Error Analysis and Precision Estimation for Floating-Point Dot-Products using Affine Arithmetic

Thang Viet Huynh
Graz University of Technology, Graz, Austria
Signal Processing and Speech Communication Laboratory
thang.huynhviet@tugraz.at

Manfred Mücke
University of Vienna, Vienna, Austria
Research Lab Computational Technologies and Applications
manfred.muecke@univie.ac.at

***Abstract*—One challenging task for VLSI and reconfigurable system design is the identification of the smallest number format possible to implement a given numerical algorithm guaranteeing some final accuracy while minimising area used, execution time and power. We apply affine arithmetic, an extension to interval arithmetic, to estimate the rounding error of different floating-point dot-product variants. The validity of the estimated error bounds is demonstrated using extensive simulations. We derive the analytical models for rounding errors over a wide range of parameters and show that affine arithmetic with a probabilistic bounding operator is able to provide a tighter bound compared to conventional forward error analysis. Due to the tight bounds, minimum mantissa bit width for hardware implementation can be determined and comparison of different dot-product variants is possible. Our presented models allow for an efficient design space exploration and are key to specialised code generators.**

***Keywords*-Floating-Point; Dot Product; Error Analysis; Bit-width estimation; Affine Arithmetic;**

I. Introduction

Advanced signal processing applications and scientific computing applications employ IEEE floating-point (FP) arithmetic to achieve high accuracy, large dynamic range and efficient application development. Floating-point operands consist of a sign bit, an exponent field and a mantissa field. Hardware cost is typically dominated by the size of the mantissa field. Knowledge on the minimal mantissa bit width (precision) is therefore vital to achieving efficient hardware implementations of floating-point algorithms.

On reconfigurable hardware, reducing the precision (mantissa bit width) of floating-point operands directly translates into increased parallelism, and if exploited correctly, to increased performance. In contrast to static architectures' short-vector SIMD units, which typically provide a linear throughput increase with reduced precision, FPGAs allow for superlinear gains in parallelism with reduced precision (see [1] for an introduction to area complexity of arithmetic operations and our recent work [2], [3] for experimental evaluation of custom-precision floating-point dot-product throughput on FPGAs and hybrid reconfigurable CPUs, respectively). However, an open research question is the estimation of optimal mantissa bit width, i.e. identifying the most suitable precision for implementation of a given algorithm, retaining the result's required accuracy while minimising a cost function including hardware resources, execution time and power consumption.

In this paper, we aim at deriving a relation between the user-specified final accuracy and a uniform minimum mantissa bit width for sequential and parallel floating-point dot product implementations over varying vector size. Similar to [4], our work makes use of probabilistic floating-point error analysis based on affine arithmetic (AA) in order to derive realistic bounds of the rounding error. In contrast to existing work, we consider a *general floating-point dot product*, investigate the impact of *implementation variants* and explore a fine-grained design-space considering precision and vector length.

The scientific contributions of this work are:

- Demonstration of an efficient and reliable Matlab-based framework for static floating-point error analysis of feed-forward algorithms based on AA modeling and its application on experimentally determining AA rounding error bounds of different floating-point dot-products.
- Presentation of analytical equations modeling AA rounding error bounds of floating-point dot-products as a function of vector length, precision and input range.
- Mantissa bit width estimation and comparison of different floating-point dot-product implementations.

The remainder of this paper is organized as follows. Section II summarises related work, followed by some background on floating-point error analysis in Section III. Section IV presents our Matlab-based framework for performing AA-based floating-point error analysis. Section V experimentally derives the AA probabilistic bounds for floating-point dot-products. The analytical equations modeling AA probabilistic error bounds are derived in Section VI. Section VII presents AA bounds for precision estimation and design space exploration of different dot-product implementations. Finally, Section VIII gives the conclusion and details future work.

II. Related Work

Error analysis is concerned with understanding and estimating the effect of parameter variations and rounding error on an algorithm's final result. In this work, we focus on the effects of rounding error. Error analysis techniques can be static or dynamic [5]. Static error analysis is based on analytical derivations, often providing conservative estimates. Dynamic error analysis is based on simulations and gives typically more realistic error estimates but requires typically long simulation times and the validity depends on the chosen test data.

Methods for static floating-point error analysis comprise classical forward error analysis, backward error analysis and range-based analysis techniques. Classical forward error analysis [6] often provides very pessimistic error estimates rendering this method impractical for bit width estimation. A

comprehensive discussion on classical forward error analysis for different floating-point algorithms can be found in [6].

Examples of range arithmetic based methods for error analysis that make use of either *interval arithmetic* or *affine arithmetic* are given in [5], [7], [8], [4]. One example of interval arithmetic for error anlysis is the work of Krämer [7]. Fang *et al.* [8], [4] employed affine arithmetic [9], a recently developed mathematical tool for refined range analysis, to model rounding error of floating-point algorithms. The key idea is to use an AA-based probabilistic bounding operator to obtain a reliable estimate of the final error bound. Lee *et al.* [5] presented MiniBit, an automatic and static approach based on AA for optimizing bit width of fixed-point feedforward designs. To our knowledge, in existing literature, there is no work that applies AA for performing error analysis of the general floating-point dot-product over a wide range of vector length, precision used, and numerical range.

III. Floating-Point Rounding Error Analysis

Floating-point arithmetic [10] is the standard approach for approximating real number arithmetic in modern computers. A floating-point number uses three fields to represent a real number r: a sign bit s, a biased exponent e and a mantissa f. Given s, e, f and the bias, using IEEE-754 conventions, a normalised binary floating-point numer evaluates to $r = (-1)^s \cdot 2^{e-\text{bias}} \cdot 1.f$. Every real number $x \in \mathbb{R}$ can be approximated by a floating-point representation $\hat{x} = fl(x)$

$$\hat{x} = fl(x) = x + x \cdot \delta, \quad |\delta| \leq u, u = 2^{-p} \tag{1}$$

with a relative error δ no larger than the unit roundoff $u = 2^{-p}$, where p is the precision (number of binary digits to represent the mantissa $1.f$) [6].

Numerical forward rounding error analysis deals with the question how the rounding operations of some given algorithm implemented in finite-precision arithmetic affects the final numerical result. Methods used include classical forward error propagation, interval arithmetic, and affine arithmetic.

a) Classical Error Propagation: Classical error propagation as used by [6] tries to derive symbolic expressions in $\delta, |\delta| \leq u$ typically concentrating on higher order terms.

b) Interval Arithmetic: Interval arithmetic (IA) [9] is a range-based model for numerical computation where each real quantity x is represented by an interval $\bar{x} = [x_{min},\ x_{max}]$. Those intervals are then added, subtracted, multiplied, etc., in such a way that each computed interval $\bar{x}$ is guaranteed to cover the unknown value of the real quantity x. In error analysis, IA can be used to derive worst-case rounding error bounds. The major disadvantage of IA is that it cannot capture correlations between variables, thereby leading to very pessimistic bounds for long computational chains.

c) Affine Arithmetic: In affine arithmetic (AA) [9] the uncertainty of any real quantity x is represented by an affine form $\hat{x} = g(\epsilon_1, ..., \epsilon_n)$, which is a first-order polynomial expressed as

$$\hat{x} = x_0 + x_1\epsilon_1 + ... + x_n\epsilon_n = x_0 + \sum_{i=1}^{n} x_i\epsilon_i, \tag{2}$$

where coefficient x_0 is the central value, while other coefficients x_i are partial deviations ($i = 1, 2, ..n$), and ϵ_i is the noise symbol whose value is assumed to lie in $[-1, 1]$.

AA can be used to model the rounding error of arithmetic operations [4]. AA is an improvement of IA, in the sense that it can keep track of correlations between quantities, thereby potentially leading to more accurate estimated ranges compared to the ones computed by IA. The limitation of AA is that linearity is not guaranteed for non-affine arithmetic operations, potentially leading to inaccurate error estimates.

IV. Matlab-based Framework for Floating-point Error Analysis using Affine Arithmetic

To derive the affine forms and to perform our experiments, we created a framework for automated AA-based floating-point error analysis using probabilistic bounding operators [4] in Matlab. Affine forms can become very complex expressions after multiple arithmetic operations have been applied to the input form. In contrast to symbolic frameworks, we represent AA models of floating-point operands as numerical row vectors. Computed result of each atomic floating-point operation is obtained by transferring AA forms of input operands to the corresponding function which evaluates the operation following affine arithmetic. The whole computational process is vector-based and can therefore be performed efficiently in Matlab. For verification of the AA error models, our framework integrates arbitrary-precision arithmetic via the GNU MPFR Library[1] (version 3.0.0). MPFR C functions are compiled into MEX files and called from Matlab.

V. AA Error Bounds for Floating-Point Dot-Products

The main limitation of simulation-based error analysis are the very long simulation times required. We investigate whether AA-based error analysis allows for deriving in less time error bounds comparable to the ones reported by simulation. In the following, we compare *error bounds* and *calculation time* of AA-based and simulation-based error analyses. For demonstration, we estimate the rounding errors of two dot-product implementation variants.

A. Dot-Product

Given two column vectors $\mathbf{x}$, $\mathbf{y}$ of length n: $\mathbf{x} = [x_1, ..., x_n]^T$, $\mathbf{y} = [y_1, ..., y_n]^T$, the dot-product of $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i \cdot y_i = x_1y_1 + ... + x_ny_n. \tag{3}$$

Different dot-product implementation variants result in different rounding errors [6]. In this paper, we perform error analysis for two different floating-point dot-product architectures: a sequential dot-product and a parallel (or binary-tree) dot-product [2] (see Fig. 1), with the use of basic floating-point operations, i.e. multiplication and addition, only.

[1] http://www.mpfr.org/

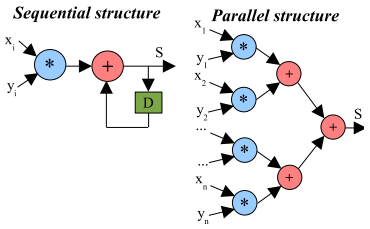

Fig. 1. Dot-product implementation variants

B. Experiment setup

The chosen numerical range is $[-128, 128]$. The precision p (mantissa bit width) of floating-point operands varies in unit step from 20 bits up to 53 bits (double-precision). The exponent bit width is fixed as 11 bits. The vector length n of two input vectors ranges from 100 to 1000 with step size of 100. The chosen confidence interval [4] for AA probabilistic bounding operator is $K = 3$ (corresponding to a probability of 99.7% the rounding error occurring will fall within the AA probabilistic bound estimated). To evaluate the accuracy of the AA probabilistic rounding error bound, we compare it against the maximum rounding error obtained by running Monte-Carlo simulation over a sufficiently large number of 10^6 samples for each pair of (p, n). For each sample, we compute the difference between the finite-precision result and its infinite-precision or accurate version, for which a 80-bit mantissa format was used. Among 10^6 computed differences, the maximum value is then chosen and considered as the maximum rounding error. The *Over-Estimate Ratio* (*OER*) is used to evaluate the reliability of AA probabilistic bound and defined as

$$OER = \frac{AA\ probabilistic\ bound}{Maximum\ error\ by\ simulation}. \quad (4)$$

All the simulations are run on an AMD Athlon 3800 Dual Core desktop CPU with clock frequency of 2.0 GHz.

C. Results

Rounding error bounds: Fig. 2 presents the contour maps of the maximum rounding error (dashed lines) obtained by simulation and the AA probabilistic rounding error bound (solid lines) of sequential dot-product. We observe that regardless of the vector length or the precision, AA probabilistic bounds are very close to realistic errors via simulation and the OER_{seq} of AA probabilistic bound of sequential dot-product is always within the range: $1.2 \leq OER_{seq} \leq 2.2$. For parallel dot-product, Fig. 3 plots the experimental result. Similarly, the OER_{par} of AA probabilistic bound for parallel dot-product is always within the range: $1.3 \leq OER_{par} \leq 4.1$.

What we learn from Fig. 2 and Fig. 3 is that the AA probabilistic bound is a reliable and efficient estimate, corresponding to the overestimation of the mantissa bit width at most $log_2(2.2) \approx 1$ bit and $log_2(4.1) \approx 2$ bits for sequential- and parallel dot-product implementations, respectively.

Calculation time: Table I reports the time (in seconds) required for approximating maximum error via simulation of 10^6 samples, and for estimating AA probabilistic bound for single-precision dot-products with the smallest vector length of 100. In terms of calculation time, the AA-based approach achieves a speedup of about three to four orders of magnitude compared to the simulation-based approach.

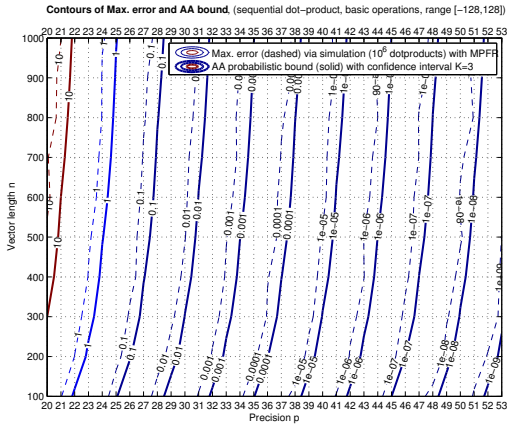

Fig. 2. Contours of maximum rounding error (dashed lines) and AA probabilistic bound (solid lines) for sequential floating-point dot-product.

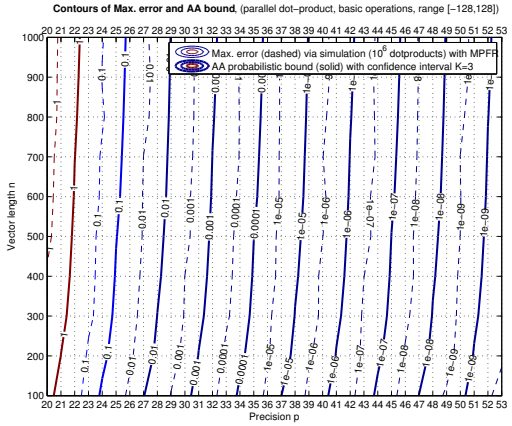

Fig. 3. Contours of maximum rounding error (dashed lines) and AA probabilistic bound (solid lines) for parallel floating-point dot-product.

VI. Analytical Equations of Rounding Error Bounds for Floating-Point Dot-Products

In this section, we make use of AA probabilistic bounding operator to derive some analytical expressions for error bounds of floating-point dot-products as a function of numerical range, precision p and vector length n. In the followings, we assume that each element of two input vectors is uniformly distributed in a symmetric range, i.e. $x_i, y_i \in [-a, +a]$.

A. Conventional forward error bound

The conventional forward error bound for a sequential dot-product [6] is presented as $|\mathbf{x}^T\mathbf{y} - fl(\mathbf{x}^T\mathbf{y})| \leq \frac{nu}{1-nu}\sum_{i=1}^{n}|x_i y_i|$, where $u = 2^{-p}$ is the unit roundoff. We use the maximal value of x_i, y_i to estimate the conventional bound, thereby $\sum_{i=1}^{n}|x_i y_i| \approx na^2$. The

TABLE I
Execution time [s] of rounding error estimation for single-precision dot product ($n = 100$)

	Simulation [s] (10^6 samples)	AA [s]	Speedup
Sequential impl.	230	0.043	5.34×10^3
Parallel impl.	3041	0.195	1.56×10^4

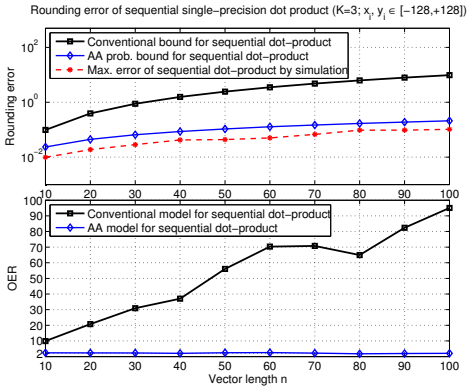

Fig. 4. Rounding error of sequential single-precision dot-product

conventional forward error bound of sequential dot-product is, therefore, estimated as

$$|\mathbf{x}^T\mathbf{y} - fl(\mathbf{x}^T\mathbf{y})| \leq \frac{nu}{1-nu} \cdot na^2. \quad (5)$$

B. Analytical expressions of AA probabilistic bounds

Similar to [4], we represent each floating-point quantity by a numerical range component and a rounding error component and apply the probabilistic bounding operator to derive analytically rounding error bounds for sequential- and parallel dot-products. The AA probabilistic rounding error bound for sequential floating-point dot-product $B_{prob,seq}$ is calculated as

$$B_{prob,seq} = 2^{-p} \cdot K \cdot \frac{\sqrt{2}}{6} \cdot a^2 \cdot \sqrt{18n + K^2(n^2+n-2)}. \quad (6)$$

For parallel floating-point dot-product using a binary-tree adder (with $\lceil log_2(n) \rceil$ adding stages), the AA probabilistic rounding error bound $B_{prob,par}$ is estimated as

$$B_{prob,par} = 2^{-p} \cdot K \cdot \frac{\sqrt{2}}{6} \cdot a^2 \cdot \sqrt{18n + 2K^2\lceil log_2(n) \rceil n}. \quad (7)$$

C. AA error model vs. Conventional error model

Fig. 4 compares rounding error bounds and corresponding *OERs* of single-precision ($p = 24$) sequential dot-product derived by using conventional model in (5) and AA probabilistic bouding operator in (6). The *OER* of conventional bound continuously increases with increasing vector length, making it useless for mantissa bit width estimation. The *OER* obtained by AA modeling remains almost unchanged (i.e., between 1 and 2) over whole range of vector length, which reveals that affine arithmetic with a probabilistic bounding operator is able to provide tighter bounds, compared to conventional error model, for floating-point error analysis.

VII. Precision Estimation and Comparison of Different Dot-Product Variants

Due to the tight bounds derived by using AA, precision estimation (minimum mantissa bit width) and comparison of different dot-product implementations are possible.

A. Precision estimation

Given a user-specified accuracy, the smallest precision for implementation of sequential- and parallel dot-products *i)* can analytically be identified by using analytical models in (6) and (7), or *ii)*, can experimentally be determined by evaluating the respective affine forms using our framework (as shown in Fig. 2 and Fig. 3).

TABLE II
Required precision for dot-product implementations to achieve a final accuracy of 10^{-7} ($x_i, y_i \in [-128, 128]$)

Vector length n	100	200	300	500	1000
Sequential impl. [bits]	45	46	47	48	49
Parallel impl. [bits]	44	45	45	45	46
Difference [bits]	**1**	**1**	**2**	**3**	**3**

B. Comparison of different dot-product implementations

Given the same numerical range and working precision, it can easily be proved from (6) and (7) that parallel dot-product implementation provides more accurate final result. Furthermore, the error bounds in Fig. 2 and Fig. 3 allow for a visual comparison of different dot-product implementations with respect to final accuracy and smallest precision required (see Table II). The superior accuracy achieved by a parallel dot-product, however, comes at the cost of more resources required for employing a parallel implementation.

VIII. Conclusion and Outlook

In this work, we have shown that affine arithmetic with a probabilistic bounding operator is able to provide a tighter error bound compared to conventional forward error analysis. More importantly, we have derived analytical models for rounding errors of different floating-point dot-product variants over a wide range of parameters. Our presented models are key to efficient design space exploration and to specialised code generators. Future work will focus *i)* on coupling estimated bit width with hardware resource usage models and *ii)* on modelling of more complex non-affine arithmetic operations.

Acknowledgment

This work has been carried out in the context of the Austrian National Research Network "Signal and Information Processing in Science and Engineering" (NFN-SISE). The authors gratefully acknowledge funding by the Austrian Science Fund (FWF) through project S10608 (NFN-SISE) and the Austrian Academic Exchange Service (ÖAD).

References

[1] B. Parhami, *Computer arithmetic: algorithms and hardware designs*. Oxford, UK: Oxford University Press, 2000.

[2] M. Mücke, B. Lesser, and W. N. Gansterer, "Peak performance model for a custom precision floating-point dot product on FPGAs," in *3rd Workshop on UnConventional High Performance Computing*, 2010.

[3] T. V. Huynh, M. Mücke, and W. N. Gansterer, "Native double-precision LINPACK implementation on a hybrid reconfigurable CPU," in *18th Reconfigurable Architectures Workshop (RAW 2011)*. IEEE, May 2011.

[4] C. F. Fang, "Probabilistic interval-valued computation: representing and reasoning about uncertainty in dsp and vlsi design," Ph.D. dissertation, Pittsburgh, PA, USA, 2005.

[5] D. U. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides, "Accuracy-Guaranteed Bit-Width Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1990–2000, October 2006.

[6] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[7] W. Kramer, "A priori worst case error bounds for floating-point computations," *IEEE Trans. on Computers*, vol. 47, no. 7, pp. 750–756, 1998.

[8] C. F. Fang, R. A. Rutenbar, M. Püschel, and T. Chen, "Toward efficient static analysis of finite-precision effects in DSP applications via affine arithmetic modeling," in *DAC '03: Proceedings of the 40th annual Design Automation Conference*. ACM, 2003, pp. 496–501.

[9] J. Stolfi and L. H. Figueiredo, "Self-validated numerical methods and appications," 1997.

[10] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.