Predicting the Impact of Functional Approximation: from Component- to Application-Level

Marcello Traiola, Alessandro Savino, Mario Barbareschi, Stefano Di Carlo, Alberto Bosio

Predicting the Impact of Functional Approximation: from Component- to Application-Level

Marcello Traiola[1], Alessandro Savino[2], Mario Barbareschi[3], Stefano Di Carlo[2] and Alberto Bosio[1]
[1]LIRMM - Univ. of Montpellier, CNRS, France, {marcello.traiola | alberto.bosio}@lirmm.fr
[2]Politecnico di Torino, Italy, {alessandro.savino | stefano.dicarlo}@polito.it
[3]University of Naples Federico II, Italy, mario.barbareschi@unina.it

***Abstract*—Approximate Computing (AxC) trades off between the level of accuracy required by the user and the actual precision provided by the computing system to achieve several optimizations such as performance improvement, energy and area reduction etc. Several AxC techniques have been proposed so far in the literature. They work at different abstraction levels and propose both hardware and software implementations. The common issue of all existing approaches is the lack of a methodology to estimate the impact of a given AxC technique on the application-level accuracy. In this paper we propose a probabilistic approach to predict the relation between component-level functional approximation and application-level accuracy. Experimental results on a set of benchmark applications show that the proposed approach is able to estimate the approximation error with good accuracy and very low computation time.**

keywords: Approximate computing; Functional approximation; Quality Metrics; Bayesian Networks;

I. Introduction

Approximate computing (AxC) refers to the idea that computer systems can let applications trade off accuracy for efficiency. Intuitively, instead of performing exact calculations, AxC aims at selectively relaxing the accuracy of the computation in order to gain in terms of lower power consumption, faster execution time, etc.

Several publications demonstrated the effectiveness of this approach when applied to algorithms showing an inherent resiliency to errors [1], [2]. Proposed solutions work both at the hardware or software level.

Among the different AxC techniques [3], *functional approximation* aims at replacing a computational function with a different implementation that closely matches (approximates) the original implementation. Let us denote with the generic term *component* the hardware or software module responsible for the implementation of a computing function. Given an application, the functional approximation allows for replacing one or more components C_p with approximate versions C_{ax}.

Despite the literature being rich of proposals for the implementation of approximate arithmetic operations [4], [5], [6], the selection of the components to approximate, and the selection of the best approximation techniques for an application remains a challenging problem.

Most of the approaches proposed in the literature simply run several times the approximate application (i.e., the application implemented with C_{ax}), and compare the outcomes with the precise application (i.e., the application implemented with C_p) [7], [8]. The comparison is achieved through the adoption of an appropriate error metric. If the accuracy of the approximate application is not satisfactory, the selected approximate component (C_{ax}) must be replaced with a different one. Every time a new C_{ax} is considered, the application must be executed and analyzed again. Since the above process iterates until a desired level of accuracy is reached, the cost depends on the final amount of runs to reach it.

A different approach that analytically formalizes the error induced by C_{ax} and how it propagates in the application has been proposed in [9]The benefit of this approach is that there is no need to execute the application every time its approximation must be evaluated. However, the formalization is clearly application dependent. Therefore, building the formal model of an application is very complex and requires to deeply analyze the algorithms implemented by the application.

Bearing in mind such considerations, this paper presents a stochastic approach to predict the impact of an approximate component C_{ax} on the accuracy of an application. First, each candidate C_{ax} is characterized as an isolated application, thus computing its approximation error distribution. Second, the knowledge of the error distribution of all considered C_{ax} is exploited to build a Bayesian Network (BN) [10] modeling the approximation error propagation through the application's data flow.

The remainder of the paper is structured as follows. Section II overviews the main concepts of the proposed approach. Section II-A presents the C_{ax} characterization, while Section II-C presents the proposed Bayesian model. Results are discussed in Section III. Finally, IV summarizes the main contributions and concludes the paper.

II. Methods

Figure 1 sketches the global modeling flow that is composed of three main steps: (i) **Component characterization**: given a library of approximate components, the goal is quantifying the error introduced by their approximation; (ii) **Bayesian network construction**: it aims at analyzing the application source code in order to build the Bayesian Network modeling the entire data flow of the application; (iii) **Approximation analysis**: the produced Bayesian network is used to analyze the approximation introduced at the application level.

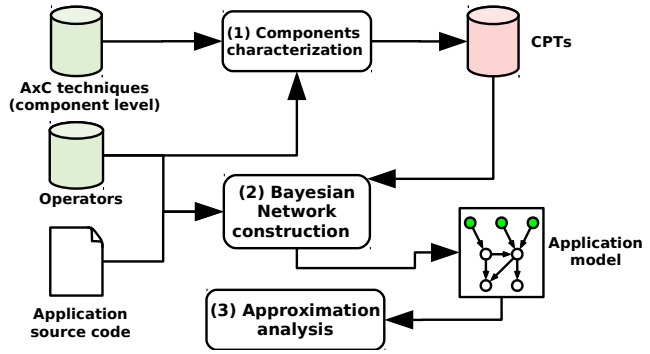

Fig. 1: Approximation estimation workflow.

A. Approximate component characterization

This section presents the approach used to characterize a given set of approximate components C_{ax}. Among the different functional approximation techniques, this paper focuses on the *precision reduction* [3] approach. Given an n bits data type, precision reduction works by reducing the size of the data type cutting its k less significant bits. Therefore, given a number x expressed on n bits, its *approximated value* $\tilde{x}$ is expressed on $n-k$ bits, without changing the original bit weights. This paper exploits the Worst Case Error (WCE) as a quality metric to assess the approximation of an operator:

$$WCE = \max_{\forall i} \left| x_{(i)} - \tilde{x}_{(i)} \right| \tag{1}$$

where (i) is the i-th value of x. Since precision reduction truncates k bit from a data type, the maximum value of WCE is equal to $2^k - 1$. Once both the error and the quality metrics are defined, it is possible to identify the following classes of approximations for $\tilde{x}$:

$$Class(\tilde{x}) = \begin{cases} P \text{ (Precise)} & if\ E = 0 \\ A \text{ (Acceptable)} & if\ E \leq WCE \\ U \text{ (Unacceptable)} & if\ E > WCE \end{cases} \tag{2}$$

As an example, let us apply precision reduction with $k = 2$. Reasonably presuming that the possible values of x are uniformly distributed, the probability to have $\tilde{x}$ in P, A or U can be computed as:

$$\begin{aligned} &P(\tilde{x}\ is\ P) = P(E = 0) = \tfrac{1}{4} \\ &P(\tilde{x}\ is\ A) = P(E \leq WCE) = \tfrac{3}{4} \\ &P(\tilde{x}\ is\ U) = P(E > WCE) = 0 \end{aligned} \tag{3}$$

Equation (3) allows us to quantify the error introduced by the precision reduction, i.e., the adopted functional approximation technique, on a single value. The problem now is to determine how this error propagates through the application when computations are performed. This in turn requires to characterize each operator manipulating the approximated numbers. Let us consider as a case study the sum operator (+). Given two precise numbers x_1 and x_2 and two approximate numbers $\widetilde{x_1}$ and $\widetilde{x_2}$, the application of the sum operators generates two different results denoted as y_{pr} and y_{axc}:

$$y_{pr} = x_1 + x_2 \tag{4}$$

$$y_{axc} = \widetilde{x_1} + \widetilde{x_2} \tag{5}$$

The error introduced by the + operator can therefore be defined as:

$$E_+ = |y_{pr} - y_{axc}| \tag{6}$$

Table Ia lists all possible combinations of events that the operator can observe at its inputs. The goal of the characterization of the operator is to classify y_{axc} as P, A or U depending on the input combinations shown in Table Ia.

TABLE I: Events classification

(a) Input Events

Event	$\widetilde{x_1}$	$\widetilde{x_2}$
1	P	P
2	P	A
3	P	U
4	A	P
5	A	A
6	A	U
7	U	P
8	U	A
9	U	U

(b) y_{axc} Classification

$E\widetilde{x_1}$	$E\widetilde{x_2}$	Ey_{axc}	**Class**
1	1	2	A
1	2	3	A
1	3	4	U
2	1	3	A
2	2	4	U
2	3	5	U
3	1	4	U
3	2	5	U
3	3	6	U

Let us start considering that events 2 and 4 are equivalent, as well as events 3 and 7 and events 6 and 8. Moreover, if at least one input is U, the output is U, otherwise if one input is A and one is P, y_{axc} will be A. Then, if both inputs are P, the output is P. The problem is reduced to the classification of y_{axc} when both inputs are A (event 5). Let us enumerate all possible cases for *Event 5* as shown in Table Ib. The first two columns report all possible errors for $\widetilde{x_1}$ and $\widetilde{x_2}$. The third column computes the error of y_{axc} as the sum of input errors of $\widetilde{x_1}$ and $\widetilde{x_2}$. The last column classifies the y_{axc} error accordingly to equation (3) (with $WCE = 3$). Table Ib allows us to easily compute the following probabilities:

$$\begin{aligned} &P(y_{axc}\ is\ P \mid (5)) = 0 \\ &P(y_{axc}\ is\ A \mid (5)) = 1/3 \\ &P(y_{axc}\ is\ U \mid (5)) = 2/3 \end{aligned} \tag{7}$$

Working with a Bayesian model, the goal of the characterization of a component is to build a Conditional Probability Table (CPT) able to model the conditional probability of having approximation errors at the output of the component, depending on the level of approximation of the inputs of the component. Table II summarizes the approximation probabilities for the + operation with respect to the two inputs in the form of a CPT. The first two rows list all possible combinations of classes for $\widetilde{x_1}$ and $\widetilde{x_2}$. The remaining rows provide the probability for the output classification based on the combination of inputs. A similar analysis can be used to characterize other operators such as difference (−), multiplication (∗) and quotient (/).

TABLE II: Conditional Probability Table for the + Operator

x_1		**P**			**A**			**U**		
x_2		**P**	**A**	**U**	**P**	**A**	**U**	**P**	**A**	**U**
y_{axc}	**P**	1	0	0	0	0	0	0	0	0
	A	0	1	0	1	1/3	0	0	0	0
	U	0	0	1	0	2/3	1	1	1	1

Despite its complexity, this activity must be performed only once for each considered AxC technique. Its results can then be reused several times.

B. Bayesian network construction

Once the impact of a single approximate component on its output data has been probabilistically characterized, this information can be used to evaluate the impact of the approximation at the full software scale. As explained in Section II, the final accuracy of an application depends on the propagation of the introduced approximations across the data of the application. Therefore, we model the application in the form of a Bayesian Network as follows:
(i) all data and operators involved in the computation are represented as nodes;
(ii) edges depict the dependency between data and operators;
(iii) for each node, a CPT express how the approximation of the parents impacts the outcome of a computation.

Once built, this model enables to analyze how errors are propagated from the root nodes down to the leaves representing the outcome of the application. To show how the application is modeled, let us consider the example depicted in Figure 2-A, consisting of a short sequence of instructions performing mathematical operations. The applied precision reduction is $k = 2$ to all the data. In order to build the BN structure, the application Data Dependence Graph (DDG) [11] is analyzed. Yellow nodes in Figure 2-B are input nodes. They express a CPT (Figure 2-C) indicating the marginal probability of the related data to be in one of the approximation classes defined in equation (2). All intermediate nodes represent the different operators that can be associated with the CPTs computed during the component characterization phase (see Section II-A). As an example, the Var2 node involves a sum of the two input nodes. Its CPT reported in Figure 2-D is therefore the one computed for the $+$ operator and reported in Table II. Finally, orange nodes identify the leaves of the network representing the output of the computation. They are the observation points in which the effect of the approximation can be probabilistically analyzed.

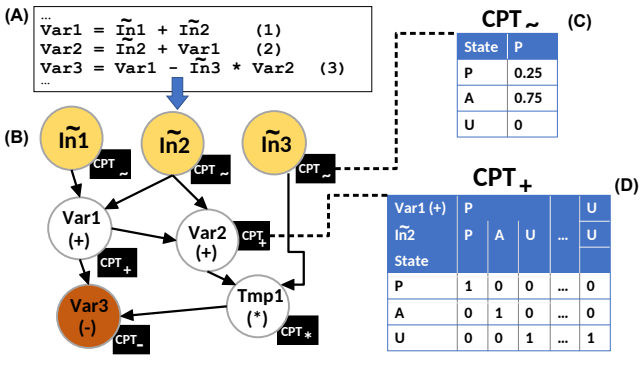

Fig. 2: Bayesian Network example

The full BN creation process has been automated using a publicly available BN library [12].

C. Approximation analysis

Once the BN is built, Bayesian inference can be used to analyze the network in order to predict the level of approximation at the output of the application [13]. The prediction can be made computing the posterior probability of the leaves of the network (orange nodes in Figure 2) to be in one of the three approximation classes defined in equation (3). This can be done by applying different update beliefs algorithms proposed in the literature. In particular, the library used to implement the proposed framework [12] provides two solvers: (1) the exact solver proposed by Lauritzen in [14] that can be used with medium size models (i.e., tens of nodes), and (2) the Estimated Posterior Importance Sampling (EPIS) approximate stochastic solver proposed in [15] that can be used with very large models (i.e., thousands of nodes). The flexibility of the proposed model can be very useful to have a quick insight on the accuracy reduction of the application, thus enabling to quickly explore different design solutions.

III. Experimental Results

The capability of the proposed evaluation approach has been tested on a set of four simple benchmarks. All benchmarks are software applications written in C language. The main purpose of the experiments is to prove the accuracy of our predictions when compared to the real execution of the application with different workloads. The bit reduction parameter k is set to 2 and, therefore, according to equation (1), WCE is equal to 3. To show how the accuracy of the proposed approach is influenced by the target application, we performed a set of experiments on different types of applications including:
Consecutive sums (CSs): a program performing a cascade of consecutive sums
Matrix multiplication (MM): a function that is widely used in several computations including linear equations solvers. Two experiments multiplying two 2×2 matrices and two 4×4 matrices (MM2 and MM4) of 8-bits elements have been performed.
Discrete cosine transform (DCT): a function important for several engineering applications (e.g., audio and images compression).

In order to show the accuracy of the prediction obtained through the proposed model, a precise and the approximate version of each application have been executed 10,000,000 times. At every execution, a random workload has been generated and provided to both versions of the application. For every execution, the result provided by the precise version and the one provided by the approximate version of the application have been compared in order to compute the approximation error as described in equation (6).

Table IIIa summarizes the result of the analysis of the selected applications. For each application, the first row (BN) reports the probabilities computed resorting to the proposed BN based model, while the second (App. Run) row reports the values computed by running the application several times (i.e., both the precise version and the approximate version). The last row (Abs. Error) quantifies the absolute error

TABLE III: Experimental Results

(a)

		P	A	U
CSs	BN	0.0015	01.167%	98.831%
	App. Run	0.0016%	0.257%	99.741%
	Abs. Error	**0.0001pp**	**0.91pp**	**0.91pp**
MM2	BN	0.4673%	0.002%	99.53%
	App. Run	0.0005	0	99.999%
	Abs. Error	**0.4668pp**	**0.002pp**	**0.469pp**
MM4	BN	0.002%	0.00002%	99.998%
	App. Run	0	0	1
	Abs. Error	**0.002pp**	**0.00002pp**	**0.002pp**
DCT	BN	28.736%	69.490%	1.773%
	App. Run	32.173%	66.07%	1.776%
	Abs. Error	**3.437pp**	**3.42pp**	**0.003pp**

(b)

	Time(s)
BN	0.002
App.Run	2.077
Gain	**99.90%**
BN	0.004
App.Run	35.874
Gain	**99.99%**
BN	0.005
App.Run	153.824
Gain	**99.99%**
BN	0.297
App.Run	198.222
Gain	**99.85%**

(c)

# Executions	# input bits
1	
10,000,000	56
1	
10,000,000	64
1	
10,000,000	256
1	
10,000,000	512

($|BN - App.Run|$), expressed in percent points (pp), observed between the two evaluation methods.

Looking at CSs, MM2 and MM4, one can immediately appreciate the capability of the proposed model that is able to estimate the accuracy of the approximate application with a negligible absolute error. For more complex applications, such as the DCT function, the error slightly increases. Indeed, modeling functions of greater complexity in a precise manner is not trivial. In particular, we employed the DCT function used in the JPEG encoder benchmark proposed within the AxBench suite [16] which receives 64 8-bits input values and produces 64 8-bits outputs. Nevertheless, also in this case, the estimation remains accurate with a worst case deviation of 3.5 pp. The reported numbers for DCT refer to the average number of P A and U predicted by the BN and produced by the running application over the 64 outputs.

The benefit of the proposed approach becomes evident when looking at the time required to analyze an application. Table IIIb reports, for each application and for each evaluation technique, the required analysis time expressed in seconds. The BN execution time includes the sum of the time needed to automatically create the BN starting from the application code and the time for its evaluation. The gain is calculated as: $100 * (App\,run\,time - BN\,time)/App\,run\,time$. Moreover, Table IIIc highlights the important gain in terms of the number of executions of the application. Indeed, while to build the BN it is enough to analyze the application only once, the comparison of the approximate and precise application requires several runs to account for the high number of possible combinations of inputs and to produce significant statistical estimations.

IV. Conclusion

In this paper, we proposed a probabilistic approach able to predict the inaccuracy of the results of a complex software application due to the functional approximation techniques applied to selected components. The prediction is performed by modeling the application data flow using a Bayesian Network model. Each operator handling approximated data is characterized only once and reused every time the operator is used in an application. The proposed approach has been tested on a set of relevant software benchmarks. Results showed that the accuracy is high for most applications with a significant gain in terms of computation time. This opens interesting paths toward the use of this model to perform design space exploration in the approximate computing domain.

References

[1] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 113.

[2] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 120.

[3] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2893356

[4] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: imprecise adders for low-power approximate computing," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*. IEEE Press, 2011, pp. 409–414.

[5] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 1, pp. 124–137, 2013.

[6] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *Computers, IEEE Transactions on*, vol. 62, no. 9, pp. 1760–1771, 2013.

[7] S. Lee, L. K. John, and A. Gerstlauer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 187–192.

[8] M. Barbareschi, F. Iannucci, and A. Mazzeo, "Automatic design space exploration of approximate algorithms for big data applications," in *IEEE International Conference on Advanced Information Networking and Applications (AINA-2016). IEEE*, 2016.

[9] K. N. Parashar, D. Menard, and O. Sentieys, "Accelerated performance evaluation of fixed-point systems with un-smooth operations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 4, pp. 599–612, April 2014.

[10] F. V. Jensen, *Introduction to Bayesian Networks*, 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996.

[11] D. L. Kuck, *Structure of Computers and Computations*. New York, NY, USA: John Wiley & Sons, Inc., 1978.

[12] BayesFusion, LLC. SMILE Engine. [Online]. Available: www.bayesfusion.com

[13] G. E. Box and G. C. Tiao, *Bayesian inference in statistical analysis*. John Wiley & Sons, 2011, vol. 40.

[14] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," *International journal of approximate reasoning*, vol. 15, no. 3, pp. 225–263, 1996.

[15] C. Yuan and M. J. Druzdzel, "An importance sampling algorithm based on evidence pre-propagation," in *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 624–631.

[16] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design Test*, vol. 34, no. 2, pp. 60–68, April 2017.