Numerical Stability of Algorithms at Extreme Scale and Low Precisions[*]

Nicholas J. Higham

Abstract

The largest dense linear systems that are being solved today are of order $n = 10^7$. Single precision arithmetic, which has a unit roundoff $u \approx 10^{-8}$, is widely used in scientific computing, and half precision arithmetic, with $u \approx 10^{-4}$, is increasingly being exploited as it becomes more readily available in hardware. Standard rounding error bounds for numerical linear algebra algorithms are proportional to $p(n)u$, with p growing at least linearly with n. Therefore we are at the stage where these rounding error bounds are not able to guarantee any accuracy or stability in the computed results for some extreme-scale or low-accuracy computations. We explain how rounding error bounds with much smaller constants can be obtained. Blocked algorithms, which break the data into blocks of size b, lead to a reduction in the error constants by a factor b or more. Two architectural features also reduce the error constants: extended precision registers and fused multiply–add operations, either at the scalar level or in mixed precision block form. We also discuss a new probabilistic approach to rounding error analysis that provides error constants that are the square roots of those of the worst-case bounds. Combining these different considerations provides new understanding of the numerical stability of extreme scale and low precision computations in numerical linear algebra.

[*] Version of September 13, 2021.

1. Introduction

We are approaching the exascale computing era, in which the world's fastest computers will be able to perform 10^{18} double precision floating-point operations (flops) per second. With the increased speed comes an increase in the size of problems that can be solved in reasonable time, provided that sufficient memory is available.

A problem of particular interest is solving a dense system of linear equations $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is nonsingular and $b \in \mathbb{R}^n$. Table 1 shows the sizes of large systems that have been solved at different points in time. The data is taken from the TOP500* list, which ranks the world's fastest computers by their speed (measured in flops per second) in solving a random linear system $Ax = b$ by LU factorization with partial pivoting. Generally, a benchmark run needs to be done with the largest n possible in order to obtain the best performance, so the tabulated values give an indication of the largest systems solved at each point in time. Table 1 suggests that the size of the largest linear systems being solved is growing by roughly a factor 10 each decade.

The standard componentwise backward error result for a solution $\widehat{x}$ computed by LU factorization in floating-point arithmetic is as follows [21, Thm. 9.4]. We write $|A| = (|a_{ij}|)$ and inequalities between matrices hold componentwise. We need the constant

$$\gamma_n = \frac{nu}{1-nu},$$

where u is the unit roundoff, which is $u = 2^{-t}$ for a base-2 floating-point arithmetic with t bits in the significand.

Theorem 1.1. *Let $A \in \mathbb{R}^{n\times n}$ and suppose that LU factorization produces computed LU factors $\widehat{L}$, $\widehat{U}$, and a computed solution $\widehat{x}$ to $Ax = b$. Then there is a matrix ΔA such that*

$$(A + \Delta A)\widehat{x} = b, \quad |\Delta A| \le \gamma_{3n}|\widehat{L}||\widehat{U}|. \tag{1.1}$$

Ideally, we would like the backward error matrix ΔA in (1.1) to satisfy $\|\Delta A\|_\infty \approx u\|A\|_\infty$. It can be shown that

$$\|\,|\widehat{L}||\widehat{U}|\,\|_\infty \le p(n)\rho_n\|A\|_\infty \tag{1.2}$$

for a quadratic polynomial p [21, Lem. 9.6], where the growth factor $\rho_n \ge 1$ measures the growth of elements during the factorization process. If, however, we make the very favorable assumption that $\|\,|\widehat{L}||\widehat{U}|\,\|_\infty \approx \|A\|_\infty$ then we obtain

$$\frac{\|\Delta A\|_\infty}{\|A\|_\infty} \lesssim \gamma_{3n} = 3nu + O(u^2). \tag{1.3}$$

For the largest n in Table 1, in IEEE double precision arithmetic (see Table 2) we have $nu \approx (2.1\times 10^7)(1.11\times 10^{-16}) \approx 2.3\times 10^{-9}$, so even with these favorable assumptions our bound indicates the potential for a significant loss of numerical stability. If we work in IEEE single precision then $nu \approx (2.1\times 10^7)(5.96\times 10^{-8}) \approx 1.25$, and our backward error bound is of order 1, suggesting the possibility of a complete loss of stability.

[*] http://www.top500.org

Table 1 Size of large linear systems solved. The data is from the TOP500.

Machine	Date	n
Fugaku	June 2021	2.1×10^7
Jaguar	June 2010	6.3×10^6
ASCI RED	June 2000	3.6×10^5
CM-5/1024	June 1993	5.2×10^4

Table 2 Parameters for floating-point arithmetics: number of bits in significand (including implicit most significant bit) and exponent (sig, exp), unit roundoff u, smallest normalized positive number $x_{\min}$, and largest finite number $x_{\max}$. The last three columns are given to three significant figures. The arithmetics whose names begin "fp" are from the IEEE standard [25].

Precision	Name	(sig, exp)	u	$x_{\min}$	$x_{\max}$
Half	bfloat16	$(8, 8)$	3.91×10^{-3}	1.18×10^{-38}	3.39×10^{38}
Half	fp16	$(11, 5)$	4.88×10^{-4}	6.10×10^{-5}	6.55×10^{4}
Single	fp32	$(24, 8)$	5.96×10^{-8}	1.18×10^{-38}	3.40×10^{38}
Double	fp64	$(53, 11)$	1.11×10^{-16}	2.22×10^{-308}	1.80×10^{308}
Double extended (Intel)		$(64, 16)$	5.32×10^{-20}	3.36×10^{-4932}	1.19×10^{4932}
Quadruple	fp128	$(113, 15)$	9.63×10^{-35}	3.36×10^{-4932}	1.19×10^{4932}

Modern hardware increasingly supports half precision arithmetic, which is attractive because of its speed, its lower energy usage, and its reduced storage and data movement costs. The two currently available half precision formats are bfloat16 [26] and IEEE half precision; see Table 2. The optimistic bound (1.3) provides useful information only if $3nu < 1$, but $3nu > 1$ for problems of order $n \geq 684$ in IEEE half precision and $n \geq 86$ in bfloat16. Yet machine learning codes routinely use half precision in inner products and matrix–vector products with $n \gg 682$ with apparent success [18], [38]. Moreover, the machine topping the HPL-AI mixed-precision benchmark [14] in the June 2021 TOP500 list solved a linear system of order 1.6×10^7 using IEEE half precision arithmetic for most of the computations, and the result was good enough to pass the benchmark's test that the residual is of order the unit roundoff for double precision.

How can this apparent mismatch between theory and practice be explained, and what are the implications for the future as the size of the largest problems continues to increase and the use of low precision arithmetic becomes more common? Have we reached the point where our techniques for analyzing rounding errors, honed over seventy years of digital computation, are unable to predict the accuracy of numerical linear algebra computations that are now routine? I will show that we can, in fact, understand to a considerable extent the behavior of extreme-scale and low accuracy computations. To do so we need to take account of a number of algorithmic design techniques and architectural features of processors that help to reduce error growth and we need to exploit a new probabilistic approach to rounding error analysis.

The main purpose of backward error analysis results such as Theorem 1.1 is to show the form of the backward error bound and to reveal the circumstances (if any) in which the backward error could be large. As Wilkinson [37], Parlett [31], and the present author [21, Sec. 3.2] have noted, the constants in a backward error bound are the least important part of it. Wilkinson recommends that if sharp error estimates are required they should be computed a posteriori [36, Sec. 12], [37]. This is indeed good advice, but it nevertheless remains valid to ask what a priori bounds can tell us—about the limits of what can be computed and about whether a successful computation can be guaranteed for a mission-critical application or one that takes up substantial computational resources. Furthermore, this question is also relevant for future benchmarking: will the HPL benchmark [13], [32] (used in the TOP500) or the HPL-AI mixed precision benchmark need modifying in the future because their criteria for successful completion can no longer be satisfied?

We assume that the floating-point arithmetic in use satisfies the standard model [21, Sec. 2.2]

$$\mathrm{fl}(x \mathbin{\mathrm{op}} y) = (x \mathbin{\mathrm{op}} y)(1+\delta), \quad |\delta| \le u, \quad \mathrm{op} = +, -, *, /. \tag{1.4}$$

This model is certainly satisfied by IEEE arithmetic, which defines $\mathrm{fl}(x \mathbin{\mathrm{op}} y)$ to be the rounded exact value. In general, we denote by $\mathrm{fl}(\mathit{expr})$ the value of the expression *expr* when it is evaluated in floating-point arithmetic.

We begin, in Section 2, by showing how the use of blocked inner products and blocked matrix factorizations reduces constants in rounding error bounds by a factor approximately equal to the block size. In Section 3 we explain how extended precision registers on Intel x86 processors and fused multiply–add operations and their mixed precision block generalizations yield reductions in the error constants. In Section 4 we explain how probabilistic rounding error analysis gives rounding error bounds with constants that are the square roots of the constants in the worst-case bounds. Some other relevant considerations are discussed in Section 5. We offer our conclusions in Section 6.

2. Blocked algorithms

Blocked algorithms* which are primarily designed to give better performance on modern computers with hierarchical memories, also lead to improved rounding error bounds, as we now explain.

2.1. Blocked inner products

Let $x, y \in \mathbb{R}^n$ and consider the inner product $s = x^T y$. If we evaluate s in the natural way as

$$s = x_1 y_1, \quad s \leftarrow s + x_k y_k, \quad k = 2\colon n, \tag{2.1}$$

then the computed result $\widehat{s}$ satisfies [21, Sec. 3.1]

$$|s - \widehat{s}| \le \gamma_n |x|^T |y|. \tag{2.2}$$

In fact, this bound holds no matter what order the terms are summed in. Another way to compute the inner product is by summing two half-length inner products, where we assume $n = 2b$ for simplicity:

$$\begin{aligned} s_1 &= x(1:b)^T y(1:b),\\ s_2 &= x(b+1:n)^T y(b+1:n),\\ s &= s_1 + s_2. \end{aligned}$$

For this formulation the error bound is

$$|s - \widehat{s}| \le \gamma_{n/2+1} |x|^T |y|,$$

so the error constant has been reduced by a factor 2. We can generalize this idea. Assuming that** $n = kb$, we can compute

$$\begin{aligned} s_i &= x((i-1)b+1 : ib)^T y((i-1)b+1 : ib), \quad i = 1 : k,\\ s &= s_1 + s_2 + \cdots + s_k, \end{aligned} \tag{2.3}$$

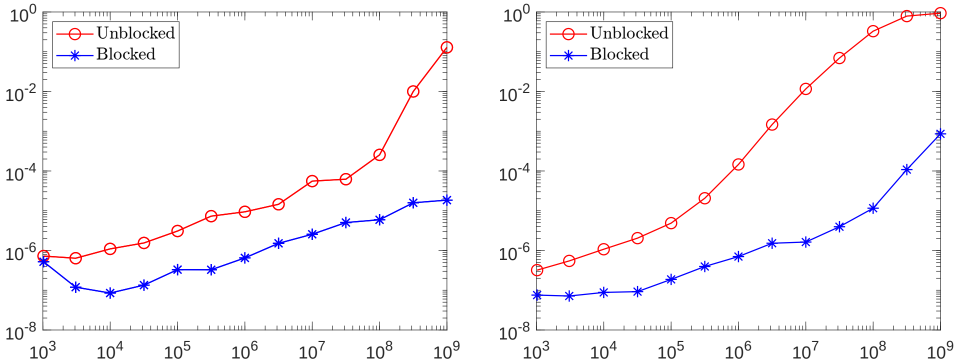
and the error bound is [21, Sec. 3.1]

$$|s - \widehat{s}| \le \gamma_{b+n/b-1} |x|^T |y|. \tag{2.4}$$

As long as $b \ll n$ the error constant has been reduced by about a factor b. The reason for the reduction is that whereas for the standard evaluation (2.1) elements of x and y take part in up to $n-1$ additions, for (2.3) they take part in at most $b + n/b - 2$ additions. The value

[*] A blocked algorithm organizes a computation so that it works on separate chunks of data. It is also commonly called a "block algorithm", but the use of "block" is best reserved for properties, factorizations, and algorithms in which scalars are generalized into blocks. For example, a block tridiagonal matrix is not, in general, tridiagonal, and a block LU factorization is different from an LU factorization because it has a block upper triangular U.

[**] This is not a practical restriction, as for general n we can compute the inner product of the last n mod b elements separately or pad the vectors with zeros so that their dimension is a multiple of b.



(a) Elements of x and y generated as $-0.25 + \text{randn}$, where randn is drawn from the normal $(0, 1)$ distribution.

(b) Elements of x and y generated from the uniform distribution on $(0, 1)$.

Figure 1 Relative errors in inner products computed in single precision with no blocking and with block size 256.

of b that minimizes the bound (2.4) is $b = \sqrt{n}$, so if we take for b the nearest integer to $\sqrt{n}$ we will have

$$|s - \widehat{s}| \lesssim \gamma_{2\sqrt{n}}|x|^T|y|.$$

By splitting the inner product into pieces, computing the partial inner products, and summing the results, we have reduced the error constant from n to $2\sqrt{n}$, which is a substantial reduction for large n.

Blocking of inner products is common in practice, though a fixed block size rather than one depending on n is normally taken [8]. It may be done in a low-level kernel for performance considerations and so may be invisible to a user.

To illustrate the benefits of blocking we show in Figure 1 the relative errors in inner products computed with and without blocking for two types of random vector. The dimension n ranges from 10^3 to 10^9, the block size is 256, and the relative errors are averaged over 10 pairs of vectors x and y for each n. The reason for shifting the normally distributed random vectors is to make the mean nonzero, as for a zero mean the errors tends to be much smaller [23]. The more rapid growth of the errors for the unblocked computation that begins around $n = 10^7$ for both distributions is due to stagnation (described in Section 4.3).

The blocking approach just described can be improved by using a combination of two different methods. We will illustrate the idea for summation, but it trivially generalizes to inner products.

Assume that we have at our disposal two summation algorithms: a fast one, referred to as the FastSum algorithm, and an accurate one, referred to as the AccurateSum algorithm. Algorithm 2.1 uses these two algorithms to compute $\sum_{i=1}^{n} z_i$ by an algorithm of Blanchard, Higham, and Mary [6]. The algorithm is called FABsum, which stands for “fast and accurate blocked summation.” To compute the inner product $x^T y$, we can take $z_i = x_i y_i$. We assume that n is a multiple of b.

Algorithm 2.1 (FABsum). *This algorithm takes as input n summands z_i, a block size b that divides n, and two summation algorithms FastSum and AccurateSum. It returns the sum $s = \sum_{i=1}^{n} z_i$.*

```
for $i = 1 : n/b$
    Compute $s_i = \sum_{j=(i-1)b+1}^{ib} z_j$ with FastSum.
end
Compute $s = \sum_{i=1}^{n/b} s_i$ with AccurateSum.
```

Note that for $b = 1$, FABsum reduces to AccurateSum and for $b = n$ it reduces to FastSum. The motivation for FABsum is that if b is chosen large enough most of the work is done by FastSum but the use of AccurateSum can lead to improved accuracy.

Assume that for a sum $s = \sum_{i=1}^{n} z_i$ the computed $\widehat{s}$ from FastSum satisfies

$$\widehat{s} = \sum_{i=1}^{n} z_i(1 + \mu_i^f), \quad |\mu_i^f| \le \epsilon_f(n) \tag{2.5}$$

and the computed $\widehat{s}$ from AccurateSum satisfies

$$\widehat{s} = \sum_{i=1}^{n} z_i(1 + \mu_i^a), \quad |\mu_i^a| \le \epsilon_a(n), \tag{2.6}$$

where $\epsilon_f(n)$ and $\epsilon_a(n)$ are $O(u)$ and depend on n and u. With these assumptions on the backward errors of the underlying summation algorithms, we have the following backward error result [6, Thm. 3.1].

Theorem 2.2. *Let $s = \sum_{i=1}^{n} z_i$ be computed by Algorithm 2.1. The computed $\widehat{s}$ satisfies*

$$\widehat{s} = \sum_{i=1}^{n} z_i(1 + \mu_i), \quad |\mu_i| \le \epsilon(n, b) = \epsilon_f(b) + \epsilon_a(n/b) + \epsilon_f(b)\epsilon_a(n/b).$$

To see the gains in accuracy Algorithm 2.1 can bring, consider the following two choices. For FastSum take recursive summation, which is the usual algorithm that computes $s = z_1 + z_2$, $s \leftarrow s + z_k$, $k = 3\colon n$. Then $\epsilon_f(b) = (b-1)u + O(u^2)$. If AccurateSum is recursive summation at twice the working precision then $\epsilon_a(n/b) = u + O(u^2)$ and so $\epsilon(n, b) = bu + O(u^2)$ is independent of n to first order. If AccurateSum is the method known as compensated summation [21, Sec. 4.3], which works entirely in the working precision and for which $\epsilon_a(n/b) = 2u + O(u^2)$, then $\epsilon(n, b) = (b+1)u + O(u^2)$, which again does not grow with n to first order. Analysis of the second order terms in [6, Sec. 3.1.2] shows that they are not significant unless n is extremely large.

Denote by $C(n,b)$ the cost in flops of Algorithm 2.1. If $C_f(n)$ and $C_a(n)$ are the costs for summing n terms by FastSum and AccurateSum, respectively, then

$$C(n,b) = \frac{n}{b}C_f(b) + C_a\Big(\frac{n}{b}\Big).$$

In particular, if the costs C_f and C_a are linear functions of the number of summands, as is usually the case, $C(n,b)$ simplifies to

$$C(n,b) = C_f(n) + \frac{1}{b}C_a(n) + O\Big(\frac{n}{b}\Big).$$

Therefore the cost of Algorithm 2.1 can be made arbitrarily close to that of FastSum by increasing the block size b. The parameter b can be tuned to achieve the highest possible performance on a given target architecture, while keeping it independent of n to avoid error growth.

We have seen that by using a blocked implementation of summation or an inner product it is possible to reduce the error constant $nu + O(u^2)$ by a constant factor, or even to reduce it to $(b+1)u + O(u^2)$ by using FABsum, while at the same time increasing the performance. The increased performance and reduced error bound go hand in hand.

2.2. Blocked matrix multiplication

The standard error bound for a matrix product $C = AB$, where $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times t}$, is

$$|C - \widehat{C}| \le \gamma_n |A||B|,$$

which is an immediate consequence of (2.2), and this bound holds for any order of evaluation. Consider Algorithm 2.3, which is a blocked implementation of matrix multiplication that amounts to computing each element of the product by the blocked inner product (2.3).

Algorithm 2.3 (Blocked matrix multiplication). *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times t}$ be partitioned into $b \times b$ blocks A_{ij} and B_{ij}, where $p = m/b_1$, $q = n/b$, and $r = t/b_2$ are assumed to be integers. This algorithm computes $C = AB$.*

```
1 for $i = 1\colon p$
2     for $j = 1\colon r$
3         $C_{ij} = 0$
4         for $k = 1\colon q$
5             $X = A_{ik}B_{kj}$
6             $C_{ij} = C_{ij} + X$
7         end
8     end
9 end
```

We have written lines 5 and 6 as shown in order to make clear that $A_{ik}B_{kj}$ is computed and then added to C_{ij}. The expression "$C_{ij} = C_{ij} + A_{ik}B_{kj}$" would be ambiguous because for an individual element of C_{ij} it has the form

$$c_{i_1,j_1} = c_{i_1,j_1} + \sum_{\ell=1}^{b} a_{i_1,\ell} b_{\ell,j_1}, \tag{2.7}$$

and the order in which the b additions are done is not specified. If the additions are done from left to right then the algorithm is numerically equivalent to standard matrix multiplication. However, in Algorithm 2.3 the addition involving c_{i_1,j_1} is done last.

A rounding error result for Algorithm 2.3 follows readily from that for a blocked inner product: the computed $\widehat{C}$ satisfies

$$|C - \widehat{C}| \le \gamma_{b+n/b-1} |A||B|. \tag{2.8}$$

Again, if $b \ll n$ then the error constant has been reduced by about a factor b. In a highly optimized matrix multiplication algorithm there may be multiple levels of blocking [17], which give a further reduction in the error bound.

We note that the FABsum algorithm (Algorithm 2.1) and its rounding error analysis trivially extend to matrix multiplication [6, Sec. 4].

2.3. Blocked matrix factorizations

The LAPACK library [3] pioneered the use of blocked algorithms that compute a matrix factorization a block at a time, where each block is square or rectangular with b columns, with the block size typically $b = 128$ or $b = 256$. These algorithms typically contain operations of the form $A_{ij} = A_{ij} - X_i Y_j$, and these are implemented as calls to a level 3 BLAS gemm (general matrix multiply) routine [12], which computes $C \leftarrow \alpha AB + \beta C$ for arbitrary matrices A, B, and C of conformable dimensions. In view of the error bound (2.8) for Algorithm 2.3, the blocked algorithm will have a constant in a (componentwise) backward error bound that is about b times smaller than for the unblocked algorithm provided that the gemm computes αAB before adding the result to βC. And of course, for block-level computations that are inner product-based, the blockings of the previous subsections can be applied with a smaller block size, giving a further reduction in error bound.

Most references do not take advantage of blocking when stating error bounds. The LAPACK manual [3] states error bounds of the form $p(n)u$ for $n \times n$ matrices, where $p(n)$ is independent of the block size. Standard texts such as those of Demmel [11], Golub and Van Loan [16], and Higham [21] give error analysis only for unblocked algorithms, so do not derive the b-dependent constants for the blocked algorithms (though [21, Sec. 13.2] derives the constants for blocked LU factorization). We suggest three reasons why error analyses for blocked algorithms are usually not provided. First, as explained in Section 1, there has long been a feeling, going back to Wilkinson, that the most important part of a bound is not the constants but the form of the bound and that optimizing constants is not worthwhile. Second, the precise constants depend on which blocked algorithm variant of a factorization is chosen (there are usually several) and precisely how it is implemented. Third, the error analysis for a blocked algorithm tends to be more complicated than for the unblocked algorithm, which can obscure the main ideas of the analysis.

The important point to note is that with a suitable implementation the constant in a backward error bound for a blocked factorization with block size b will be reduced by a factor of order b or more.

3. Architectural features

A number of features of modern processors contribute to reducing the error in numerical computations.

3.1. Extended precision registers

Intel x86 processors support an 80-bit extended precision format with a 64-bit significand (see Table 2), which is compatible with that specified in the IEEE standard [25], [27, Sec. 4.2.2], [29, Sec. 3.4.3]. When a compiler uses this format with 80-bit registers to accumulate sums and inner products it is effectively working with a unit roundoff of 2^{-64} rather than 2^{-53} for double precision, giving error bounds smaller by a factor up to $2^{11} = 2048$. We note, however, that extra precision registers can lead to strange rounding effects, in particular because of double rounding [21, sec. 2.3, Probs. 27.1, 27.3], [29].

3.2. Fused multiply–add

Another architectural feature that provides benefits to accuracy is a fused multiply–add (FMA) operation, which computes $x + y * z$ with just one rounding error instead of two. Without an FMA,

$$\mathrm{fl}(x + y * z) = \big(x + y * z(1 + \delta_1)\big)(1 + \delta_2), \quad |\delta_1| \le u, \quad |\delta_2| \le u,$$

whereas with an FMA,

$$\mathrm{fl}(x + y * z) = (x + y * z)(1 + \delta), \quad |\delta| \le u,$$

which means that the result is computed with a relative error bounded by u. The motivation for an FMA is speed, as it is implemented in such a way that it takes the same time as a single multiplication or addition. With the use of an FMA standard error bounds for inner product-based computations are reduced by a factor 2. It should be noted, though, that an FMA can lead to unexpected results when applied to certain expressions [21, sec. 2.6].

3.3. Mixed precision block fused multiply–add

A mixed precision block FMA takes as input matrices $A \in \mathbb{R}^{b_1 \times b}$, $B \in \mathbb{R}^{b \times b_2}$, and $C \in \mathbb{R}^{b_1 \times b_2}$, where A and B are provided in a given precision u_{low} and C is either in precision u_{low} or in a higher precision u_{high}, and computes

$$\underbrace{D}_{u_{\mathrm{low}} \text{ or } u_{\mathrm{high}}} = \underbrace{C}_{u_{\mathrm{low}} \text{ or } u_{\mathrm{high}}} + \underbrace{A}_{u_{\mathrm{low}}} \; \underbrace{B}_{u_{\mathrm{low}}}, \tag{3.1}$$

returning D in precision u_{low} or u_{high}. We will assume that the internal computations are at precision u_{high}. The output matrix D can be used as the input C to a subsequent FMA, so by

Table 3 Processing units or architectures equipped with mixed-precision fused multiply–add accelerators. Matrix dimensions are expressed as $b_1 \times b \times b_2$, where b_1 is the number of rows in A, b is the number of columns in A and rows in B, and b_2 is the number of columns in B. The input and output precisions u_{low} and u_{high} are defined in (3.1). Sources [4], [9], [30].

Year of release	Device	Matrix dimensions	u_{low}	u_{high}
2016	Google TPU v2	$128 \times 128 \times 128$	bfloat16	fp32
2017	Google TPU v3	$128 \times 128 \times 128$	bfloat16	fp32
2017	NVIDIA V100	$4 \times 4 \times 4$	fp16	fp32
2018	NVIDIA T4	$4 \times 4 \times 4$	fp16	fp32
2019	ARMv8.6-A	$2 \times 4 \times 2$	bfloat16	fp32
2020	NVIDIA A100	$8 \times 8 \times 4$	bfloat16	fp32
		$8 \times 8 \times 4$	fp16	fp32
		$8 \times 4 \times 4$	TensorFloat-32	fp32
		$2 \times 4 \times 2$	fp64	fp64

chaining FMAs together in this way, larger matrix products can be computed [5, Alg. 3.1]. Table 3 gives the precisions and matrix dimensions for block FMAs available in hardware. These block FMAs are designed to give one result per cycle and so can give significant performance benefits. For example, on the NVIDIA V100 GPU, whose tensor cores implement block FMAs, half precision arithmetic on the tensor cores runs 8 times faster than single precision arithmetic, which in turn runs at twice the speed of double precision arithmetic.

When C and D in (3.1) are taken at the higher precision, u_{high}, mixed precision block FMAs give an increase in accuracy compared with computations carried out at the lower precision, u_{low}. Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times t}$ be given in precision u_{high} and partitioned into $b_1 \times b$ blocks A_{ij} and $b \times b_2$ blocks B_{ij}, respectively, where $p = m/b_1$, $q = n/b$, and $r = t/b_2$ are assumed to be integers. When the product $C = AB$ is computed by a sequence of chained block FMAs using [5, Alg. 3.1] (which has the same general form as Algorithm 2.3), it can be shown [5, Thm. 3.2] that the computed $\widehat{C}$ satisfies

$$|C - \widehat{C}| \le f(n, b, u_{\mathrm{low}}, u_{\mathrm{high}})|A||B|, \tag{3.2}$$

where the first order part of f is given in Table 4. We see that for $n < 2u_{\mathrm{low}}/u_{\mathrm{high}} =: n_*$ the block FMA constant is independent of n. It is always smaller than the constant for standard multiplication at precision u_{low} and of similar magnitude to the constant for standard multiplication at precision u_{high} for $n > n_*$. When u_{low} corresponds to fp16 or bfloat16 and u_{high} to fp32, we have $n_* = 16{,}384$ and $n_* = 131{,}072$, respectively. Hence while a mixed precision block FMA takes inputs at precision u_{low}, for large n it produces results as good as if the computation was done at precision u_{high}.

Table 4 First order part of constant term $f(n, b, u_{\text{low}}, u_{\text{high}})$ in error bound (3.2) for matrix multiplication with and without use of a mixed precision block FMA.

Evaluation method	Bound
Standard in precision u_{low}	$(n+2)u_{\text{low}}$
Block FMA, u_{high} internally	$2u_{\text{low}} + nu_{\text{high}}$
Standard in precision u_{high}	nu_{high}

Note that (3.2) assumes that, in the notation of Algorithm 2.3, lines 5 and 6 are evaluated as $C_{ij} = C_{ij} + A_{ik}B_{kj}$, in left to right order; if the evaluation uses lines 5 and 6 as stated then the u_{high} term in Table 4 for the block FMA is further reduced. However, NVIDIA tensor cores in the Volta, Turing, and Ampere microarchitectures with $b = 4$ do not use a fixed order when evaluating each individual element $c_{ij} + a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + a_{i4}b_{4j}$ in (3.1), but rather evaluate the expression starting with the largest magnitude term [15].

4. Probabilistic rounding error analysis

We have now seen two main reasons why standard rounding error bounds may be pessimistic: first, they do not account for block algorithms and second, architectural features of the computer may provide increased accuracy for certain types of operations. We now discuss a third reason, which has to do with the very nature of rounding error bounds.

In the model (1.4), the relative error δ in $\text{fl}(x \text{ op } y)$ is typically strictly less than u in magnitude, and of course it is zero if $x \text{ op } y$ happens to be a floating-point number. Rounding error analyses apply (1.4) repeatedly. Typically, a product of $1+\delta_i$ terms appears, which can be handled by the next lemma [21, Lem. 3.1].

Lemma 4.1. *If $|\delta_i| \le u$ and $\rho_i = \pm 1$ for $i = 1 : n$, and $nu < 1$, then*

$$\prod_{i=1}^{n} (1+\delta_i)^{\rho_i} = 1 + \theta_n, \quad |\theta_n| \le \gamma_n. \tag{4.1}$$

This lemma, combined with some useful identities satisfied by the γ_k and θ_k [21, Lem. 3.3], provides a convenient way to carry out rounding error analyses. However, the proof of the lemma involves multiple uses of the triangle inequality and so the bound $|\theta_n| \le \gamma_n$ can be expected to be potentially weak.

For a given algorithm and a given set of data we would like to be able to say that there exists a set of rounding errors δ_i that, if they occur, produce an error of roughly the same size as the rounding error bound. This is usually not the case, but it can be true if in every invocation of (1.4) δ has the same sign. For basic kernels it may be possible to show that the error bound is approximately attainable for a special choice of the data, as is the case for recursive summation [21, Prob. 4.2], [35, p. 19], but such examples do not indicate the quality of the bound in typical cases.

In an early paper on rounding error analysis, Wilkinson derives rounding error bounds for Gaussian elimination, Givens QR factorization, and Householder QR factorization and then states that [34, p. 318]

> *"The bounds we have obtained are in all cases strict upper bounds. In general, the statistical distribution of the rounding errors will reduce considerably the function of n occurring in the relative errors. We might expect in each case that this function should be replaced by something which is no bigger than its square root and is usually appreciably smaller."*

He makes similar statements in [35]. For many years, primarily because of Wilkinson's comments, it has been regarded as a rule of thumb that a worst-case rounding error bound $f(n)u$ is more realistic if it is replaced by $\sqrt{f(n)}u$. No proof has been given to make this rule of thumb rigorous, but one can argue as follows:

- linearize the error into a sum $e = \sum_{i=1}^{p} t_i\delta_i$, where the δ_i are rounding errors and the t_i depend on the data;
- assume that the δ_i are independent random variables of mean zero;
- apply the central limit theorem to deduce that the probability distribution of $e/(\sum_{i=1}^{n} t_i^2)^{1/2}$ tends towards a normal distribution of mean zero and standard deviation $\sigma \le u$;
- conclude that for sufficiently large n, the probability that $|e|$ will not exceed $u(\sum_{i=1}^{n} t_i^2)^{1/2}$ times a small constant is very high.

Compared with the worst-case constant $\sum_{i=1}^{n} |t_i|$, the quantity $(\sum_{i=1}^{n} t_i^2)^{1/2}$ can be smaller by a factor up to $\sqrt{n}$. This argument, however, has a number of weaknesses. First, it is essentially forward error-based, whereas we prefer to work with backward errors if possible. Second, the argument is based on the first-order part of the error, so says nothing about higher order terms. Third, it is not clear how large n must be for the application of the central limit theorem to be valid.

Despite the weaknesses of a central limit theorem argument, a probabilistic approach seems to be necessary to obtain substantially better bounds than the worst-case ones. Indeed, as Stewart [33] has noted,

> *"To be realistic, we must prune away the unlikely. What is left is necessarily a probabilistic statement."*

We will discuss probabilistic rounding error analysis in the next two subsections.

4.1. Error analysis for nonrandom data

Higham and Mary [22] introduced a new probabilistic rounding error analysis, making use of a concentration inequality. This work was extended by Higham and Mary for random data [23], and by Ipsen and Zhou [28], and Connolly, Higham, and Mary [10], all of whom use martingales. We will present the most general results for nonrandom data, which are those from [10].

We need the following probabilistic version of Lemma 4.1 [10, Lem. 4.2], [22, Thm. 2.4], which includes the constant

$$\widetilde{\gamma}_n(\lambda) = \exp\left(\frac{\lambda\sqrt{n}u + nu^2}{1-u}\right) - 1 = \lambda\sqrt{n}u + O(u^2). \tag{4.2}$$

Theorem 4.2. *Let $\delta_1, \delta_2, \dots, \delta_n$ be random variables of mean zero with $|\delta_k| \le u$ for all k such that $\mathbb{E}(\delta_k \mid \delta_1, \dots, \delta_{k-1}) = \mathbb{E}(\delta_k) = 0$ for $k = 2\colon n$. Then for $\rho_i = \pm 1$, $i = 1\colon n$ and any constant $\lambda > 0$,*

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n, \quad |\theta_n| \le \widetilde{\gamma}_n(\lambda) \tag{4.3}$$

holds with probability at least $1 - 2\exp(-\lambda^2/2)$.

The key difference between (4.1) and (4.3) is that, to first order, the bound in (4.3) is proportional to $\sqrt{n}u$ rather than nu.

Next, we need the following model of rounding errors.

Model 4.3 (probabilistic model of rounding errors). *Let the computation of interest generate rounding errors $\delta_1, \delta_2, \dots$ in that order. The δ_k are random variables of mean zero such that $\mathbb{E}(\delta_k \mid \delta_1, \dots, \delta_{k-1}) = \mathbb{E}(\delta_k)$ $(= 0)$.*

The model says that the rounding errors δ_i are mean independent and of mean zero, but they do not need to be from the same distribution. Mean independence is a weaker condition than independence: if the rounding errors are independent then they can be shown to be mean independent, but the converse implication does not hold. Under the model, Theorem 4.2 holds and allows us to bound rounding error terms that appear in analyses of inner product-based computations. This leads to the following three results [10, Thms. 4.8–4.10], in which

$$Q(\lambda, n) = 1 - 2n\exp(-\lambda^2/2).$$

Theorem 4.4 (inner products). *Let $s = x^T y$, where $x, y \in \mathbb{R}^n$, be evaluated in floating-point arithmetic. Under Model 4.3, no matter what the order of evaluation the computed $\widehat{s}$ satisfies*

$$\widehat{s} = (x + \Delta x)^T y = x^T(y + \Delta y), \quad |\Delta x| \le \widetilde{\gamma}_n(\lambda)|x|, \ |\Delta y| \le \widetilde{\gamma}_n(\lambda)|y| \tag{4.4}$$

with probability at least $Q(\lambda, n)$.

Theorem 4.5 (matrix products). *Let $C = AB$ with $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$. Under Model 4.3, the jth column of the computed $\widehat{C}$ satisfies*

$$\widehat{c}_j = (A + \Delta A_j)b_j, \quad |\Delta A_j| \le \widetilde{\gamma}_n(\lambda)|A|, \quad j = 1\colon n, \tag{4.5}$$

with probability at least $Q(\lambda, mn)$, and hence

$$|C - \widehat{C}| \le \widetilde{\gamma}_n(\lambda)|A||B| \tag{4.6}$$

with probability at least $Q(\lambda, mnp)$.

Theorem 4.6 (linear system). *Let $A \in \mathbb{R}^{n\times n}$ and suppose that LU factorization and substitution produce computed factors $\widehat{L}$ and $\widehat{U}$ and a computed solution $\widehat{x}$ to $Ax = b$. Then, under Model 4.3,*

$$(A + \Delta A)\widehat{x} = b, \quad |\Delta A| \le \bigl(3\widetilde{\gamma}_n(\lambda) + \widetilde{\gamma}_n(\lambda)^2\bigr)|\widehat{L}||\widehat{U}| \tag{4.7}$$

holds with probability at least $Q(\lambda, n^3/3 + 3n^2/2 + 7n/6)$.

Matrix multiplication and LU factorization both have triply nested loops, which can be ordered in $3! = 6$ ways. Theorems 4.5 and 4.6 both hold no matter which ordering of the loops is taken.

Let us now focus our attention on Theorem 4.6. For $n = 10^8$, the function $Q(\lambda, n^3/3 + 3n^2/2 + 7n/6)$ approaches 1 rapidly as λ increases and is approximately $1 - 10^{-3}$ for $\lambda = 11$ and $1 - 10^{-8}$ for $\lambda = 12$. Moreover, as shown in [22, Sec. 3.5], $Q(\lambda, f(n))$ remains independent of n as long as λ increases proportionally to $\log n$. Experiments show that the probability $Q(\lambda, f(n))$ is actually very pessimistic and in practice the bounds usually hold with $\lambda = 1$.

Theorems 4.2– 4.6 provide a rigorous proof of the rule of thumb stated by Wilkinson, under the assumptions of Model 4.3.

Probabilistic error analysis can also be applied to blocked algorithms, with the blocking and the probabilistic approach combining to reduce the error constant. For example, the error constant $(b + n/b - 1)u + O(u^2)$ in (2.4) for a blocked inner product translates to $(\sqrt{b} + \sqrt{n/b})u + O(u^2)$ in a probabilistic bound.

4.2. Error analysis for random data

Numerical experiments show that the bounds in Theorems 4.4–4.6 reflect the actual rate of growth of the error with n for some problems [22], [23], but the bounds can nevertheless be pessimistic. Higham and Mary [23] investigate the case where the data is random. They use the following model for the data, which is denoted by d_j, $j = 1\colon n$.

Model 4.7 (probabilistic model of the data). *The d_j, $j = 1\colon n$, are independent random variables sampled from a given distribution of mean μ_x and satisfy $|d_j| \le C_x$, $j = 1\colon n$, where C_x is a constant.*

A modified version of Model 4.3 is needed.

Model 4.8 (modified probabilistic model of rounding errors). *Let the computation of interest generate rounding errors $\delta_1, \delta_2, \dots$ in that order. The δ_i are random variables of mean zero and, for all k, the δ_k are mean independent of the previous rounding errors and of the data, in the sense that*

$$\mathbb{E}(\delta_k \mid \delta_2, \dots, \delta_{k-1}, d_1, \dots, d_n) = \mathbb{E}(\delta_k)\ (= 0). \tag{4.8}$$

Under these models, Higham and Mary [23] obtain error bounds for an inner product, a matrix–vector product, and a matrix product. We state the result for matrix products [23, Thm. 3.4].

Theorem 4.9. *Let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ satisfy Model 4.7 with means μ_A, μ_B and bounds C_A, C_B, and let $C = AB$. Under Model 4.8, the computed $\widehat{C}$ satisfies*

$$\max_{i,j} |(C - \widehat{C})_{ij}| \le \big(\lambda|\mu_A\mu_B|n^{3/2} + (\lambda^2 + 1)C_AC_Bn\big)u + O(u^2) \tag{4.9}$$

with probability at least $P(\lambda) = 1 - 2mnp\exp(-\lambda^2/2)$.

The rate of growth of the bound (4.9) is $n^{3/2}$ except when μ_A or μ_B is small or zero, in which case it is just n. Thus the error bound depends on the means of the data. Furthermore, it is shown in [23, Thm. 3.3] that for an inner product x^Ty in which either x or y has zero mean the backward error is bounded by $c_1u + O(u^2)$ instead of $c_2\sqrt{n}u + O(u^2)$ as in Theorem 4.4, where c_1 and c_2 are constants.

We note that extending this analysis with random data to the solution of linear systems by LU factorization is an open problem, as noted in [23, Sec. 5].

4.3. Limitations

It is important to realise that the assumptions of the probabilistic rounding error analysis may not hold: the rounding errors may be dependent or may have nonzero mean, and in these cases the error may grow as nu rather than $\sqrt{n}u$. Consider a sum $\sum_{i=1}^n x_i$ computed by recursive summation, where the x_i are positive and decrease with i. For a large enough i, the summand x_i may be so small that it does not change the current partial sum in floating-point arithmetic. From this point on no summand changes the sum so the rounding errors are all negative and Model 4.3 does not hold, and in this circumstance the worst-case linear growth can be achieved, as can be shown by numerical examples [10], [22]. This problem is called stagnation. A cure for stagnation is to randomize the rounding using stochastic rounding [10], which ensures that the sum can increase. Indeed with stochastic rounding, Model 4.3 is always satisfied and so by Theorem 4.4 the error in the sum grows as $\sqrt{n}u$ instead of nu with high probability.

5. Other considerations

5.1. Sharpness of error bounds

The bound (1.1) of Theorem 1.1 is not the best we can obtain. In the proof of the bound in [21] it is first shown that $A + \Delta A_1 = \widehat{L}\widehat{U}$, where

$$|\Delta A_1| \le \begin{bmatrix} \gamma_1 & \gamma_1 & \cdots & \cdots & \gamma_1 \\ \gamma_1 & \gamma_2 & \cdots & \cdots & \gamma_2 \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \gamma_{n-1} & \gamma_{n-1} \\ \gamma_1 & \gamma_2 & \cdots & \gamma_{n-1} & \gamma_n \end{bmatrix} \circ |\widehat{L}||\widehat{U}| \equiv H \circ |\widehat{U}|, \tag{5.1}$$

where $\circ$ is the Hadamard product: $A \circ B = (a_{ij}b_{ij})$. The bound (1.1) corresponds to replacing every element of H by γ_n. Analogous replacements are made in the part of the analysis dealing with the solution of the triangular systems by substitution. Clearly, then, not all n^2 inequalities in (1.1) are sharp. The same is true of (4.7), as its proof is analogous to that of (1.1). However, (5.1) still contains a term γ_n and so this sharper bound does not bring any significant benefits.

5.2. Growth factor at low precisions

Wilkinson showed that with partial pivoting the growth factor ρ_n for LU factorization is bounded by 2^{n-1}, and he noted that ρ_n is nevertheless usually small in practice. Many years of experience have confirmed that ρ_n is indeed usually less than 50 (say) in practice. The growth factor directly affects the backward error bounds, through the size of the elements of $\widehat{U}$ and (1.2). When we are working in half precision, with unit roundoff $u \approx 5 \times 10^{-4}$ for fp16 or $u \approx 4 \times 10^{-3}$ for bfloat16, element growth can have a much bigger relative effect on the quality of a solution than for single precision or double precision. Matrices that give large growth factors for partial pivoting are known, and a class of random matrices of arbitrary condition number that typically have $\rho_n \geq n/(4 \log n)$ was recently identified by Higham, Higham, and Pranesh [20]. For the latter class of matrices, growth alone can cause a complete loss of numerical stability for $n \geq 10^5$ in fp16—and, to complicate matters, it can also cause overflow in fp16 [24].

5.3. Iterative refinement

If we solve $Ax = b$ in half precision arithmetic then of course we cannot expect a backward error smaller than the unit roundoff u_h for half precision. However, we can obtain a numerically stable solution at higher precision by using the computed half precision solution as a first approximation that we improve by iterative refinement at the higher precision, using the half precision LU factors. This procedure is guaranteed to work only for condition numbers $\kappa(A) = \|A\|\|A^{-1}\|$ up to u_h^{-1}. GMRES-based iterative refinement solves the update equation by GMRES preconditioned by the LU factors and can tolerate much more ill conditioned A. See [2], [7], [19] for details of GMRES-based iterative refinement. Although this mixed precision algorithm uses higher precision to raise the quality of the initial solution, the conditions for success rest on the rounding error bounds for the factorization, and so the considerations of this paper contribute to our understanding of the algorithm.

6. Conclusions

We have seen that several factors combine to make errors in inner-product based computations much smaller than worst-case rounding error bounds suggest. Block algorithms can reduce error bounds by a factor of the block size b, and if blocking is used at multiple levels then the reduction factors can accumulate. Extended precision registers and (block) FMAs can give automatic accuracy boosts. With a block size $b = 256$ and the 80-

bit registers on Intel x86-64 processors a reduction in an error bound by a constant factor $256 \times 2048 = 5.2 \times 10^5$ is possible for large problems.

The rate of growth of the error can be much smaller than the worst-case bounds because of statistical effects. If the rounding errors are mean independent and of mean zero then, as explained in Section 4.1, for inner products, matrix–vector products, matrix products, and the solution of linear systems by LU factorization, the constant $\gamma_n = nu + O(u^2)$ in a worst-case componentwise backward error bound can be replaced by $\widetilde{\gamma}_n = \sqrt{n}u + O(u^2)$ to obtain a bound that holds with high probability. Even these bounds can be pessimistic because, as explained in Section 4.7, when the data is random with zero mean the error bound reduces further—to a constant independent of n for an inner product.

Together, these aspects go a considerable way to explaining why linear systems, and other linear algebra problems, are able to be successfully solved with ever growing dimensions and with the use of low precision arithmetics (perhaps within a mixed precision algorithm [1]).

It is pleasing to note that blocked algorithms and (block) FMAs, which were introduced to boost performance, also yield smaller rounding error bounds. It will be important to analyze future developments in algorithms and computer architectures to understand their effects on rounding error analysis.

Acknowledgments

I thank Jack Dongarra, Massimilano Fasi, Sven Hammarling, Theo Mary, and Mantas Mikaitis for their comments on a draft of this paper.

Funding

This work was supported by Engineering and Physical Sciences Research Council grant EP/P020720/1, the Royal Society, and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

[1] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B. F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y. M. Tsai, and U. M. Yang, A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. *Int. J. High Performance Computing Applications* **35** (2021), no. 4, 344–369.

[2] P. Amestoy, A. Buttari, N. J. Higham, J.-Y. L'Excellent, T. Mary, and B. Vieublé, Five-precision GMRES-based iterative refinement. MIMS EPrint 2021.5, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, 2021, URL http://eprints.maths.manchester.ac.uk/id/eprint/2807.

[3] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen, *LAPACK users' guide*. Third edn., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[4] *Arm architecture reference manual. Armv8, for Armv8-A architecture profile*. ARM DDI 0487F.b (ID040120), ARM Limited, Cambridge, UK, 2020.

[5] P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh, Mixed precision block fused multiply-add: Error analysis and application to GPU tensor cores. *SIAM J. Sci. Comput.* **42** (2020), no. 3, C124–C141.

[6] P. Blanchard, N. J. Higham, and T. Mary, A class of fast and accurate summation algorithms. *SIAM J. Sci. Comput.* **42** (2020), no. 3, A1541–A1557.

[7] E. Carson and N. J. Higham, Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. Sci. Comput.* **40** (2018), no. 2, A817–A847.

[8] A. M. Castaldo, R. C. Whaley, and A. T. Chronopoulos, Reducing floating point error in dot product using the superblock family of algorithms. *SIAM J. Sci. Comput.* **31** (2008), 1156–1174.

[9] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, NVIDIA A100 tensor core GPU: Performance and innovation. *IEEE Micro* **41** (2021), no. 2, 29–35.

[10] M. P. Connolly, N. J. Higham, and T. Mary, Stochastic rounding and its probabilistic backward error analysis. *SIAM J. Sci. Comput.* **43** (2021), no. 1, A566–A585.

[11] J. W. Demmel, *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[12] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, A set of Level 3 basic linear algebra subprograms. *ACM Trans. Math. Software* **16** (1990), no. 1, 1–17.

[13] J. J. Dongarra, P. Luszczek, and A. Petitet, The LINPACK benchmark: Past, present and future. *Concurrency Computat. Pract. Exper.* **15** (2003), 803–820.

[14] J. J. Dongarra, P. Luszczek, and Y. M. Tsai, HPL-AI Mixed-Precision Benchmark. https://icl.bitbucket.io/hpl-ai/.

[15] M. Fasi, N. J. Higham, M. Mikaitis, and S. Pranesh, Numerical behavior of NVIDIA tensor cores. *PeerJ Comput. Sci.* **7** (2021), e330(1–19).

[16] G. H. Golub and C. F. Van Loan, *Matrix computations*. Fourth edn., Johns Hopkins University Press, Baltimore, MD, USA, 2013.

[17] K. Goto and R. A. van de Geijn, Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Software* **34** (2008), no. 3, 1–25.

[18] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, Deep learning with limited numerical precision. In *Proceedings of the 32nd international conference on machine learning*, pp. 1737–1746, JMLR: Workshop and Conference Proceedings 37, 2015.

[19] A. Haidar, H. Bayraktar, S. Tomov, J. Dongarra, and N. J. Higham, Mixed-precision iterative refinement using tensor cores on GPUs to accelerate solution of linear systems. *Proc. Roy. Soc. London A* **476** (2020), no. 2243, 20200110.

[20] D. J. Higham, N. J. Higham, and S. Pranesh, Random matrices generating large growth in LU factorization with pivoting. *SIAM J. Matrix Anal. Appl.* **42** (2021), no. 1, 185–201.

[21] N. J. Higham, *Accuracy and stability of numerical algorithms*. Second edn., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[22] N. J. Higham and T. Mary, A new approach to probabilistic rounding error analysis. *SIAM J. Sci. Comput.* **41** (2019), no. 5, A2815–A2835.

[23] N. J. Higham and T. Mary, Sharper probabilistic backward error analysis for basic linear algebra kernels with random data. *SIAM J. Sci. Comput.* **42** (2020), no. 5, A3427–A3446.

[24] N. J. Higham, S. Pranesh, and M. Zounon, Squeezing a matrix into half precision, with an application to solving linear systems. *SIAM J. Sci. Comput.* **41** (2019), no. 4, A2536–A2551.

[25] *IEEE standard for floating-point arithmetic, IEEE Std* 754-2019 (*revision of IEEE* 754-2008). The Institute of Electrical and Electronics Engineers, New York, USA, 2019.

[26] Intel Corporation, BFLOAT16—Hardware Numerics Definition. 2018, URL https://software.intel.com/en-us/download/bfloat16-hardware-numerics-definition, white paper. Document number 338302-001US.

[27] Intel Corporation, Intel 64 and IA-32 architectures software developer's manual. Volume 1: Basic Architecture. 2021, https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture.html.

[28] I. C. F. Ipsen and H. Zhou, Probabilistic error analysis for inner products. *SIAM J. Matrix Anal. Appl.* **41** (2020), no. 4, 1726–1741.

[29] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres, *Handbook of floating-point arithmetic*. Second edn., Birkhäuser, Boston, MA, USA, 2018.

[30] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, The design process for Google's training chips: TPUv2 and TPUv3. *IEEE Micro* **41** (2021), no. 2, 56–63.

[31] B. N. Parlett, The contribution of J. H. Wilkinson to numerical analysis. In *A history of scientific computing*, edited by S. G. Nash, pp. 17–30, Addison-Wesley, Reading, MA, USA, 1990.

[32] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, HPL: A portable implementation of the High-Performance Linpack Benchmark for distributed-memory computers, Version 2.3. 2018, URL https://www.netlib.org/benchmark/hpl/.

[33] G. W. Stewart, Stochastic perturbation theory. *SIAM Rev.* **32** (1990), no. 4, 579–610.

[34] J. H. Wilkinson, Error analysis of direct methods of matrix inversion. *J. ACM* **8** (1961), 281–330.

[35] J. H. Wilkinson, *Rounding errors in algebraic processes*. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963.

[36] J. H. Wilkinson, Modern error analysis. *SIAM Rev.* **13** (1971), no. 4, 548–568.

[37] J. H. Wilkinson, Numerical linear algebra on digital computers. *IMA Bulletin* **10** (1974), no. 2, 354–356.

[38] P. Zamirai, J. Zhang, C. R. Aberger, and C. De Sa, Revisiting bfloat16 training. ArXiv:2010.06192, 2020, URL https://arxiv.org/abs/2010.06192, revised March 2021.

Nicholas J. Higham

Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK,
nick.higham@manchester.ac.uk