that the implementation actually meets the constraint. On the contrary, computing the confidence level CL on meeting the time constraint with Equation 16, we obtain a relatively low level of confidence: 78.32%. This means that the analyzed solution is not so good: there is a not small probability (21.68%) that the actual execution time of the application does not meet the constraint, and so a significant probability that the application does not meet the constraint according to WCET analysis. Since WCET analysis can be very time consuming and we do not have a very good expectation on its outcome, we would not apply it to the application as it is but we would optimize the application before.

We now show the correctness of the previous considerations by simulating on TSIM the application and by analyzing it with *aiT*[20]. Actual execution time of the analyzed application is much closer to the constraint ($95.452 \cdot 10^6$) and the obtained WCET is $112.473 \cdot 10^6$. So, even if according to punctual

TABLE II

RESULTS OBTAINED WITH THE PROPOSED METHODOLOGY. FOR EACH MODEL AND FOR EACH PREDICTION LEVEL WE REPORT: *BPI*, THE NUMBER OF BENCHMARKS WHOSE REAL EXECUTION TIME IS IN ITS OWN COMPUTED PREDICTION INTERVAL, *Avg.* AND *Max*, THE AVERAGE AND THE MAXIMUM RATIO BETWEEN PREDICTION INTERVAL SIZE AND PREDICTED EXECUTION TIME.

	90% Interval			95% Interval			99% Interval		
Model	BPI	Avg.	Max.	BPI	Avg.	Max.	BPI	Avg.	Max.
All-0	90.47%	35.12%	333.88%	95.63%	54.54%	398.13%	98.33%	84.97%	524.15%
Sel-0	90.83%	19.35%	80.00%	96.21%	25.01%	93.34%	98.41%	36.09%	136.03%
All-1	90.97%	39.52%	345.09%	95.18%	55.52%	345.09%	99.56%	82.03%	454.31%
Sel-1	89.17%	24.79%	86.67%	95.28%	29.63%	97.96%	98.68%	36.90%	142.11%
All-2	93.89%	53.21%	346.94%	95.70%	72.23%	403.43%	97.91%	90.01%	550.42%
Sel-2	90.31%	31.16%	82.25%	94.74%	35.23%	96.14%	98.61%	43.21%	123.38%

estimation the proposed implementation seemed a promising solution, we have to discard the implementation as correctly suggested by confidence level CL.

V. CONCLUSIONS

In this paper we presented a performance estimation methodology which, extending punctual techniques based on linear regression, allows to compute the prediction interval of the estimated execution time and, given a time constraint, a confidence level on meeting that time constraint. Both the characteristics of the applications used to build the model and the characteristics of the new application are taken into account in building the prediction intervals and the confidence levels on constraints meeting, producing less uncertainty for applications expected to be better predicted. Extending linear regression technique, the proposed methodology can be applied each time the performance of an application can be modeled as linear combination of some numerical features.

The results show how the proposed methodology is effectively able to capture the estimation error potentially introduced by the model, building prediction intervals whose size reflects the cross validation error of the model itself.

REFERENCES

[1] Choonseung Lee, Sungchan Kim, and Soonhoi Ha. A systematic design space exploration of mpsoc based on synchronous data flow specification. *J. Signal Process. Syst.*, 58:193–213, February 2010.

[2] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem. overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7:36:1–36:53, May 2008.

[3] K. Suzuki and A. Sangiovanni-Vincentelli. Efficient software performance estimation methods for hardware/software codesign. In *DAC '96: 33rd Design Automation Conference*, pages 605–610, Jun, 1996.

[4] J. R. Bammi, E. Harcourt, W. Kruijtzer, L. Lavagno, and M. T. Lazarescu. Software performance estimation strategies in a system-level design tool. In *CODES 2000: the Eighth International Workshop on Hardware/Software Codesign.*, pages 82–86, 2000.

[5] Zhonglei Wang, Kun Lu, and A. Herkersdorf. An approach to improve accuracy of source-level tlms of embedded software. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1 –6, march 2011.

[6] Per Bjuréus and Axel Jantsch. Performance analysis with confidence intervals for embedded software processes. In *Proceedings of the 14th international symposium on Systems synthesis*, ISSS '01, pages 45–50, New York, NY, USA, 2001. ACM.

[7] Guillem Bernat, Antoine Colin, and Stefan M. Petters. Wcet analysis of probabilistic hard real-time systems. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, RTSS '02, pages 279–, Washington, DC, USA, 2002. IEEE Computer Society.

[8] S.M. Ross. *Introduction to probability and statistics for engineers and scientists*. Elsevier Academic Press, 2004.

[9] Leon 3 Processor. http://www.gaisler.com.

[10] The PandA framework. https://trac.ws.dei.polimi.it/panda.

[11] GCC - GNU Compiler Collection. http://gcc.gnu.org.

[12] Marco Lattuada and Fabrizio Ferrandi. Performance modeling of embedded applications with zero architectural knowledge. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, pages 277–286, New York, NY, USA, 2010. ACM.

[13] Tsim leon simulator. http://www.gaisler.com.

[14] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[15] V. živojnović, J. M. Velarde, C. Schläger, and H. Meyr. DSPSTONE: A DSP-oriented benchmarking methodology. In *ICSPAT '94: International Conference on Signal Processing and Technology*, 1994.

[16] David Bailey, Tim Harris, William Saphir, Rob Van Der Wijngaart, Alex Woo, and Maurice Yarrow. The nas parallel benchmarks 2.0. Technical report.

[17] A. J. Dorta, C. Rodriguez, F. de Sande, and A. Gonzalez-Escribano. The openmp source code repository. In *PDP*, pages 244–250, 2005.

[18] A. Malik, B. Moyer, and D. Cermak. A low power unified cache architecture providing power and performance flexibility (poster session). In *ISLPED '00: Proceedings of the 2000 international symposium on Low power electronics and design*, pages 241–243, New York, NY, USA, 2000. ACM.

[19] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA*, pages 24–36, 1995.

[20] ait: worst-case execution time analyzers. http://www.absint.com/ait.