A low-cost approach for determining the impact of Functional Approximation

Marcello Traiola[1], Alessandro Savino[2], Stefano Di Carlo[2]
[1]LIRMM - France
Email: {marcello.traiola}@lirmm.fr
[2]Politecnico di Torino - Italy
Email: {alessandro.savino | stefano.dicarlo}@polito.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 level 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 application 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 to 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.

The resiliency of a computation to approximation errors tightly depends on the application domain. Algorithms dealing with noisy real-world input data (e.g., sensor data processing, speech recognition, etc.), applications whose outputs are interpreted by humans (e.g., digital signal processing of images or audio), data analytics algorithms, web search algorithms and wireless communications are all examples of applications resilient to approximation [3], [4], [5].

Among the different AxC techniques [6], *functional approximation* aims at replacing a computational function with an equivalent 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 is rich of proposals for the implementation of approximate arithmetic operations [7], [8], [9], the selection of the components to approximate, and the selection of the best approximation techniques for an application remains a challenging problem. Indeed, while it is pretty much simple to quantify the impact of an AxC technique on the performance or power budget of an application, measuring the error introduced on the result of the computation is still an open challenge.

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) [10], [11], [12], [13], [14]. The comparison is achieved trough the adoption of an appropriate error metric (e.g., the structural similarity index [15] in the case of image processing). 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. The above process must be iterated until a good enough approximation is identified. The cost of this process depends on the amount of runs of the application required to reach the desired level of accuracy.

A different approach that analytically formalizes the error induced by C_{ax} and how it propagates in the application has been proposed in [16], [17]. 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. The approach is based on the *divide et impera* paradigm. 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) [18] modeling the approximation error propagation through the application's data flow. In the considered model, the network nodes represent the application data and components. Hence, they embed the error distribution computed during the component characterization. Eventually, by analyzing the network using the Bayesian inference theory, it is possible to estimate the error distribution of the application.

The advantages of the proposed approach, compared to the state-of-the-art, are: (i) the characterization of the different approximate components is done only one time, (ii) the Bayesian model of the application can be automatically constructed by syntactically analyzing the application's code without requiring a case by case analysis of the application.

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 bayesian proposed Bayesian model. Results are discussed in Section III. Finally, IV summarizes the main contributions and concludes the paper.

II. Methods

As introduced in Section I, the goal of the proposed approach is to develop a stochastic method able to assess the accuracy of an application exploiting a set of functional approximate components. Figure 1 sketches the global modeling flow that is composed of three main steps:

1) Component characterization,
2) Bayesian network construction, and
3) Approximation analysis.

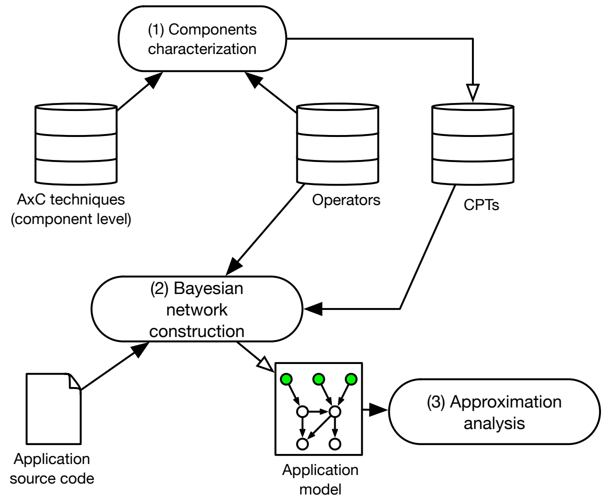
The first step characterizes a library of approximate components quantifying the error introduced by their approximation. Working with a Bayesian model, the goal of the characterization of a component is to build a Conditional Probability Table (CPT) able to model the conditional probability of having approximation errors at the output of the component, depending on the level of approximation of the inputs of the component. Despite its complexity, this activity must be performed only once for each considered AxC technique. Its results can then be reused several times.

The second step is application dependent. It aims at analyzing the application source code in order to build the Bayesian Network modeling the entire data flow of the application. The nodes represent the data and the operators, while edges model the relationships between them. Each node is associated to one of the CPTs computed during the first step.

Finally, in the third step the Bayesian model of the application can be used to analyze the approximation introduced at the application level.

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* [6] approach. Given a n bits data type, precision reduction works by reducing the size of the data type cutting its k less significant bits. Therefore, given a



(1) Components characterization
AxC techniques (component level)
Operators
CPTs
(2) Bayesian network construction
Application source code
Application model
(3) Approximation analysis

Fig. 1: Approximation estimation workflow.

number x expressed on n bits, its *approximated value* $\tilde{x}$ on $n-k$ bits can be expressed as:

$$\tilde{x} = x(n\ downto\ k) \times 2^k \tag{1}$$

From the definition of approximated value in equation (1), the introduced approximation error con be computed as:

$$E = |x - \tilde{x}| \tag{2}$$

This paper exploits the Worst Case Error (WCE) as a quality metric to assess the approximation of an operator:

$$WCE = \max_{\forall i} \left| x_{(i)} - \tilde{x}_{(i)} \right| \tag{3}$$

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 $\tilde{x}$:

$$Class(\tilde{x}) = \begin{cases} P\ \text{(Precise)} & if\ E = 0 \\ A\ \text{(Acceptable)} & if\ E \leq WCE \\ U\ \text{(Unacceptable)} & f\ E > WCE \end{cases} \tag{4}$$

As an example, let us apply precision reduction with $k = 2$. According to equation (2), the approximation error E can vary from 0 to 3, and the $WCE = 2^2 - 1 = 3$. For each possible value of E, the corresponding class of approximation computed according to equation (4) is reported in Table I.

TABLE I: Error Enumeration

E	**Class**
0	P
1	A
2	A
3	A

Starting from the information reported in Table I, and 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:

$$\begin{array}{l} P(\tilde{x}\ is\ P) = P(E=0) = \frac{1}{4} \\ P(\tilde{x}\ is\ A) = P(E \leq WCE) = \frac{3}{4} \\ P(\tilde{x}\ is\ U) = P(E > WCE) = 0 \end{array} \tag{5}$$

Equation 5 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. Indeed, the local error can be either masked or amplified, leading to P, A or U application results. 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 $\tilde{x_1}$ and $\tilde{x_2}$, the application of the sum operators generates two different results denoted as y_{pr} and y_{axc}:

$$y_{pr} = x_1 + x_2 \tag{6}$$

$$y_{axc} = \tilde{x_1} + \tilde{x_2} \tag{7}$$

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

$$E_+ = |y_{pr} - y_{axc}| \tag{8}$$

Table II 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 II. This can be formally expressed as:

$$\begin{array}{l} P(y_{axc}\ is\ P \mid (1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7)) \\ P(y_{axc}\ is\ A \mid (1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7)) \\ P(y_{axc}\ is\ U \mid (1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7)) \end{array} \tag{9}$$

The probabilities reported in equation (9) can be further simplified by considering that, if one of the two inputs is A, y_{axc} will be A as well. Similarly, if one of the two inputs is U, y_{axc} will be U. Moreover, conditions 2 and 3 are equivalent, as well as conditions 5 and 6. This reduces the problem to the classification of y_{axc} when both inputs are A (condition 4).

TABLE II: Input Events

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

To do this, let us enumerate all possible cases as shown in Table III. The first two columns report all possible errors for $\tilde{x_1}$ and $\tilde{x_2}$. The third column computes the error of y_{axc} as the sum of input errors of $\tilde{x_1}$ and $\tilde{x_2}$. The last column classifies the y_{axc} error accordingly to equation (5). Table III allows us to easily compute the following probabilities:

$$\begin{array}{l} P(y_{axc}\ is\ P \mid (4)) = 0 \\ P(y_{axc}\ is\ A \mid (4)) = 1/3 \\ P(y_{axc}\ is\ U \mid (4)) = 2/3 \end{array} \tag{10}$$

Eventually, Table IV 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 $\tilde{x_1}$ and $\tilde{x_1}$. 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 III: y_{axc} Classification

$E\tilde{x_1}$	$E\tilde{x_2}$	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

TABLE IV: CPT_+: Conditional Probability Table for the + Operator

x1		**P**			**A**			**U**		
x2		**P**	**A**	**U**	**P**	**A**	**U**	**P**	**A**	**U**
yaxc	**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

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, the application must be modeled in such a way to formally represent:

- all data and operators involved in the computation;
- all relations between data and operators;
- the mechanisms that propagate approximation errors through the data of the application.

To achieve this goal we model the application in the form of a Bayesian Network in which:

- nodes represent both data and operators;
- edges depict the dependency between data and operators;
- each node is associated with a CPT able to express how the approximation of the parents nodes impacts the outcome of a computation.

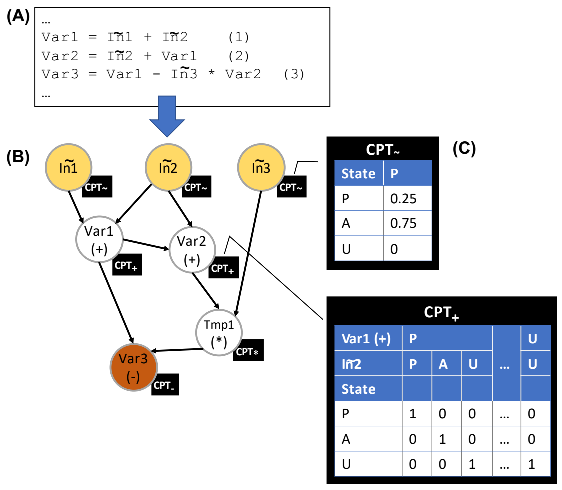
Once built, this model enables to analyze how errors are propagated from the root nodes down to the leaves representing the outcome of the application.

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.

In order to build the BN structure, thus identifying all required nodes and edges, we start by analyzing the application following its Data Dependence Graph (DDG) [19]. Conversely, the DDG of an application associates every data object of the application (e.g., variables, registers, etc.) to a node of the graph and depicts the dependencies among data (i.e., data are computed resorting to other data) by creating edges between nodes. Figure 2-B shows the DDG of the considered example. Whenever multiple operations are performed into a single instruction, intermediate nodes are created to properly model the intermediate results (e.g., tmp1 in Figure 2-B). Each node with a parent is associated with the information of the operator used to compute its value.

Yellow nodes in Figure 2-B are input nodes. They must be associated to a CPT indicating the marginal probability of the related data to be in one of the approximation classes defined in equation (4). If we apply precision reduction with k equal to 2 to the three data, according to equation (5) the nodes can be associated to the CPT reported in Figure 2-C. All intermediate nodes, represent computations involving different operators implemented by components analyzed during the characterization phase (see Section II-A). They can therefore be associated with the CPTs computed for each operator 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 IV.

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.



```
...
Var1 = Ĩn1 + Ĩn2     (1)
Var2 = Ĩn2 + Var1    (2)
Var3 = Var1 - Ĩn3 * Var2   (3)
...
```

$CPT_{\sim}$

State	P
P	0.25
A	0.75
U	0

CPT_{+}

Var1 (+)	P				U
Ĩn2	P	A	U	...	U
State					
P	1	0	0	...	0
A	0	1	0	...	0
U	0	0	1	...	1

Fig. 2: Bayesian Network example

The full BN creation process has been automated using a publicly available BN library and engine [20].

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 [21].

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 (5). This can be done by applying different update beliefs algorithms proposed in the literature. In particular, the library used to implement the proposed framework [20] provides two solvers: (1) the exact solver proposed by Lauritzen in [22] 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 [23] 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.

Let us first describe the experimental set-up by using a simple case study corresponding to a function computing the sum of the N elements of a vector, whose source code is reported in Listing 1.

```c
AxC_short_t AxC_VectorSum (AxC_short_t *vector, int
    N) {
  AxC_short_T sum;
  sum = vector[0];
  for (i = 1; i < N; i++) {
      sum += vector[i];
  }
}
```

Listing 1: VectorSum Source Code

The "AxC_short_T" data type implements the precision reduction technique described in Section II-A. For this case study, the bit reduction parameter k is set to 2, and therefore according to equation (3) WCE is equal to 3. For the sake of simplicity, the number N of elements in the vector for which we show the model is 4.

According to the concepts introduced in Section II-C, the application can be modeled by the Bayesian network depicted in Figure 3. The first 4 nodes (from $\widetilde{M}0$ to $\widetilde{M}3$) represent the approximated input values. From this point forward, the data are approximate. The network depicts the DDG of the application in which four sums are performed. The orange node corresponds to the sum variable representing the output of the application.

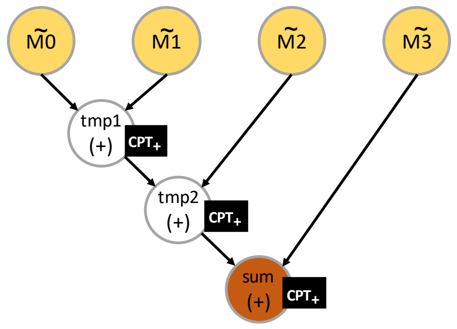


Fig. 3: Linear Vector Sum Bayesian Network

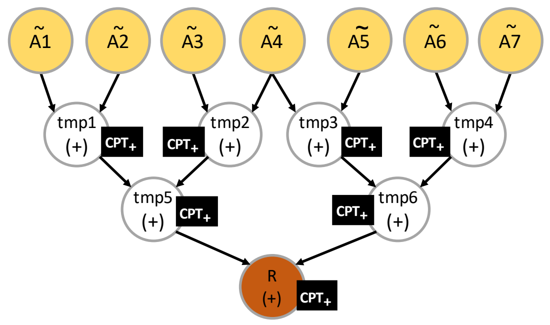


Fig. 4: Subsequent sums Bayesian Network

In order to show the accuracy of the prediction obtained through the proposed model, a precise and the approximate version of the VectorSum 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 (8).

Table V reports the obtained results in terms of probabilities to obtain a precise (P), acceptable (A) or un-acceptable (U) result. The first row reports the probabilities computed resorting to the proposed BN based model, while the second row reports the values computed by running the application several times (i.e., both the precise version and the approximate version). The last row quantifies the absolute error ($|BN - App.Run|$), expressed in percent points (pp), observed between the two evaluation methods. As shown in the table, the worst case error is about 4 pp for the A and U classes. Overall, this can be considered a good result given the significant difference in computational complexity for the two types of analysis.

TABLE V: Linear Vector Sum Validation

	P	**A**	**U**
BN	0.391%	17.578%	82.031%
App. Run	0.398%	13.473%	86.129%
Abs. Error	0.007pp	4.105pp	4.098pp

To show how the accuracy of the proposed approach is influenced by the target application, the same set of experiments (using the same parameters for the approximation) has been performed on different types of applications including:

- *Consecutive sums (CSs)*: a program performing a cascade of consecutive sums whose model is reported in Figure 4;
- *Matrix multiplication (MM)*: a function that is widely used in several computations including linear equations solvers. A two 4x4 matrices (MM4) multiplication of 8-bits elements has been performed.
- *Discrete cosine transform (DCT)*: a function important for several engineering applications (e.g., audio and images compression).

Table VIa summarizes the result of the analysis of the selected applications.

Looking at CSs and MM4, it can be immediately appreciated 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 [24] 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 VIb reports, for each application and for each evaluation technique, the required analysis time expressed in seconds. In both cases, the analysis has been executed on a personal computer with an Intel Core i7 7500U / 2.7 GHz and 8 GB of RAM at 1866 MHz and Ubuntu 17.10. 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:

$$Gain = \frac{App\,run\,time\;-\;BN\,time}{App\,run\,time} * 100 \qquad (11)$$

Moreover, Table VIc highlights the important gain in terms of 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.

As shown in Table VI, the proposed approach provides a significant gain in the analysis time (over 99%) with a very limited inaccuracy (up to 3.5pp).

IV. Conclusion

In this paper, we proposed a probabilistic approach able to analyze the impact of the application of functional approximation techniques to selected components of a complex software

TABLE VI: 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**
MM4	BN	0.4673%	0.002%	99.53%
	App. Run	0.0005	0	99.999%
	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	**99.99%**
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	256
1	
10,000,000	512

application. The proposed approach is able to estimate the effect of the approximation to the precision of the results of the computation. The prediction is performed by modeling the application data flow using a Bayesian Network model. Each operator handling approximated data is characterized only once. The outcome of the characterization is then reused every time the operator is used in an application. The proposed approach has been tested on a set of relevant software benchmarks. To assess the accuracy of the obtained estimations, the same benchmarks have been analyzed by comparing the precise and approximate application under a large amount of executions with random data. 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] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, “Accept: A programmer-guided compiler framework for practical approximate computing.”

[4] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” in *Test Symposium (ETS), 2013 18th IEEE European*. IEEE, 2013, pp. 1–6.

[5] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, “Data mining with big data,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, no. 1, pp. 97–107, 2014.

[6] 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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] K. Nepal, Y. Li, R. Bahar, and S. Reda, “Abacus: A technique for automated behavioral synthesis of approximate computing circuits,” in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 361.

[12] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, “Enerj: Approximate data types for safe and general low-power computation,” *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, 2011.

[13] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, “Precimonious: Tuning assistant for floating-point precision,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 27.

[14] 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.

[15] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[16] 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.

[17] R. Rocher, D. Menard, P. Scalart, and O. Sentieys, “Analytical approach for numerical accuracy estimation of fixed-point systems based on smooth operations,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2326–2339, Oct 2012.

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

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

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

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

[22] 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.

[23] 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.

[24] 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.