Compiler-Driven Error Analysis for Designing Approximate Accelerators

Jorge Castro-Godínez[1,2], Sven Esser[1], Muhammad Shafique[3], Santiago Pagani[1,4], Jörg Henkel[1]

[1]Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
[2]School of Electronics Engineering, Instituto Tecnológico de Costa Rica, Costa Rica
[3]Institute of Computer Engineering, Vienna University of Technology (TU Wien), Austria
[4]ARM Ltd., Cambridge, United Kingdom

Corresponding Author: jorge.castro-godinez@kit.edu

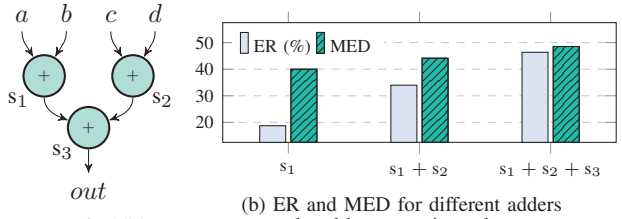
***Abstract*—Approximate Computing has emerged as a design paradigm suitable to applications with inherent error resilience. This paradigm aims to reduce the associated computing costs (such as execution time, area, or energy) of exact calculations by reducing the quality of their results. Several approximate arithmetic circuits have been proposed, which can be used to implement hardware blocks such as approximate accelerators. However, to satisfy quality constraints in these accelerators, it is imperative to assess how the errors introduced by approximate circuits propagate through other exact and approximate computations, and finally accumulate at the output. This is, in particular, crucial to enable high-level synthesis of approximate accelerators. This work proposes a compiler-driven error analysis methodology to evaluate the behavior of errors generated from approximate adders in the design of approximate accelerators. We present CEDA, a tool to perform a static analysis of the error propagation. This tool uses #pragma-based annotated C/C++ source code as input. With these annotations, exact additions are replaced by approximate ones during the code analysis to estimate the error at the output. The error estimations produced by our tool are comparable to those obtained through simulations.**

***Index Terms*—Approximate computing, error analysis, design tools.**

I. Introduction

Power and energy efficiency have become major design concerns for modern computing systems. This has motivated the coming forth of new design techniques looking to reduce the power and energy requirements. In recent years, Approximate Computing has emerged as a novel design paradigm relevant to applications with inherent resilience to errors [1]. By accepting *good enough* results caused by imprecise calculations, e.g., in image and video processing where the human perception plays a major role, the computational *quality* (accuracy of results) is traded-off to reduce the required computational *effort* (execution time, area, power, or energy). This paradigm can be applied at different abstractions layers, from the circuit layer up to the application layer, each one presenting different benefits and challenges [2].

Recent works have proposed to exploit inherent resilience to errors in applications by using approximate accelerators [1], i.e., specialized hardware blocks that execute highly frequent and error-tolerant sections of an application, while the rest of the program is executed by a host processor. One approach proposed is to implement these accelerators as neural networks and take advantage of the approximate nature of the results produced by this computational model [3], [4]. Another approach proposes to use approximate arithmetic circuits, such as approximate adders and multipliers, to replace exact calculations in hardware accelerators [2], [5].



(a) Tree of additions

(b) ER and MED for different adders replaced by approximated ones

Fig. 1: Motivation example for up to three additions that can be replaced by approximate ones. The ER and MED increase with the number of approximate adders.

Despite the tolerance to errors, approximate accelerators must limit the output error to satisfy quality constraints, whether defined by the application, the developer, or even the end-user. The use of approximate arithmetic circuits in accelerators imposes the challenge of understanding how the error introduced by each approximate circuit behaves and propagates through other exact and approximate computations, and finally how it accumulates at the output. This is visible in the following motivational case study.

Motivation Example: Consider three additions arranged as presented in Figure 1a. Each adder (s_1, s_2, s_3) can be replaced by an approximate adder that presents an Error Rate (ER) of 0.19 and a Mean Error Distance (MED) of 40.0. The ER is defined as the likelihood of erroneous outputs, while the MED is the weighted mean value of all error distances in an approximate adder [6], being the error distance (ED) the difference between an accurate and an approximate computation.

Figure 1b depicts the increment of ER and MED as the number of additions performed approximated grows. When just s_1 is replaced by the approximate adder, the error metrics at *out* correspond to the ones produced by such adder (ER $= 0.19$, MED $= 40.0$). When all adders are replaced, the ER and MED at *out* reach 0.46 and 48.5, respectively. Estimating these values make possible the selection of adequate approximate circuits to comply with a defined error tolerance. For instance, if the additions presented can tolerate up to 45.0 as MED, then replacing s_1 and s_2 for the mentioned approximate adder can satisfy this constraint, due to these two approximate additions produce a MED of 44.1, as depicted in Figure 1b. These error estimations are required to guide the design space exploration of approximate accelerators, and thus select the approximate circuits that meet the error tolerance of a design while, for instance, reducing delay or power consumption.

Concept Overview and Novel Contributions: In this paper, we present a novel compiler-driven methodology for estimat-

978-3-9819263-0-9/DATE18/©2018 EDAA

ing the error propagation to the outputs of an approximate accelerator. Since the design of accelerators is carried out in conjunction with the source code of the application, our methodology proposes to use the source code of the function to be accelerated as an input. A custom defined `#pragma` directive is defined to annotate the code and to indicate which accurate operations are replaced by approximate ones. A modified compiler is proposed in order to handle these annotations and to append metadata to the intermediate representation of the code. This information is later used to statically analyze the code, using error propagation models previously determined, to obtain an estimation of the error at the output (see Fig. 6).

The main novel contributions of this work are:

- A model of the error distribution propagation. Using as a base a set of heuristics to model error propagation for individual calculations, we define models to estimate the propagation of error distributions represented as a probability distribution. These rules consider the propagation of errors through other arithmetic computations.
- An algorithm to estimate the error propagation in approximate accelerator designs. Based on Depth-First Search (DFS) algorithm, our algorithm uses the defined propagation models to estimate the error at the output(s).
- A tool capable of performing static analysis of annotated source code, corresponding to a function to be accelerated in an approximate manner, and to assess the error propagated to the output.

Implementation and Open Source Contribution: We integrate our methodology into a tool that we called CEDA (Compiler-driven Error Analysis for Designing Approximate Accelerators). Our tool provides the possibility to explore approximate accelerator designs on an early stage and to determine their error characteristics even before the actual hardware is implemented.

CEDA currently employs existing state-of-the-art approximate adders as source of approximations. This is because addition is the most common operation in many applications, and there are several approximate adders proposed in the literature, whose error characteristics are analytically modeled. However, by including the required error models and error propagation rules, this tool can be easily extended to integrate other approximate circuits such as multipliers.

CEDA is developed as open source and it is available for download at https://git.scc.kit.edu/CES/CEDA.

II. Related Work

Previous works have proposed different approaches to estimate the output error at component and accelerator level. Statistical and analytic models have been presented to characterize the error produced by approximate circuits such as adders [7], [8] and multipliers [9]. Given that the focus of our work is on approximate accelerators, these models [7–9] can serve as an input to our work.

For approximate accelerators, some existing works employ exhaustive simulations or Monte Carlo simulations [10], while others define an input data set [11] or use interval and affine arithmetic [12], [13]. Capturing error statistics through simulations produce accurate characterizations, but they are time-consuming, particularly when considering diverse configurations for an arithmetic circuit. For the other cases, the precision of the characterization is limited because the error metrics are sensitive to the input distribution. Other works use analytical formulations [14], [15] but limiting the applicability to a small set of approximate circuits. Once the error characterization for an arithmetic circuit is achieved, the output error in a hardware block, involving exact and approximate calculations, is determined by defining error propagation rules [10], [12], [13], or by performing regression analysis [14] or curve fitting [11] to establish prediction equations. For the latter approaches, in case that the design or the circuits used changes, it is necessary to obtain new prediction equations by performing again the proposed methods.

Most of these works require simulations either to determine the error characteristics of single arithmetic circuits or to estimate final error propagation. This is a limitation to explore a vast design space when multiple configurations of approximate circuits are considered, for instance, while performing high-level synthesis of approximate accelerators. Our work proposes an analytical estimation of the error propagation not requiring simulations.

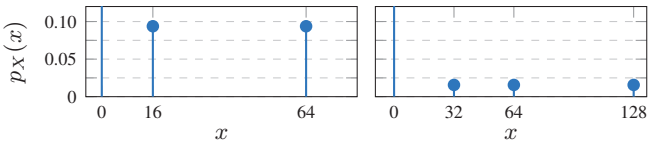
III. Error Representation

Existing works have proposed different approximate arithmetic circuits, mainly approximate adders [16] and multipliers [17]. Various metrics have been used and proposed to model and represent the error produced by such components, e.g, ER, MED and ED [6]. In this work, we focus our attention on using approximate adders as a source of approximations because addition is the most common operation, particularly high-performance approximate adders, and we use Probability Mass Function (PMF) to represent the error distribution of such circuits.

Approximate adders: Existing approximate adders can be classified as low-power or high-performance. The former replaces accurate 1-bit additions for simplified versions of a full adder, and uses them to compute the less significant bits of the addition, reducing the power consumption of the adder. The latter approximates the results by breaking the carry propagation chain of the addition and it uses multiple sub-adders to produce the summation result, thus reducing the latency of the adder but increasing the required area.

Most of the proposed high-performance approximate adders, such as ACA-I [18], ETAII [19] and GDA [20], can be implemented as a configuration of GeAr [21], which helps to reduce the representation of diverse high-performance approximate adders to a single design. GeAr performs an n-bit addition using multiple sub-adders of smaller size. The most significant r-bits of the sub-adders are considered as resultant bits and used in the result, while the remaining p-bits, the previous bits, are used to estimate the carry propagation to upper bits. The error distribution of GeAr adder, i.e., the value and probability of errors produced by one configuration of the adder, have been analytically modeled, which eliminates the need of time-consuming simulations [7]. In this work, we focus on this type of approximate adders as source of errors. However, this work can be extended to other approximate adders and arithmetic circuits.

Probability Mass Function (PMF): In this work, we use PMF as the main form to represent the error distribution of approximate adders. PMF represents the error characteristics of approximate circuits in greater detail, and other commonly used metrics, such as MED and ER, can be obtained from it. For instance, Figure 2 presents two PMFs for an ETAII and a GeAr adder. PMF denotes the probability $\mathbf{P}$ of a discrete



(a) ETAII (ER = 0.1875) (b) GeAr (ER = 0.0468)

Fig. 2: PMF for two 8-bit high-performance approximate adders. ETAII [19] is configured with 2 sub-adders, while GeAr [21] is configured to 1 bit for result and 4 bits for prediction. The values in the x-axis represent the ED, i.e., the error values produced by each approximate adder. The ER denotes the total probability of error for such adder. The probability of having no errors, i.e., $p_X(0)$, is $1 - \text{ER}$ (out of the axis in the figure). This applies for all error distributions presented in this work.

random variable X, to be equal to a determined value x. This is expressed as $p_X(x) = \mathbf{P}(X = x)$. For an approximate adder, it allows the representation of ED and its probability of occurrence. In each case, the ED and its probability depend on the adder configuration and its bit width; similarly for ER. The probabilistic analysis presented in [7] is capable to analytically estimate the PMF for a set of high-performance approximate adders.

IV. Propagation Modeling

Error propagation estimation requires models that represent the interaction between errors produced by approximate adders and other accurate and approximate computations. From these models, a set of propagation rules or patterns can be established. We performed exhaustive computations, using results produced by approximate adders, to set heuristics for the estimation the error propagation. Considering the diagram in Figure 3, the inputs a, b, c and d are error-free values added by two approximate adders, whose error distribution is known. The error in x and y depends on whether the addition produces an error or not, which depends on their input values and the adder configuration. Here, Op is replaced by accurate operations: addition, subtraction, multiplication, shift left, or shift right. Also, an approximate addition can be considered for Op.

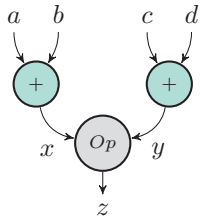


Fig. 3: Results x and y, produced by approximate adders, are used as inputs to other accurate and approximate computations to model the error at the output z.

By analyzing the results of the individual computations for every replacement of Op, we have derived a set of heuristics to estimate the error propagation. These heuristics are presented in Table I. For the addition, the output error (e_z) is defined as the addition of the input error of both addends (e_x and e_y). In the case that Op is replaced by an approximate addition, the output error considers the error introduced by the approximate adder itself (e_+), along with the error of the addends (to the current state of this work, all operations are considered as unsigned). For the subtraction, the output error is the difference between the error of both operands. As in a regular subtraction, the order of the operands affect the output error, and this can lead to errors that can be reduced and even eliminated.

TABLE I: Error propagation rules, heuristically determined from exhaustive experimentation.

Operation	e_z
Addition	$e_x + e_y + e_+$
Subtraction	$e_x - e_y$
Multiplication	$x \cdot e_y + y \cdot e_x + e_x \cdot e_y$
Shift Left	$e_x \cdot 2^y$
Shift Right	$e_x / 2^y$

The multiplication presents a peculiarity. The output error has an intrinsic dependency on the input value, as can be noticed from the rule formulation in Table I.[1] This means that the error propagated through an accurate multiplication does not just depend on the input error, as the addition, but also on the values to be multiplied. To avoid this dependency, in our work, we approximate the value of x and y as:

$$x = x_{\text{max}} - e_x \qquad y = y_{\text{max}} - e_y \tag{1}$$

where x_{max} and y_{max} are the maximum value that x and y can take, which is delimited by the bit width of the input values. So, for a 8-bit input x, $x_{\text{max}} = 256$. This approximation is useful as a guardband to consider an upper error bound in the estimation. In case that the multiplication is between a value with errors and a constant, the propagated error is the product of the error and the constant value.

For shift left and shift right, the error propagates through the computation experimenting increment or decrement of its magnitude, respectively. For the case in Figure 3, y represents the number of bits that x needs to be shifted. Thus, the input error of x is multiplied or divided by 2 to the power of y, if a shift left or right is required.

These models determined for single error values can be applied to error distributions represented by a PMF. Considering the PMFs of the errors produced by approximate adders as distributions of independent discrete and random variables, the addition of two PMFs is the convolution of their probability distributions. If $p_Z(z) = \mathbf{P}(Z = z)$ is the error distribution after an accurate addition (as for Op in Figure 3), and $Z = X + Y$, then

$$p_Z(z) = \sum_y p_X(z - y) p_Y(y) \tag{2}$$

being $p_X(x)$ and $p_Y(y)$ the PMFs for x and y. This formulation also applies to the subtraction. However, before performing the convolution, it is required to invert the ED values for the subtrahend PMF. For the product of two PMFs, it is calculated as the joint probability of $p_X(x)$ and $p_Y(y)$, $Z = XY$, for two independent discrete random variables. Then,

$$\begin{aligned} p_Z(z) = p_{X,Y}(x, y) &= \sum_{x \in X, y \in Y} \mathbf{P}(X = x, Y = y) \\ &= \sum_{x \in X, y \in Y} \mathbf{P}(X = x)\mathbf{P}(Y = y) \end{aligned} \tag{3}$$

considering all pairs of values (x, y) that X and Y take. For shift left and right, the ED of the input error distribution is scaled by a factor of 2^y, as previously described. So,

[1]The propagation rules obtained for addition and multiplication confirm previous presented equations in [10].

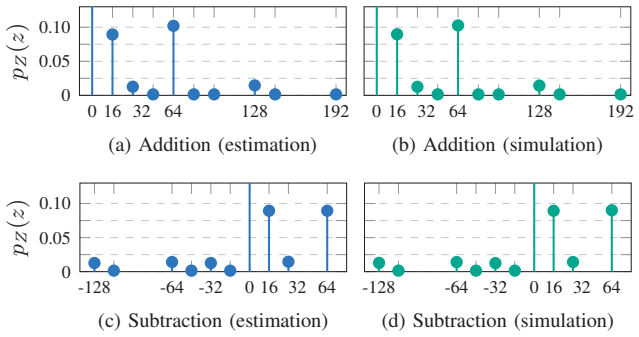


Fig. 4: Error propagation estimation after an accurate addition (a) and a subtraction (c). The error distribution of the approximate adders in Figure 2 are used in this example as inputs x and y.

$$p_Z(z) = 2^y \cdot p_X(x) \tag{4}$$

$$p_Z(z) = \frac{p_X(x)}{2^y} \tag{5}$$

represent the error propagation after shift left and shift right, respectively. In these shift operations, the ER for each ED does not change. When the multiplication is between a constant value and one with errors, the effect in the error propagated is similar as for shift left.

Figure 4 depicts the PMFs for the error propagated after an accurate addition and subtraction. For this example, the PMFs from the previously presented approximate adders (see Fig. 2) are used as x and y. The results obtained by applying the propagation models (Fig. 4a and 4c) produce identical results as for Monte Carlo simulations (Fig. 4b and 4d).

V. Error Propagation Estimation

Using the previous rules, by exploring where the errors are introduced and how they propagate, it is possible to estimate the error propagation at the output(s) of an accelerator design. Representing the accelerator as a Data Flow Graph (DFG), we propose an algorithm based on Depth-First Search (DFS) [22] to explore all the nodes in the DFG, from the output to the inputs. All the nodes on which the output has a data dependency must be visited.

Consider the DFG for a 3×3 Gaussian kernel, as depicted in Figure 5. The output error is calculated starting from the shift right at the output. From it, the error of its predecessors is requested, in this case the adder s_8. This adder, in turn, needs the error propagated by the adder s_7, and so on until the inputs are reached. For instance, considering that just adders s_1 and s_2 are replaced by approximate adders, once those nodes are reached, the error they generate is determined according to its configuration. Because the inputs are considered as error-free, the error propagated after s_1 is the error generated by itself. The same occurs for s_2. Both inputs to adder s_5 now present errors, and the error propagated after s_5 is computed for the case of an accurate addition of two input error distributions, as introduced in Eq. (2). This error is propagated through s_7 and s_8 with no change, because no other errors are generated in other computations, and the final error distribution propagates is just affected by the shift right at the end. This estimation process is generalized in Algorithm 1.

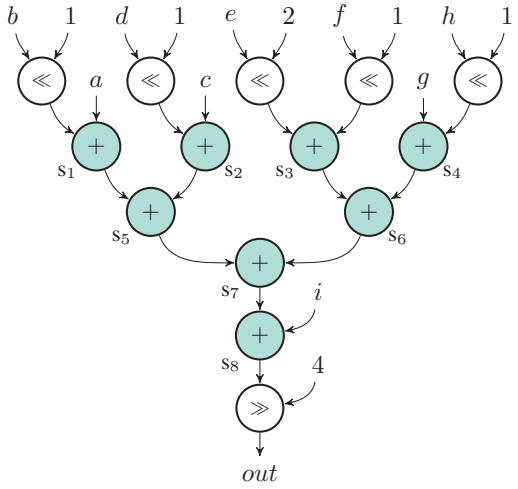


Fig. 5: DFG to represent a 3×3 Gaussian kernel. The additions are potentially replaced by approximate adders.

Algorithm 1 DFS-based Error Propagation Estimation

```
1: COMPUTEERROR(node)
2: foreach parent ∈ node.parents do
3:   if parent == instruction then
4:     parentError = COMPUTEERROR(parent)
5:   else
6:     parentError = 0
7: ERRORPROPAGATION(node.error, node.parentsErrors)
```

VI. CEDA Tool

We implement our methodology into a tool and we call it CEDA: Compiler-driven Error Analysis for Designing Approximate Accelerators. Figure 6 shows its major components. An annotated C/C++ source code and an error model file are required to perform an estimation of the error propagation. The source code corresponds to the application where the introduction of errors in a function or kernel is explored. A custom directive, `#pragma approx`, indicates to a modified Clang compiler when an accurate addition is replaced by an approximate one, as depicted in the code example of Figure 7. In this Figure, the `gear wrp 8 1 4` of the pragma points the type of adder used for the evaluation (in this case GeAr) with its configuration (bit width `w`, resultant `r`, and propagation bits `p`; in this example 8, 1, and 4, respectively).

The modified Clang compiler is capable of handling this pragma and to append this information as metadata in the Intermediate Representation (IR) of the code, as can be seen in Figure 7. The error analysis stage (Fig. 6) takes the code as IR and creates a DFG representation. The metadata allows to identify which additions, represented as DFG nodes, are then approximated. The error model input corresponds to the definition of the error representation (i.e., as PMF) and the set of error propagation models previously presented. The error analysis follows the proposed methodology to estimate the error propagated to the output as a PMF. This information can be then related to specific metrics such as MED, Peak Signal-to-Noise Ratio (PSNR) or Structural Similarity (SSIM) index, depending on the application nature. The error propagation analysis is performed statically, eliminating the need for simulations and the associated computing time.

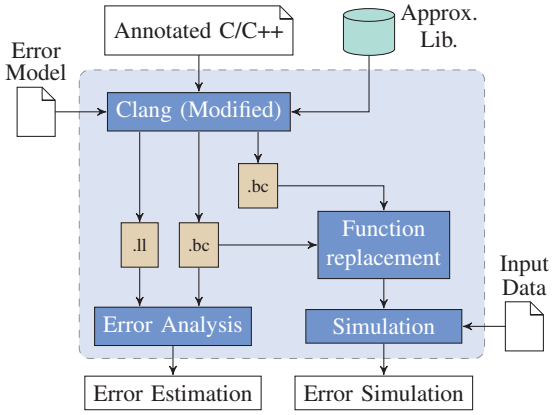


Fig. 6: Main components of our CEDA tool. The .bc and .ll files contain the LLVM intermediate representation (IR) of the annotated source code, the error model, and the approximate library. Both error analysis and simulation are carried out using the IR.

```
void prog(int64_t* out, int64_t a, int64_t b, int64_t c) {
  #pragma approx gear wrp 8 1 4
  int64_t r = a + b;
  int64_t m = r + c;
  out[0] = r;
  out[1] = m;
}
```

```
define void @prog(i64* nocapture %out, i64 %a, i64 %b,
i64 %c) #0 {
entry:
  %add = add nsw i64 %a, %b, !approx !1
  %add1 = add nsw i64 %add, %c
  store i64 %add, i64* %out, align 8
  %arrayidx2 = getelementptr inbounds i64, i64* %out, i64 1
  store i64 %add1, i64* %arrayidx2, align 8
  ret void
}

!1 = metadata !{metadata !"gear", i32 8, i32 1, i32 4}
```

Fig. 7: Example code with custom pragma annotation (above) and its corresponding IR (below). Notice the metadata added regarding the approximation.

CEDA provides the possibility to obtain output error distributions through simulations. The metadata in the IR is used to determine which additions to replace by a function invocation and to provide the parameters required. This function mimics the execution of an approximate adder. For this, a library of approximate adders is required as an input to our tool, as showed in Figure 6, which has functional models that can produce the results of the approximate adders. Once this replacement is performed, the IR codes for accurate and approximate versions of the application are executed using the LLVM-JIT (just-in-time) [23] compiler and a given input data set. The error statistics are determined by comparing the results. Results obtained through simulations provide the possibility to compare the accuracy of the results provide by the proposed methodology, and to validate the propagation models.

VII. Experimental Evaluations

We compare the results obtained using our methodology against user-defined input data sets and Monte Carlo simulations. Results for estimations and simulations are generated with CEDA. Figure 8 presents the estimation and simulation results for the aforementioned Gaussian kernel. In this example, the adders s_4 and s_7 have been replaced by GeAr adders. Our estimation (Fig. 8a) shows almost identical error propagation on 1000000 input combinations generated by a Monte Carlo simulation (Fig. 8b). The inputs generated used by the simulation correspond to potential pixel values as in an image. There is a difference in the probability of the ED $= 2$, which is also reflected in the 0.02 difference in the ER, because the values in the simulation do not cause the approximate adders to produce the errors with the exact same probability as considered by the analytic error modelling [7], which is used by our methodology. These differences are also noticeable when using a smaller input data set, as a picture for this example, due to the distribution of the inputs provided by each one. Two test pictures, *peppers* and *cameraman*, were used as inputs. For the case of *peppers* (Fig. 8c), the results show the same ED values but with lower value in their probability. For the *cameraman* (Fig. 8d), the ER is basically equal to the simulation results, but it is appreciable the difference of the probability for ED $= 8$ and ED $= 32$.

The required time for the estimation does not overpass some tens of milliseconds, compared to the seconds required for the simulation, which is particularly important if this analysis is required for a diverse set of configurations of approximate adders. The results using a smaller data set can reduce the required simulation time, but they produce inaccurate results depending on the values provided by the image(s) used.

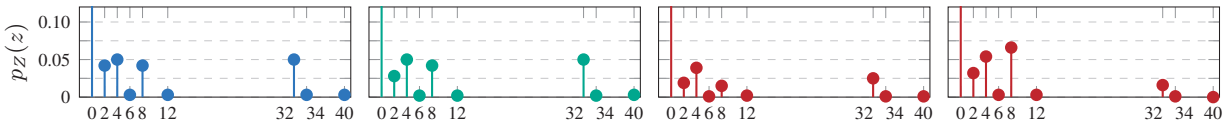
Our tool is able to accurately estimate for different amount of adders replaced by approximated ones. For example, Figure 9 shows the MED and ER for a Gaussian kernel, considering that the adders replaced grows from 1 (just one adder) to 8 (all additions). The MED is calculated from the resulting error propagated PMF. Our estimation shows slight differences regarding simulation, because this simulation is not exhaustive, and therefore does not consider all input combinations.

Figure 10 shows an example for a Laplacian image filter, where two additions have been replaced by GeAr adders. In this Figure, the accuracy of the estimation with respect to simulation can be noticed.

Low-power approximate adders: Even though in this work we have focused on high-performance approximate adders as source of approximations, our methodology can be also applied to low-power approximate adders. Figure 11 shows an example for a Laplacian kernel, where two additions are approximated using AMA1 and AMA5 approximate adders, with 2 and 3 approximate bits, respectively [24]. The error distributions for these adders were precomputed through simulations. As depicted, the difference between the estimation and the simulation is negligible.

VIII. Conclusion

In this paper, we have presented a compiler-based methodology to estimate the error propagation on approximate accelerators designs. This methodology employs heuristic models to describe the error propagation through accurate arithmetic computations. These models are used in a DFS-based algorithm and integrated into CEDA, a tool which we released as an open-source contribution. Our experiments on various image processing kernels have shown very accurate estimations with respect to picture-based input data sets and Monte Carlo simulations. CEDA, by analyzing the source code of the approximation tolerant accelerator, enables a faster and more extensive explorations on the error propagation caused by approximate computations.



(a) Estimation (ER = 0.193) (b) Simulation (ER = 0.178) (c) *peppers* (ER = 0.102) (d) *cameraman* (ER = 0.176)

Fig. 8: Error propagation estimation (a) and simulation (b) for a 3×3 Gaussian kernel. Two adders are replaced by GeAr adders. Results for a defined data set, such as test image *peppers* (c) and *cameraman* (d) are presented.

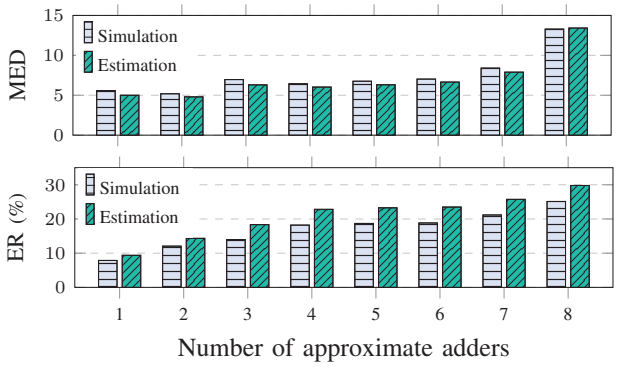
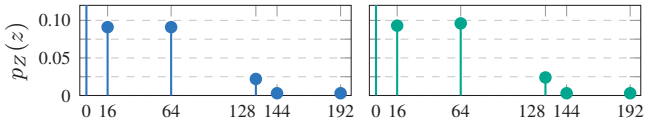
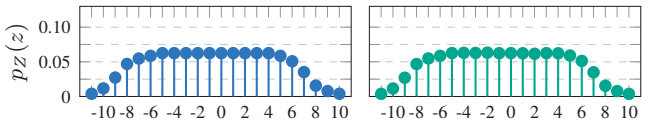


Fig. 9: MED and ER for different error estimations and simulation of a 3×3 Gaussian kernel. The amount of approximate adders varies from 1 to 8.



(a) Estimation (ER = 0.210) (b) Simulation (ER = 0.219)

Fig. 10: Error propagation estimation (a) and simulation (b) for a Laplacian image filter. Two adders are replaced by GeAr adders.



(a) Estimation (ER = 0.9375) (b) Simulation (ER = 0.9375)

Fig. 11: Error propagation estimation (a) and simulation (b) for a Laplacian image filter where two adders are replaced by approximate low-power adders.

The work presented in this paper focuses on approximate accelerator design. However, our methodology can be used to explore the error at the output of general approximate hardware blocks designs by using a software-based representation, and for the exploration of error resiliency in functions considered as potential approximate accelerators. The use of our CEDA tool can be further extended to other approximate circuits and to perform automatic design space exploration of approximate accelerators for giving error constraints or optimization goals, for instance in terms of delay, area, or power consumption.

ACKNOWLEDGMENT

This work was supported in part by the Costa Rica Institute of Technology.

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, Feb 2016.

[2] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited - cross-layer approximate computing: From logic to architectures," in *the 53rd Design Automation Conference (DAC)*, 2016.

[3] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *the 45th Intl. Symposium on Microarchitecture (MICRO)*, 2012.

[4] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, "Snnap: Approximate computing on programmable socs via neural acceleration," in *the 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[5] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "An area-efficient consolidated configurable error correction for approximate hardware accelerators," in *the 53rd Design Automation Conference (DAC)*, 2016.

[6] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on Computers (TC)*, Sept 2013.

[7] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, and J. Henkel, "Probabilistic error modeling for approximate adders," *IEEE Transactions on Computers (TC)*, March 2017.

[8] M. K. Ayub, O. Hasan, and M. Shafique, "Statistical error analysis for low power approximate adders," in *the 54rd Design Automation Conference (DAC)*, 2017.

[9] S. Mazahir, O. Hasan, R. Hafiz, and M. Shafique, "Probabilistic error analysis of approximate recursive multipliers," *IEEE Transactions on Computers (TC)*, 2017.

[10] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *the 52nd Design Automation Conference (DAC)*, 2015.

[11] A. Kapare, H. Cherupalli, and J. Sartori, "Automated error prediction for approximate sequential circuits," in *the 35th Intl. Conference on Computer-Aided Design (ICCAD)*, 2016.

[12] J. Huang, J. Lach, and G. Robins, "Analytic error modeling for imprecise arithmetic circuits," in *Silicon Errors in Logic - System Effects*, 2011.

[13] ——, "A methodology for energy-quality tradeoff using imprecise hardware," in *the 49th Design Automation Conference (DAC)*, 2012.

[14] W.-T. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits," in *the 31st International Conference on Computer Design (ICCD)*, 2013.

[15] S. Lee, D. Lee, K. Han, E. Shriver, L. K. John, and A. Gerstlauer, "Statistical quality modeling of approximate hardware," in *the 17th Intl. Symposium on Quality Electronic Design (ISQED)*, 2016.

[16] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *the GLSVLSI*, 2015.

[17] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "A comparative evaluation of approximate multipliers," in *the Intl. Symposium on Nanoscale Architectures (NANOARCH)*, 2016.

[18] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *the Design, Automation and Test in Europe (DATE)*, 2008.

[19] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *the 12th International Symposium on Integrated Circuits (ISIC)*, 2009.

[20] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *the Intl. Conference on Computer-Aided Design (ICCAD)*, 2013.

[21] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *the 52nd Design Automation Conference (DAC)*, 2015.

[22] R. Sedgewick and K. Wayne, *Algorithms*. Addison-Wesley, 2011.

[23] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *CGO*, 2004.

[24] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Jan 2013.