Analytic Error Modeling for Imprecise Arithmetic Circuits

Jiawei Huang[1], John Lach[1], and Gabriel Robins[2]
[1]Charles L. Brown Department of Electrical and Computer Engineering, [2]Department of Computer Science
University of Virginia, Charlottesville, VA 22903, {jh3wn, jlach, robins}@virginia.edu

***Abstract*—Imprecise hardware challenges the traditional notion that correctness is an immutable priority, by systematically trading off efficacy (precision) against efficiency (power, area, performance, and cost). Evaluating the impact of such tradeoffs on output quality using, e.g., Monte Carlo simulations is a time-consuming and non-deterministic process. This paper presents two analytic modeling techniques for evaluating error properties and output quality in imprecise arithmetic circuits, based on Interval Arithmetic and Affine Arithmetic. Experiments show that these techniques offer significant speedups over previous methods, as well as promising estimation accuracy.**

I. Introduction

Power has become the limiting factor in digital systems. The demand for higher computing capacity continues to rise, but power has reached a limit due to thermal and delivery issues. Imprecise hardware (IHW), also known as *stochastic computation* [1, 2], or *Better than Worst-Case Design* [3], has been proposed as a solution to tackle this problem. Compared to traditional hardware designed to always compute correctly, IHW allows occasional computation errors but achieves higher performance and/or lower power. Any resulting errors can be relegated to dedicated error correction circuits and software, or even left uncorrected, given the error-tolerant nature of many applications. This paper focuses on deterministic arithmetic IHW, which alters the logic function of the performed computation either at design time, or dynamically at runtime. Examples include supporting only a subset of the input combinations [4, 5], voltage overscaling [1, 2], and data width reduction. The introduced errors can be uniquely determined by the input but the mapping function is not easy to determine.

The benefits of IHW are largely determined by (1) the level of performance/power improvement offered by imprecise computation, and (2) the amount of output quality degradation caused by errors. Significant power reduction and performance increase from IHW have been reported [4, 5, 6], and new IHW techniques are being developed. However, the consequences of IHW on output quality require further investigation. Typically a system with IHW components is simulated with random inputs in order to obtain an output profile with acceptable quality. This is a time-consuming task, especially for complex systems. Imprecise hardware expands the design space by relaxing the correctness requirement, thus faster error estimation is crucial to the effective exploration of the space. Moreover, simulations are usually nondeterministic and unreliable as output profiles can be influenced by the size and scope of the simulations.

This paper presents two methods that enable analytic modeling of the statistical distributions and bounds of errors introduced by IHW. These methods are based on Interval Arithmetic and Affine Arithmetic [7] and are modified to handle the characteristics of IHW errors. Compared to simulations, they provide orders-of-magnitude speedup and have the potential for stronger error bound guarantees.

II. Characterizing Imprecise Hardware Errors

A. Error as a Distribution

Although the error is a deterministic function of the input in deterministic IHW, many applications are concerned with the output error statistics (e.g. error rate). Such errors have been modeled as a random additive signal independent of the input, following a certain distribution. Fixed point error distributions can be effectively described by a Probability Mass Function (PMF) [8] which is a discrete function in the error-frequency / error-magnitude plane. Each bar (e.g., see Fig 1) indicates a non-zero error probability. The base of a bar on the x-axis indicates the magnitude range of the error and the height of the bar indicates its error frequency. The error-free condition is also represented on the plot as two bars next to ε. Thus the y-values of all bars sum to 1, because a PMF includes all error-free and error-present conditions, and ε partitions the x-axis into negative (left) and positive errors (right). The x-axis is $\log_2$-based, so for example, a bar bounded by marker -8 and -7 to the right of ε represents errors with magnitude between 2^{-8} and 2^{-7}. The error is assumed to be uniformly distributed inside each interval bounded by two adjacent markers. Errors in a floating-point system can be similarly characterized by a probability density function (PDF). PMFs and PDFs can also model the distribution of regular data during computation.

Fig. 1 shows sample PMFs of two types of IHW-induced errors: (a) frequent small-magnitude (FSM) errors, and (b) infrequent large-magnitude (ILM) errors. In these PMFs, the binary point is assumed to be 12 bits to the left of the least significant bit (i.e., the smallest possible error magnitude is 2^{-12}). These PMFs are generated by simulating the ETAIIM adder [4] and the Almost-correct adder [5] respectively, with 2 million random inputs uniformly drawn from [-1, 1].

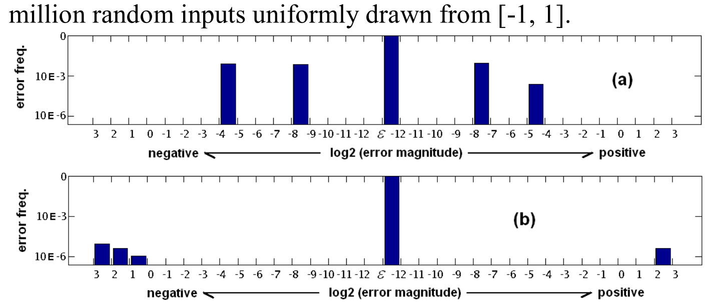


Fig. 1. PMF examples: (a) FSM errors by ETAIIM adder, and (b) ILM errors by Almost-correct adder

B. Interval Arithmetic and Affine Arithmetic

Two classical methods to estimate variable ranges during numerical computations are *Interval Arithmetic* (IA) and *Affine Arithmetic* (AA) [7]. The former uses a single interval $[x_l, x_r]$ to represent each variable, while the latter uses a so-called *affine form*: $\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots x_n\varepsilon_n$, where x_0 is the central (mean) value of the distribution, ε_i are independent uniformly-distributed variables in the range [-1, 1], and $x_1 \cdots x_n$ are coefficients.

IA is easy to compute, but the ranges produced are often too conservative, i.e. the bounds are not tight. AA improves on this by considering the first-order correlations of error signals through the sharing of ε_i. AA generally produces tighter bounds than IA but uses more complex computations. However, both techniques are only capable of representing symmetric distributions. Highly asymmetric distributions such as the errors produced by IHW (Fig. 1) are not representable by either IA or AA.

In order to represent asymmetric error distributions using PMF, we propose some modifications to IA/AA. Instead of representing the entire distribution with a single interval or affine form, we use *multiple* intervals or affine forms.

C. *Modified Interval Arithmetic*

Modified Interval Arithmetic (MIA) represents every bar of the PMF as a uniformly distributed interval. Thus, the entire PMF can be expressed as:

$$PMF_X(n) = \begin{cases} \Pr(2^{\varepsilon+n-1} \le X \le 2^{\varepsilon+n}), \text{if n} > 0 \\ \Pr(-2^{\varepsilon+n+3} \le X \le -2^{\varepsilon+n+2}), \text{if n} < \text{-1} \\ \Pr(0 \le X \le 2^{\varepsilon}), \text{if n} = 0 \\ \Pr(-2^{\varepsilon} \le X \le 0), \text{if n} = \text{-1} \end{cases}, n \in Z \quad (1)$$

Errors with magnitude below 2^{ε} are considered error-free. The specific value of ε is application-dependent but usually can be set to match the ULP (unit in the last place) of the system. Operations in MIA can be decomposed into simple operations between two intervals, which can be easily analyzed. However, MIA has a serious limitation called range explosion. Since intervals carry no information about variable correlation, a variable which appears multiple times in one expression will be treated as a new variable every time it is encountered. This may cause the final estimated range to be overly pessimistic.

D. *Modified Affine Arithmetic*

Affine arithmetic considers first-order variable correlation and yields tighter bounds. Modified Affine Arithmetic (MAA) uses a collection of affine forms to represent one error distribution, one for each bar in the PMF:

$$PMF_X = \begin{cases} P_1 : x_{1\text{-}0} + x_{1\text{-}1}\boxed{\alpha_0} + x_{1\text{-}2}\boxed{\beta_0} + \cdots \\ P_2 : x_{2\text{-}0} + x_{2\text{-}1}\boxed{\alpha_1} + x_{2\text{-}2}\boxed{\beta_1} + \cdots \\ P_3 : x_{3\text{-}0} + x_{3\text{-}1}\boxed{\alpha_2} + x_{3\text{-}2}\boxed{\beta_2} + \cdots \\ \qquad \vdots \qquad\qquad \text{Exclusive sets} \end{cases} \quad (2)$$

where $x_{i,j}$ are the coefficients and $\alpha, \beta, \cdots$ are error symbols. Each affine form occurs with probability P_i.

When two PMFs expressed in MAA operate with each other, every affine form of the first PMF operates with every affine form of the second. In order to preserve the probability equivalence, we introduce a concept called *exclusive set*, which refers to a group of affine symbols that originate from the same distribution. Any variable which appears for the first time will produce a new exclusive set: the symbols used in all of its affine forms are mutually exclusive. They are denoted using the same symbol, such as α or β in (2), with a unique subscript. Through repeated and cross computations, these symbols will be occur in many derived PMFs. When two PMFs operate, only those affine forms with no conflicting symbols (symbols that belong to the same exclusive set) are allowed to operate with each other. The reason is that symbols in the same exclusive set are originally an integral part of the same distribution (denoted as D). If a certain affine form of a derived variable contains a symbol in the exclusive set, it means this interval is a result of taking a value in that original part of D. Having two conflicting symbols in the same affine form is thus an impossible event.

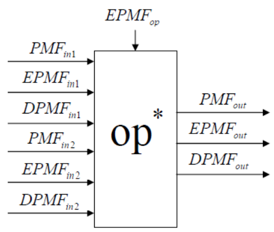
For affine forms, addition, subtraction and scaling by a constant are *affine operations* which result in a perfect affine form. Multiplication and division are *non-affine operations* whose result cannot be exactly represented in an affine form. Approximations are performed to convert the result into a closest affine form, but overshoot and undershoot can occur. In certain situations, the range estimates of AA can be worse than those of IA. For example, for two variables A and B uniformly distributed in the interval of (-1, 0) and (0, 1) respectively, their product in affine form, is represented by the form $(-0.5+0.5\varepsilon_1)(0.5+0.5\varepsilon_2) = -0.25+0.25\varepsilon_1 - 0.25\varepsilon_2 + 0.25\varepsilon_3$ which spans the range (-1, 0.5), but the actual range of the product is (-1, 0).

MAA also suffers from a practical issue of *storage explosion*. When two PMFs containing N and M affine forms operate, in general the resulting PMF will contain N×M affine forms. The size of intermediate PMFs can thus grow exponentially. This problem does not reduce estimation accuracy, but implementation inefficiency may diminish the practical value of MAA. There are several ways to address this issue. First, we can use a larger scaling factor between neighboring intervals (e.g. 4 instead of 2) while constructing the MAA. This reduces the number of terms of each MAA. Second, affine forms with low frequency or low magnitude can be merged into their neighboring affine form. Finally, all single-use variables can be represented with only one PMF.

MIA and MAA can model hardware errors resulting from sources other than IHW. For example, errors from algorithmic approximation or PVT variations can all be modeled as such. Once the errors are represented in one of the PMF formats mentioned above, their hardware details become irrelevant. The error statistical properties are preserved in PMF, so the following error analysis is implementation-independent.

III. ERROR PROPAGATION MODEL

This section presents a primitive model for error propagation across a single operation, to derive the output PMF using input PMF. Fig. 2 shows the model setup. $DPMF_{in}$ and $DPMF_{out}$ denote input/output error-free data PMF (assuming all operators are precise). $EPMF_{in}$ and $EPMF_{out}$ are pure error PMF. The actual PMFs involving imprecise operations are denoted PMF_{in} and PMF_{out}. $EPMF_{op}$ is the error PMF introduced by the imprecise operator Op^*, which can be characterized by *a priori* simulations and is also a function of design parameters *params*. The relationships among the various quantities are also shown in Fig. 2.



Rules

$$DPMF_{out} = DPMF_{in1} \text{ op } DPMF_{in2} = f(\overline{DPMF_{in}}, \text{op})$$
$$PMF_{out} = PMF_{in1} \text{ op}^* \, PMF_{in2} = u(\overline{PMF_{in}}, \text{op, params})$$
$$EPMF_{op} = error(PMF_{in1} \text{ op}^* \, PMF_{in2}) = v(\overline{PMF_{in}}, \text{op, params})$$
$$EPMF_{out} = g(\overline{DPMF_{in}}, \overline{EPMF_{in}}, EPMF_{op}, \text{op})$$

Fig. 2. Error propagation model of an imprecise arithmetic 2-operand operation

The principal task of error propagation is to determine the transfer functions *f*, *g*, *u* and *v* for common operations (ADD, SUB, MUL, and DIV). More complex operations can be studied by decomposing them into the four basic operations.

Functions *u* and *v* are pre-characterized in order to reduce derivation time. The imprecise operators in the design are pre-characterized with extensive simulation, and the results are stored in lookup tables. The table indices are single-interval distributions of the two operands, and the entry contains the PMF and EPMF resulting from the imprecise operation. For example, entry (*i*, *j*) contains the PMF_{out} and $EPMF_{op}$ for the two operands between $[-2^i, -2^{i-1}]$ and $[-2^j, -2^{j-1}]$, respectively. Operations with constants (e.g. 0.2x) are characterized similarly, but the result is a characterization vector instead of a table. Simulations cover a range no smaller than the dynamic range during the derivation. Functions *u* and *v* take the full-spectrum input PMFs and iterate over each interval. For every interval pair, we perform a lookup in the characterization table. The returned PMF_{out} and $EPMF_{op}$ will be merged to form the final full-spectrum PMF_{out} and $EPMF_{op}$. Any imprecise operator with the same setting needs to be characterized only once. With the $EPMF_{op}$ obtained as above, we derive $DPMF_{out}$ and $EPMF_{out}$ (functions *f* and g) as in Fig. 3.

$$\textbf{ADD}: DPMF_{out} = DPMF_1 * DPMF_2 \text{ (convolution)}$$
$$EPMF_{out} = EPMF_1 * EPMF_2 * EPMF_{ADD}$$
$$\textbf{SUB}: DPMF_{out} = DPMF_1 \; xcorr \; DPMF_2 \text{ (cross-correlation)}$$
$$EPMF_{out} = EPMF_1 \; xcorr \; EPMF_2 * EPMF_{SUB}$$
$$\textbf{MUL}: multiplication-convolution: f ** g(t) \equiv \int_{-\infty}^{\infty} f(\tau) g(t/\tau) \mathrm{d}\tau$$
$$DPMF_{out} = DPMF_1 ** DPMF_2$$
$$EPMF_{out} = (DPMF_1 ** EPMF_2) * (EPMF_1 ** DPMF_2) * (EPMF_1 ** EPMF_2) * EPMF_{MUL}$$
$$\textbf{DIV}: division-convolution: f // g(t) \equiv \int_{-\infty}^{\infty} f(\tau) g(t\tau) \mathrm{d}\tau$$
$$DPMF_{out} = DPMF_1 // DPMF_2$$
$$EPMF_{out} = [EPMF_1 \; xcorr \; (DPMF_{out} * EPMF_2)] // (DPMF_2 * EPMF_2) * EPMF_{DIV}$$

Fig. 3. Performing functions *f* and g for imprecise ADD, SUB, MUL & DIV

If we use an MIA-based representation, the propagation follows the rules in Figs. 2 and 3. For MAA-based representation, only DPMF and EPMF are represented in MAA form. DPMF and EPMF propagations are simply basic AA operations [7] with exclusive-set rules applied. The rules in Fig. 3 must also be modified with convolution and cross-correlation replaced by AA addition/multiplication. The required $EPMF_{op}$ is obtained by doing the same table lookup as in MIA and converting the full-spectrum PMF into MAA form.

IV. System-level Error Aggregation

In an IHW-based system, the designer is ultimately interested in the quality of the final output. Typically it is measured by comparing the DPMF and EPMF of the final output. The one-op propagation rules presented previously can be used as a building block to propagate the DPMF and EPMF from the primary input to the primary output, in five steps:

1) Construct the characterization vector and table by simulating the IHW with inputs being constants or drawn from various $[-2^i, -2^{i-1}]$ intervals.
2) Propagate DPMF assuming precise operations and obtain DPMF for every data path using the rules from Fig. 3.
3) Propagate the PMF and $EPMF_{op}$ of every IHW component by looking up the characterization vector/table.
4) Propagate EPMF using DPMF and $EPMF_{op}$ obtained in Step 2 and 3 and rules from Fig. 3.
5) Calculate output quality metric using the final output DPMF and EPMF.

Even though this process takes five steps, it is still much faster than simulation (see Fig. 4) for two reasons. First, only distributions are propagated and no actual computation is needed. The simulation used to generate the characterization table can be pre-computed with no overhead. Second, Steps 2 and 3 are independent and thus can be performed in parallel. Pure simulation is only parallelizable across different input data at the cost of additional expensive hardware.

Since the output quality metric is as diverse as an application can be, its definition is beyond the scope of this paper. As an example, assume the quality metric is signal-to-noise ratio (SNR), to calculate it from output DPMF and EPMF, we have:

$$SNR = \left(\frac{A_{signal}}{A_{noise}}\right)^2 = \left(\frac{E(DPMF)}{E(EPMF)}\right)^2$$

V. Experimental Results

Two types of computations are used for the experiment: *consecutive addition* and *FIR filter*. The former performs the summation of eight 20-bit inputs using a tree structure, and the latter involves four repeated multiply-accumulate (MAC) operations on a single 24-bit input. In both cases the input data are drawn from independent uniform distributions. The adders and multipliers involved are all IHW. The IHW adders have two forms: ETAIIM adder and Almost-Correct adder. The IHW multiplier is constructed from an imprecise adder following simple partial product generation + Wallace tree + final adder scheme. All imprecise adders and MIA are implemented as MATLAB functions. MAA is implemented in C++ by expanding the *libaffa* [9] open source library. An optimized multiplication rule is adopted from [10].

The goal of these experiments is to compare the estimation accuracy and runtime of simulation (sim), MIA, MAA and affine arithmetic (AA). Because theoretical numbers are not easily obtainable, simulation results with 1 million random drawings are regarded as the true error distribution. Hence the estimation accuracy for each method can be established by checking how close its estimate is to the simulation estimate.

The two most important attributes of any error distribution are error rate (*f*) and maximum error magnitude (*M*). These can be evaluated from the PMF using the following formula:

1) error rate : $1-\sum f_n, |M_n| < \varepsilon$

2) maximum error magnitude : $\max(|M_n|), f_n > 0$

Estimation time is affected by the computation complexity and, for simulation, the input size. We use the number of consecutive additions to represent computation complexity (ops) and tested complexity of 2, 4, 8 and 16. For simulation, we also vary the random input size.

Fig. 4 shows that simulation time grows linearly with input size but the proposed analytic methods are unaffected. All methods consume more time with increased complexity, but MAA grows the fastest (exponentially) due to storage explosion, making it a poor candidate for complex designs. In any case, MIA is at least 50 times faster than simulation.

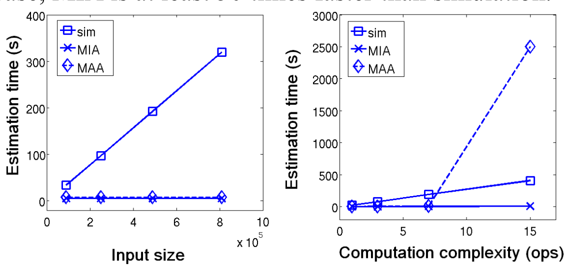


Fig. 4. Estimation time against input size and computation complexity

Table 1. Comparison of estimated error rate

Test-case	Consecutive add				Order-4 FIR			
	sim	MIA	MAA	AA	sim	MIA	MAA	AA
1	0.10	0.10	0.10	0.97	0.19	0.19	0.22	1
2	3.0e-4	2.9e-4	2.8e-4	1	8.2e-5	2.3e-4	1.1e-4	1
3	0.047	0.046	0.046	1	0.04	0.05	0.05	1

Table 2. Comparison of estimated maximum error magnitude

Test-case	Consecutive add				Order-4 FIR			
	sim	MIA	MAA	AA	sim	MIA	MAA	AA
1	0.012	0.063	0.011	0.024	1.1	2.0	2.0	4.1
2	8	128	8	70	0.06	8.0	5.0	38.5
3	8	128	8	71	0.06	8.0	8.0	16.0

Table 1 and Table 2 compare the estimated error rate and maximum error magnitude of the four estimation methods. For each computation class (*Consecutive add* and *Order-4 FIR*), three architectural configurations are evaluated: (1) all ETAIIM adders/multipliers, (2) all ACA, and (3) a mixture of both.

Qualitatively and across all estimation results, the ACA configuration has the lowest error rate and biggest error magnitude; while ETAIIM is the opposite. This observation conforms to the error characteristics of ETAIIM/ACA units (Fig. 1). For different estimation methods, affine arithmetic (AA) clearly has the worst estimation accuracy due to its inability to model asymmetric data distributions. MIA exhibits fairly high estimation accuracy on error rate, but does poorly on error magnitudes. Compared to MIA, MAA has similar estimation accuracy on error rate, but gives considerable tighter error magnitude bounds. Since even simulation does not guarantee error bounds due to the large input size required to induce extreme errors, the true error bounds should be between the simulation and MAA estimates. The original IA is not compared because it models all data with a flat uniform distribution (interval), thus providing little information on error rate. Based on the simulated error rate and the data size (1 million), we calculate that the true error rate lies in the ±15% confidence interval around the simulation value with at least 85% confidence level, thus simulation *is* a reasonable baseline for error rate estimation.

MIA and MAA are supposed to give upper bounds for the error magnitude. However, during the course of the experiments, we performed truncation of the MAA to reduce the estimation time. The number of affine forms in this case grows to over 8 million after 4 MAC operations, and we truncate low frequency MAA terms to make the runtime manageable, at the cost of estimation accuracy. It is necessary to develop techniques that mitigate the storage explosion problem for MAA to be useful in more complex applications. Table 3 summarizes the properties of the three error estimation methods.

Table 3. Overview of error estimation methods

Method	Simulation	MIA	MAA
Pros	Statistically accurate	Fastest, reasonably accurate	Fast if low complexity, tight bounds
Cons	Slow, bounds too tight	Loose bounds if variables correlate	Storage explosion if high complexity

VI. CONCLUSION

Imprecise hardware is a viable emerging technology for power reduction. In order to safely deploy IHW in a system, we need to evaluate the IHW-induced error statistics. This work proposes two analytic error estimation methods that offer significant speedup over simulation with reasonable estimation accuracy. Their limitations and possible solutions are also addressed. In future works we will continue to refine the models to improve estimation accuracy, extend the methods to model errors induced by voltage scaling and guardband reduction, and addressing the storage explosion problem.

REFERENCES

[1] N.R. Shanbhag, R.A. Abdallah, R. Kumar, D.L. Jones, "Stochastic computation," *Design Automation Conference*, pp. 859 –864, 2010

[2] S. Narayanan, J. Sartori, R. Kumar, D. Jones, "Scalable stochastic processors," *Design, Automation and Test in Europe*, pp. 335-338, 2010

[3] T. Austin, V. Bertacco, D. Blaauw, T. Mudge, "Opportunities and challenges for better than worstcase design," *Asia and South Pacific Design Automation Conference*, vol. 1, pp. I/2 – I/7, 2005

[4] N. Zhu, W.L. Goh, K.S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," *International Symposium on Integrated Circuits,* pp. 69–72, 2009

[5] A. Verma, P. Brisk, P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," *Design, Automation and Test in Europe*, pp. 1250–1255, 2008

[6] J. Huang, J. Lach, "Exploring the fidelity-efficiency design space using imprecise arithmetic," *Asia and South Pacific Design Automation Conference*, pp. 579-584 , 2011

[7] L.H. de Figueiredo, J. Stolfi, "Self-validated numerical methods and applications," *Brazilian Mathematics Colloquium, IMPA*, 1997

[8] E.P. Kim, R.A. Abdallah, N.R. Shanbhag, "Soft NMR: Exploiting statistics for energy-efficiency," *International Symposium on SOC*, pp. 52-55, 2009

[9] http://savannah.nongnu.org/projects/libaffa

[10] L.V. Kolev, "An improved interval linearization for solving non-linear problems," *Numerical Algorithms*, vol. 37, pp. 213-224, 2004