Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications

Probabilistic Estimation of the Application-Level Impact of Precision Scaling in Approximate Computing Applications

Marcello Traiola

LIRMM - University of Montpellier / CNRS - France

Alessandro Savino, Stefano Di Carlo

Politecnico di Torino - Italy

Abstract

The introduction of Approximate Computing (AxC) into software allows to achieve several optimizations such as performance improvement, energy reduction, and area reduction by paying with a decreased precision of the computed results. In order to find an effective balance in the application of approximate operators within the software, the approximation techniques proposed in the literature run several versions of the software with different configurations of the given technique. It is very clear how the common issue of this kind of approaches is the lack of a methodology to estimate the impact on the application-level accuracy without facing time intensive simulations.

In this paper, we evaluate the application-level accuracy by means of a Bayesian network modeling the propagation of the approximation across data. Once modeled the classes of accuracy required, the approach predicts the probability of the outcomes to reach each accuracy class. We performed experiments on a set of well-known target applications, both resilient and non-resilient. Specifically, as case study applications we used matrix multiplication, Discrete Cosine Transform (DCT), a Finite Impulse Response (FIR) filter and the image blending algorithm. Results show that the proposed approach is able to estimate the approximation error with good accuracy ($98 - 99\%$) and very low computation time (i.e., few seconds, in the worst case).

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

1. Introduction

Approximate computing (AxC) is an approach widely used in error-tolerant applications (i.e., they may accept results with a controlled level of inaccuracy) to trade-off accuracy for efficiency. The idea is to relax the accuracy of the results to gain benefits in other design parameters, e.g., power consumption, execution time, etc.

Approximate computing techniques try to maintain a controlled level of error by exploiting the resiliency of the computation to the approximation, which 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 analytic algorithms, web search algorithms and wireless communications algorithms are all examples of applications resilient to errors [1, 2, 3].

The literature is rich of publications demonstrating how this computing paradigm is effectively applied to: algorithms (e.g., to reduce the number of iterations) [4, 5], data (e.g., to reduce the storage burden in data-intensive

scenarios) [6, 7, 8] or hardware (e.g., to reduce area or delay by accepting internal inaccuracies) [2, 9, 10, 11]. Recently, Barrois et al. compared these approaches with the goal of improving awareness when choices are taken [12].

Among the different approximation techniques, this paper focuses on a class of approaches named *precision scaling*. These approaches work in general by changing the precision (bit-width) of input or intermediate operands of an operation in order to reduce storage/computing requirements. Despite the literature is rich of different techniques to perform precision scaling (the reader may refer to [13] for a review of these techniques), techniques to select the operators to approximate are still not well explored. The general approach is to rely on large and computation intensive campaigns of executions and profiling of the target application.

In order to provide design space exploration tools to support the design of approximate computing applications, Barbareschi et al. [14, 15] proposed IDEA, a tool to generate variants of a C/C++ application including different AxC techniques. The space of possible variants is analyzed using a branch and bound technique. Variants are generated manipulating the Clang version of the application. While this approach solves the problem of automating the application of different AxC techniques to a program, the problem of assessing the impact of the approximation applied to each generated variant still remains.

A totally different approach was proposed in [16]. The paper models the approximation error as a white noise and uses the noise linearity property to evaluate the output quality of DSP approximated blocks. However, as shown in [17], the white noise assumption is not always correct, since the error is dependent on the input. Moreover, the noise linearity assumption limits its application to approximate adders, only.

When considering a complex application, modeling the effect of an AxC operation on the application result requires to correlate its error propagation with the behavior of all other operations. This paper proposes a stochastic approach based on a Bayesian prediction model able to estimate the error affecting the result of a complex software application when precision scaling is applied to different portions of the computation. Bayesian networks (BN) [18] are a class of graphical stochastic models well suited to describe stochastic phenomena represented by a set of correlated random variables. The approximation is obtained by scaling the precision of hardware operations and data registers. The application is modeled in the form of a BN whose nodes represent data and operators, while arcs model the data-flow. Overall, the network models the error propagation along the data-flow of the application. To construct the model, the application is profiled only once to extract its data-flow. Then, the operators are characterized with required probabilities to be embedded in the model. Finally, the error distribution of the application can be quickly estimated exploiting the Bayesian inference theory.

An extensive set of experiments shows the proposed technique at work, highlighting its accuracy and low computation time when compared with state-of-the-art techniques.

The remainder of this paper is organized as follows: Section 2 overviews related publications in the field, Section 3 presents the proposed approach and Section 4 presents results of the application of the proposed technique on a set of case studies. Finally, Section 5 summarizes the main contributions and concludes the paper.

2. Background

When dealing with the introduction of the approximation into an application, two challenging problems must be faced: the selection of the portion of the application to approximate and the identification of the best combination of AxC techniques to use. In both cases, the goal is to satisfy the accuracy requirements of the final result and the investigation is usually very expensive.

Several publications address the above mentioned problems by profiling the executing of the application several times after applying different combinations of AxC operators. The results of the executions are then compared with those produced by a golden (i.e., approximation-free) implementation [19, 6, 20, 21, 22]. The comparison is done exploiting metrics that suit the specific application domain (e.g., the structural similarity index in the image processing domain [23]).

A first attempt to model the approximation resorting to statistic in order to reduce the impact of the analysis can be found in [24]. The paper proposes a circuit level model to systematically characterize different AxC circuits. Despite the impact of the use of these circuits in a full application is not analyzed, the paper introduces the concept of *error profile*, which opens a path to the use of statistical methods in AxC characterization.

A different approach is reported in [25, 26], where the authors try to formalize the error introduced by different implementations of AxC operators to model its propagation within the application. Despite this approach does not require several executions of the applications, the formal model is complex and the formalization is rather application dependent.

In general, the idea of studying the statistics of errors due to approximations is not completely new in literature. However, available papers consider only approximate hardware components such as adders and multipliers. Ayub et al. [27] proposed an algorithm to assess the error probability for a specific adder implementation, i.e., the Low Power Approximate Adder (LPAA). The proposed approach takes into account the different probability distributions of the input bits of each two-bits adder composing the full implementation. Also in this case, the paper does not consider a full application using the adder as a computing block.

Mazahir et al. proposed a methodology to evaluate the error distribution at the output of an approximate adder [11] and an approximate multiplier [17]. Both works improve previous approaches by considering not only single adders or multipliers but evaluating the error at the end of a cascade of elements of the same type. Nevertheless, both papers still do not consider cases in which different sequences of approximate operations are performed. Eventually, the same authors in [28] used a similar concept to provide run-time adaptation to the approximate arithmetic data-path.

More recently, Wu et al. [29] proposed a methodology to calculate the error statistics of block-based approximate adders. The approach enumerates all possible output deviations and then evaluates their occurrences. This is interesting for the error characterization of a single component. However, the work focuses only on approximate adders and it does not take into account precision scaling techniques. Moreover, the paper evaluates the error probability for a single operation, without considering cumulative effects of several operations. Therefore, it does not allow assessing the approximation impact on the overall application error.

Overall, solutions able to model how the error distribution at the output of an approximate component propagates while executing the operations composing a complex application are still not well explored.

The general idea of the model proposed in this paper was first introduced by the same authors in [30]. However, despite moving a first step toward constructing a Bayesian model able to estimate the error introduced by precision scaling operators in a complex program, the model proposed in [30] suffered from inaccuracies for those applications in which the same approximated data is reused in long cascades of operations. In fact, to reduce complexity, the model proposed in [30] assumed that a given operator is always characterized by the same error profile at its output, regardless its position in the computation flow. However, this assumption does not hold in real applications, and despite it introduces acceptable estimation errors in several cases, it can lead to strong inaccuracies for selected cases as will be discussed later in Section 4.

This paper relaxes this assumption adapting the description of each operator in the model to its depth level and thus overall improving the accuracy of the model. This comes at the cost of an additional effort in characterizing the operators, and an additional complexity of the model that slightly increase the computation time. However, the model still provides estimations with a computation time orders of magnitude smaller than performing multiple application executions.

A second strong limitation introduced by our preliminary model proposed in [30] was the introduction of a strong link between the number k of bits reduced when applying precision scaling operators and the maximum error accepted at the output of the application. In particular, the assumption was to accept as maximum error at the output of the application the Worst Case Error (WCE) of an operator ($WCE = 2^k - 1$). However, this assumption is too strong. If applied, most of the time the applications generate unacceptable results after few operations. In real cases, given an accepted approximation error α, the designer must be able to apply precision scaling techniques reducing the number of bits in such a way that $WCE = 2^k - 1 < \alpha$. This paper extends the model to take this into account.

3. Methods

Figure 1 sketches the workflow proposed in this paper to estimate the effect of precision scaling applied to different operations of a complex computation. The workflow is composed of three main phases:

1. *Model construction*: this phase is strictly application dependent. It analyzes the source code of the application in order to extract its data-flow. The data-flow of the application is than used to build a BN where nodes represent data and operators, and edges model the relationships among them. Both precise and approximate operators can be used in this model. The Conditional Probability Table (CPT) of each node models the propagation of the approximation error through the data-flow and is computed in the operators characterization phase.
2. *Operators characterization*: this phase builds a library of characterized operators (e.g., mathematical operators) to be used in the model construction phase. Each operator is characterized in order to quantify the error introduced at its outputs when applying different precision scaling factors at its inputs or intermediate

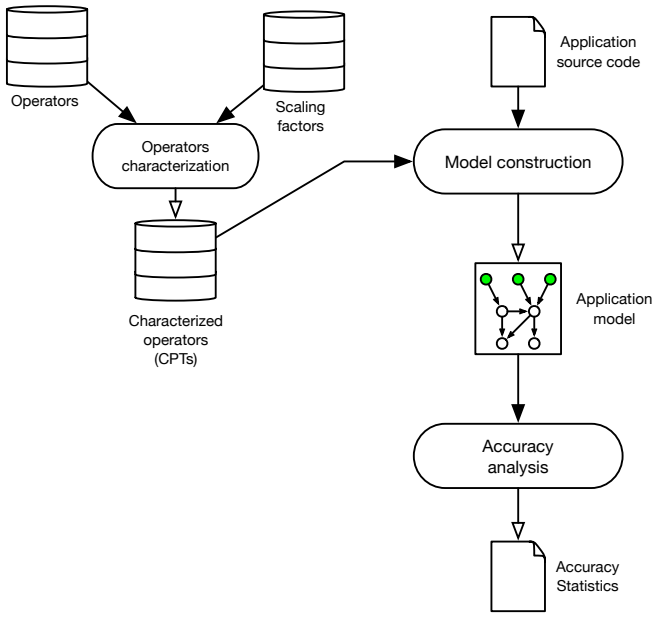


Figure 1: Approximation estimation workflow.

operands. This includes the characterization of precise operators working on approximated data. To be useful in a Bayesian model, each operator is characterized in terms of a CPT modeling the conditional probability of the operator to produce an approximation error at its output, depending on the level of approximation of its inputs. Despite its complexity, this activity must be performed only once for each considered AxC technique. All results can then be reused several times.

3. *Accuracy analysis*: this phase estimates the approximation introduced at the application level by the use of instructions exploiting hardware blocks introducing precision scaling. This is done by classifying the approximation into a set of accuracy classes. Bayesian inference is used here to quickly estimate how errors propagate to the output of the application [31]. This operation can be performed several times evaluating different combinations of approximated operators, thus providing a valuable tool to perform Design Space Exploration (DSE) in the AxC domain.

3.1. Model construction

The final accuracy of an application depends on the propagation of the error introduced by approximate operators at intermediate steps of the computation. To properly capture this error propagation, 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, the application is modeled in the form of a BN in which:

- nodes represent both data and operators;
- edges model 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 the computation (see Subsection 3.2)

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. The construction of the BN structure starts by profiling the target software application. The application is instrumented in order to extract a trace of all performed operations. Each operation in the trace is recorded together with the list of used inputs and outputs resources. The generated application trace is used as input of Algorithm 1 to generate the related BN model.

The algorithm processes one operation at the time (lines 1-24), first computing its inputs and outputs resources (lines 2-3). For each output resource (lines 6-9) a new node is created and the node is annotated with the operation. For each input resource (lines 11-20), the algorithm checks whether a new node must be instantiated or not. Since each resource can be used in the application multiple times, the BN can be populated with more than one node for each resource. If an input resource i was previously written (line 12), its associated node (n_i) is already present and can be connected to all output nodes (lines 17-19). If not (lines 13-16), a new node is created and then connected with the output nodes as in the previous case.

Finally, all output nodes are marked as the last write operation of the related resources (lines 21-23).

Algorithm 1: Bayesian Network generation algorithm

Data: AC = Application Trace

Result: BN = Algorithm's Bayesian Network

```
1  foreach operation ∈ AC do
2      in[] = inputs of the operation;
3      out[] = outputs of the operation;
4      op = operation_label;
5      outnodes[] = ∅ ;
6      foreach o ∈ out do
7          n_o = add_node_to_BN(o);
8          annotate_node(n_o, op);
9          outnodes[o] = n_o
10     end
11     foreach i ∈ in do
12         n_i = Get_Last_Written_Instance(i);
13         if n_i does not exist then
14             n_i = add_node_to_BN(i);
15             Set_Last_Written_Instance (i, n_i) ;
16         end
17         foreach n_o ∈ outnodes do
18             add_edge_to_BN(n_i, n_o);
19         end
20     end
21     foreach n_o ∈ outnodes do
22         Set_Last_Written_Instance (o, n_o)
23     end
24 end
```

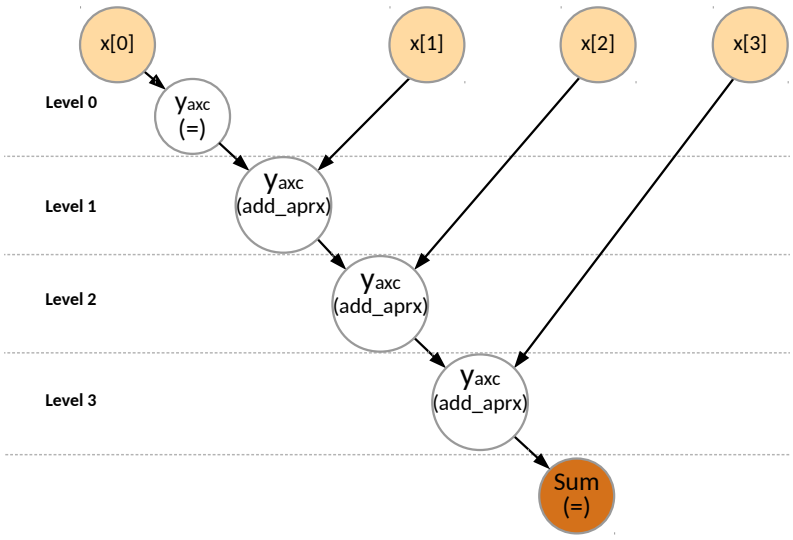
To better understand the model construction phase, let us consider the example depicted in Figure 2-a. The considered application is composed of a short sequence of instructions computing the sum of the values contained in a vector of four elements.

Figure 2-b shows the BN architecture for the considered example. For simplicity, the figure does not report those resources not involved in the approximation (e.g., the integer counter variable i). Whenever multiple operations are performed into a single instruction, intermediate nodes are created to properly model the intermediate results. Each node with a parent is associated with the information of the operator used to compute its value in order to keep track of all hardware operators required by the computation.

Figure 2-b depicts three node types. Yellow nodes are input nodes. All intermediate nodes (in white), represent

```
int x[4] = { random data };
//for all vector elements
int yaxc = x[0];
for (int i = 1; i < 4; i++) {
  yaxc = add_aprx(x[i], yaxc);
}
int Sum = yaxc;
```

(a)



(b)

Figure 2: Vector element sum code and associate BN model

computations involving different operators. Eventually, orange nodes identify the leaves of the network representing the output of the computation. They are the observation points from which the accuracy of the computation must be observed. The full BN creation process has been automated using a publicly available BN library and engine SMILE [32]. The SMILE library is a C++ library that supports the creation of complex BNs, and offers several already implemented solving algorithms. The developed tool extracts from an annotated code the list of approximate-related instructions and parses them to create the network resorting to the SMILE engine.

3.2. Operators characterization

As mentioned in Section 1, among the different functional approximation techniques presented in the literature, this paper focuses on the *precision scaling* approach [13]. This subsection shows how the impact of precision scaling on selected operators can be modeled in terms of CPTs in the proposed model. Each node of a BN is characterized in terms of the set of states it can assume. In the proposed model different states are associated to different accuracy levels. The characterization of an operator means computing the probability of the output of the operator to be in a certain state (accuracy level) conditioned to the state of its parent nodes.

In the proposed model, we define three different classes of accuracy (states) for each node: (i) *precise*, when the result is error free regardless of the precision scaling operation; *acceptable*, when the introduced error remains under a user-define threshold (α); *unacceptable*, when the introduced error cannot be tolerated. Given the aforementioned classes, the CPT of an operator expresses the probability that the outcome of an operator belongs to a specific class, conditioned to the state of its inputs. Looking at a full application, the final accuracy depends on the probabilities of its data to belong to one of the three accuracy classes.

3.2.1. Input nodes characterization

Given a general n bits data type, precision scaling works by reducing the size of the data type cutting its k less significant bits. Therefore, given a number x (which can be any of the inputs in Figure 2-b) expressed on n bits, its *approximated value* $\tilde{x}$ on $n-k$ bits can be expressed as:

$$\tilde{x} = \sum_{i=k}^{n-1} x^i \cdot 2^i \tag{1}$$

where x^i is the i-th bit of x. From the definition of approximated value in eq. (1), the introduced approximation error can be computed as:

$$E_x = |x - \tilde{x}| = \sum_{i=0}^{k-1} x^i \cdot 2^i \leq 2^k - 1 \tag{2}$$

The error E_x can be used to assess whether the introduced approximation is tolerable or not.

Let us denote with α the threshold indicating the maximum value of E_x that can be tolerated by the application domain. Based on its approximation error E_x, each data $\tilde{x}$ can be classified into three different accuracy classes defined as:

$$Class(\tilde{x}) = \begin{cases} P \text{ (Precise)} & if\ E_x = 0 \\ A \text{ (Acceptable)} & if\ E_x \leq \alpha \\ U \text{ (Unacceptable)} & if\ E_x > \alpha \end{cases} \tag{3}$$

The value of α must be selected based on a given quality metric for the application. However, following eq. (2), it should be grater than $2^k - 1$ to avoid the generation of data with unacceptable accuracy at the beginning of the application.

As an example, let us apply a 2 bits precision scaling ($k = 2$) on an integer number and let us choose $\alpha = 4$. According to eq. (2), the approximation error E_x can vary from 0 to 3. For each possible value of E_x, Table 1 reports the corresponding class of approximation computed according to eq. (3).

$\mathbf{E_x}$	**Class**
0	P
1	A
2	A
3	A

Table 1: Error Enumeration

Starting from the enumeration reported in Table 1, 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:

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

Eq. (4) allows us to quantify the probability of introducing an error on a single value by using precision scaling $k = 2$ and a threshold value $\alpha = 4$. This can be translated in the CPT reported in Table 2. This CPT can be used to characterize all inputs nodes (the Yellow ones in Figure 2) of the model. The Input Data CPT defines the input

Output	**P**	0.25
	A	0.75
	U	0

Table 2: Input Data CPT

data distribution over the three classes (P, A and U) due to precision reduction on original data. It is the only CPT that does not need to be conditioned on the status of its parent nodes.

3.2.2. Operation nodes characterization

Operation nodes propagate the error introduced on precision scaled data by performing a specific computation. Their characterization requires determining how this propagation takes place. Indeed, the local error can be either masked or amplified, changing the way the results of the application are distributed in the three accuracy classes (P, A or U).

In order to explain the approach to compute the CPT of a single operator, we refer to the example shown in Figure 2.

The characterization of the assignment operator (=) is trivial because the input data is replicated at the output without alterations. The probability of each output state ($OS \in \{P, A, U\}$) can be evaluated using the Bayes Theorem with respect to the probability of its parents ($inputs_i$) to be in a given state ($IS \in \{P, A, U\}$):

$$P(output = OS) = \sum_{\forall IS \in \{P,A,U\}} \left(P(output = OS \mid input_0 = IS_0, \cdots, input_n = IS_n) \cdot \prod_{\forall i \in \{0,\cdots,n\}} P\left(input_i = IS_i\right) \right) \tag{5}$$

This leads to the CPT shown in Table 3.

input		**P**	**A**	**U**
output	**P**	1	0	0
	A	0	1	0
	U	0	0	1

Table 3: Assignment operator CPT

Let us now focus on the characterization of the accumulation operator (+ =). Since the assignment operator does not alter the accuracy of the data, the accumulation operator can be treated as a sum operator (+). Given two precise numbers x_0 and x_1 and their approximated versions $\tilde{x_0}$ and $\tilde{x_1}$ (see eq. (1)), the application of the sum operator generates two different results denoted as y_{pr} and y_{axc}:

$$\begin{aligned} y_{axc} &= \tilde{x_0} + \tilde{x_1} \\ y_{pr} &= x_0 + x_1 = (\tilde{x_0} + E_{x_0}) + (\tilde{x_1} + E_{x_1}) \end{aligned}$$

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

$$E_{y_{axc}} = |y_{pr} - y_{axc}| = E_{x_0} + E_{x_1} \tag{6}$$

According to eq. (2), we can express eq. (6) based on the k parameter as follows:

$$E_{y_{axc}} = \sum_{i=0}^{k-1} (x_0^i + x_1^i) \cdot 2^i \leq 2 \cdot (2^k - 1) \tag{7}$$

Eq. (7) shows that the effect of a sum can double the error introduced by the precision scaling technique to the original data. Consequently, depending on the value of α, a sum operator can generate results in the U accuracy class.

In the + operator case, the CPT determines the conditional probability of the result (y_{axc}, in this case) to belong to one of the three aforementioned accuracy classes (P, A, U), given the classes of the operands ($\tilde{x_0}$ and $\tilde{x_1}$, in this case). Therefore, we need to evaluate the following equations:

$$P(y_{axc}\ is\ P) = P(E_{y_{axc}} = 0) = P(E_{x_0} + E_{x_1} = 0) \tag{8}$$

$$P(y_{axc}\ is\ A) = P(E_{y_{axc}} \leq \alpha) = P(E_{x_0} + E_{x_1} \leq \alpha) \tag{9}$$

$$P(y_{axc}\ is\ U) = P(E_{y_{axc}} > \alpha) = P(E_{x_0} + E_{x_1} > \alpha) \tag{10}$$

To evaluate eq. (8), we can simply observe that, since errors E_x are additive, the only possible combination of errors leading to sum = 0 is $E_{x_0} = E_{x_1} = 0$. All other equations require a further analysis. Eq. (10) is surely satisfied if at least one operand among $\tilde{x_0}$ and $\tilde{x_1}$ belongs to U, i.e., $E_{\tilde{x_0}} > \alpha$ or $E_{\tilde{x_1}} > \alpha$. Similarly, eq. (9) is surely satisfied when one among $\tilde{x_0}$ and $\tilde{x_1}$ belongs to A and the other to P. Both eq. (9) and eq. (10) also include a subset of cases when both $\tilde{x_0}$ and $\tilde{x_1}$ belong to the A class. In these cases, depending on the errors, $E_{x_0} + E_{x_1}$ can still be below α or raising above it. All combinations are summarized in Table 4.

Considering the case of $\alpha = 4$, Table 5 enumerates all possible acceptable accuracy cases.

Table 5 allows us to easily compute the following conditional probabilities by enumerating all possible cases:

$$\begin{aligned} &P(y_{axc}\ is\ P \mid \tilde{x_0}\ is\ A\ and\ \tilde{x_1}\ is\ A) = 0 \\ &P(y_{axc}\ is\ A \mid \tilde{x_0}\ is\ A\ and\ \tilde{x_1}\ is\ A) = 2/3 \\ &P(y_{axc}\ is\ U \mid \tilde{x_0}\ is\ A\ and\ \tilde{x_1}\ is\ A) = 1/3 \end{aligned} \tag{11}$$

Eventually, Table 6 reports the full CTP of the + operator.

x_0		**P**			**A**			**U**		
x_1		**P**	**A**	**U**	**P**	**A**	**U**	**P**	**A**	**U**
y_{axc}	**Possible States**	P	A	U	A	A if $E_{x_0} + E_{x_1} \leq \alpha$ U if $E_{x_0} + E_{x_1} > \alpha$	U	U	U	U

Table 4: Accuracy state of y_{axc} after a + operator depending on the accuracy of x_0 and x_1

$\tilde{\mathbf{x_0}}$ **class**	$\tilde{\mathbf{x_1}}$ **class**	$\mathbf{E_{x_0}}$	$\mathbf{E_{x_1}}$	$\mathbf{E_{y_{axc}}}$	$\mathbf{y_{axc}}$ **class**
A	A	1	1	2	A
A	A	1	2	3	A
A	A	2	1	3	A
A	A	1	3	4	A
A	A	2	2	4	A
A	A	3	1	4	A
A	A	2	3	5	U
A	A	3	2	5	U
A	A	3	3	6	U

Table 5: y_{axc} classification when both $\tilde{x_0}$ and $\tilde{x_1}$ belong to the A class, sorted by $E_{y_{axc}}$, according to eq. (6) and the resulting y_{axc} class according to eq. (4).

The first two rows list all possible combinations of classes for $\tilde{x_0}$ and $\tilde{x_1}$. The remaining rows provide the probability for the output classification based on the combination of inputs. In absolute terms, the maximum possible approximation error at this level is 6. This case corresponds to having $E_{x_0} = 3$ and $E_{x_1} = 3$.

According to eq. (6) and (7), if y_{axc} is used as input of a new computation, its error will be now $\leq 2 \cdot (2^k - 1)$, which is bigger then the maximum error introduced by an input operator when evaluating the previous CPT_+. Therefore, it is necessary to define the CPT for a second sum operator +. We refer to it as a level-two operator. Hence, by considering the second sum in Figure 2, we have to analyze the following equation:

$$y_{axc} = y_{axc} + \tilde{x_2} \tag{12}$$

First, we can apply the same observations we made for eq. (8), (9) and (10) with a wider set of error values

x_0		**P**			**A**			**U**		
x_1		**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	2/3	0	0	0	0
	U	0	0	1	0	1/3	1	1	1	1

Table 6: CPT_+: Conditional Probability Table for the + Operator

for one operator. Thus, we have to analyze the cases when both the operands (y_{axc} and $\tilde{x_2}$) belong to the A class. Table 7 shows what happens in those cases.

$\mathbf{y_{axc}}$ **class**	$\mathbf{\tilde{x_2}}$ **class**	$\mathbf{E_{y_{axc}}}$	$\mathbf{E\tilde{x_2}}$	**final** $\mathbf{E_{y_{axc}}}$	**final** $\mathbf{y_{axc}}$ **class**
A	A	1	1	2	A
A	A	1	2	3	A
A	A	2	1	3	A
A	A	1	3	4	A
A	A	2	2	4	A
A	A	3	1	4	A
A	A	2	3	5	U
A	A	3	2	5	U
A	A	4	1	5	U
A	A	3	3	6	U
A	A	4	2	6	U
A	A	4	3	7	U

Table 7: y_{axc} Classification when both $\tilde{x_0}$ and $\tilde{x_1}$ belong to the A class

According to Table 7, we can compute the following probabilities:

$$\begin{aligned} &P(y_{axc}\ is\ P \mid both\ operands\ are\ A) = 0 \\ &P(y_{axc}\ is\ A \mid both\ operands\ are\ A) = 1/2 \\ &P(y_{axc}\ is\ U \mid both\ operands\ are\ A) = 1/2 \end{aligned} \tag{13}$$

Finally, in Table 8, we report the CPT for the level-two sum of the analyzed application.

y_{axc}		**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	1/2	0	0	0	0
	U	0	0	1	0	1/2	1	1	1	1

Table 8: CPT_+: Conditional Probability Table for the level-two + Operator

In absolute terms, the maximum possible approximation error at this level is 9. This case corresponds to having $E_{y_{axc}} = 6$ (thus $E_{x_0} = 3$ and $E_{x_1} = 3$) and $E_{x_2} = 3$.

At this point, referring to Fig. 2, the CPT of the addiction reported at Level 2 is ready but the same operator recurs also a third time (the Level 3 accumulation). This means that the characterization must be done again, following the same approach, in order to be able to properly model the results of the following equation:

$$y_{axc} = y_{axc} + \tilde{x_3} \tag{14}$$

In general, the way an operator is characterized depends on: the level where input operands have been computed, the precision scaling cut value k and threshold α for the accuracy classification. This requires to characterize all necessary levels, increasing the operator characterization time. Nevertheless, as we will show in Section 4, this extra time still makes the model faster than running the application.

A similar analysis can be performed to characterize other operators such as difference $(-)$, multiplication $(*)$ and quotient $(/)$.

3.3. Accuracy analysis

Once the BN is built, Bayesian inference can be used to analyze the network in order to predict the level of accuracy at the output of the application [31].

The prediction can be made computing the posterior probability of the leaves of the network (orange nodes in Figure 2-b) to be in one of the three accuracy approximation classes defined in eq. (4). The posterior probability that a node y of the BN is in one of the three accuracy classes (i.e., P, A and U) can be computed resorting to the Bayes Theorem reported in eq. 5).

Depending on the BN structure the eq. (5) can be a recursive equation since the probability of each parent (referred as $P(input_i = IS_i)$ in the equation) may denote a conditional probability itself. The equation takes into account the states of all operators from the bottom of the model up to the top. It can be solved by applying different update beliefs algorithms proposed in the literature. In particular, the library used to implement the proposed framework already provides two solvers: (1) the exact solver proposed by Lauritzen in [33] 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 [34] 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 level of the application, thus enabling to quickly explore different design solutions.

Moreover, we can also resort to the BN capability of updating all beliefs of the network based on the knowledge (evidence) that selected nodes of the network are in a given state. This allows us to evaluate the effect of selected functional approximations on the application. Such approach enables to explore how the presence or the absence of the approximation, i.e., an operator that produces only non-approximate (*evidence* set on P state) values or an operator with approximate output (*evidence* set on A state), impacts the final output. This allows us to perform a very simple and preliminary design exploration that can help taking actions on the program implementation and usage of approximate operators.

4. Experimental Results

4.1. Experimental setup

The capability of the proposed model was evaluated on five software applications with different complexity (use cases). They were implemented in C/C++ to easily simulate hardware precision scaling implementations using different operators. The list of use cases is as follows:

- *Consecutive sums (CSs)*: a simple program performing a cascade of three consecutive sums (see Section 3.1) on integer numbers.

- *Matrix multiplication (MM)*: a function that is widely used in several computations including linear equations solvers. Two variants of this use case working on 2x2 and 4x4 matrices (MM2 and MM4) of 8- and 16-bits elements were considered.

- *Discrete cosine transform (DCT)*: a transformation that can be considered similar to the discrete Fourier transform (DFT), but using only real numbers. The function has important applications in science and engineering, such as audio (e.g., lossy compression of audio) and image (e.g., JPEG compression algorithm) processing.

- *Finite impulse response (FIR)*: a convolution-based filter that can be implemented either in software or hardware. It was applied to an image processing application (box blur).

- *Image Blending (IB)*: an algorithm that blends two input images of the same size. The value of each pixel in the output image is a linear combination of the corresponding pixels in the input images. They are 24-bit per pixel images, where each byte represents a chromatic component.

Two sets of experiments were performed. The first set aimed at assessing the accuracy of the estimations performed using the proposed model and the analysis time required to obtain the estimations. All five use cases were analyzed by considering different configurations of the precision reduction factor (k) and the accuracy threshold (α), defined based on the characteristics of the use case.

The considered range of values for the accuracy threshold α was selected based on the bit width of the output values produced by the use case. For all use cases the minimum considered acceptable error was 1 and the maximum was $2^{\frac{bit-width}{2}}$. Table 9 summarizes this parameter for all use cases. Please note that for MM2 and MM4 this parameter indicates the bit width of each element composing the output matrix. Similarly, for DCT, FIR and IB, the parameter represents the width of each single pixel component (R,G,B) of the resulting image.

For each value of α, different precision scaling factors k were considered. Starting from $k = 1$, for each use case, the maximum k was defined as the one generating unacceptable results (U class) for the higher value of α. The ration behind this setup is to stop the analysis when, even with a very relaxed value of α, the approximation is so high to produce unacceptable results.

The second set of experiments reproduced the experimental campaign proposed in [30] to fairly compare the capability of the proposed model with those of our previously presented approach.

In all experiments, we built the BN model of the selected use cases and estimated the probabilities of each output value to be in the three accuracy classes defined in eq. (3).

To evaluate the accuracy of the estimations, we compared the obtained results with those computed resorting to a campaign of executions of the applications. This required the implementation of both an approximation free (golden) and an approximate version of each use case. For every execution, a random workload was generated and provided to both the golden and the approximate version of the application in order to estimate the error on each

output value and classify the output according to eq. (3). The number of samples required to have meaningful results was defined using the randomized block design approach [35] and it is reported in Table 9. To prevent the number of samples to become too high to be analyzed on time, we added a threshold to evaluate the deviation in the error estimation between two consecutive samples, stopping the analysis as soon as this deviation was smaller than 0.001% for all classes. Table 9 also reports the mathematical operators involved in each case study together with the overall number of instances per case.

Case Study	**CSs**	**MM2**		**MM4**		**DCT**	**FIR**	**IB**
		8-bit	16-bit	8-bit	16-bit			
Output bit-width	11	16	32	16	32	8	8	8
α **range**	$[2^0 - 2^6]$	$[2^0 - 2^8]$	$[2^0 - 2^{16}]$	$[2^0 - 2^8]$	$[2^0 - 2^{16}]$	$[2^0 - 2^4]$	$[2^0 - 2^4]$	$[2^0 - 2^4]$
Number of samples	10^8	$2 \cdot 10^8$	$2 \cdot 10^8$	$2 \cdot 10^8$	$2 \cdot 10^8$	10^8	10^8	$5.5 \cdot 10^7$
Involved Operators	4 Add	4 Add 8 Mul	16 Add 64 Mul	4 Add 8 Mul	16 Add 64 Mul	12 Add 20 Mul 2 Sqrt* 16 Cos*	1536 Add 1728 Mul 192 Div	12288 Add 24576 Mul

* by means of a pre-caculated LUT

Table 9: Experimental setup

4.2. Accuracy of the model

To evaluate the accuracy of the proposed approach, for each output value of a use case, we computed the absolute error between the probabilities estimated using the Bayesian model (denoted as $\hat{P}(y\ in\ C)$) and the probabilities computed by executing campaigns of executions of the application ($P(y\ in\ C)$):

$$e_C(y) = \hat{P}(y\ in\ C) - P_\alpha(y\ in\ C),\ \text{where } C \in \{P, A, U\} \tag{15}$$

Table 10 reports for each benchmark, for each accuracy class (see eq. (3)) and for each value of k, the mean and the standard deviation of the absolute error defined in eq. (15) computed over all outputs and values of α. Results show that, on average, the error is negligible with a very small variance, thus confirming that the proposed model can be used as an instrument for the designer to qualitatively explore the resiliency of an application to precision scaling without executing it several times. This is favored by the fast analysis time of the proposed approach, which is one of the strength of this technique. Few experiments report a 0 value both in mean and standard deviation: those cases are due to the fact that the application did not produced outputs belonging to the specific class, while the model correctly predicted a 0 probability for the same class.

4.3. Analysis time

Table 11 shows the average time expressed in seconds required to analyze each use case. Reported times represent average values computed among all experiments performed for each use case. The table compares the analysis time

required to analyze the use cases using campaign of executions (denoted with App) and the time required to perform the same operation using the proposed model (denoted with BN). The percentage of reduction has been calculated as:

$$\% \text{ Reduction} = \frac{\text{App Time} - \text{BN time}}{\text{App Time}} \cdot 100 \tag{16}$$

The model execution times are split into: (i) Operators characterization, (ii) BN generation and (iii) BN analysis. This enables to appreciate the low computation time required to perform the BN generation and the final analysis that are the only steps that must be executed several times for the different applications and for different parameters. Results were collected on a workstation equipped with an Intel Core i7 7500U / 2.7 GHz, 8 GB RAM at 1866 MHz and running Linux Ubuntu 18.04.

Reported times clearly show how the proposed approach is faster compared to precise evaluation through repeated application runs.

4.4. Use cases comparison

Given the accuracy of the proposed model and the low analysis time, a very interesting aspect is the possibility of using it to quickly perform design exploration when applying precision reduction to an application. Figures 3, 4, 5, 6, 7 and 8 show, for each use case, the average classification projection of the output values (i.e., the average probability of the output values to be in one of the accuracy classes) for all considered values of α and k.

Figure 3 shows that CS is quite resilient to precision scaling, enabling to increase k up to four bits when $\alpha = 64$. This is due to the fact that this use case does not include any multiplication. In fact, multiplications quickly amplify the approximation error,thus leading to unacceptable results even for small values of k. This is for example the case for MM2 and MM4 in Figures 4 and 5.

Concerning both MM2 and MM4, for the sake clarity we did not plot some the of the α values for the 16-bit versions. As already mentioned, due to the massive use of multiplications, MM is not resilient to precision scaling. Indeed, already for $k = 1$ the outputs contain unacceptable values for all considered α values.

Looking at the results' projection of the DCT (Figure 6) and the FIR (Figure 7), again the two applications are quite resilient to precision scaling. This result is particularly unexpected for FIR since it includes approximate multiplications. However, it is fair to highlight that the FIR maximum acceptable error ($2^4 = 16$) was lower than the CS one ($2^6 = 64$).

Finally, the Image Blending case study shows interesting resilience to the precision scaling (Figure 8). In particular, for $k < 3$ this use case is the one with higher probability to generate precise results.

This analysis is just an example of how the proposed model can be efficiently explored to analyze different design options for a given application.

4.5. Comparison with previous works

Eventually, the proposed model was compared in terms of accuracy and computation time with the preliminary model presented in [30]. In order to provide a fair comparison, we considered the same set of applications (i.e., CSs, MM2, MM4 and DCT) and the same experimental setup (i.e., $k = 2$ and $\alpha = 2^k - 1 = 3$) presented in [30]. Table 12 reports both error and execution time for the two considered models.

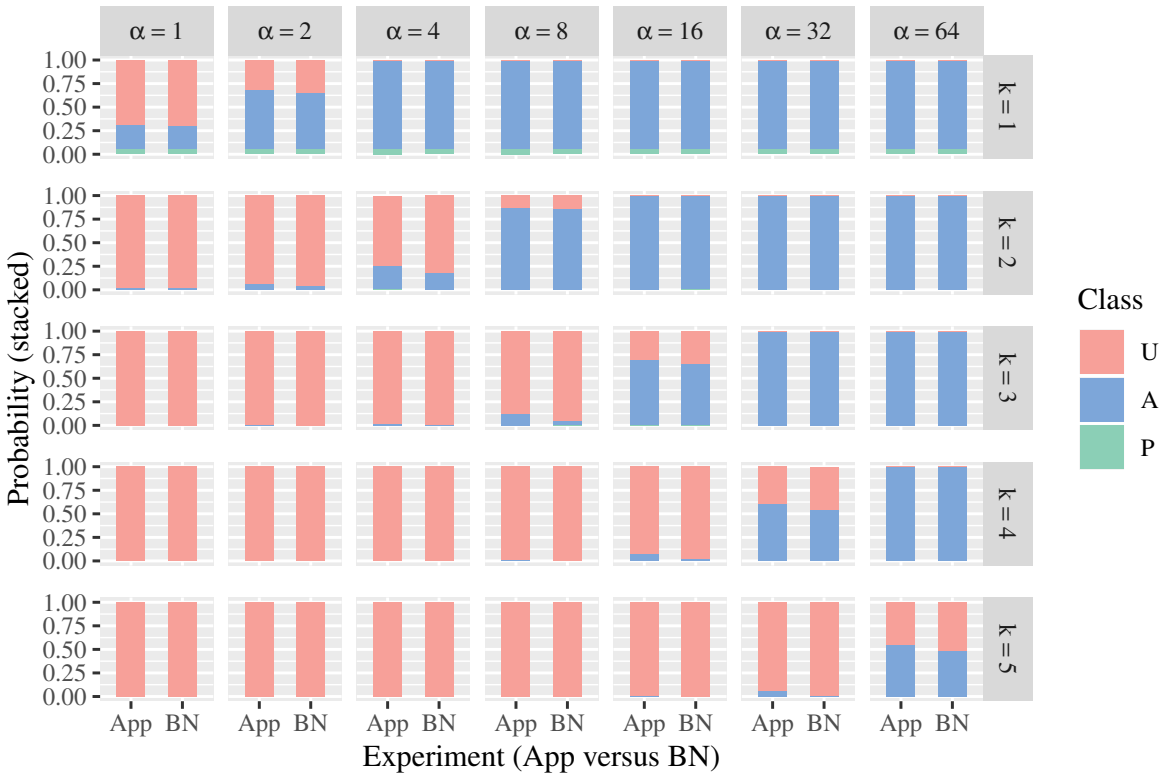


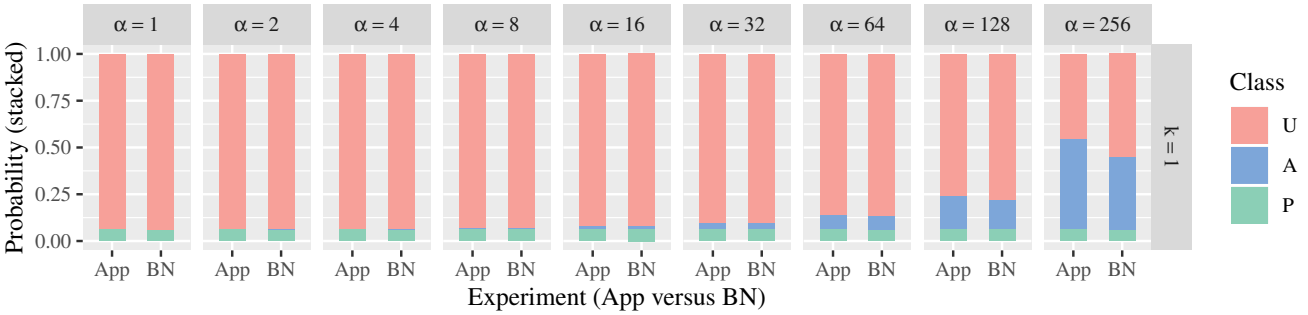
Figure 3: CS results projection

Overall, the results presented in Table 12 show that the proposed model is more accurate w.r.t. the model presented in [30]. Few cases report a slightly higher error on one class that is always compensated by improving the accuracy on all other classes. The gain can be particularly appreciated for complex applications such as the DCT, in which the combinations of different operations are less trivial than other uses cases.

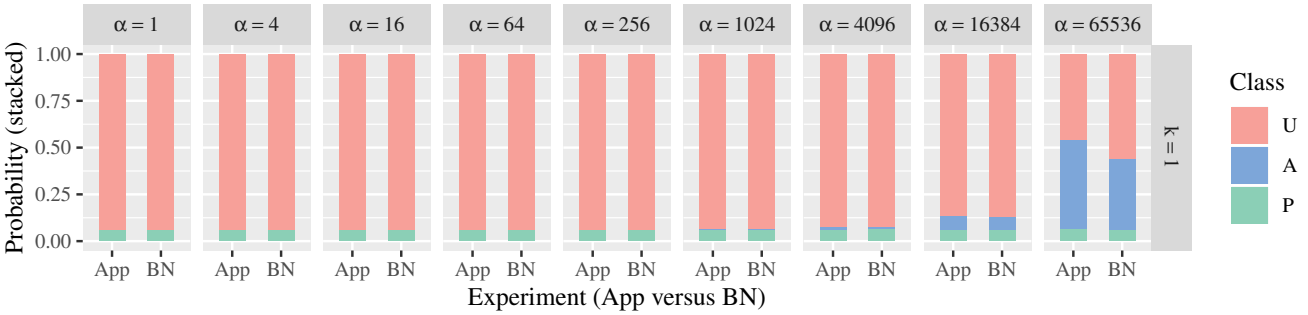
Interestingly enough, the improved precision does not significantly impact the execution time, with a time overhead always lower than < 0.1 seconds.

5. Conclusion

The paper proposes a probabilistic model able to analyze the impact of precision scaling techniques on the accuracy of a software application. The approach is based on the characterization of a library of approximated operators used to build a Bayesian model able to predict how errors introduced by precision scaling propagate to the outcome of the computation. This is done without executing the application several times. This enables to obtain fast predictions that can be used at design time to take informed decisions on how to apply precision scaling approximation techniques. The validation was performed on a set of relevant software case studies. To assess the accuracy of the obtained estimations, the same case studies were analyzed by comparing the precise and approximated application under a large amount of executions with random data. Experimental results show that the proposed model enables qualitative evaluations of the accuracy of the application with a remarkable gain in terms of computation time. This makes the model a candidate instrument to perform design space exploration in



(a) 8-bit version with maximum $\alpha = 2^8 = 256$



(b) 16-bit version with maximum $\alpha = 2^{16} = 65536$

Figure 4: MM2 results projection

the approximate computing domain. There are several interesting research directions that can be further pursued to improve the model. New functional approximation techniques can be included to expand the investigation capabilities of the model. Further, it may be interesting to model techniques that approximate the control flow, such as loop perforation.

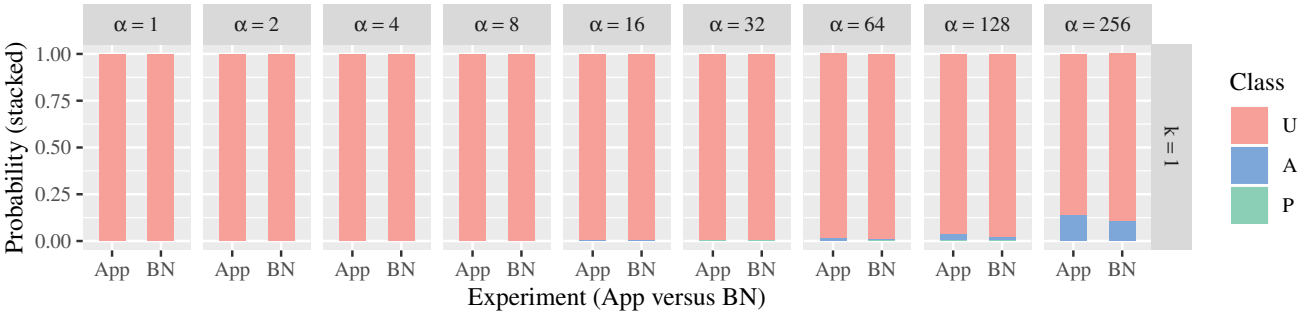
References

[1] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, M. Oskin, Accept: A programmer-guided compiler framework for practical approximate computing, University of Washington Technical Report UW-CSE-15-01 1 (2) (2015).

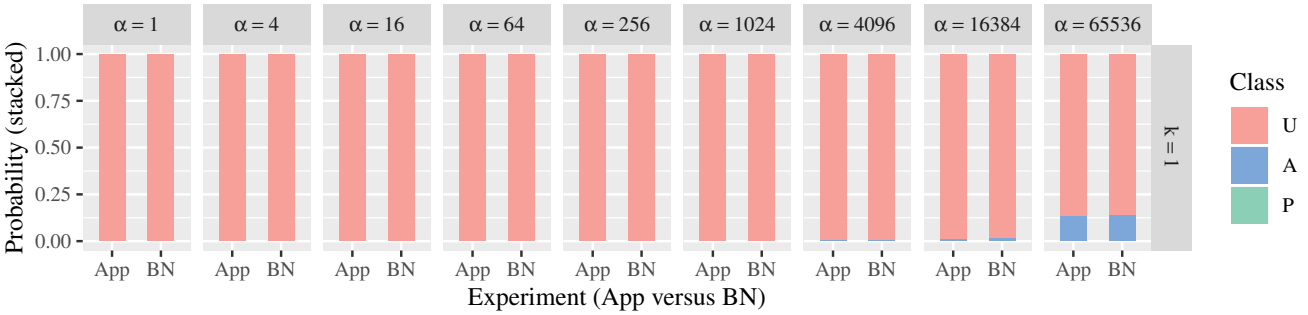
[2] J. Han, M. Orshansky, Approximate computing: An emerging paradigm for energy-efficient design, in: Test Symposium (ETS), 2013 18th IEEE European, IEEE, 2013, pp. 1–6 (2013).

[3] X. Wu, X. Zhu, G. Wu, W. Ding, Data mining with big data, IEEE Transactions on Knowledge and Data Engineering 26 (1) (2014) 97–107 (Jan 2014). doi:10.1109/TKDE.2013.109.

[4] V. K. Chippa, S. T. Chakradhar, K. Roy, 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 (2013).



(a) 8-bit version with maximum $\alpha = 2^8 = 256$



(b) 16-bit version with maximum $\alpha = 2^{16} = 65536$

Figure 5: MM4 results projection

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

[6] K. Nepal, Y. Li, R. Bahar, 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 (2014).

[7] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, A. Raghunathan, Approximate storage for energy efficient spintronic memories, in: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015, pp. 1–6 (June 2015). doi:10.1145/2744769.2744799.

[8] B. Barrois, O. Sentieys, Customizing fixed-point and floating-point arithmetic — a case study in k-means clustering, in: 2017 IEEE International Workshop on Signal Processing Systems (SiPS), 2017, pp. 1–6 (Oct 2017). doi:10.1109/SiPS.2017.8109980.

[9] F. S. Snigdha, D. Sengupta, J. Hu, S. S. Sapatnekar, Optimal design of jpeg hardware under the approximate computing paradigm, in: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2016, pp. 1–6 (June 2016). doi:10.1145/2897937.2898057.

[10] Q. Fan, S. S. Sapatnekar, D. J. Lilja, Cost-quality trade-offs of approximate memory repair mechanisms for

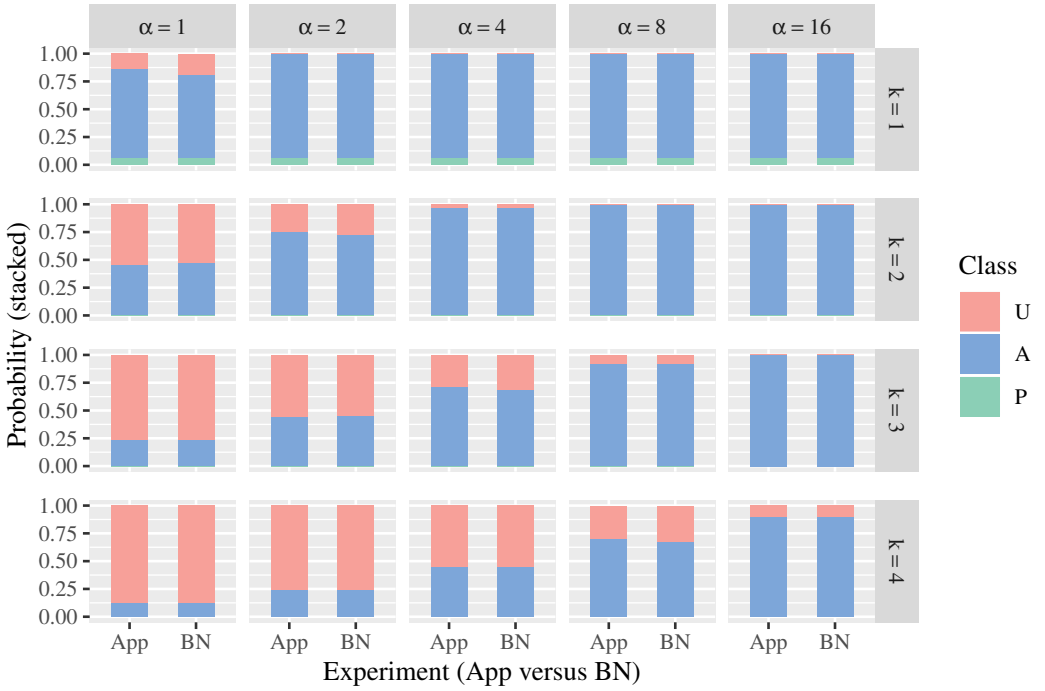


Figure 6: DCT results projection

image data, in: 2017 18th International Symposium on Quality Electronic Design (ISQED), 2017, pp. 438–444 (March 2017). doi:10.1109/ISQED.2017.7918355.

[11] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, J. Henkel, Probabilistic error modeling for approximate adders, IEEE Transactions on Computers 66 (3) (2017) 515–530 (March 2017). doi:10.1109/TC.2016.2605382.

[12] B. Barrois, O. Sentieys, D. Menard, The hidden cost of functional approximation against careful data sizing — a case study, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, 2017, pp. 181–186 (March 2017). doi:10.23919/DATE.2017.7926979.

[13] S. Mittal, A survey of techniques for approximate computing, ACM Comput. Surv. 48 (4) (2016) 62:1–62:33 (Mar. 2016). doi:10.1145/2893356.
URL http://doi.acm.org/10.1145/2893356

[14] M. Barbareschi, F. Iannucci, A. Mazzeo, A pruning technique for b&b based design exploration of approximate computing variants, in: VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on, IEEE, 2016, pp. 707–712 (2016).

[15] M. Barbareschi, F. Iannucci, A. Mazzeo, An extendible design exploration tool for supporting approximate computing techniques, in: 2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), IEEE, 2016, pp. 1–6 (2016).

[16] M. Pashaeifar, M. Kamal, A. Afzali-Kusha, M. Pedram, A theoretical framework for quality estimation and

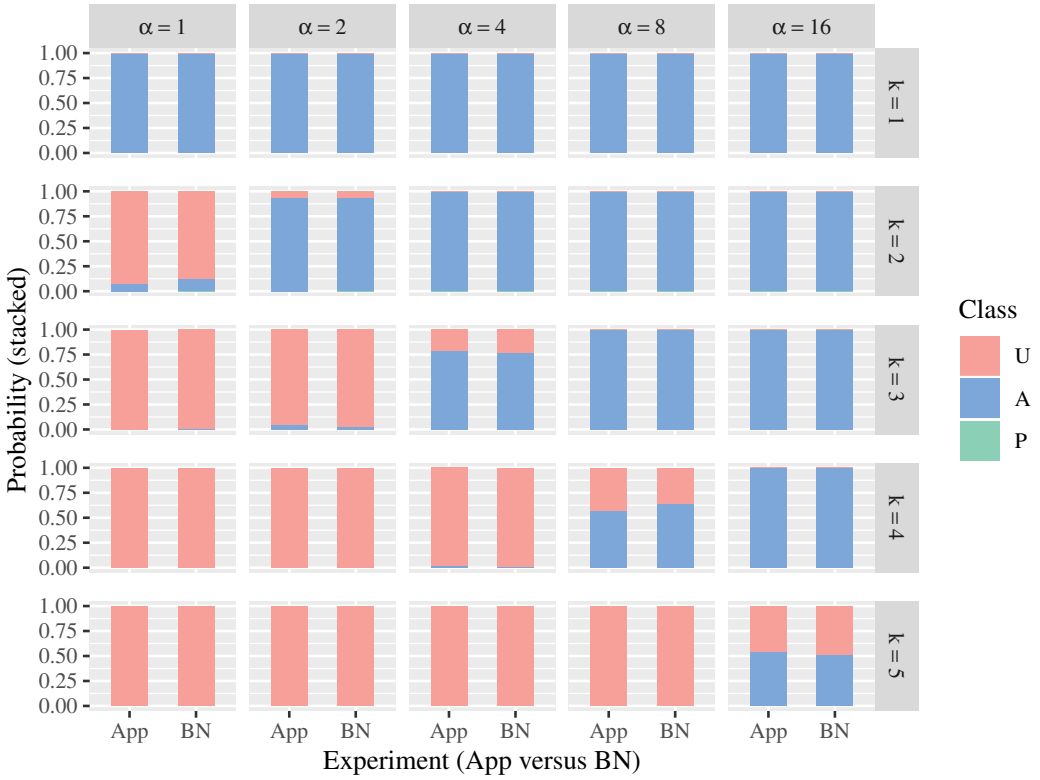


Figure 7: FIR filter results projection

optimization of dsp applications using low-power approximate adders, IEEE Transactions on Circuits and Systems I: Regular Papers 66 (1) (2019) 327–340 (Jan 2019). doi:10.1109/TCSI.2018.2856757.

[17] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, Probabilistic error analysis of approximate recursive multipliers, IEEE Transactions on Computers 66 (11) (2017) 1982–1990 (Nov 2017). doi:10.1109/TC.2017.2709542.

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

[19] S. Lee, L. K. John, A. Gerstlauer, High-level synthesis of approximate hardware under joint precision and voltage scaling, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, 2017, pp. 187–192 (March 2017). doi:10.23919/DATE.2017.7926980.

[20] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, D. Grossman, Enerj: Approximate data types for safe and general low-power computation, ACM SIGPLAN Notices 46 (6) (2011) 164–174 (2011).

[21] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, 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 (2013).

[22] M. Barbareschi, F. Iannucci, A. Mazzeo, Automatic design space exploration of approximate algorithms for big

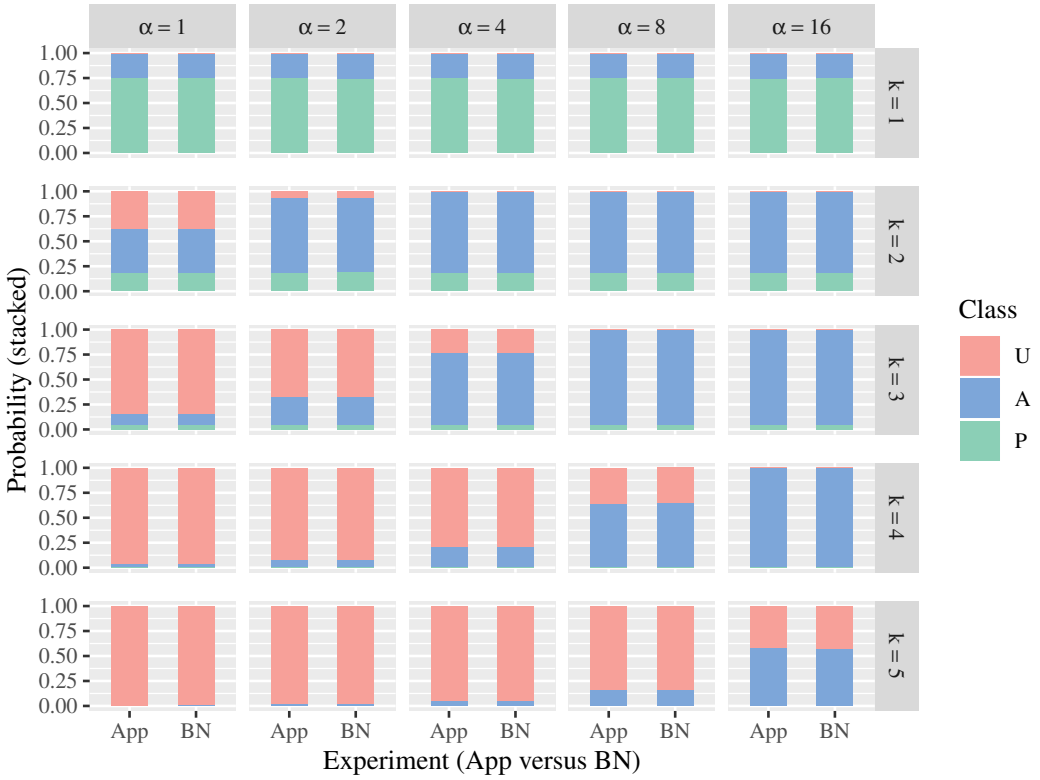


Figure 8: IB results projection

data applications, in: IEEE International Conference on Advanced Information Networking and Applications (AINA-2016). IEEE, 2016 (2016).

[23] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Transactions on Image Processing 13 (4) (2004) 600–612 (April 2004). doi:10.1109/TIP.2003.819861.

[24] R. Venkatesan, A. Agarwal, K. Roy, A. Raghunathan, Macaco: Modeling and analysis of circuits for approximate computing, in: 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2011, pp. 667–673 (Nov 2011). doi:10.1109/ICCAD.2011.6105401.

[25] K. N. Parashar, D. Menard, O. Sentieys, Accelerated performance evaluation of fixed-point systems with un-smooth operations, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 33 (4) (2014) 599–612 (April 2014). doi:10.1109/TCAD.2013.2292510.

[26] R. Rocher, D. Menard, P. Scalart, 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 59 (10) (2012) 2326–2339 (Oct 2012). doi:10.1109/TCSI.2012.2188938.

[27] M. K. Ayub, O. Hasan, M. Shafique, Statistical error analysis for low power approximate adders, in: 2017

54th ACM/EDAC/IEEE Design Automation Conference (DAC), 2017, pp. 1–6 (June 2017). doi:10.1145/3061639.3062319.

[28] S. Mazahir, O. Hasan, M. Shafique, Adaptive approximate computing in arithmetic datapaths, IEEE Design Test 35 (4) (2018) 65–74 (Aug 2018). doi:10.1109/MDAT.2017.2772874.

[29] Y. Wu, Y. Li, X. Ge, Y. Gao, W. Qian, An efficient method for calculating the error statistics of block-based approximate adders, IEEE Transactions on Computers 68 (1) (2019) 21–38 (Jan 2019). doi:10.1109/TC.2018.2859960.

[30] M. Traiola, A. Savino, M. Barbareschi, S. D. Carlo, A. Bosio, Predicting the impact of functional approximation: from component- to application-level, in: 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), 2018, pp. 61–64 (July 2018). doi:10.1109/IOLTS.2018.8474072.

[31] G. E. Box, G. C. Tiao, Bayesian inference in statistical analysis, Vol. 40, John Wiley & Sons, 2011 (2011).

[32] BayesFusion, LLC. SMILE Engine [online].

[33] C. Huang, A. Darwiche, Inference in belief networks: A procedural guide, International journal of approximate reasoning 15 (3) (1996) 225–263 (1996).

[34] C. Yuan, 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 (2002).

[35] S. E. Maxwell, Randomized block designs, Encyclopedia of Statistics in Behavioral Science (Oct 2005). doi:10.1002/0470013192.bsa535.
URL http://dx.doi.org/10.1002/0470013192.bsa535

Case Study	k	P		A		U	
		MAE[1]	SD[2]	MAE[1]	SD[2]	MAE[1]	SD[2]
CSs	1	$1.05{\cdot}10^{-03}$	$1.08{\cdot}10^{-03}$	$5.03{\cdot}10^{-03}$	$9.41{\cdot}10^{-03}$	$4.24{\cdot}10^{-03}$	$9.80{\cdot}10^{-03}$
CSs	2	$1.96{\cdot}10^{-04}$	$1.71{\cdot}10^{-04}$	$1.56{\cdot}10^{-02}$	$2.96{\cdot}10^{-02}$	$1.56{\cdot}10^{-02}$	$2.97{\cdot}10^{-02}$
CSs	3	$6.14{\cdot}10^{-05}$	$3.85{\cdot}10^{-05}$	$1.74{\cdot}10^{-02}$	$2.65{\cdot}10^{-02}$	$1.75{\cdot}10^{-02}$	$2.65{\cdot}10^{-02}$
CSs	4	$2.14{\cdot}10^{-05}$	$1.57{\cdot}10^{-05}$	$1.77{\cdot}10^{-02}$	$2.78{\cdot}10^{-02}$	$1.77{\cdot}10^{-02}$	$2.77{\cdot}10^{-02}$
CSs	5	$4.29{\cdot}10^{-06}$	$5.35{\cdot}10^{-06}$	$1.76{\cdot}10^{-02}$	$2.93{\cdot}10^{-02}$	$1.76{\cdot}10^{-02}$	$2.93{\cdot}10^{-02}$
MM2 8-bit	1	$1.39{\cdot}10^{-03}$	$1.16{\cdot}10^{-03}$	$1.43{\cdot}10^{-02}$	$3.22{\cdot}10^{-02}$	$1.54{\cdot}10^{-02}$	$3.19{\cdot}10^{-02}$
MM2 16-bit	1	$4.80{\cdot}10^{-04}$	$3.90{\cdot}10^{-04}$	$1.23{\cdot}10^{-02}$	$3.43{\cdot}10^{-02}$	$1.24{\cdot}10^{-02}$	$3.41{\cdot}10^{-02}$
MM4 8-bit	1	$1.76{\cdot}10^{-04}$	$1.19{\cdot}10^{-04}$	$6.18{\cdot}10^{-03}$	$1.23{\cdot}10^{-02}$	$6.34{\cdot}10^{-03}$	$1.23{\cdot}10^{-02}$
MM4 16-bit	1	$2.91{\cdot}10^{-04}$	$2.59{\cdot}10^{-04}$	$1.08{\cdot}10^{-03}$	$2.09{\cdot}10^{-03}$	$1.35{\cdot}10^{-03}$	$2.05{\cdot}10^{-03}$
DCT	1	$3.44{\cdot}10^{-04}$	$1.69{\cdot}10^{-04}$	$9.27{\cdot}10^{-03}$	$2.00{\cdot}10^{-02}$	$9.11{\cdot}10^{-03}$	$2.04{\cdot}10^{-02}$
DCT	2	$1.01{\cdot}10^{-04}$	$1.02{\cdot}10^{-04}$	$7.24{\cdot}10^{-03}$	$1.03{\cdot}10^{-02}$	$7.15{\cdot}10^{-03}$	$1.03{\cdot}10^{-02}$
DCT	3	$1.68{\cdot}10^{-05}$	$1.59{\cdot}10^{-05}$	$6.33{\cdot}10^{-03}$	$1.11{\cdot}10^{-02}$	$6.32{\cdot}10^{-03}$	$1.11{\cdot}10^{-02}$
DCT	4	$3.73{\cdot}10^{-06}$	$1.98{\cdot}10^{-06}$	$6.06{\cdot}10^{-03}$	$1.14{\cdot}10^{-02}$	$6.06{\cdot}10^{-03}$	$1.14{\cdot}10^{-02}$
FIR	1	$3.90{\cdot}10^{-03}$	$1.23{\cdot}10^{-04}$	$3.90{\cdot}10^{-03}$	$1.23{\cdot}10^{-04}$	0	0
FIR	2	$1.00{\cdot}10^{-05}$	$7.07{\cdot}10^{-06}$	$1.00{\cdot}10^{-02}$	$2.01{\cdot}10^{-02}$	$1.00{\cdot}10^{-02}$	$2.01{\cdot}10^{-02}$
FIR	3	0	0	$8.35{\cdot}10^{-03}$	$1.14{\cdot}10^{-02}$	$8.35{\cdot}10^{-03}$	$1.14{\cdot}10^{-02}$
FIR	4	0	0	$1.46{\cdot}10^{-02}$	$2.89{\cdot}10^{-02}$	$1.46{\cdot}10^{-02}$	$2.89{\cdot}10^{-02}$
FIR	5	0	0	$7.39{\cdot}10^{-03}$	$1.62{\cdot}10^{-02}$	$7.39{\cdot}10^{-03}$	$1.62{\cdot}10^{-02}$
IB	1	$1.08{\cdot}10^{-03}$	$7.56{\cdot}10^{-04}$	$1.08{\cdot}10^{-03}$	$7.56{\cdot}10^{-04}$	0	0
IB	2	$1.00{\cdot}10^{-03}$	$6.25{\cdot}10^{-04}$	$1.03{\cdot}10^{-03}$	$8.31{\cdot}10^{-04}$	$7.26{\cdot}10^{-04}$	$1.37{\cdot}10^{-03}$
IB	3	$3.86{\cdot}10^{-04}$	$4.29{\cdot}10^{-04}$	$9.65{\cdot}10^{-04}$	$7.44{\cdot}10^{-04}$	$7.87{\cdot}10^{-04}$	$1.00{\cdot}10^{-03}$
IB	4	$1.04{\cdot}10^{-04}$	$1.33{\cdot}10^{-04}$	$9.58{\cdot}10^{-04}$	$1.13{\cdot}10^{-03}$	$1.04{\cdot}10^{-03}$	$1.15{\cdot}10^{-03}$
IB	5	$3.34{\cdot}10^{-04}$	$1.85{\cdot}10^{-04}$	$2.12{\cdot}10^{-03}$	$3.20{\cdot}10^{-03}$	$2.30{\cdot}10^{-03}$	$3.32{\cdot}10^{-03}$

[1]MAE = Mean Absolute Error [2]SD = Standard deviation

Table 10: Accuracy analysis. For each benchmark and value of k, the table reports the mean and the standard deviation of the absolute error for each class, computed among all outputs and considered values of α. When both MAE and SD are 0, no differences between the BN predictions and the actual application values were observed. This happened only when both the application and the BN did not produce any output belonging to the related class.

Case study	App Time (s)	BN Time (s)				% Reduction
		Operator Characterization	Generation	Analysis	Total	
CS	4.79	0.00310	0.00033	0.00002	0.00345	99.93%
MM2 8-bit	30.24	0.02500	0.00043	0.00004	0.02547	99.92%
MM2 16-bit	31.7	1.56940	0.00051	0.00003	1.56994	95.05%
MM2 8-bit	54.8	0.02280	0.00043	0.00004	0.02327	99.96%
MM2 16-bit	56.3	4.12880	0.00051	0.00003	4.12934	92.67%
DCT	130.7	0.05360	0.00261	0.30990	0.36611	99.72%
FIR	101.6	0.00560	0.00067	0.03961	0.04588	99.95%
IB	120.1	0.00270	0.00038	0.01398	0.01706	99.99%

Table 11: Average execution time comparison for all use cases.

Case study		Classes			BN Exec. Time (s)
		P	A	U	
CSs	Abs. Error [30]	$1.00{\cdot}10^{-06}$	$9.10{\cdot}10^{-03}$	$9.10{\cdot}10^{-03}$	0.002
	Abs. Error in current work	$6.20{\cdot}10^{-05}$	$1.27{\cdot}10^{-03}$	$1.20{\cdot}10^{-03}$	0.004
	Δ	$-6.10{\cdot}10^{-05}$	$7.83{\cdot}10^{-03}$	$7.90{\cdot}10^{-03}$	-0.002
MM2	Abs. Error [30]	$4.67{\cdot}10^{-03}$	$2.00{\cdot}10^{-05}$	$4.69{\cdot}10^{-03}$	0.004
	Abs. Error in current work	$7.99{\cdot}10^{-04}$	$1.40{\cdot}10^{-06}$	$8.00{\cdot}10^{-04}$	0.007
	Δ	$3.87{\cdot}10^{-03}$	$1.86{\cdot}10^{-05}$	$3.89{\cdot}10^{-03}$	-0.003
MM4	Abs. Error [30]	$2.00{\cdot}10^{-05}$	$2.00{\cdot}10^{-08}$	$2.00{\cdot}10^{-05}$	0.005
	Abs. Error	$9.00{\cdot}10^{-07}$	$2.00{\cdot}10^{-10}$	$1.00{\cdot}10^{-06}$	0.019
	Δ	$1.91{\cdot}10^{-05}$	$1.98{\cdot}10^{-08}$	$1.90{\cdot}10^{-05}$	-0.014
DCT	Abs. Error [30]	$3.44{\cdot}10^{-02}$	$3.42{\cdot}10^{-02}$	$3.00{\cdot}10^{-05}$	0.297
	Abs. Error in current work	$1.56{\cdot}10^{-04}$	$8.14{\cdot}10^{-03}$	$8.30{\cdot}10^{-03}$	0.35
	Δ	$3.42{\cdot}10^{-02}$	$2.61{\cdot}10^{-02}$	$-8.27{\cdot}10^{-03}$	-0.053

Table 12: Comparison between current model and the one presented in [30]