Appears in the Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE), 2017

Gaussian Mixture Error Estimation for Approximate Circuits

Amin Ghasemazar and Mieszko Lis
The University of British Columbia
{aming,mieszko}@ece.ubc.ca

***Abstract*—In application domains where perceived quality is limited by human senses, where data are inherently noisy, or where models are naturally inexact, approximate computing offers an attractive tradeoff between accuracy and energy or performance. While several approximate functional units have been proposed to date, the question of how these techniques can be systematically integrated into a design flow remains open.**

Ideally, units like adders or multipliers could be automatically replaced with their approximate counterparts as part of the design flow. This, however, requires accurately modelling approximation errors to avoid compromising output quality.

Prior proposals have either focused on describing errors per-bit or significantly limited estimation accuracy to reduce otherwise exponential storage requirements. When multiple approximate modules are chained, these limitations become critical, and propagated error estimates can be orders of magnitude off.

In this paper, we propose an approach where both input distributions and approximation errors are modelled as Gaussian mixtures. This naturally represents the multiple sources of error that arise in many approximate circuits while maintaining reasonable memory requirements. Estimation accuracy is significantly better than prior art (up to 7.2× lower Hellinger distance) and errors can be accurately propagated through a cascade of approximate operations; estimates of quality metrics like MSE and MED are within a few percent of simulation-derived values.

I. Introduction and background

Over the past decade, power and energy considerations have become a limiting factor for digital hardware design. Recently, approximate computing techniques [1–4] have been proposed to address this: the idea is that smaller, less complex circuits that compute *almost* the correct result are acceptable in many application domains. Domains that rely on sampling inherently noisy signals — ranging from biochemistry to radar processing — are tolerant of small errors, as are machine learning techniques such as deep neural networks.

A wide range of custom-designed approximate computing elements have been proposed, ranging from functional units (*FUs*) like approximate adders [5–9] or multipliers [10–12] to architectural components like approximate caches [13]. Overclocking and undervolting techniques can turn exact circuits into approximate variants. Methodologies for automatically approximating FUs in a design flow have also been proposed [10, 14]; the idea is that modules (e.g., adders) can be replaced with their approximate equivalents to optimize for a specific objective (e.g., energy) provided that an *output quality constraint* (known as *QoS*) for the entire circuit is not violated.

Key to such flows is an accurate estimation of the error introduced by *approximate* functional units (*aFUs*). The QoS depends on both the error *generation* (by approximate FUs) and error *propagation* across the rest of the circuit. Since inputs can vary a great deal, approximation errors are typically represented as statistical distributions [10, 14–18]; a key research question, therefore, is how to represent such distributions with sufficient accuracy and acceptable space overhead.

Much prior work has focused on modelling the *bit error rate* of approximate circuits — i.e., the number of bits that differ from the exact result [17, 19, 20]; this is appropriate when each bit represents an independent value (e.g., pixel), but substantially underestimates errors when bit vectors represent numerical quantities. Methods that attempt to model numerical values either model error distributions as single Gaussians [14, 16], or discretize the distribution in exponentially-sized intervals $(2^n, 2^{n+1}]$, with a single count for all errors in an interval [10, 15]. These are compact, but offer a close estimate only around one central value (such as 0), and have poor accuracy when propagated across multiple aFUs.

In this paper, we observe that, while errors are not confined to small range around a fixed centre, they are also not spread out randomly across the full range of possible values. Instead, both error and output values can be efficiently represented and analytically propagated as a set of Gaussian distributions, i.e., a *Gaussian mixture model*. This formulation also lends itself naturally to characterizing error distributions for arbitrary aFUs (such as those generated through synthesis-like flows [20]), as estimation techniques for such models and their convergence properties are well known [21–23].

Specifically, this paper makes the following contributions:

- we propose a novel error estimation model based on Gaussian mixtures that represents both output and error distributions of aFUs using only a few Gaussians;
- we show that this approach is an effective model for value distributions observed in real applications;
- we develop rules to propagate output and error distributions through multiple precise and approximate FUs;
- we describe a space-efficient representation of the intrinsic error introduced by an aFU;
- we show how aFUs can be characterized through well-known machine-learning methods like expectation maximization (EM).

II. Gaussian mixture error estimation (GMEE)

Consider a hypothetical approximate synthesis flow that takes a precise RTL design and automatically replaces some of the

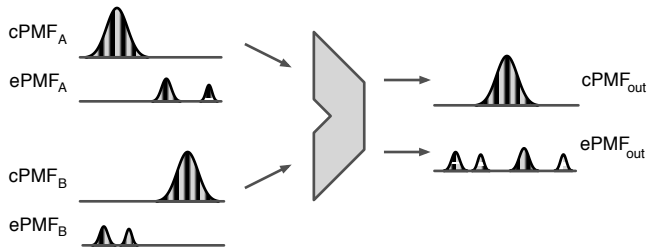

Fig. 1. Propagation through an aFU. The element's transfer function generates a correct PMF and an error PMF (from the input *ePMF*s and *cPMF*s).

computation units (e.g., adders or multipliers) with approximate versions. Such a flow can be cast as an optimization problem in which energy is minimized — subject to a minimum output quality constraint — by replacing some FUs with approximate versions [10, 14, 16, etc].

A key question that arises is how to estimate the output quality given a specific choice of approximate compute units. Exhaustive simulation and simulation-based Monte-Carlo sampling [19] require extensive computation effort and are practical only for small circuits (such as the FUs themselves) [15]. This motivates the need for a statistical approach, where *statistical distributions* are propagated through the approximate circuit to estimate the final error metric.

The steps of a statistics-based estimation flow are as follows:

1) obtain expected distributions for circuit inputs (e.g., through high-level simulation);
2) propagate the input distributions through the approximate circuit, keeping track of correct and incorrect values;
3) compute the final error metric from the output distribution.

In this section, we describe how to separately model the distributions of correct and incorrect values produced by an aFU, and how to propagate them across the entire circuit.

Reference and approximate distributions. Throughout the paper we will describe the statistical distributions of correct and erroneous results as probability mass functions (PMFs), using the following shorthands:

- The **reference PMF (*rPMF*)** refers to the values (e.g., outputs) observed in a *non-approximate* FU. We measure rPMFs directly by executing the relevant precise operation (e.g., addition) on the input data.
- The **approximate PMF (*aPMF*)** refers to the values produced by an aFU. We measure these by simulating an RTL implementation of the relevant aFU.
- The **correct-part PMF (*cPMF*)** are the *correct* outputs produced by an aFU, i.e., where the output matches that of the non-approximate version for the same inputs.
- The **error-part PMF (*ePMF*)** are the *incorrect* outputs produced by an aFU. Together, $cPMF + ePMF = aPMF$.

Gaussian mixture error model. We represent each PMF as a sum of a finite number K of weighted Gaussians:

$$PMF = \sum_{i=1}^{K} w_i \mathcal{N}\left(\mu_i, \sigma_i^2\right).$$

In addition, we make the assumption that the component Gaussians are independent; this heuristic makes propagation computations tractable, and fits well with empirical data.

Propagation through non-approximate FUs. The proposed error model, shown in Figure 1, separately propagates the *cPMF*s and *ePMF*s through the datapaths in the design. Each FU is represented by a *value transfer function* that depends on the mathematical operation performed by the unit: this function combines the unit's input distributions to produce an output distribution. For example, consider a *non-approximate* adder with two approximate inputs (denoted A and B), and corresponding input distributions:

$$aPMF_A = cPMF_A + ePMF_A$$
$$aPMF_B = cPMF_B + ePMF_B.$$

(Note that even though the non-approximate adder will not *introduce* any errors, it will propagate input errors to its output.)

For adders, the output PMF is the distribution of the sum, and so $aPMF_{out}$ is the convolution of the input PMFs:

$$aPMF_{out} = aPMF_A * aPMF_B.$$

The Gaussian mixture representation and the independence assumption make it straightforward to propagate the *cPMF*s and *ePMF*s through the transfer functions. For example, the correct-part PMF of an adder output is

$$\begin{aligned} cPMF_{out} &= cPMF_A * cPMF_B \\ &= \sum_{i=1}^{K}\sum_{j=1}^{L} w_i \mathcal{N}\left(\mu_i, \sigma_i^2\right) * w_j \mathcal{N}\left(\mu_j, \sigma_j^2\right) \\ &= \sum_{i=1}^{K}\sum_{j=1}^{L} w_i w_j \mathcal{N}\left(\mu_i + \mu_j, \sigma_i^2 + \sigma_j^2\right). \end{aligned}$$

PMF propagation through approximate functional units. In addition to *propagating* errors, aFU models also represent errors *introduced* by the aFU. Given two *cPMF* inputs, where a non-approximate adder would produce a *cPMF*, an approximate adder model will produce both a *cPMF* and an *ePMF*.

We model each aFU as a combination of its non-approximate equivalent and filters that separate the output into a *correct filter function cFF* and an *error filter function eFF*. These filters are convolved with the *output* of the equivalent non-approximate adder to obtain the correct and erroneous distributions. Figure 2 illustrates this principle.

For example, the *cPMF* and *ePMF* outputs of an approximate adder are modelled as

$$cPMF_{out} = cFF * (cPMF_A * cPMF_B)$$
$$ePMF_{out} = cFF * (ePMF_A * cPMF_B + cPMF_A * ePMF_B + ePMF_A * ePMF_B) + eFF * (cPMF_A * cPMF_B + \cdots)$$

Although in principle *cFF* and *eFF* could be represented by any computable function, we restrict them to weighted sums of Gaussian distributions and Dirac δ functions; this ensures that *aPMF* and *ePMF* remain Gaussian mixtures and can be propagated as described above.

Compact representation of PMFs. To represent PMFs concisely, we observe that most Gaussian components of a PMF are in the *ePMF* part of the output and have passed through multiple *eFF*s, and so contribute very little to the overall mixture. In Sec. III, we quantify the accuracy loss due

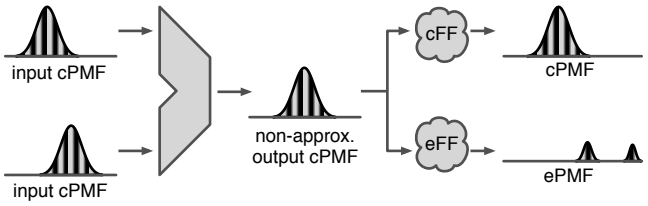

Fig. 2. Approximate functional units are modelled as a combination of a non-approximate equivalent with filter functions *cFF* and *eFF* applied to the output to produce the correct and error components.

to limited representation size, and discard all but six component Gaussians without noticeable loss of fidelity.

Automatically characterizing approximate components. Automatically generating aFUs through gate-level synthesis techniques [20] requires a characterization procedure to estimate the *cFF* and *eFF* filters without human intervention. Characterization is complicated by the fact that the errors generated by the aFU may be dependent on the input value range. The natural solution here is a lookup table addressed by input distribution parameters; because in GMEE those distributions can have many parameters (e.g., five $\langle \mu, \sigma, w \rangle$ triples), however, such a table would have prohibitively many dimensions.

To reduce the number of dimensions, we take advantage of two insights. Firstly, we note that in practice the *rPMF* suffices to capture the dependency, and the table need only be addressed by the parameters of *rPMF* (which can be computed since the non-approximate equivalent unit is known). Secondly, because *rPMF* is also a Gaussian mixture and convolution is distributive, *cFF* and *eFF* can be applied *separately* to each Gaussian component, and the results combined linearly to produce the complete output *cPMF* and *ePMF*. This means that the *cFF* and *eFF* lookup tables for each component have only two dimensions and can be addressed by a $\langle \mu, \sigma \rangle$ tuple.

The distributivity of convolution over *rPMF* also allows approximate module characterization to be performed on single-Gaussian inputs. To characterize such a module, we:

1) sweep Gaussian parameters μ and σ for each input so that eventually all combinations are covered for all inputs;
2) repeatedly sample the input distributions, and for each set of samples compute *rPMF* and *aPMF* by simulating the non-approximate and approximate versions of the module;
3) calculate the filter functions *aFF* and *eFF* for each *output* $\langle \mu, \sigma \rangle$ by deconvolution of *rPMF* with *cPMF* and *ePMF*;
4) compute Gaussian mixture representations for *aFF* and *eFF* via Expectation Maximization [22].

This results in an $\langle aFF, eFF \rangle$ pair for each reference output $\langle \mu, \sigma \rangle$. Since both are represented as Gaussian mixtures, they require three numbers for each component Gaussian, a total of fifteen numbers for the $\langle aFF, eFF \rangle$ pair. With 1,000 values for each μ and σ, ~57 MB of storage space are required for each approximate component — well within the footprint of a design resource such as an ASIC library.

III. Results and discussion

Methods. We used RTL Verilog to implement circuits with one and two levels of 32-bit approximate ETAIIM adders with 4 bits per block and three-block carry chains [5, Fig. 3]; we used RTL-level simulation to obtain the *cFF* and *eFF* tables as well as to obtain true simulation results to evaluate GMEE. Bluespec RTL-level simulation of a H.264 decoder [24] was used to obtain realistic input data for the approximate adders. The remainder of the error models were implemented in Matlab.

As a baseline, we used modified interval arithmetic (MIA) [15] and a single normal distribution (SAM) [16]. We used the EM algorithm to fit all Gaussians in GMEE. For test inputs, we selected three single Gaussian distributions with a mean of 0 and different standard deviations (σ = 32, 4096, 32768), a best-case scenario for MIA and SAM. We also included real input distributions observed at the inputs of two critical-path adders in the interpolation module of the H.264 decoder to represent real-world behaviour. We sampled each distribution at least 100,000 times.

GMEE representations survive across propagations. Figures 3a and 3b show the output of one-adder and two-adder fragments of the H.264 interpolation pipeline where the adders were replaced with ETAIIM versions, and the MIA, SAM, and GMEE estimates for the output distribution. The SAM model does poorly because σ-only propagation quickly increases σs to cover errors introduced by approximation: σ increases 16× after the first adder and 8× more after the second. The MIA model is a close estimate around 0, but its granularity becomes very coarse in this example where most values exceed 2,000. Because GMEE can model multiple distribution peaks, propagation results closely correspond to actual simulation outputs.

A small number of Gaussians suffices to represent realistic distributions. Figure 3c shows the Hellinger distance (a measure of distribution similarity where 0 is identical and 1 signifies no overlap [25]) of a GMEE model fit to the input distribution observed at the two H.264 adders, and the computation cost (in terms of thousands of convolutions) required to propagate the distributions across one approximate adder. As few as 5 to 7 Gaussians per *aPMF* bring the Hellinger distance close to 0; all other diagrams in this paper use a mixture of 5 Gaussians.

GMEE closely tracks different input distributions. Figure 3d quantifies the fit quality of the three estimates for different input distributions after propagating through one and two approximate adders. Even in cases that favour MIA and SAM (μ = 0, small σ), the Hellinger distance between the GMEE estimate and the actual simulation output is much lower.

Quality metrics computed from GMEE representations are better estimates. Figures 3e and 3f show the mean squared error (MSE) and mean error distance (MED [26]) metrics estimated from the three estimation techniques, normalized to the corresponding metric obtained from directly simulating the approximate circuit. In all cases, the quality metric estimate derived from GMEE is within a few percent of the simulation-based value: the maximum MSE deviation is 17% (gmean 2.4%), while the MED never deviates by more than 0.3%.

References

[1] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," in *MICRO*, 2012.

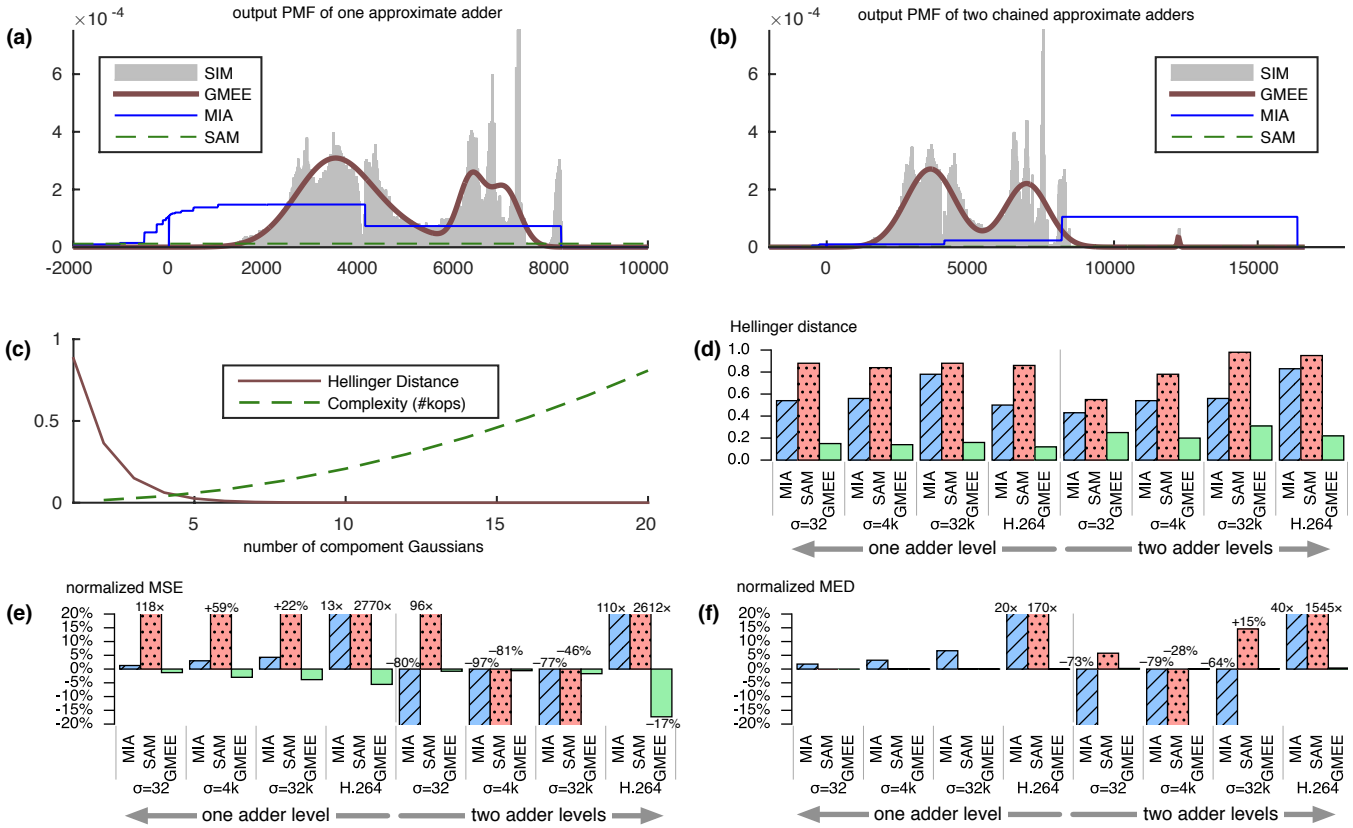

Fig. 3. (a) The output of an approximate adder in the H.264 decoder (grey = simulation results) compared to GMEE (thick red line), MIA (thin blue line), and SAM (dashed green line) estimates; (b) shows the results and estimates propagated across two approximate adders. (c) illustrates the tradeoff between accuracy (Hellinger distance) and computation complexity (1000s of convolutions) as the count of Gaussians in GMEE increases. (d) shows the Hellinger distance of the three error estimates with different input distributions; the left half shows propagation across one approximate adder, and the right half shows propagation across two. (e) shows the mean squared error, and (f) the mean error distance, of the three estimates, normalized to the MSE and MED obtained from simulation.

[2] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate Logic Synthesis Under General Error Magnitude and Frequency Constraints," in *ICCAD*, 2013.

[3] M. Kamal, A. Ghasemazar, A. Afzali-Kusha, and M. Pedram, "Improving efficiency of extensible processors by using approximate custom instructions," in *DATE*, 2014.

[4] J. S. Miguel and N. E. Jerger, "The Anytime Automaton," in *ISCA*, 2016.

[5] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed Adder For Error-Tolerant application," in *ISIC*, 2009.

[6] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of Low-Power High-Speed Truncation-Error-Tolerant Adder and Its Application in Digital Signal Processing," *Trans. VLSI*, vol. 18, pp. 1225–1229, 2010.

[7] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: IMPrecise adders for low-power approximate computing," in *ISPLED*, 2011.

[8] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC*, 2012.

[9] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *DATE*, 2012.

[10] J. Huang, J. Lach, and G. Robins, "A Methodology for Energy-quality Tradeoff Using Imprecise Hardware," in *DAC*, 2012.

[11] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *DATE*, 2014.

[12] G. Zervakis, K. Tsoumanis, S. Xydis, N. Axelos, and K. Pekmestzi, "Approximate Multiplier Architectures Through Partial Product Perforation: Power-Area Tradeoffs Analysis," in *GLSVLSI*, 2015.

[13] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: A cache for approximate computing," in *MICRO*, 2015.

[14] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint Precision Optimization and High Level Synthesis for Approximate Computing," in *DAC*, 2015.

[15] J. Huang, J. Lach, and G. Robins, "Analytic Error Modeling for Imprecise Arithmetic Circuits," in *SELSE*, 2011.

[16] 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 *ICCD*, 2013.

[17] D. Sengupta and S. S. Sapatnekar, "FEMTO: Fast error analysis in Multipliers through Topological Traversal," in *ICCAD*, 2015.

[18] S. Lee, D. Lee, K. Han, E. Shriver, L. K. John, and A. Gerstlauer, "Statistical quality modeling of approximate hardware," in *ISQED*, 2016.

[19] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and Analysis of Circuits for Approximate Computing," in *ICCAD*, 2011.

[20] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic Logic Synthesis of Approximate Circuits," in *DAC*, 2012.

[21] N. E. Day, "Estimating the components of a mixture of normal distributions," *Biometrika*, vol. 56, pp. 463–474, 1969.

[22] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society (B)*, vol. 39, pp. 1–38, 1977.

[23] L. Xu and M. I. Jordan, "On Convergence Properties of the EM Algorithm for Gaussian Mixtures," *Neural Computation*, vol. 8, pp. 129–151, 1996.

[24] K. Fleming, C. C. Lin, N. Dave, Arvind, G. Raghavan, and J. Hicks, "H.264 Decoder: A Case Study in Multiple Design Points," in *MEMOCODE*, 2008.

[25] E. Hellinger, "Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen," *Journal für die reine und angewandte Mathematik*, vol. 136, pp. 210–271, 1909.

[26] J. Liang, J. Han, and F. Lombardi, "New Metrics for the Reliability of Approximate and Probabilistic Adders," *Trans. Computers*, vol. 62, pp. 1760–1771, 2013.