FLOATING-POINT ERROR ANALYSIS BASED ON AFFINE ARITHMETIC

Claire Fang Fang, Tsuhan Chen, Rob A. Rutenbar

Dept. of Electrical and Computer Engineering
Carnegie Mellon University
{ffang, tsuhan, rutenbar}@ece.cmu.edu

ABSTRACT

During the development of floating-point signal processing systems, an efficient error analysis method is needed to guarantee the output quality. We present a novel approach to floating-point error bound analysis based on affine arithmetic. The proposed method not only provides a tighter bound than the conventional approach, but also is applicable to any arithmetic operation. The error estimation accuracy is evaluated across several different applications which cover linear operations, non-linear operations, and feedback systems. The accuracy decreases with the depth of computation path and also is affected by the linearity of the floating-point operations.

1. INTRODUCTION

With the constant advance in VLSI technology, more and more floating-point signal processing algorithms are being ported from software solutions, with virtually infinite precision, to hardware solutions where precision is highly limited by the low-power constraint. Custom floating-point format for video, audio and speech applications and its benefit in power saving are studied in [1-3]. In order to prevent numerical catastrophe and assist design decision making, a method that can keep track of floating-point error is needed during the system development. Unlike the case of fixed-point arithmetic, where the round-off error could be modeled as a white noise sequence independent of the input signal, the round-off error in floating-point arithmetic is strongly correlated with the signal magnitude determined only at run-time, which complicates the error analysis.

Research in floating-point error analysis has been going on for about three decades. Previous publications take approaches that fall into two categories. One is to predict statistical property of the computed signal, namely error variance, given the system transfer function *H(z)* [5-9]. The other is to estimate the error bound based on floating-point error propagation models [10]. However, both of them have limitations when applied to the current scenario. In [5], a floating-point round-off error variance model is introduced for digital filters, which requires *H(z)* and the autocorrelation matrix of the inputs. More recently, a simplified model is proposed in [9] based on the same methodology. The main drawback of these statistical approaches is that the derivation of the model is only applicable to vector inner products, and is not scalable to a larger set of signal processing algorithms involving operations such as division, square, square root, etc. Further, knowing only the error variance is not enough to avoid numerical failures coming from the worst case error. For example, in the IEEE standard for the implementation of Inverse Discrete Cosine Transform in image and video processing, both the peak error and the mean square error are considered [11]. On the other hand, error bound estimation presented in [10] employs interval arithmetic (IA) to build round-off error bound propagation models for not only addition and multiplication, but also for division and square root. Although its accuracy is not shown in the paper, it is very likely to lead to unacceptable overestimation given the fact that IA performs poorly when signals have correlations among each other.

In this paper, we develop an error bound analysis method that not only gives a tighter bound than previous work regardless of correlations between signals, but also is applicable to more floating-point computations than just vector inner products. We present a novel approach based on affine arithmetic, a recent development in range arithmetic, and show its advantages over previous approaches in terms of accuracy and scalability.

The remainder of the paper is organized as follows. In Section 2, background on range arithmetic is briefly introduced. Based on the affine arithmetic (AA) model in range arithmetic, we present our AA-based floating-point error model in section 3. Section 4 provides experimental results to show the accuracy and applicability of the proposed method. More discussions about related issues are given in section 5. Finally, some concluding remarks are given in Section 6.

2. BACKGROUND – RANGE ARITHMETIC

Range arithmetic is widely used in approximate numerical computations. It also plays an important role in floating-point error analysis due to the following reasons. First, in order to keep track of the error bound of each quantity during computation, ideally, an accurate estimate of the quantity value is required, because the floating-point error bound strongly depends on the magnitude of the quantity [4]. However, such information is impossible to obtain prior to run-time. Therefore, a less accurate, but more practical feature - range of the quantity - is estimated and propagated through computations. Second, estimated error can be expressed as either a single value (bound), or a range. By choosing range as a representation of error, we are able to model the error propagation more precisely, with the help of range arithmetic.

2.1 Interval Arithmetic

Interval arithmetic (IA), also known as *interval analysis*, is invented in the 1960s by Moore [12] as a simple tool to solve range problems. The interval of quantity x is represented by $\bar{x} = [\bar{x}.lo, \bar{x}.hi]$, meaning that the "true" value of x is known to satisfy $x.lo \le x \le x.hi$.

For each operation f: $R^m \to R$, there is a corresponding range extension $\bar{f}$: $\bar{R}^m \to \bar{R}$. An important property of the range extension is the *fundamental invariant*:

$$x \in \bar{x} \;\Rightarrow\; f(x) \in \bar{f}(\bar{x})$$

The fundamental invariant guarantees that the operation output lies in the range estimated by the range extension. For example, the sum of two intervals $\bar{x}$ and $\bar{y}$ is computed as

$$\bar{z} = \bar{x} + \bar{y} = [\bar{x}.lo + \bar{y}.lo, \bar{x}.hi + \bar{y}.hi] \quad (1)$$

According to the fundamental invariant, the value of quantity z lies in the interval $\bar{z}$. Analogous formulas can be derived for multiplication, division, square root, and all other common mathematical functions [13].

The main problem of IA is overestimation, especially when intervals are correlated with each other. To illustrate the problem, suppose in (1) $\bar{x}$ = [-1, 1], $\bar{y}$ = [-1, 1], and the quantities x and y have the relation $y = -x$. According to (1), $\bar{z}$ = [-2, 2], while $z = x + y \equiv 0$! The effect of overestimation accumulates along the computation chain, and finally may result in range explosion.

2.2 Affine Arithmetic

Affine arithmetic (AA), or *affine analysis,* is developed as a solution to the overestimation in IA. It not only keeps track of intervals, but also preserves correlations between them. In affine arithmetic, a quantity x is represented by an *affine form* $\hat{x}$, which is a first-degree polynomial:

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + ... + x_n\varepsilon_n \quad \text{with } -1 \le \varepsilon_i \le 1 \qquad (2)$$

Each noise symbol ε_i stands for an independent component of the total uncertainty of the quantity x; the corresponding coefficient x_i gives the magnitude of that component. The source of the uncertainty may be either "external" (due to variation of the quantity, numerical approximation), or "internal" (due to arithmetic round-off or other numerical errors committed in the computation of $\hat{x}$) [13]. Similar to IA, affine arithmetic also has the fundamental invariant property.

For the linear operations $\hat{x} \pm \hat{y}, a \pm \hat{x}$, and $a\hat{x}$ on affine forms $\hat{x}, \hat{y}$ and real number a, the resulting affine forms are easily obtained by applying (2). For any other operation f: $R^m \to R$, the resulting function $f^*(\varepsilon_1,...\varepsilon_n)$ is no longer a linear combination of ε_i. In order to preserve the affine form of the result, we first select an approximate linear function $f^a(\varepsilon_1,...\varepsilon_n)$ according to a certain rule, e.g. *Chebyshev approximation theory,* then a new noise term ε_k indicating the approximation error is estimated and added to the final affine form.

The key feature of the AA model is that the same noise symbol may contribute the uncertainty of two or more quantities, indicating the correlations among them. This advantage of AA is especially noticeable in computations subject to range cancellation or of great arithmetic depth. In the example in Section 2.1, x and y have the following affine forms

$$\hat{x} = 0 + 1\varepsilon \;\text{ and }\; \hat{y} = -\hat{x} = 0 - 1\varepsilon$$

The resulting affine form $\hat{z} = \hat{x} + \hat{y} = 0$ perfectly agrees with the range that the quantity z actually falls in. To show the difference between IA and AA along a computation chain, we apply both on the Inverse Discrete Cosine Transform (IDCT), implemented according to the structure in [14] and fed by inputs generated in the range [-128, 128]. There are six stages along the computation path, including the inputs and the outputs. Fig. 1 shows the range for each stage in a particular path. Thanks to the extra information embedded in the affine form about the correlations, the ranges estimated by AA grow much slower than those by IA.

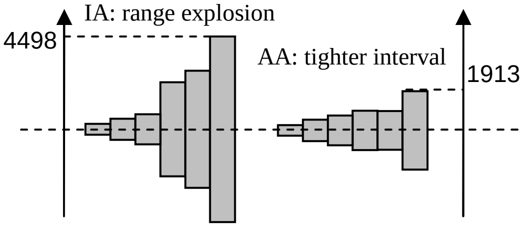

Fig. 1 Estimated range growth in a computation path in IDCT

3. FLOATING-POINT ERROR MODELS USING AFFINE ARITHMETIC

Floating-point representations are computer approximations of real numbers, with errors caused by input quantization, or rounding committed at each arithmetic operation. In this section, we first review a conventional model that people have been using to analyze floating-point error. We then develop a new model that nicely fits into the affine form for any arithmetic operation.

3.1 Conventional error models

Throughout the paper, we assume that floating-point numbers are stored in the form (sign)·μ·2^v, where v is called the *exponent*, and μ, with the value between 1 and 2, is called the *mantissa.*. For each input quantity x, x_f denotes its floating-point approximation. The notation $fl(\cdot)$ is the floating-point approximation for an operation. The approximations by input quantization and rounding (real rounding is assumed in the paper) are modeled as the following [4, 5]:

$$x_f = x(1+\alpha) \qquad (3)$$

$$fl(x \circ y) = (x \circ y)(1+\beta) \qquad (4)$$

, where the error variables α, β are usually assumed to be uniformly distributed within [-2^{-q}, 2^{-q}], where q is the number bits used in the mantissa.

3.2 AA-based error models

If the quantity x is distributed in a range and represented by an affine form $x = x_0 + x_1\varepsilon_1$, from (3) we can see that the error of x is bounded by max(|x|)·2^{-q}. In this case, x_f can be written in the affine form

$$\hat{x}_f = x_0 + x_1\varepsilon_1 + \max(|x|)\cdot 2^{-q}\cdot\delta_1,\quad \varepsilon_1,\delta_1\in[-1,1] \quad (5)$$

We call ε_1 the *variation symbol* and δ_1 the *error symbol* according to their different causes.

For any operation *f*: $R^m \to R$, the output quantity *z* can also have an affine form. Here we use a binary operation $z = f(x \circ y)$ to illustrate. According to (4), the error of *z* has a bound of max(|z|)·2^{-q}. Therefore the affine form for z_f is

$$\hat{z}_f = (\hat{x}_f \circ \hat{y}_f) + \max(|z|)\cdot 2^{-q}\cdot\delta,\quad \delta\in[-1,1], \qquad (6)$$

As discussed in the last section, the first term $(\hat{x}_f \circ \hat{y}_f)$ is either in a precise affine form, or an approximate affine form, depending on the linearity of the operation. Combining (5) and (6), for any quantity (input, output, or intermediate result) during floating-point computations, it can be represented in an affine form

$$\hat{u}_f = u_0 + \sum v_i\varepsilon_i + \sum w_i\delta_i,\quad \varepsilon_i,\delta_i\in[-1,1]$$

, where the variation symbol ε_i denotes the variations from all related input quantities, and the error symbol δ_i represents the errors from input quantization, rounding, and affine approximation.

If two or more quantities share the same error symbol, it is possible for them to be cancelled during computation. Hence, error analysis using AA is more accurate than using IA. Differences between them on the same example, IDCT, are shown in Fig. 2. Six vertical bars are the estimated error range for each stage along a computation path. The error ranges estimated by AA are much tighter than those by IA.

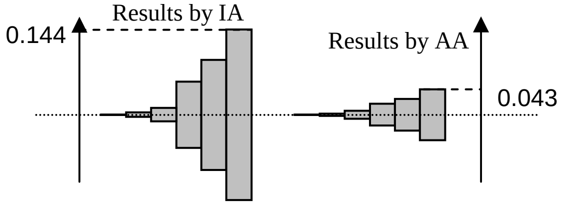

Fig. 2 Estimated error ranges in a computation path in IDCT

4. EXPERIMENTAL RESULTS

Based on the AA floating-point error model, we build a C++ library that automatically keeps track of the floating-point errors. We test the applicability and accuracy of the proposed error analysis method on several frequently-used signal processing tasks, among which Walsh-Hadmard Transform (WHT), FIR and IDCT are all essentially linear transforms, or vector inner products. To show that the AA error analysis is also applicable to non-linear operations, we conduct an experiment on the Gaussian distribution distance calculation task ($y = \sum (x - m_i)^2 / v_i$) commonly used in pattern recognition algorithms. In addition, IIR filter is evaluated as an example of feedback systems.

4.1 Accuracy

To evaluate the AA-based error analysis, we define

Accuracy = real error / estimated error

, where the real error is obtained by measuring the maximum difference between a 64-bit double precision result and a 16-bit custom floating-point result, simulated using *Cmufloat* custom floating-point library [1, 2].

Since the estimated error provides an error bound, it is always expected to be larger than the real error. The closer this measurement is to 1, the more accurate the error analysis.

	# of adds	# of mults	AA accuracy
WHT4	3	0	0.958
WHT64	63	0	0.799
FIR (4-tap)	3	4	0.777
FIR (25-tap)	24	25	0.564
IDCT8	13	6	0.473
Dist. calc.	11	4	0.39

Table 1. Accuracy of AA-based error analysis

In Table 1, we show the accuracy of six benchmarks. Their accuracy is affected by the number of arithmetic operations along the computation path and the linearity of the operations. Comparing WHT and FIR, we can see that multiplications reduce the accuracy more than additions because of the affine approximations taken in multiplications [13]. Gaussian distance calculation has the lowest accuracy since it involves non-linear operations.

We also obtain the error analysis accuracy of IDCT using IA-based method. The AA-based error analysis is significantly better (124%) than the IA-based method due to a large number of correlations. For example, in the IDCT structure shown in Fig. 3, two quantities denoted by the grey dots are correlated because they are both dependent on x_1. This is where overestimate in the IA-based method takes place.

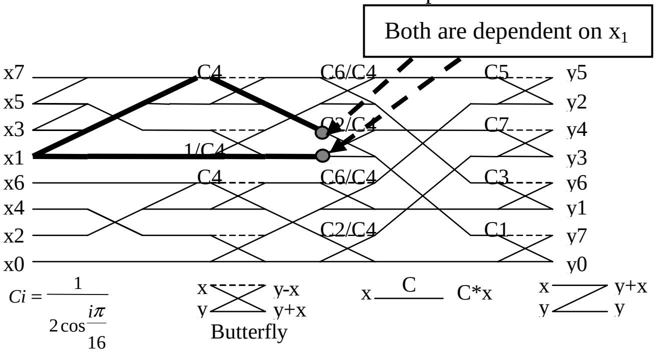

Fig. 3 Structure of IDCT

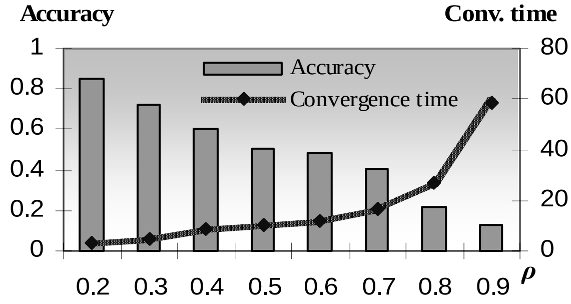

Fig. 4 Accuracy of error analysis on IIR filters

4.2 Applying to feedback systems

As the error estimation accuracy decreases with the depth of the computation chain, one may ask whether this method is suitable for a feedback system which can be viewed as having infinite computation depth. We conduct the experiment on an IIR filter with poles inside the unit circle. It's specified by $y_n = x_n - (-\sqrt{2}\rho y_{n-1} + \rho^2 y_{n-2})$. We find that after a sufficient amount of time, the estimated error converges (the error range stops growing). The accuracy of the asymptotic error and the convergence time depends on the location of the poles, as depicted in Fig. 4. The larger the ρ, the past computations have more influence on the present computation. Therefore it takes longer to converge and results in worse accuracy.

5. DISCUSSION

In all the experiments above, the inputs are generated according to uniform distribution assumption. A different distribution of real application data will certainly worsen the error estimation. In the same IDCT experiment, if the inputs are from a Gaussian distribution, the accuracy is decreased from 0.473 to 0.304. The performance can be boost by feeding a histogram, not just a range of the inputs. The result is a histogram of error in this case. The final error bound is chosen to cover 90% confidence interval. From Table 2, we show the accuracy is improved to 0.987 by having a 5-bin histogram. However, the runtime grows exponentially with the number of bins. Hence, it's not worth having even more complicated histogram.

	Accuracy	runtime (sec)
Simple analysis	0.304	0.03
3-bin histogram	0.822	5
5-bin histogram	0.987	13889

Table 2. AA error analysis with histogram inputs

The AA-based error estimation can also be used in floating-point custom format optimization. In the optimization algorithm proposed in [2], it relies on simulation to evaluate the goodness of the current format setting. This time-consuming step can be replaced by static error estimation. Although it does not have 100% accuracy, the final format can be achieved by a little local tuning after the optimization. Further, by using AA-based error estimation, it is very easy to determine which quantity contributes the most in the final error, and hence where to allocate more bits in the next iteration during the optimization.

Finally, we want to point out that similar fixed-point error models can also be built upon affine arithmetic. Since the quantization and round-off error in fixed-point arithmetic is independent on the magnitude of the quantity, the AA-based fixed-point error models will be simpler than floating-point arithmetic.

6. CONCLUSION

In this paper we have described a novel approach to analyze floating-point propagation errors using affine arithmetic. A general affine form is developed for floating-point error bound estimation independent of the type of operations. We have shown that the advantage of this method is significant when a large number of correlations are involved in the intermediate computations. However, the accuracy decreases with the computation complexity and is also affected by the linearity of the computations. It is also applicable to feedback systems, with convergence time and accuracy dependent on the pole positions. In case of non-uniform distributed inputs, the accuracy can be boost by feeding histogram inputs.

Armed with the AA-based error model, we can integrate error analysis and custom format optimization together for both floating-point and fixed-point arithmetic, and ultimately provide a powerful developing environment for signal processing algorithms.

7. REFERENCE

[1] F. Fang, T. Chen, R. Rutenbar, "Lightweight Floating-Point Arithmetic: Case Study of Inverse Discrete Cosine Transform", EURASIP Journal on Signal Processing, Special Issue on Applied Implementation of DSP and Communication Systems, 2002

[2] F. Fang, T. Chen, R. Rutenbar, "Floating-point Bit-width Optimization for Low-Power Signal Processing Applications," Proc. International Conf. on Acoustic, Speech and Signal Processing, pp 3208-3211, 2002

[3] R. Chamberlain, Y. H. Chew, V. DeAlwis, et al., "Power consumption of customized numerical representations for audio signal processing", Sixth Annual Workshop on High Performance Embedded Computing, pp 41-43, 2002

[4] P. H. Sterbenz, "Floating-point computation", Prentice-Hall

[5] B. Liu, T. Kaneko, "Error analysis of digital filters realized with floating-point arithmetic", Proc. IEEE, vol. 57, pp 1735-1747, Oct.1969

[6] C. Weinstein, A. V. Oppenheim, "A comparison of roundoff noise in floating point and fixed point digital filter realizations", Proc. IEEE, vol. 57, pp 1181-1183, Jun. 1969

[7] T. L. Laakso, L. B. Jackson, "Bounds for floating-point roundoff noise", IEEE Trans. Circ. and Syst.-II: Analog and Digital Signal Processing, vol. 41, pp 424-426, Jun. 1994

[8] C. Tsai, "Floating-point roundoff noises of first- and second-order sections in parallel form digital filters", IEEE Trans. Circ. and Syst.-II: Analog and Digital Signal Processing, vol. 44, pp 774-779, Sep. 1997

[9] B. D. Rao, "Floating point arithmetic and digital filters", IEEE, Trans. Signal Processing, vol 40, pp 85-95, Jan, 1992

[10] W. Kramer, "A prior worst case error bounds for floating-point computations", IEEE Trans. Computers, vol. 47, pp 750-756, Jul. 1998

[11] "IEEE-standard specifications for the implementations of 8X8 inverse discrete cosine transform," IEEE Std 1180-1990, Institute of Electrical and Electronics Engineers, Inc, 1990

[12] R. E. Moore. "Interval analysis", Prentice-Hall, 1966

[13] L. H. de Figueiredo and J. Stolfi, "Self-validated numerical methods and applications", Brazilian Mathematics Colloquium monographs. IMPA, Rio de Janeiro, Brazil, Jul. 1997

[14] A. Artieri, O. Colavin, "A chip set core for image compression," IEEE Trans. on Consumer Electronics, vol. 36, pp.395 -402, Aug. 1990