Testing the Sharpness of Known Error Bounds on the Fast Fourier Transform

Nicolas Brisebarre
CNRS, LIP, Université de Lyon
Lyon, France
Email: nicolas.brisebarre@ens-lyon.fr

Jean-Michel Muller
CNRS, LIP, Université de Lyon
Lyon, France
Email: jean-michel.muller@ens-lyon.fr

Joris Picot
ENS de Lyon, LIP, Université de Lyon
Lyon, France
Email: joris.picot@ens-lyon.fr

***Abstract*—The computation of Fast Fourier Transforms (FFTs) in floating-point arithmetic is inexact due to roundings, and for some applications it can prove very useful to know a tight bound on the final error. Although it can be almost attained by specifically built input values, the best known error bound for the Cooley-Tukey FFT seems to be much larger than most actually obtained errors. Also, interval arithmetic can be used to compute a bound on the error committed with a given set of input values, but it is in general considered hampered with large overestimation. We report results of intensive computations to test the two approaches, in order to estimate the numerical performance of state-of-the-art bounds. Surprisingly enough, we observe that while interval arithmetic-based bounds are overestimated, they remain, in our computations, tighter than general known bounds.**

Keywords: Fast Fourier Transform (FFT), Floating-Point arithmetic, Error analysis.

I. Introduction

The Fast Fourier Transform (FFT) is a ubiquitous tool in many areas of science and engineering. Although it is known [1] to be a fairly accurate algorithm, several applications require a very careful error analysis. A typical example we are interested in is the multiplication of very large integers, or of polynomials of very large degree with integer coefficients, using the Schönhage-Strassen algorithm introduced in [2] or a variant of it: since we know that the exact final values (either digits or polynomial coefficients) are integers, if we are certain that the final error of the (inherently inaccurate) calculation of them in floating-point arithmetic through Fourier transforms is less than $1/2$, we can retrieve them. In such cases, the classical error bounds of the form $\alpha\, u + O(u^2)$ that are omnipresent in numerical analysis (see later on for the definition of u) do not suffice: we need to be *certain* that the error is not larger than a given threshold. For the very same reason, probabilistic error bounds are not considered here, although they are useful in other contexts.

To that purpose, we consider two possible approaches:

1) use *global*, *a priori* error bounds, i.e., valid for all possible input values, such as the ones given in [3], and then compute FFTs in conventional floating-point arithmetic (if the global bound is less than the threshold);
2) by computing the FFTs in interval arithmetic (IA) [4], [5], obtain *on the fly* a *local* error bound (i.e., valid only for the considered input value).

The first approach is simple and fast. Its main drawback is that by nature a global, uniform, error bound is in general much larger than the actual error we encounter in practice: we may therefore (when the bound is larger than the threshold) erroneously consider, for a given input value, that the floating-point calculation will not be accurate enough. The second approach is by nature significantly slower (the ratio depends much on the environment: assuming the precision p is the same—see definition below—interval arithmetic is between around 2.5 and 10 times slower [6], [7] than floating-point arithmetic), and there are two antagonistic effects to consider:

- since the computed error bound is *local* there is hope of obtaining a bound significantly smaller than the global bound;
- on the other hand, the tendency of IA to overestimate the size of intervals may result in computed local error bounds significantly larger [8] than the actual error one would obtain by using floating-point arithmetic with the same input values.

To choose which of the two approaches should be used for a given problem, we need to know which of these antagonistic effects prevails over the other one, i.e., we need to compare the best known global bounds with local bounds estimated with interval arithmetic. This is the major purpose of this paper.

The literature has a long history [3] of successively-refined global bounds for the Cooley-Tukey FFT algorithm [9] in fixed-point or floating-point arithmetic, that started [10] just one year after Cooley and Tukey's seminal paper. An excellent recent overview of Fourier-related transforms is given in [11].

In [3], the authors present the state-of-the-art global error bounds. They also build a bad case, i.e., an input vector for the FFT for which particularly large errors are obtained, showing that their global bounds cannot be refined much more.

In the following, we compare actually attained errors of the Cooley-Tukey algorithm in binary floating-point arithmetic with global and local bounds obtained through IA, by means of intensive numerical computations.

This work is partly sponsored by the NuSCAP (ANR-20-CE48-0014) project of the *Agence Nationale de la Recherche.*

II. Presentation and notation

A. Floating-point arithmetic

We assume that we compute Fourier Transforms in binary, precision-p, floating-point (FP) arithmetic, assuming an unbounded exponent range (hence our results apply to usual FP arithmetics such as the ones specified by the IEEE 754 Standard [12] provided that underflow and overflow do not occur). Throughout this paper, a FP number is a number of the form $x = \frac{M}{2^{p-1}} \cdot 2^e$, where $|M| \leq 2^p - 1$ is an integer. Typical values of p are 24 (for binary32/single-precision arithmetic) and 53 (for binary64/double-precision arithmetic). Since the exact sum and product of two FP numbers are not, in general, FP numbers, they must be rounded. Assuming rounded-to-nearest operations (which is the default in IEEE-754 arithmetic), the implementations $\oplus$ and $\otimes$ of the addition and multiplication are such that (barring underflow/overflow) for any FP numbers x and y,

$$\begin{aligned} |(x \oplus y) - (x + y)| &\leqslant u\,|x + y|\,, \\ |x \otimes y - x \times y| &\leqslant u\,|x \times y|\,, \end{aligned} \tag{1}$$

where $u = 2^{-p}$. Eq. (1) is sometimes called the "standard model" of FP arithmetic, and a large part of numerical error analysis is based on it [1]. In the following, $\mathbb{F}$ is the set of the FP numbers, and $\mathbb{G}$ is the set of the complex numbers whose real and imaginary parts are FP numbers.

B. Discrete Fourier Transform and Notations

For any positive integer N, the discrete Fourier transform takes $X = (x_0, \ldots, x_{N-1}) \in \mathbb{C}^N$ and uses the N-th roots of unity $\omega_N^k = \exp\left(i\, 2\pi \frac{k}{N}\right)$ to return $Y \in \mathbb{C}^N$ defined by

$$y_k = \textstyle\sum_{j=0}^{N-1} x_j\, \omega_N^{-jk}.$$

An FFT algorithm takes an input vector $X \in \mathbb{G}^N$ and uses FP approximations of ω_N^k and (rounded) arithmetic operations to return a FP approximation of Y, noted $\hat{Y} \in \mathbb{G}^N$. Our main concern is to bound the error $|\hat{Y} - Y|$. As there are several possible variants of the FFT algorithm, we consider the one presented in Fig. 1. We assume that N is a power of two.

C. Different measures of FFT errors

The input and output values of the FFT belong to $\mathbb{C}^N$. In that domain, different norms can be used to express distances between vectors (and therefore errors). It is therefore important to precisely establish which norms are used, to avoid confusions. Unfortunately, the "natural norm" for Fourier transforms (i.e., the one for which calculations are easiest) is not the norm that most interests us for our applications. The natural norm for FFT is the 2-norm, used to derive the *2-norm relative error:*

$$\begin{aligned} e_2 &= \|\hat{Y} - Y\|_2 / \|Y\|_2, \\ \text{where } \|Z\|_2 &= \sqrt{|z_0|^2 + \cdots + |z_{N-1}|^2}. \end{aligned}$$

However, the 2-norm relative error does not tell us much about the accuracy of a particular coefficient. To get that, we prefer using the following *component-wise ∞-norm input-scaled absolute error:*

$$\begin{aligned} e_\infty^\perp &= \|\hat{Y} - Y\|_\infty^\perp / \|X\|_\infty^\perp, \\ \text{where } \|Z\|_\infty^\perp &= \max_{k=0}^{N-1} \left\{ \max\left\{ |\mathfrak{Re}(z_k)|\,, |\mathfrak{Im}(z_k)| \right\} \right\}. \end{aligned} \tag{2}$$

The choice of scaling by $\|X\|_\infty^\perp$ comes from the fact that it makes $e_\infty^\perp$ comparable to e_2 and, since $\|X\|_\infty^\perp$ is known exactly, an absolute error bound can easily be obtained from (2). In the literature, many results are on e_2 but few on $e_\infty^\perp$. Hopefully, one can use:

$$e_\infty^\perp \leqslant \sqrt{2N}\, e_2, \tag{3}$$

to compare $e_\infty^\perp$ with e_2. Note that for large integer multiplication applications, $\|X\|_\infty^\perp$ cannot be arbitrarily large: it is straightforwardly bounded using the value of the largest allowed digit.

```
Function reverse(n, j)
    return Σ_{k=0}^{n-1} ((j&(2^n − 1) ≫ k)  mod 2) ≪ (n − 1 − k)
Function OneStep(x, k, n)
    bs = 2^k;                                   /* Block size */
    N_blocks = 2^n/bs;                          /* Number of blocks */
    for bi from 0 to N_blocks − 1 do            /* Block index */
        first_index = bi · bs;
        for j from 0 to bs/2 − 1 do
            j1 = j + first_index;
            j2 = j1 + bs/2;
            y[j1] = x[j1] + ω_bs^j · x[j2];
            y[j2] = x[j1] − ω_bs^j · x[j2];
    return y
Function FFT(x, n)
    for j from 0 to 2^n − 1 do /* Butterfly permutation */
        y[j] = x[reverse(n, j)];
    for k from 1 to n do
        y = OneStep(y, k, n);
    return y
```

Fig. 1. Pseudocode for the radix-2 FFT algorithm, extracted from [3], with permission. Here, `reverse(n, j)` is the `n`-bit number whose binary representation is the mirror image of the representation of `j`, and $\mathtt{N_{blocs}}$ is the number of independent order-2^k FFTs.

III. Global bounds

In this section, we remind some of the results presented in [3]. There are two main sources of errors in an FP implementation of the Cooley-Tukey algorithm: the roundings in the arithmetic operations, and the fact that, except in trivial cases (namely, ± 1 and $\pm i$), the roots of unity are not exactly representable in FP arithmetic.

A. Rounding in arithmetic operations

The FFT algorithm uses complex additions and multiplications. From Eq. (1), one easily deduces that for $x, y \in \mathbb{G}$, the relative error of the floating-point addition $\oplus$ satisfies

$$\left|(x \oplus y) - (x + y)\right| \leqslant u\,|x + y|.$$

Complex multiplication is less simple because there are two implementations to consider: one uses the fused multiply-add (FMA) operation ($\otimes_{\text{FMA}}$), while the other does not ($\otimes_{\text{MUL}}$).

They are described in [3]. For any $x, y \in \mathbb{G}$, the errors bounds, proven in [13] and [14] respectively, are

$$\frac{|x \otimes_{\text{MUL}} y - x \times y|}{|x \times y|} \leqslant \sqrt{5}\,u, \quad \frac{|x \otimes_{\text{FMA}} y - x \times y|}{|x \times y|} \leqslant 2u.$$

In the following, we avoid to specify which implementation is used, and we simply write this bound ρ.

B. Rounded values of roots of unity

FFT algorithms use precision-p FP approximations $\hat{\omega}_N^k$ of the roots of unity. We note δ_n the maximum absolute error of the FP approximations of the order-2^n roots of unity:

$$\delta_n = \max_{k=0}^{n-1} \left|\hat{\omega}_{2^n}^k - \omega_{2^n}^k\right|.$$

C. Global error bound on the Cooley-Tukey algorithm

The state-of-the-art 2-norm global relative error bound is provided by [3]. The authors of that paper wrote (with notations adapted):

> Let $\hat{Y}$ be the computed 2^n-point FFT of $X \in \mathbb{C}^{2^n}$, and let Y be the exact value. Then
> $$\frac{\|\hat{Y} - Y\|_2}{\|Y\|_2} \leqslant (1+u)^n \prod_{j=1}^{n} (1+g_j) - 1,$$
> where $g_1 = g_2 = 0$, $g_j = \delta_j + \rho\,(1+\delta_j)$ for $j \geq 3$.

An ∞-norm input-scaled absolute error can be deduced using Eq. (3):

$$\|\hat{Y} - Y\|_\infty^\perp / \|X\|_\infty^\perp \leqslant b_n,$$
$$\text{with } b_n = \sqrt{2} \cdot 2^n \left((1+u)^n \prod_{j=1}^{n} (1+g_j) - 1 \right). \tag{4}$$

D. A bad case

It is not known if (4) is tight, i.e., if there exists a *worst case* X for which b_n nearly equals the actual FP error. However, a *bad case,* i.e., an input X for which the FP error is particularly large, is built in [3], and the corresponding error is

$$w_n = \left(\frac{2^n}{27} + (15n+14) - \frac{5}{9}\cos\left(\frac{n\pi}{3}\right) + \frac{\sqrt{3}}{9}\sin\left(\frac{n\pi}{3}\right) + \frac{(-1)^n}{27} \right) u \tag{5}$$

One can see in Fig. 2 that b_n and w_n have a similar asymptotic behavior with an asymptotic ratio b_n/w_n of 8.

IV. Local, IA-based, bounds

To obtain local bounds, we use a straightforward implementation of the FFT algorithm using interval arithmetic (IA). In IA, a number is represented by an interval $\boldsymbol{z} = [\underline{z}, \overline{z}]$. Arithmetic operations in IA, are defined such that the intervals always contain the exact result. In the FFT, we replace the input X by a vector of intervals, the roots ω_N^k by intervals, and all intermediate operations by their IA equivalents, then we obtain an interval result that contains the exact result: $Y \in \boldsymbol{Y}$.

In computer implementations, the bounds of an interval are represented by FP numbers, and IA operations use directed roundings to make sure that the exact result remains contained in the output interval. If we use the same precision in conventional FP computation and IA, the result of the FP computation is also included in the interval result: $\hat{Y} \in \boldsymbol{Y}$. The error Eq. (2) is bounded by

$$\|\tilde{Y} - Y\|_\infty^\perp / \|X\|_\infty^\perp \leqslant \sup\left(\|\boldsymbol{Y} - Y\|_\infty^\perp / \|X\|_\infty^\perp\right)$$

where $\sup([\underline{z}, \overline{z}]) = \overline{z}$.

Unfortunately, this bound cannot be computed because Y is not known. However, we can use the inequalities $\sup\left(|\boldsymbol{z}|\right) \leqslant 2\,\mathrm{rad}(\boldsymbol{z}) \leqslant 2\sup\left(|\boldsymbol{z}|\right)$, where $2\,\mathrm{rad}([\underline{z}, \overline{z}]) = \overline{z} - \underline{z}$, which apply when $0 \in \boldsymbol{z}$. Applying this to each component of $\boldsymbol{Y} - Y$, which contains 0, yields

$$\sup\left(\frac{\|\boldsymbol{Y} - Y\|_\infty^\perp}{\|X\|_\infty^\perp}\right) \leqslant \frac{2\,\mathrm{rad}_\infty^\perp(\boldsymbol{Y})}{\|X\|_\infty^\perp} \leqslant 2\sup\left(\frac{\|\boldsymbol{Y} - Y\|_\infty^\perp}{\|X\|_\infty^\perp}\right) \tag{6}$$
$$\text{where } \mathrm{rad}_\infty^\perp(\boldsymbol{Z}) = \max_{k=0}^{N-1} \left\{\mathrm{rad}_\infty(\boldsymbol{z}_k)\right\},$$
$$\text{and } \mathrm{rad}_\infty(\boldsymbol{z}_k) = \max\left\{\mathrm{rad}\left(\mathfrak{Re}(\boldsymbol{z}_k)\right), \mathrm{rad}\left(\mathfrak{Im}(\boldsymbol{z}_k)\right)\right\}.$$

In (6), the middle part of the inequality is computable, and it cannot be more than twice the value of the actual error bound.

Therefore the error obtained with IA computations is always larger than the errors obtained through FP computations, but of course knowing in which extent is difficult and highly dependent on the kind of computation being performed.

V. Numerical investigations

We now test the two approaches presented in sections III and IV by means of intensive numerical computations. The Cooley-Tukey algorithm is implemented in Julia (v1.7.2). Julia's generic programming allows our code to be implemented for all formats consistently, which helps fairer performance comparisons. All tests are performed in binary64/double-precision arithmetic ($p = 53$) for FP and IA approximations, and we used complex multiplications *with FMA*, so that $\rho = 2u$. IA is implemented with the `IntervalArithmetic` library[1]. To estimate errors, we need "exact" values to serve as reference. To that purpose, we used Johansson's Arblib library [15] (through Julia's `Arblib` library[2]) to obtain high-precision enclosure of the exact values.

A. Evaluation of errors

Given an input vector X of FP coefficients, we first compute an accurate enclosure of Y using Arblib in high-precision ($p = 256$), then we compute a FP (resp. IA) approximation $\hat{Y}$ (resp. $\boldsymbol{Y}$) in double-precision arithmetic ($p = 53$). To perform these computations, we obtained beforehand high-precision enclosures of the coefficients ω_N^k, from which we were able to deduce correctly-rounded FP approximations $\hat{\omega}_N^k$, and tight enclosures $\boldsymbol{\omega}_N^k$. Knowing Y allows the computation of usually inaccessible errors

$$e_n^{\text{FP}} = \|\hat{Y} - Y\|_\infty^\perp / \|X\|_\infty^\perp, \quad e_n^{\text{IA}} = \sup(\|\boldsymbol{Y} - Y\|_\infty^\perp) / \|X\|_\infty^\perp,$$

using high-precision Arb arithmetic. The local bound $r_n^{\text{IA}} = 2\,\mathrm{rad}_\infty^\perp(\boldsymbol{Y}) / \|X\|_\infty^\perp$, is also computed. This procedure is applied to $N_{\text{stat}} = 65\,536$ randomly-chosen input vectors, and we retain the maximum value of each error e_n^{FP}, e_n^{IA}, and r_n^{IA}.

[1] https://github.com/JuliaIntervals/IntervalArithmetic.jl/tree/v0.20.8
[2] https://github.com/kalmarek/Arblib.jl/tree/v0.8.1

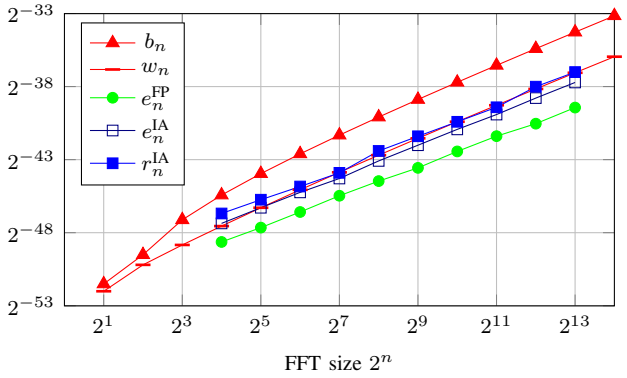

Fig. 2. In binary64/double-precision arithmetic: best known bound b_n (Eq. 4), bad case w_n (Eq. 5), and maximum errors e_n^{FP}, e_n^{IA}, and r_n^{IA} obtained over a sample of 65 536 randomly-chosen inputs.

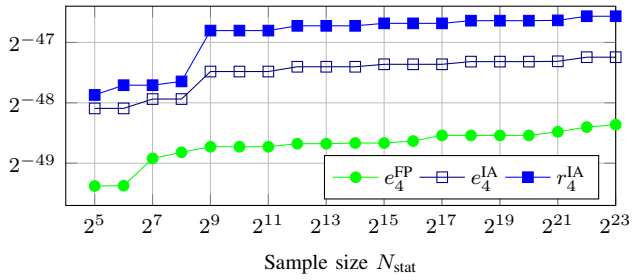

Fig. 3. Maximum errors e_4^{FP}, e_4^{IA}, and r_4^{IA} for increasing values of the number of samples N_{stat}.

B. Results

Figure 2 shows the maximum values obtained for e_n^{FP}, e_n^{IA}, and r_n^{IA} over a random set of 65 536 inputs and for sizes 2^n with $n = 1, 2, \ldots, 13$, allowing the following observations:

- All computed FP approximations have error smaller than the "bad case", which clearly is "exceptionally bad".
- As expected, the computed IA approximations have much larger errors than the FP approximations, and the obtained local bound is slightly larger.
- Nevertheless, the local, IA-based, bounds remain smaller than the global bounds.

In double-precision, the number of possible inputs makes exhaustive tests impossible, hence a natural question is whether we performed enough tests to get a reasonable estimate of the errors one meets in practice. Figure 3 shows e_4^{FP} for increasing values of the number of samples N_{stat}, up to 2^{24}. It still slowly increases with N_{stat}, indicating that larger errors in the FP calculation are still of non-negligible probability.

Fig. 4 shows the timing ratios between FP and IA arithmetic (and between FP and Arb arithmetic) on the environment we used (Julia). The drop for FFTs of size around 2^{11} probably comes from cache effects: for these values what we measure is more the delay of memory transfers than the delay of arithmetic operations. This indicates that for very large FFTs, the time penalty due to IA is between a factor of 2 and 10.

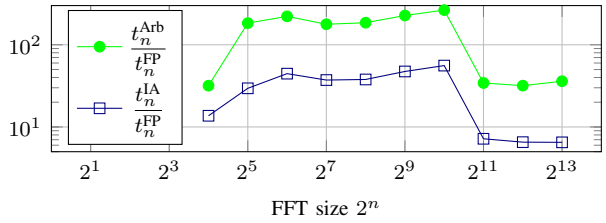

Fig. 4. Timing ratios between computation with high-precision Arb (p = 256) / FP arithmetic and IA / FP arithmetic for increasing input sizes.

VI. Conclusion

Our experiments show that, despite the inherent interval-growth of interval arithmetic, local error bounds obtained through interval arithmetic are significantly smaller than the state-of-the-art global error bounds. As a consequence, unless either IA is slow on the platform being used or the global bound is smaller than the error threshold required by the considered application, computing FFTs in IA can be a interesting alternative.

Moreover, tests with various different interval libraries may allow us to refine our results.

References

[1] N. J. Higham, *Accuracy and stability of numerical algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[2] A. Schönhage and V. Strassen, "Schnelle Multiplikation großer Zahlen," *Computing*, vol. 7, no. 3–4, pp. 281–292, 1971.

[3] N. Brisebarre, M. Joldeș, J.-M. Muller, A.-M. Naneș, and J. Picot, "Error Analysis of Some Operations Involved in the Cooley-Tukey Fast Fourier Transform," *ACM TOMS*, vol. 46, no. 2, pp. 1–27, May 2020.

[4] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*. Philadelphia, PA, USA: SIAM, Jan. 2009.

[5] W. Tucker, *Validated Numerics: A Short Introduction to Rigorous Computations*. Princeton, NJ, USA: Princeton University Press, 2011.

[6] B. Lambov, "Interval arithmetic using SSE-2," in *Reliable Implementation of Real Number Algorithms: Theory and Practice*, ser. Lecture Notes in Computer Science, no. 5045. Dagstuhl Castle, Germany: Springer Berlin Heidelberg New York, January 2006, pp. 102–113.

[7] J. Rivera, F. Franchetti, and M. Püschel, "An interval compiler for sound floating-point computations," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Seoul, South Korea, February 2021, pp. 52–64.

[8] G. Liu and V. Kreinovich, "Fast convolution and Fast Fourier Transform under interval and fuzzy uncertainty," *J. Comput. System Sci.*, vol. 76, no. 1, pp. 63–76, Feb. 2010.

[9] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comp.*, vol. 19, pp. 297–301, 1965.

[10] W. M. Gentleman and G. Sande, "Fast Fourier Transforms: for fun and profit," in *Proceedings of the November 7-10, 1966, fall joint computer conference on XX - AFIPS '66 (Fall)*. San Francisco, California: ACM Press, 1966, p. 563.

[11] G. Plonka, D. Potts, G. Steidl, and M. Tasche, *Numerical Fourier Analysis*, ser. Applied and Numerical Harmonic Analysis. Cham, Switzerland: Birkhäuser, 2018.

[12] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std., Rev. 754-2019, July 2019. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?punumber=8766227

[13] R. P. Brent, C. Percival, and P. Zimmermann, "Error bounds on complex floating-point multiplication," *Math. Comp.*, vol. 76, no. 259, pp. 1469–1481, 2007.

[14] C.-P. Jeannerod, P. Kornerup, N. Louvet, and J.-M. Muller, "Error bounds on complex floating-point multiplication with an FMA," *Math. Comp.*, vol. 86, no. 304, pp. 881–898, Mar. 2017.

[15] F. Johansson, "Arb: efficient arbitrary-precision midpoint-radius interval arithmetic," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1281–1292, 2017.