Decision Tree-based Adaptive Approximate Accelerators for Enhanced Quality

Mahmoud Masadeh, Alain Aoun, Osman Hasan, and Sofiène Tahar
Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada
Email:{m_masa, a_alain, o_hasan, tahar}@ece.concordia.ca

***Abstract*—Hardware accelerators are used for parallel computation with the tendency to accept inexact results. Such accelerators are used extensively in big-data processing applications, and thus can be designed approximately for reduced power consumption, area and processing time. However, since for some inputs the output errors may reach unacceptable levels, the main challenge is to *ensure the quality* of the approximated results. Towards this goal, in this paper, we propose a fine-grained input-dependent decision tree-based adaptive approximate design to meet the output quality constraints set by the user. For illustration purposes, we use a library of 16-bit approximate array multipliers with 20 different settings. The proposed methodology has been evaluated for audio and image processing applications. The simulation result, demonstrate the effectiveness of the proposed methodology, utilizing a lightweight decision tree-based *design selector* where the proposed adaptive design achieves the user-specified target output quality with a relatively low overhead.**

I. Introduction

Hardware accelerates, with a reduced power consumption, latency and increased parallelism, are able to process big-data more efficiently than software processing [1]. Thus, they are quite suitable for image and digital signal processing (DSP). Approximate computing (AC) or inexact computing is an emerging computing paradigm for error-resilient applications where computing accuracy could be scarified for gaining design efficiency in terms of reduced area, power and delay. Approximate arithmetic components, i.e., adders [2], dividers [3] and multipliers [4], can be utilized to construct *approximate hardware accelerators*, which are suitable for error-resilient computationally intensive application, e.g., big-data and image/signal processing. Such applications have the following approximation-enabling characteristics [5]: 1) the input data is noisy with iterative-refinement nature, 2) there is no golden or unique result where a set of outcomes are acceptable, 3) the best solution is not required or guaranteed where good-enough result is sufficient, and 4) the inexact result is consumable by human perception.

For a static approximate design, the approximation error persists during the operational-life of an approximate accelerator. Moreover, the error magnitude depends on the applied inputs [6]. This mandates contemporary techniques to reduce approximation error with minimal implementation overhead. Thus, it is substantial to consider this critical issue at the early design stage by accommodating to change the architecture of the approximate accelerators. This could be realized by two techniques; i) adapting the architecture of the approximate components, i.e., approximate multipliers, which constitutes the basic build block for the accelerators. For example, the quality of AC can be controlled through error compensation by predicting error magnitude for specific applied inputs utilizing machine learning techniques [7], ii) switching between different implementations of approximate components, where [8] proposed to change the approximate design dynamically for different input data to satisfy a user-defined target output quality (TOQ). We consider the second technique of switching between different implementations of approximate components. In this paper, to assure approximation quality, we propose a decision tree (DT)-based model to select the most suitable approximate design based on the applied inputs and user-specified quality constraints (TOQ). We consider *approximate accelerators* with 20 different settings for a 16-bit approximate array multipliers [4]. We implement a fully-automated tool chain for the proposed methodology, which we validated based on audio and image processing applications.

The rest of the paper is structured as follows: Section II introduces the related work. Section III explains our proposed methodology to assure the accuracy of approximate accelerators by design adaptation. Section IV provides the experimental results of audio and image processing application. Section V concludes the paper and highlights the future work.

II. Related Work

The research efforts in the field of quality assurance of approximate computing are scarce compared to the designing of approximate components. Such efforts can be classified into software and hardware techniques. However, to the best of our knowledge, there are very few works targeting the quality assurance of approximate accelerators based on fine-grained of the applied inputs. While most prior works focus on error prediction, in this paper, we aim to overcome the degraded quality of approximation through design adaptation based on the applied inputs.

In [9], the authors approximated different designs given as behavioral descriptions based on the expected coarse-grained input data distributions. These approximate designs are used to build an adaptive accelerator based on the applied workload. However, the real workload may differ completely from the training one since not all possible workload distributions can be precharacterized. Xu et al. [10] also presented a self-tuneable adaptive approximate architecture that is suitable for application-specific integrated circuit (ASIC) designs. The

978-1-7281-5365-0/20/$31.00 ©2020 IEEE

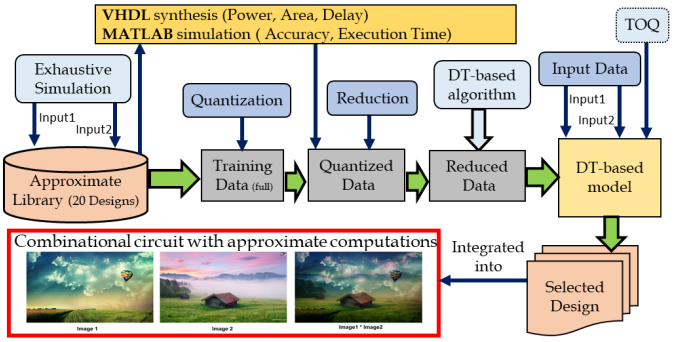

Figure 1: Proposed Methodology

used approximation techniques are variable-to-variable (V2V) and variable-to-constant (V2C) optimization only. A design-space exploration of state-of-the-art approximate designs has been conducted in [11], where a flow for designing approximate coarse-grained reconfigurable arrays (CGRAs) has been proposed.

Green [12] and SAGE [13] check the output quality of approximate programs through sampling techniques, and use a more accurate configuration if the approximation error is high. However, they are inadequate for fine-grained input data. In [14], a self-compensating accelerator has been proposed by combining approximate components with their complementary designs, which have the same error magnitude with opposite polarity, however, obtaining such complementary components is not always guaranteed. Compensation of approximation error was proposed in [7], where it utilizes a single approximate design only while other designs may exist with superior characteristics.

Motivated by the above, in this paper, we present an input-dependent quality assurance methodology for approximate computing. To the best of our knowledge, it is the first holistic work that explicitly considers both a fine-grained input-data and user preferences, as detailed in the following sections.

III. Methodology

Some of the main challenges of approximate computing include [15]: 1) selecting the most suitable approximate design based on the applied inputs and user preferences, 2) minimizing the approximation results with large error magnitude , and 3) continuously monitoring the output quality to compensate the error. Thus, we propose a methodology to continuously monitor the inputs and select the most suitable approximate design accordingly. For that, we build a *lightweight* decision tree-based *design selector*. Figure 1 shows the proposed methodology with the main building blocks. Overall, the proposed methodology consists of the following main steps:

1- Library of approximate designs: In [4], a set of energy-efficient approximate multipliers, both 8 and 16-bits designs, based on two levels of approximation and 11 different types of basic building blocks have been designed, i.e., full adders (FAs). Then, based on the obtained results, the best 5 types know as "approximate mirror adders" (AMA) [16] have been selected and another 2 levels of approximations have been added. This ended up with 20 designs with different settings, based on 5 types and 4 levels of approximations, which constitutes our approximate library. Design Type = $\{AMA1, AMA2, AMA3, AMA4, AMA5\}$, while approximation Level/Degree = $\{D1, D2, D3, D4\}$, where D1 has 15 bits approximated out of the 32-bit result, while D2, D3, and D4 have 16, 17, and 32 approximated bits, respectively

2- Training Data: Exhaustive simulation of 16-bit array multiplier includes $2^{32} = 4.29$ x 10^9 input combinations. Thus, the training data for the 20 designs of the approximate library includes $8.5\text{x}10^{10}$ instances. Each combination evaluates the error distance (ED) which represents the difference between the exact and approximate result as given by Eq. 1. However, ED evaluates design accuracy for a single inputs only. Thus, we have to evaluate average error metrics, such as peak signal to noise ratio, which is the metric that we used to build the DT-based *design selector*. Considering an approximate design with two inputs, of *n*-bit each, where the exact result is (P) and the approximate result is (P'), some of the error metrics of the approximate result include [17]:

- Error Distance (ED): The arithmetic difference between the exact output and the approximate output for a given input. ED can be given by:

$$ED = |P - P'| \tag{1}$$

- Mean Error Distance (MED): The average of ED values for a set of outputs obtained by applying a set of inputs, which is obtained as:

$$MED = \frac{1}{2^{2n}} \sum_{i=1}^{2^{2n}} |ED_i| \tag{2}$$

- Mean Square Error (MSE): It is defined as the average of the squared ED values:

$$MSE = \frac{1}{2^{2n}} \sum_{i=1}^{2^{2n}} |P_i - P'_i|^2 = \frac{1}{2^{2n}} \sum_{i=1}^{2^{2n}} |ED_i|^2 \tag{3}$$

- Peak Signal-to-Noise Ratio (PSNR): The peak signal-to-noise ratio is a fidelity metric used to measure the quality of the output images, and given by:

$$PSNR = 10 * log_{10}(\frac{2^{2n}}{MSE}) \tag{4}$$

Any of these metrics could be used to evaluate the quality of approximation results based on the application. However, in this work, we use PSNR in building the DT-based model, which is applied to image and audio processing applications.

3- Quantization of Training Data: In order to evaluate average error metrics, e.g., mean square error (MSE), peak signal to noise ratio (PSNR) and normalized mean error distance (NMED), a set of consecutive applied inputs should be grouped together as a single cluster. For that, we propose to consider 256 inputs as a single cluster. Thus, the training data has $2^{24}\text{x}20 = 3.35\text{x}10^8$ instances rather than $4.29\text{x}10^9$.

Table I: Design Characteristics of the 16-bit Approximate Library, including Power, Area, Delay and Power-Area-Delay Product (PADP)

Design	Degree	Dynamic Power(mW)	Area (Slice)	Area (LUT)	Delay (ns)	PADP	Priority
AMA1	D1	290	166	552	18.297	3809.80	19
AMA1	D2	259	165	536	18.472	3353.76	17
AMA1	D3	230	151	487	13.620	1998.6	11
AMA1	D4	52	53	115	7.547	65.93	3
AMA2	D1	318	165	504	18.479	3931.26	20
AMA2	D2	300	153	483	18.690	3560.45	18
AMA2	D3	289	148	473	18.329	3289.49	15
AMA2	D4	98	80	207	8.221	231.22	5
AMA3	D1	309	156	451	17.796	3337.87	16
AMA3	D2	292	147	467	18.876	3204.95	14
AMA3	D3	271	133	415	17.134	2544.54	13
AMA3	D4	93	38	63	7.330	68.85	4
AMA4	D1	268	143	439	15.109	2356.64	12
AMA4	D2	249	128	423	14.434	1980.33	10
AMA4	D3	222	128	413	14.366	1725.39	8
AMA4	D4	32	27	34	6.787	13.25	1
AMA5	D1	287	128	413	14.366	1725.39	8
AMA5	D2	270	99	312	13.989	1552.36	7
AMA5	D3	241	93	255	13.343	1119.05	6
AMA5	D4	74	23	24	6.046	21.03	2
Exact	-	**473**	**183**	**603**	**19.008**	**7066.76**	-

4- Pre-processing of Training Data: Data pre-processing is an often neglected but a significant step in the data mining process [18]. Building an efficient machine learning-based model utilizing $3.35\text{x}10^8$ training instances would be visionary where we intend to design a model with low implementation overhead, i.e., area, power and execution time. Thus, we perform a multi-criteria data processing based on the characteristics of our approximate library.

Table I shows the synthesis results, i.e., area, power and delay, of the 20 approximate designs. Based on that, they are prioritized based on their Power-Area-Delay Product (PADP). The design with (*Type=AMA4*, *Degree=D4*) has the minimum PADP which is 0.18% of the value of the exact multiplier design. Thus, it has the highest priority which is 1. On the other hand, the design with (*Type=AMA2*, *Degree=D1*) has the maximum PADP which is 55.63% of the value of the exact multiplier design. Thus, it has the least priority which is 20. Such design prioritization is indispensable for the proposed data pre-processing to reduce the size of the training data to be manageable.

For every distinctive applied input, we examine the proposed approximate designs to reduce their generated training data based on their accuracy and priority. The pre-processing includes the following steps:

- Discard a training instance if there exists a design with the same or better accuracy and have a higher priority. This reduction keeps 26.89% of the training instances.
- Eliminate training instances with very high accuracy, e.g., PSNR $> 70dB$, if there exists another instance with higher priority given that its PSNR $> 70dB$. This reduction combined with the previous one accumulatively conserves 16.93% of the training instances.
- Remove training instances with very low accuracy, e.g., PSNR $\leq 15dB$, which is not acceptable in real applications. Thus, the cumulative reduced training data includes 9.64% of the original data.

Table II: The Number of Training Instances of each Approximate Design after Pre-processing

	AMA1	AMA2	AMA3	AMA4	AMA5
Degree1	264	12,677	7	441,404	1,859,393
Degree2	0	0	0	29,437	315,541
Degree3	0	0	0	75,752	16,761,883
Degree4	0	1,315,321	1,493,624	4,987,277	2,230,190

The remaining data includes ≈ 3.24 x 10^7 instances which are used to build the proposed decision tree-based *design selector*. Table II shows the number of training instances which remains after reduction, for each approximate design.

5- Building Decision Tree-based Model: Such model functions as a *design selector* which enables design adaptation based on the applied inputs, where error distribution is input-dependent [19]. In [8], we designed and evaluated various machine learning (ML) models based on several algorithms, i.e., linear regression (LM), decision tree (DT), random forest (RF) and neural network (NN), developed in the statistical computing language R [20]. These models represent the *design selector* for our adaptive design. Then, for an 8-bit approximate array multipliers, we implemented and evaluated two versions of the *design selector*, based on decision tree and neural network models. We discarded the linear regression and random forest models since they have no performance benefits over decision tree and neural network.

The built models have been evaluated for their accuracy and execution time. Also, we evaluated their power, area, delay, frequency and energy since hardware implementation of the work is also considered besides the software implementation. Based on the obtained results, the execution time of NN-based model is 1.31X higher than the DT, while its average accuracy is 98% of the accuracy achieved by the DT-based model. Other design metrics, including power, slice look-up table (LUTs), occupied slices, period and energy have a magnitude of 8.6X, 13.93X, 11.74X, 1.81 and 13.6X, respectively, compared to the DT-based model. Surprisingly, in our case, the DT-based model is better than the NN-based model in all design characteristics including accuracy and execution time.

Since DT has proven superiority, in this work, for 16-bit approximate array multipliers, we implement and evaluate *design selector* based on decision tree model, utilizing judiciously reduced training data. We use MATLAB's Classification Learner Toolbox [21] to build a decision tree-based model with an accuracy of 83.9%. For the prediction, a response from the decision tree is achieved with a minimum of 5 nodes and a maximum of 9 nodes as shown in Figure 2. The simple structure of the generated tree creates a negligible overhead compared to the quality and performance achieved by the proposed design. The full textual description of the generated decision tree model can be found on *https://github.com/hvg-concordia/DTAAA*. Next, we evaluate the model performance based on audio and image blending applications.

IV. Results and Discussion

This section evaluates the effectiveness of the proposed adaptive approximate design to assure output quality. It includes the DT-based *design selector* within the library of 16-bit

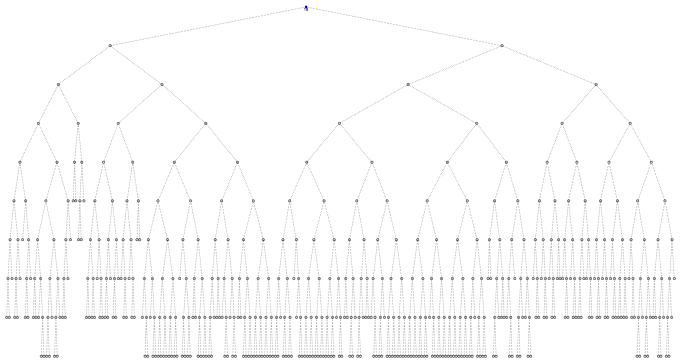

Figure 2: The Structure of the Constructed DT-Model

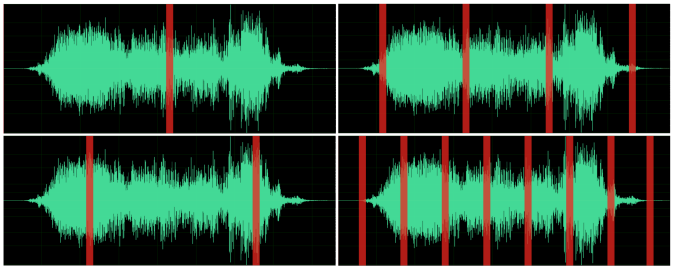

Figure 3: Sequence Used to Sample Audio Files

approximate multipliers. Multimedia services are classified as error-tolerant applications, while having a great impact on the computer industry, e.g., video gaming and video streaming. We implement audio and image blending based on the multiplication mode, where we monitor the results to measure its final quality. For that, we perform $2\text{x}10^9$ and $12\text{x}10^6$ multiplication operations for audio and image samples, respectively. We execute the experimental work for a wide variation in the target PSNR as well as the number of samples, which vary from 1 sample to the maximum possible value i.e., 2^n. For every run, the samples are averaged out and then passed to the DT-model along with target PSNR for design prediction.

1- Audio Processing: Sounds are propagating waves that could be saved in a binary format with a depth of 16-bit. Such high number of bits per sample would be able to cover a wide range of amplitudes with enhanced quality. We use a database of *WAV* sound files which are available at [22]. For each used file, we perform data sampling where Figure 3 shows the sequence of sampling when an audio processing test is conducted. Audio files are sampled in such a way to cover a wide time-line of the audio while ensuring that the samples are equidistant from each other. Figure 4 shows the obtained PSNR for executing audio blending where the TOQ varies from *15dB* to *70dB*. For each run, the number of samples, i.e., 2^n, varies from n=0 to n=17. Generally, all selected designs have a satisfying final output quality compared to the TOQ. Thus, the proposed methodology can assure the final output quality. Moreover, we notice a clear variation in the obtained output quality for a various number of samples and their corresponding evaluation overhead.

2- Image Processing: Images are managed in a similar fashion while being represented with three channels, i.e., Red,

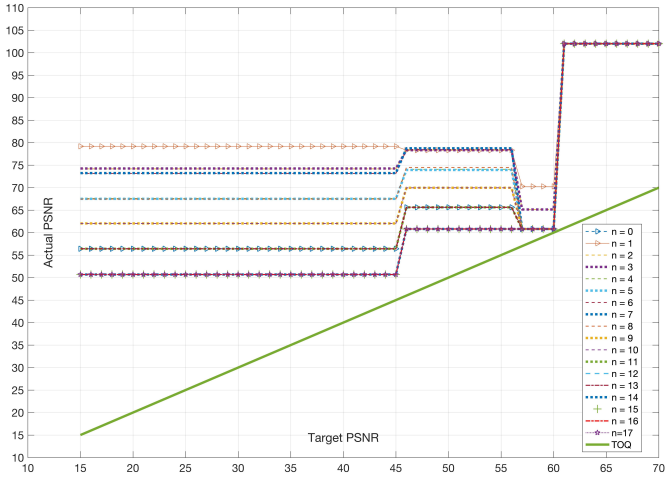

Figure 4: TOQ versus Obtained Output Quality for Adaptive Audio Blending

Green and Blue (RGB), of the same size, i.e., the number of pixels. Each pixel has a binary representation with the number of bits defining different unique colors, e.g., 16-bit red channel means 2^{16} unique red colors. For image processing the sequence of sampling is shown in Figure 5, where the sampling sequence would double the number of samples every run. Figure 6 shows the obtained PSNR for executing image blending where the TOQ varies from *15dB* to *70dB*. For each run, the number of samples, i.e., 2^n, varies from n=0 to n=12. Generally, all selected designs have a satisfying final output quality compare to the TOQ, except the case when n=0, i.e., one sample taken from each channel. Thus, the proposed methodology can assure the final output quality, with a clear variation in the obtained output quality for a various number of samples. Similar to audio processing, image processing shows that on average a higher number of samples would result in either similar or inferior output quality. For instance, for a target PSNR of *57dB* to *60dB*, n=12 resulted in a PSNR of almost $64dB$. On the other hand, with the same parameters held in position, n=5 resulted in a much better PSNR of almost *93dB*. Figure 7 shows an example of images that we used in the blending operation.

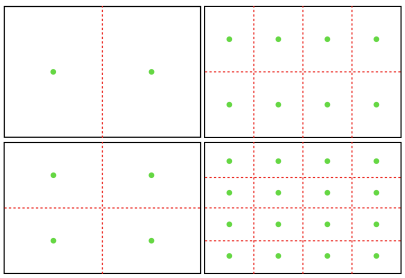

Figure 5: Sequence Used to Sample Images

The obtained results show that the achieved PSNR is higher than the TOQ because of the pre-processing step discussed in Section III, which resulted in keeping designs that have better quality while targeting the lowest PADP. Thus, image and

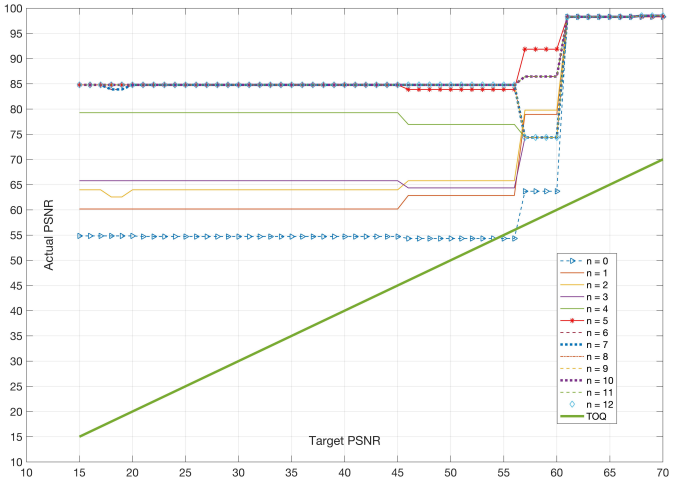

Figure 6: TOQ versus Obtained Output Quality for Adaptive Image Blending

audio blending could be used together for full-video blending. The designs we use have only 4 degrees of approximation. However, additional degrees might be discovered and added to the decision tree model so that the library would become more significant and better results might be achieved.

V. Conclusion

The emerging approximate computing paradigm reduces energy consumption and execution time of error-resilient applications by sacrificing the quality constraints. For a static approximate design, when the input data varies, the associated error magnitude varies accordingly, and thus it may reach unacceptable levels for some inputs for a static design. In this work, targeting approximate programs, we proposed and implemented a novel fine-grained input-based adaptive design based on decision tree models, which only use 10% of the generated training data, that have the highest priority. The proposed approach applies to both hardware and software designs, where we were able to satisfy the TOQ with negligible energy and delay overhead most of the time. Our ongoing work seeks to expand the approximate library to encompass other approximation techniques. For follow up work, we are targeting a full hardware implementation of our proposed system with different error-resilient applications.

References

[1] A. G. Scanlan, "Low power and mobile hardware accelerators for deep convolutional neural networks," *Integration*, vol. 65, pp. 110 – 127, 2019.

[2] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate xor/xnor-based adders for inexact computing," in *IEEE International Conference on Nanotechnology*, 2013, pp. 690–693.

[3] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, "Truncapp: A truncation-based approximate divider for energy efficient dsp applications," in *Design, Automation Test in Europe Conference Exhibition*, 2017, pp. 1635–1638.

[4] M. Masadeh, O. Hasan, and S. Tahar, "Comparative study of approximate multipliers," in *Great Lakes Symposium on VLSI*. ACM, 2018, pp. 415–418.

[5] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Design Automation Conference*, 2015, pp. 1–6.

Figure 7: Example of 16-bit Blended Images [23]

[6] M. Laurenzano, P. Hill, M. Samadi, S. Mahlke, J. Mars, and L. Tang, "Input responsiveness: Using canary inputs to dynamically steer approximation," in *Programming Language Design and Implementation*. ACM, 2016, pp. 161–176.

[7] M. Masadeh, O. Hasan, and S. Tahar, "Machine Learning-Based Self-Compensating Approximate Computing," *CoRR*, vol. abs/2001.03783, 2018, https://arxiv.org/abs/2001.03783.

[8] ——, "Using machine learning for quality configurable approximate computing," in *Design, Automation & Test in Europe*. IEEE/ACM, 2019, pp. 1554–1557.

[9] S. Xu and B. C. Schafer, "Approximate reconfigurable hardware accelerator: Adapting the micro-architecture to dynamic workloads," in *International Conference on Computer Design*. IEEE, 2017, pp. 113–120.

[10] ——, "Toward self-tunable approximate computing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 27, no. 4, pp. 778–789, 2018.

[11] M. Brandalero, L. Carro, A. C. S. Beck, and M. Shafique, "Approximate on-the-fly coarse-grained reconfigurable acceleration for general-purpose applications," in *Design Automation Conference*. ACM, 2018, pp. 160:1–160:6.

[12] W. Baek and T. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," *SIGPLAN Notices*, vol. 45, no. 6, pp. 198–209, Jun. 2010.

[13] M. Samadi, J. Lee, D. Jamshidi, A. Hormati, and S. Mahlke, "SAGE: Self-tuning approximation for graphics engines," in *International Symposium on Microarchitecture*, 2013, pp. 13–24.

[14] S. Mazahir, O. Hasan, and M. Shafique, "Self-compensating accelerators for efficient approximate computing," *Microelectronics Journal*, vol. 88, pp. 9 – 17, 2019.

[15] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Quality control for approximate accelerators by error prediction," *IEEE Design Test*, vol. 33, no. 1, pp. 43–50, 2016.

[16] 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*, vol. 32, no. 1, pp. 124–137, 2013.

[17] M. Masadeh, O. Hasan, and S. Tahar, "Error analysis of approximate array multipliers," *CoRR*, vol. abs/1908.01343, 2019, https://arxiv.org/pdf/1908.01343.pdf.

[18] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2015.

[19] W. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits," in *International Conference on Computer Design*, 2013, pp. 47–53.

[20] "The R project for statistical computing," 2020, https://www.r-project.org/, Last accessed on 2020-02-15.

[21] MATLAB and Classification Learner Toolbox Release, Natick, Massachusetts, United States:The MathWorks, Inc., 2018b.

[22] "BBC Sound Effects," 2020, http://bbcsfx.acropolis.org.uk/, Last accessed on 2020-06-16.

[23] "RAW-Samples," 2020, http://rawsamples.ch/, Last accessed on 2020-06-16.