Error estimation and correction using the forward CENA method

Paul D. Hovland$^{1[0000-0002-0907-2567]}$ and Jan Hückelheim$^{1[0000-0003-3479-6361]}$

Mathematics and Computer Science Division
Argonne National Laboratory
Lemont, IL, USA
`jhueckelheim@anl.gov` | `hovland@mcs.anl.gov`

Abstract. The increasing use of heterogeneous and more energy-efficient computing systems has led to a renewed demand for reduced- or mixed-precision floating-point arithmetic. In light of this, we present the *forward CENA method* as an efficient roundoff error estimator and corrector. Unlike the previously published CENA method, our forward variant can be easily used in parallel high-performance computing applications. Just like the original variant, its error estimation capabilities can point out code regions where reduced or mixed precision still achieves sufficient accuracy, while the error correction capabilities can increase precision over what is natively supported on a given hardware platform, whenever higher accuracy is needed. CENA methods can also be used to increase the reproducibility of parallel sum reductions.

Keywords: CENA method · Roundoff error · Mixed-precision arithmetic · Reproducibility

1 Introduction

Roundoff error is inevitable in floating-point arithmetic; but rigorous error analysis is difficult even for numerical analysis experts, and such experts are in short supply. This situation leads to two main strategies: perform computations in the highest-available precision, possibly sacrificing time and energy savings available at lower precision, or perform computations in low precision and hope for the best. A third strategy is to employ roundoff error estimation in order to characterize and possibly correct roundoff errors. The correction des erreurs numériques d'arrondi method of Langlois [16] (hereafter, original CENA or reverse CENA) is one method for roundoff error estimation and correction but suffers from memory requirements proportional to the number of floating-point operations and an operations count that grows linearly with the number of output variables. We introduce a forward variant of the CENA method (hereafter, forward CENA or CENA) that suffers neither of these deficiencies.

CENA computes local roundoff errors from individual operations and uses automatic, or algorithmic, differentiation (AD) to estimate their cumulative effect on the final output. This estimate is often precise enough to be used as an effective error correction, by subtracting the estimated error from the computed result. The corrected results are not only more accurate than results obtained without the CENA method, but also more reproducible, since they are less affected by the non-associativity of floating-point operators. CENA can easily be implemented and deployed in an existing program by using operator overloading, for example as offered by C++. We believe that CENA can be most useful during the development of numerical software. For example, the error estimates can be used to choose error bounds for regression tests, or the error corrections can be used during the regression tests themselves to remove the nondeterministic effects of parallel sum reductions. Furthermore, CENA could be used to increase the precision of results compared with the best-available precision that is natively supported on a given platform.

The next section summarizes previous work on roundoff error estimation and correction, followed by a brief introduction to AD in Section 3. Then, in Section 4 we provide a description of the forward CENA and its relationship to reverse CENA, and in Section 5 we discuss implementation details. In Section 6 we present experimental results, and we conclude in Section 7 with a brief summary and discussion of future work.

2 Related work

Many techniques for estimating or reducing the effects of roundoff error have been developed [4, 6, 11–13, 16–19, 21–24]. The forward CENA method builds on the reverse CENA method of Langlois [16]. In contrast to this and related techniques, forward CENA has an operations overhead independent of the number of output variables and a memory overhead independent of the number of operations; it is also much easier to parallelize. Like reverse CENA and in contrast to many other techniques, forward CENA computes deterministic local error estimates and combines them with derivatives to compute a global error correction. Forward CENA requires no source code analysis or transformation and can therefore be implemented as a drop-in replacement numeric type requiring no external tool support. We note that forward CENA can be seen as a way to generalize certain algorithms for accurate summation [1, 7, 14, 20] to other types of computation, and in Section 6.1 we compare forward CENA with Kahan's compensated summation algorithm.

3 Brief introduction to AD

Automatic, or algorithmic, differentiation (AD) is a technique for computing the derivatives of functions defined by algorithms [8]. It computes partial derivatives for each elementary operator and combines them according to the chain rule of differential calculus, based on the control flow of the program used to compute the function. In the so-called forward mode of AD, the derivatives are combined in an order that follows the control flow of the function. In the so-called reverse mode, the derivatives are combined in an order that reverses the control flow of the function. The forward mode computes a Jacobian-matrix product, $JS = \frac{\partial y}{\partial x} S$, at a cost proportional to the number of columns in the so-called seed matrix, S, while the reverse mode computes a matrix-Jacobian product, WJ, at a cost proportional to the number of rows in W. The forward mode is therefore efficient for computing Jacobian-vector products, Jv, and the reverse mode is efficient for computing transposed-Jacobian-vector products, $J^T v$ and the gradients of scalar functions.

4 Forward CENA method

The forward and reverse CENA methods approximate the error Δ_y in a result y using the formula

$$\Delta_y \approx E_y = \sum_i \frac{\partial y}{\partial x_i} \delta_i, \tag{1}$$

where x_i is the result of each instruction i used in computing y and δ_i is the local round-off error in computing x_i. In the reverse CENA method [16], one computes the derivatives using reverse mode AD. The number of operations is proportional to the number of operations in the function evaluation. Unfortunately, employing the reverse mode also incurs a storage cost proportional to the number of operations in the function evaluation.

Instead of reverse mode AD, we can employ forward mode AD, using a seed matrix (vector) $\delta = [\delta_1 \delta_2 \ldots \delta_n]^T$ to directly compute the inner product $\frac{\partial y}{\partial x}^T \delta = \sum_i^n \frac{\partial y}{\partial x_i} \delta_i$. This is most easily comprehended by using the *buddy variable* approach [3]:

```
xibuddy = 0.0
xi = fi(xj,xk) + xibuddy
```

which, after differentiating and initializing the seed matrix for `xibuddy`, yields

```
xibuddy = 0.0
ad_xibuddy = deltai
xi = fi(xj,xk) + xibuddy
ad_xi = (dfidxj*ad_xj + dfidxk*ad_xk) + ad_xibuddy .
```

We note that `deltai` (the roundoff error in computing `xi`) may not in general be available until after the computation of `xi`; however, one can easily simplify the derivative computation to

```
xi = fi(xj,xk)
ad_xi = (dfidxj*ad_xj + dfidxk*ad_xk) + deltai.
```

Theorem If for each statement $x_i = \phi_i(x_1, x_2, \ldots, x_{i-1})$ we compute

$$E_i = \delta_i + \sum_{j=1}^{i-1} \frac{\partial \phi_i}{\partial x_j} E_j,$$

then E_i satisfies Equation 1. That is,

$$E_i = \sum_{j=1}^{i} \frac{\partial y}{\partial x_j} \delta_j .$$

Proof By induction. $E_1 = \delta_1$. Assume that

$$E_i = \delta_i + \sum_{j=1}^{i-1} \frac{\partial \phi_i}{\partial x_j} E_j = \sum_{j=1}^{i} \frac{\partial x_i}{\partial x_j} \delta_j$$

for all $i \le n$. Then,

$$\begin{aligned}
E_{n+1} &= \delta_{n+1} + \sum_{j=1}^{n} \frac{\partial \phi_{n+1}}{\partial x_j} E_j = \delta_{n+1} + \sum_{j=1}^{n} \frac{\partial \phi_{n+1}}{\partial x_j} \sum_{k=1}^{j} \frac{\partial x_j}{\partial x_k} \delta_k \\
&= \delta_{n+1} + \sum_{j=1}^{n} \frac{\partial \phi_{n+1}}{\partial x_j} \sum_{k=1}^{n} \frac{\partial x_j}{\partial x_k} \delta_k = \delta_{n+1} + \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{\partial \phi_{n+1}}{\partial x_j} \frac{\partial x_j}{\partial x_k} \delta_k \\
&= \delta_{n+1} + \sum_{k=1}^{n} \sum_{j=1}^{n} \frac{\partial \phi_{n+1}}{\partial x_j} \frac{\partial x_j}{\partial x_k} \delta_k = \delta_{n+1} + \sum_{k=1}^{n} \sum_{j=k}^{n} \frac{\partial \phi_{n+1}}{\partial x_j} \frac{\partial x_j}{\partial x_k} \delta_k \\
&= \delta_{n+1} + \sum_{k=1}^{n} \frac{\partial \phi_{n+1}}{\partial x_k} \delta_k = \frac{\partial \phi_{n+1}}{\partial \phi_{n+1}} \delta_{n+1} + \sum_{k=1}^{n} \frac{\partial \phi_{n+1}}{\partial x_k} \delta_k = \sum_{k=1}^{n+1} \frac{\partial \phi_{n+1}}{\partial x_k} \delta_k .
\end{aligned}$$

We note that the proof ignores roundoff errors in the computation of E_i and therefore holds only if the E_i and partial derivatives are computed in real arithmetic. This is sufficient to fulfill our goal of demonstrating equivalence between forward and reverse CENA, which both ignore roundoff errors in the computation of the derivatives.

5 Implementation

We created a C++ type that overloads the standard operators to compute error estimates and corrections using the forward CENA method. They can be used just like any other number type as long as no unsupported operators are used. Currently, our library supports the usual operators, such as `+`, `-`, `*`, and `/` (in addition to their compound operators, such as `+=`); comparison operators, including `<`, `<=`, and `==`; assignment operators, cast to and from native types, and so on. In addition, we support the `sqrt`, `sin`, and `cos` functions and the `<<` streaming operator to output a textual representation of the number, error estimate, and error correction. One example operator is shown in Fig. 1. Furthermore, we used the OpenMP `declare reduction` pragma to allow parallel reductions over CENA types.

```
template<typename T>
class freal{
  private:
    T val, err;
    static T addition_error(T a, T b, T x) {
        T corr = x - a;
        return ((x-corr)-a)+(corr-b);
    }

  public:
    freal<T>(T value, T error) : val(value), err(error) { }

    void operator+=(const freal<T> rhs) {
      T value = this->val + rhs.val;
      T localerror = addition_error(this->val,rhs.val,value);
      this->val = value;
      this->err += rhs.err + localerror;
    }
};
```

Fig. 1. Part of the CENA class, showing only the compound addition operator and the internal helper function to compute the local error produced by that operation. The actual implementation used in this work supports many more operators.

In our experiments we use the GNU MPFR library [5] to test the CENA method at arbitrary floating-point precision, in addition to the natively supported single, double, extended double precision, and quad precision as supported by the GNU libquadmath library. All MPFR operations are guaranteed to use the exact precision that was specified, which allows us not only to perform tests at high precision and obtain accurate reference results but also to simulate half, quarter, or more esoteric low-precision number types.

6 Test cases and experimental results

In this section we show the effectiveness of the CENA method on three test cases. The first, shown in Section 6.1, uses CENA to obtain reproducible results in parallel sum reductions. Then, in Section 6.2 we use CENA to obtain error estimates within an OpenMP-parallel benchmark derived from a cosmology code. Next, in Section 6.3 we use CENA to reduce roundoff errors in various implementations of the matrix-matrix-product. Finally, in Section 6.4 we apply CENA to a pathological example, the Muller recurrence. All test cases were compiled by GCC 9.2 with flags `-O3 -std=c++11 -fopenmp` and executed on a 28-core/56-thread Intel® Xeon® Platinum 8180 Processor (“Skylake”).

6.1 Sum reduction

Sum reductions are ubiquitous in numerical programs, for example during the computation of a dot-product, matrix-vector or matrix-matrix product, or numerical quadrature. In this test case we look at reductions in isolation, but two of the subsequent larger test cases also contain sum reductions.

One often wishes to compute large sum reductions in parallel, for example when forming the dot product of two large vectors. This is typically done by accumulating partial sums on each thread in parallel, followed by some strategy to combine these into one overall result. A common problem with parallel sum reductions is the lack of reproducibility because of the non-associativity of the `+` operator for floating-point numbers. This can result in the same correctly implemented and data-race-free program producing different results every time it is executed, because of the nondeterministic scheduling of the summation. This causes problems, for example, in regression testing, where distinguishing floating-point roundoff from other small errors or race conditions can be difficult.

In this test case we demonstrate how CENA can help reduce nondeterminism caused by a change in summation order. To this end, we initialize an array of 1 million pseudorandom numbers. The same hard-coded seed is always used for the pseudorandom number generator, to ensure that the set of generated numbers remains the same between runs. However, the vector of numbers is then shuffled randomly, using a different seed and thus ensuring a different summation order each time. Any resulting changes are therefore due to roundoff.

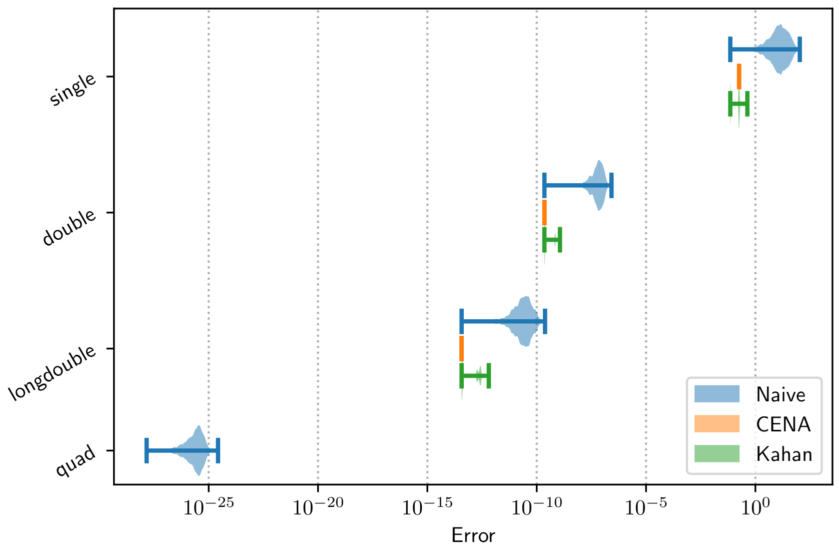

Fig. 2. The result of the sum reduction fluctuates because of the non-associativity of floating-point summations. The CENA-corrected results (and to a slightly lesser extent the Kahan results) are consistent across runs, and more accurate than the uncorrected results. Increasing the working precision has a larger benefit than using Kahan or CENA for accuracy, but not for reproducibility.

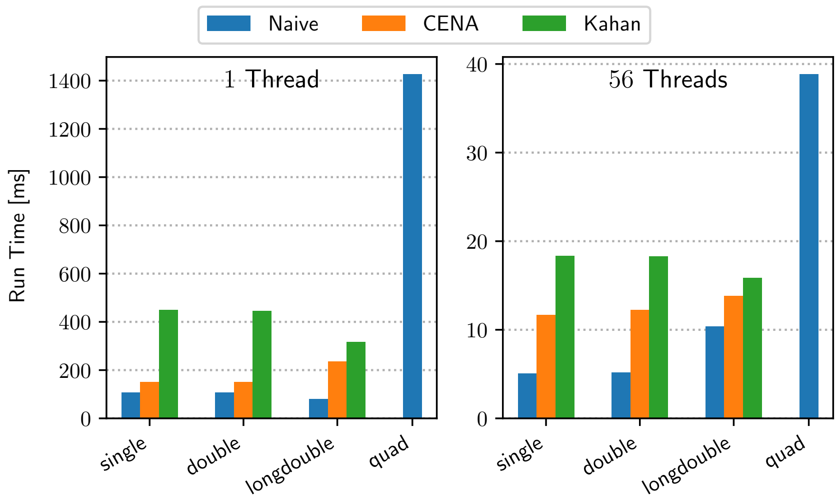

Fig. 3. Run time of conventional, CENA, and Kahan summation on 1 (left) or 56 (right) threads. CENA types are slightly slower than built-in number types but do not affect parallel scaling significantly and still work well on higher thread counts.

We perform this test 1,000 times in IEEE754 single, double, and quadruple precision, as well as 80-bit extended precision, using "naive" summation (adding numbers one by one to an accumulator), Kahan summation, and summation using CENA types. We note that because all of the derivatives in summation are equal to 1, the CENA method (forward or reverse) reduces to Algorithm 4.1 in [20], a form of compensated summation based on Knuth's two-sum algorithm, but without needing to modify the implementation of summation, beyond using our CENA type. The input numbers are generated with a quad-precision mantissa consisting of a uniformly sampled random bit pattern, a random sign bit, and an exponent from a uniform distribution in $[2^{-16}...2^{16}]$ ($\approx [10^{-5}...10^{5}]$). The inputs for the lower-precision tests are obtained by type casting. Reference results are computed by using the aforementioned MPFR library with a mantissa length of 200 bits, well above the 113-bit mantissa of quad precision and almost four times that of double-precision numbers.

Figure 2 shows the errors for each of these settings. CENA and Kahan summation have comparable effects on the mean errors, although CENA is often slightly superior and reduces the variability of errors by many orders of magnitude. In Figure 3 we show the run times of our tested summation approaches, for sequential or parallel summation on 56 threads. Using CENA instead of built-in number types increases the time by a small factor, typically below 2. Kahan summation is slower, but this is probably a deficiency in our implementation, since Kahan requires fewer operations than CENA. Because of the lack of hardware support, the quad precision summation is very slow.

6.2 HACCmk

The Hardware Accelerated Cosmology Code (HACC) [9] helps in understanding the evolution of the Universe, by simulating the formation of structure and the behavior of collision-less fluids under gravity.

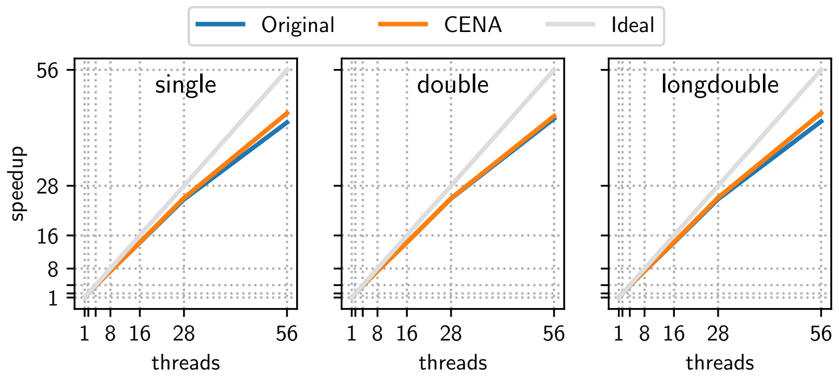

Fig. 4. HACCmk scalability is excellent with and without CENA. The absolute run time between CENA and normal execution differs by a factor of ca. 10 (see Fig. 5).

HACCmk is a compute-intensive kernel routine extracted from HACC that calculates force values for each particle in an OpenMP `parallel for` loop. Our forward CENA method can be used simply by changing the number type through a `typedef`. The only other modification is to replace the `pow(·)` function by `1/·*sqrt(·)`, since `pow` is currently not supported by our implementation.

The HACCmk code scales well on our system with and without CENA, as shown in Figure 4. Absolute run times in Figure 5 show that CENA increases run time by a factor of ca. $10\times$ (slightly less for single/double and slightly more for long double precision). CENA estimates the actual error well enough to be able to improve the result by an order of magnitude. Correction is again slightly more effective for even-length mantissas, see Figure 6.

6.3 Classic and Strassen matrix multiplication

In this section we investigate CENA in the context of matrix-matrix multiplications using either Strassen's algorithm or classic multiplication using a triple-nested loop. We briefly summarize results from previous literature showing that Strassen's algorithm produces higher roundoff errors but is faster than classic multiplication for large matrices. We then present numerical experiments.

Strassen's algorithm was the first published way of computing the product

$$\mathbf{C} = \mathbf{AB} \qquad \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n \times n}$$

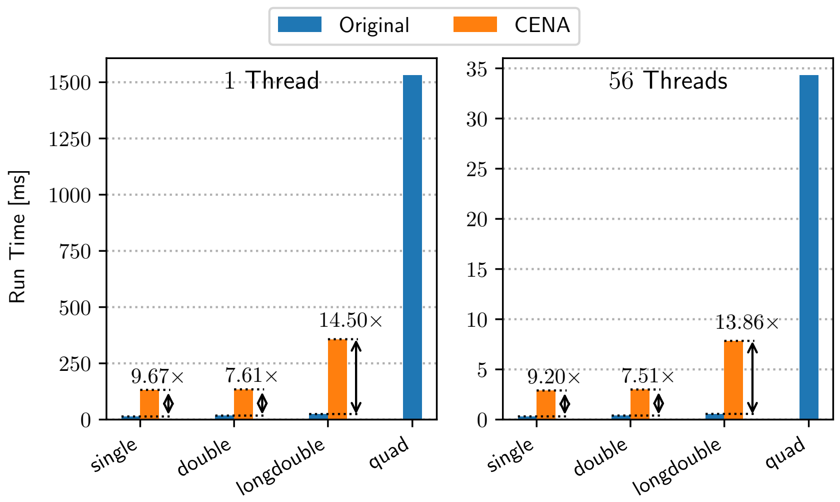

Fig. 5. Absolute run times for HACCmk with and without using CENA. The CENA type slows down execution by one order of magnitude.

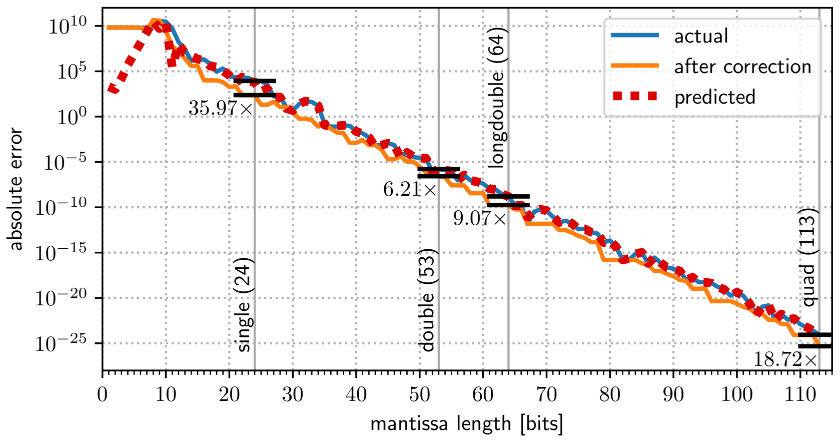

Fig. 6. Errors for HACCmk with and without CENA correction, and CENA error estimate, for various mantissa lengths, compared to a 200-bit mantissa reference. CENA correctly computes the exponent and some mantissa bits of the actual errors.

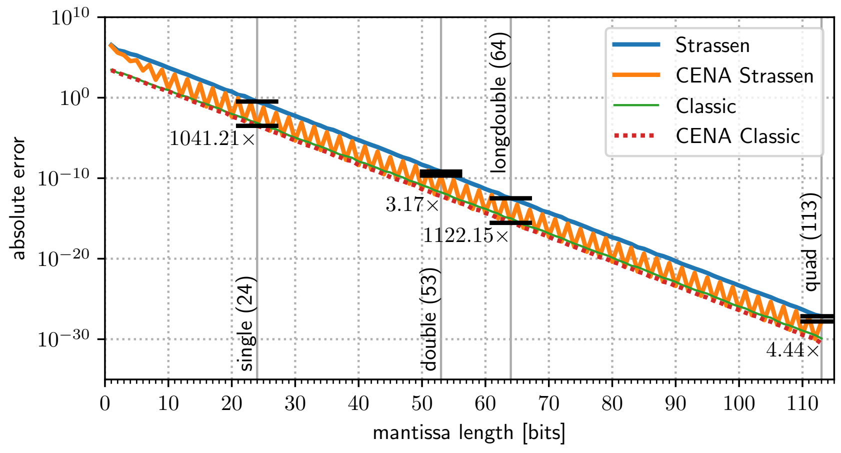

Fig. 7. Absolute error of classic and Strassen multiplication for the entire range of mantissa lengths from 1 bit up to 113 bits (IEEE754 quadruple precision), for a 64×64 matrix. Without CENA correction, Strassen multiplication produces an error that is about $500\times$ larger than that of classic multiplication. CENA reduces the error of classic multiplication by a factor of ca. $2\times$ and that of Strassen multiplication by a factor of ca. $3\times$ for odd mantissa lengths and above $1000\times$ for even mantissa lengths.

with a time complexity of less than $\mathcal{O}(n^3)$. Strassen's algorithm and other subsequently discovered subcubic algorithms have been studied extensively, regarding both their run time and their numerical stability. Previous studies have found that Strassen's and related algorithms are generally stable, although their error bounds are slightly worse than those of the classic matrix multiplication [2, 10].

We summarize here the error bounds given in [10]. For classic multiplication $\mathbf{C} = \mathbf{AB}$, the error in a computed matrix $\hat{\mathbf{C}}$ is bounded by

$$\left\|\hat{\mathbf{C}} - \mathbf{C}\right\| \leq n^2 \epsilon \|\mathbf{A}\| \|\mathbf{B}\| + \mathcal{O}(\epsilon^2), \tag{2}$$

where ϵ is the unit roundoff error and $\|\cdot\|$ is the maximum norm. Note that [10] also provides a tighter bound for the error in each element in $\mathbf{C}$ that is linear in the matrix size n, while [2] gives an even tighter bound if the summation is performed by using pairwise summation. The authors also remark that no such elementwise error bound can exist for fast (subcubic) matrix multiplication algorithms, whose error is bounded by

$$\left\|\hat{\mathbf{C}} - \mathbf{C}\right\| \leq \left[\left(\frac{n}{n_0}\right)^{\log_2 12} (n_0^2 + 5n_0) - 5n\right] \epsilon \|\mathbf{A}\| \|\mathbf{B}\| + \mathcal{O}(\epsilon^2), \tag{3}$$

where n_0 is the threshold at which the small partition matrices are multiplied using the classic algorithm (the recursion base case).

We note that $\log_2 12 \approx 3.58496$, resulting in a significantly faster growth of roundoff errors than the quadratic growth in the classic algorithm. We also

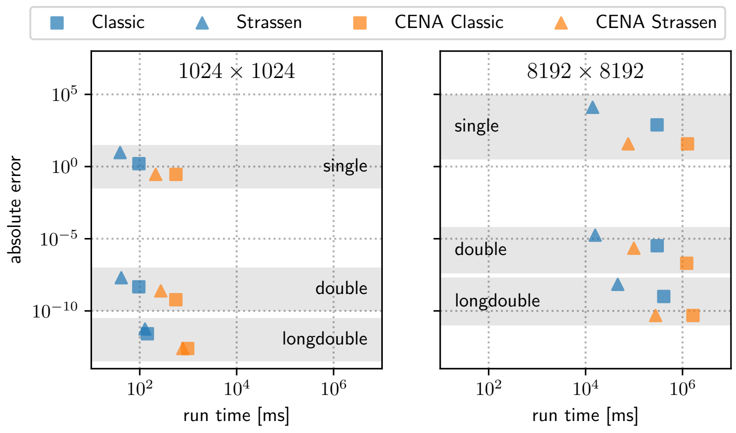

Fig. 8. Error vs run time for classic and Strassen multiplication, for various working-precision settings, with and without CENA (left: 1024×1024, right: 8192×8192 matrix). CENA enables new trade-offs between accuracy and run time. For example, in the large test case Strassen with CENA offers better run time and accuracy than does classic multiplication without CENA.

note that for $n_0 = n$ (i.e., a threshold so large that the classic algorithm is used without previous partitioning) the bounds in Equations (2) and (3) are identical.

Our experiments use input matrices filled by the same number generator as in Section 6.1. After multiplication with either a classic or Strassen's algorithm, we compute the maximum of all elementwise absolute errors by comparing with a reference result obtained through MPFR with a 200-bit mantissa. Our classic implementation is accelerated through the use of OpenMP. Our Strassen implementation supports only matrix sizes that are a power of 2 (a restriction that could be lifted by a better implementation) and is parallelized through the use of the parallized classic multiplication as a recursion base case. We use the best-performing base case size n_0 for our experiments, which we determined to be between 128 and 512 on 56 threads, depending on the working precision and whether CENA was used. We note that the implementation, parallelization, and hardware platform do not change the asymptotic trends and merely affect the break-even point beyond which Strassen's outperforms classic multiplication.

In our experiments, Strassen multiplication reduces run time and increases error, while CENA has the opposite effect. CENA more than offsets the accuracy loss from Strassen multiplication for the tested matrices. For mantissas with even bit length (e.g. single and long double), but not for those with odd length (e.g. double or quad), CENA produces equally good results for either multiplication method. The reason for this even-odd oscillation, shown in Figure 7, is unclear but can also be observed to a smaller extent in the other test cases.

Since CENA increases run time by a constant factor and Strassen instead reduces the run time complexity class, there is necessarily a break-even point at which Strassen combined with CENA outperforms classic multiplication without CENA. The result is a method that is faster, but at the same time more accurate, than classic non-CENA multiplication. Figure 8 illustrates this effect by showing run time and accuracy for two different problem sizes, one that is smaller and one that is larger than this break-even point.

6.4 Muller Recurrence

Error estimation and correction are not a panacea. We applied CENA to a pathological example from [15], the Muller recurrence

$$x_{n+1} = 108 - (815 - 1500/x_{n-1})/x_n$$

using initial values $x_0 = 4.0$ and $x_1 = 4.25$. This recurrence is carefully designed so that the solution is of the form

$$x_n = (\alpha 3^{n+1} + \beta 5^{n+1} + \gamma 100^{n+1})/(\alpha 3^n + \beta 5^n + \gamma 100^n)$$

where the specific values of α, β, and γ depend on x_0 and x_1. Our initial conditions correspond to $\alpha = 1$, $\beta = 1$, and $\gamma = 0$, Therefore, the recurrence has the solution $x_n = (3^{n+1} + 5^{n+1})/(3^n + 5^n)$ and ought to converge to 5 in the limit. However, the slightest roundoff error causes the recurrence to match the solution for a nonzero γ and the recurrence converges to 100 in the limit.

Figure 9 shows the actual errors as well as the CENA estimate for a number of precisions, for each iteration, compared with the correct value at that iteration. When the sequence first diverges from the true solution with $\gamma = 0$, CENA estimates large roundoff errors up to O(100). However, eventually the nonzero γ terms in the recurrence come to dominate and CENA estimates a very small roundoff error. Thus, consistent with the title of [15], mindless application of the CENA correction would leave roundoff error in the computation of x_{30} undetected. However, monitoring error estimates at each iteration would allow detection of a significant roundoff problem. While real computational science applications are unlikely to exhibit such extreme behavior as the Muller recurrence, it may nonetheless be advisable to monitor error estimates throughout the computation rather than relying exclusively on the final terms.

7 Conclusions

We introduced the forward CENA method and an efficient implementation. Using CENA to estimate and to some extent correct roundoff errors in numerical programs can be as easy as replacing the number types with our overloaded CENA number type. We showed that forward CENA does not negatively affect the scalability of parallel codes but has an overhead factor of 2–15. Future work includes analyzing the reasons for the observed superior error correction for even mantissa lengths compared with odd mantissa lengths. We also plan to investigate whether forward-mode AD can be employed in other error estimation methods that have historically relied on reverse-mode AD.

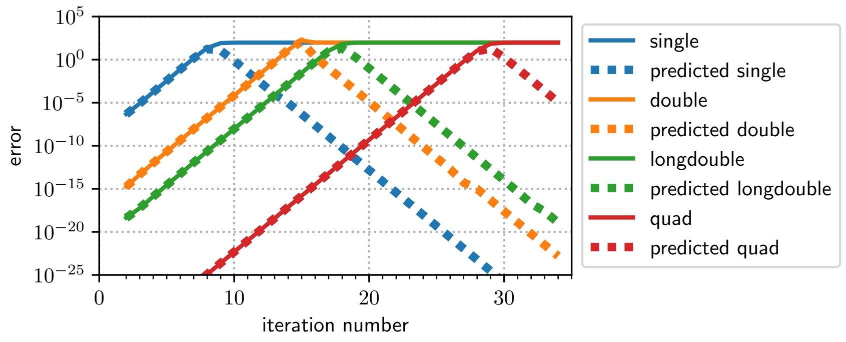

Fig. 9. CENA accurately estimates the roundoff errors up to the point where the recurrence starts to converge to the "wrong" fixed point. Using more precise number types only delays, but does not prevent this problem.

Acknowledgments

We thank Vincent Baudoui for introducing us to the CENA method and constructive conversations about roundoff error. This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under contract number DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

References

1. Ahrens, P., Demmel, J., Nguyen, H.D.: Algorithms for efficient reproducible floating point summation. ACM Trans. Math. Softw. **46**(3) (Jul 2020)
2. Ballard, G., Benson, A.R., Druinsky, A., Lipshitz, B., Schwartz, O.: Improving the numerical stability of fast matrix multiplication. SIAM Journal on Matrix Analysis and Applications **37**(4), 1382–1418 (Jan 2016)
3. Bischof, C.H., Carle, A., Hovland, P.D., Khademi, P., Mauer, A.: ADIFOR 2.0 user's guide (Revision D). Argonne Technical Memorandum 192 (1998)
4. Christianson, B.: Reverse accumulation and accurate rounding error estimates for Taylor series coefficient. Optimization Methods and Software **1**(1), 81–94 (1992)
5. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Trans. Math. Softw. **33**(2) (June 2007)
6. Garcia, R., Michel, C., Rueher, M.: A branch-and-bound algorithm to rigorously enclose the round-off errors. In: Simonis, H. (ed.) Principles and Practice of Constraint Programming. pp. 637–653. Springer International Publishing, Cham (2020)
7. Graillat, S., Ménissier-Morain, V.: Accurate summation, dot product and polynomial evaluation in complex floating point arithmetic. Information and Computation **216**, 57–71 (2012)

8. Griewank, A., Walther, A.: Evaluating Derivatives. Society for Industrial and Applied Mathematics (Jan 2008). https://doi.org/10.1137/1.9780898717761
9. Habib, S., Pope, A., Finkel, H., Frontiere, N., Heitmann, K., Daniel, D., Fasel, P., Morozov, V., Zagaris, G., Peterka, T., Vishwanath, V., Lukić, Z., Sehrish, S., keng Liao, W.: HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. New Astronomy **42**, 49–65 (Jan 2016)
10. Higham, N.J.: Exploiting fast matrix multiplication within the Level 3 BLAS. ACM Trans. Math. Softw. **16**(4), 352–368 (Dec 1990)
11. Iri, M., Tsuchiya, T., Hoshi, M.: Automatic computation of partial derivatives and rounding error estimates with applications to large-scale systems of nonlinear equations. J. Computational and Applied Mathematics **24**(3), 365–392 (Dec 1988)
12. Jézéquel, F., Graillat, S., Mukunoki, D., Imamura, T., Iakymchuk, R.: Can we avoid rounding-error estimation in hpc codes and still get trustworthy results? In: Christakis, M., Polikarpova, N., Duggirala, P.S., Schrammel, P. (eds.) Software Verification. pp. 163–177. Springer International Publishing, Cham (2020)
13. Jézéquel, F., Chesneaux, J.M.: CADNA: a library for estimating round-off error propagation. Computer Physics Communications **178**(12), 933–955 (2008)
14. Kahan, W.: Pracniques: Further remarks on reducing truncation errors. Commun. ACM **8**(1), 40 (Jan 1965). https://doi.org/10.1145/363707.363723
15. Kahan, W.: How futile are mindless assessments of roundoff in floating-point computation? (2006), `http://www.cs.berkeley.edu/~wkahan/Mindless.pdf`
16. Langlois, P.: Automatic linear correction of rounding errors. BIT Numerical Mathematics **41**(3), 515–539 (June 2001)
17. Linnainmaa, S.: Taylor expansion of the accumulated rounding error. BIT Numerical Mathematics **16**(2), 146–160 (June 1976)
18. Martel, M.: Semantics of roundoff error propagation in finite precision calculations. Higher-Order and Symbolic Computation **19**(1), 7–30 (March 2006)
19. Menon, H., Lam, M.O., Osei-Kuffuor, D., Schordan, M., Lloyd, S., Mohror, K., Hittinger, J.: ADAPT: Algorithmic differentiation applied to floating-point precision tuning. In: Proceedings of SC '18. pp. 48:1–13. IEEE Press, Piscataway, NJ (2018)
20. Ogita, T., Rump, S.M., Oishi, S.: Accurate sum and dot product. SIAM Journal on Scientific Computing **26**(6), 1955–1988 (Jan 2005)
21. Solovyev, A., Baranowski, M.S., Briggs, I., Jacobsen, C., Rakamarić, Z., Gopalakrishnan, G.: Rigorous estimation of floating-point round-off errors with symbolic Taylor expansions. ACM Trans. Program. Lang. Syst. **41**(1), 2:1–39 (Dec 2018)
22. Tienari, M.: A statistical model of roundoff error for varying length floating-point arithmetic. BIT Numerical Mathematics **10**(3), 355–365 (Sep 1970)
23. Vassiliadis, V., Riehme, J., Deussen, J., Parasyris, K., Antonopoulos, C.D., Bellas, N., Lalis, S., Naumann, U.: Towards automatic significance analysis for approximate computing. In: 2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). pp. 182–193 (March 2016)
24. Vignes, J.: Discrete stochastic arithmetic for validating results of numerical software. Numerical Algorithms **37**(1-4), 377–390 (Dec 2004)