Providing Accuracy Guarantees for the Fixed-Point Emulation of Matrix Multiplication

Cole Brower
NVIDIA Corporation
Santa Clara, California, USA
cbrower@nvidia.com

Peter Caday
NVIDIA Corporation
Santa Clara, California, USA
pcaday@nvidia.com

John Gunnels
NVIDIA Corporation
Santa Clara, California, USA
jgunnels@nvidia.com

November 13, 2026

Abstract

Fixed-point methods such as Ozaki-II emulate double-precision matrix multiplication (GEMM) on fast low-precision hardware. Software that chooses the mantissa bit count automatically is numerically safe when the emulated product keeps the error guarantee of the floating-point GEMM it replaces. We analyze the worst-case and average-case error of fixed-point and floating-point dot products, define exponent span capacities that determine the extra mantissa bits an input requires, and approximate them with a low-precision GEMM. We prove that 53 bits plus such a capacity and a small constant preserve the classical worst-case error bound for a floating-point dot product, and we demonstrate this guarantee in software on several test cases. In a developer build of cuBLAS on an NVIDIA B200, each implemented bit-count rule earns grade A on Test 2c of "Grading the BLAS" at every size and exponent range tested, falling back to double precision at the widest ranges; in exact arithmetic, the bound holds on every adversarial and random input tested.

1 Introduction

On modern GPUs, low-precision general matrix multiplication (GEMM) outperforms traditional IEEE-754 single- and double-precision arithmetic by many orders of magnitude, driven by the computational demands of artificial intelligence. Recently-introduced algorithms, in particular the Ozaki-I and Ozaki-II schemes [15, 14, 16], leverage this trend, using fast low-precision hardware to accelerate higher-precision GEMMs. In both algorithms, floating-point input matrices are scaled and quantized to integer matrices, and the resulting high-precision integer GEMM is reduced to a series of lower-precision integer GEMMs. A final reconstruction step undoes the scaling to produce a final floating-point product. We denote this class of algorithms *fixed-point methods*.

Fixed-point methods' accuracy is primarily driven by the precision of the integer quantization (fixed-point mantissa bit count), which is a tunable parameter. A natural question is how to choose the mantissa bit count for a particular application, and in particular, how to match the accuracy of floating-point GEMM. As the accuracy of floating-point GEMM is itself an involved topic [6, 7, 3], this is a subtle question. In a previous paper [18], the *exponent span capacity* (ESC) was proposed as a measure of the number of additional bits of integer precision beyond the 53 mantissa bits of native FP64 values. Fasi et al. [4] recently proved that this choice (with a fixed number of additional bits) ensures that the *worst-case* fixed-point error matches the *worst-case* floating-point GEMM error.

In this paper, we carry the accuracy analysis further, analyzing both average-case and worst-case error for fixed-point and floating-point dot products. Our results naturally lead to two new variants of ESC that provide tighter bounds on the number of required fixed-point mantissa bits to match floating-point accuracy. We show that these variants can be efficiently approximated using a low-precision (bfloat16) GEMM, and also derive an efficient new approximation algorithm for the original ESC that is more robust and often provides tighter ESC bounds than the coarsened approximation of [18]. Finally, we discuss a new

approach to scaling in Ozaki-II and demonstrate that our implementations pass known GEMM accuracy tests from Demmel et al [3], as well as additional proposed stress tests which demonstrates the robustness of our ESC algorithms.

2 Dot Product Accuracy

In this section, we present worst-case and average-case error analyses for both fixed-point and floating-point dot products. Our assumption throughout is that integer and fixed-point arithmetic is performed exactly, so that the only errors in fixed-point methods arise from discretization. Note that this is true of the Ozaki-II scheme but not of the original Ozaki-I scheme, where some lower-order contributions to the integer product are not computed. At the cost of reduced performance, Ozaki-I may be readily modified to compute the integer product in full precision, in which case our analysis applies.

We will also consider the fundamental uncertainty of the dot product, that is, error due to pre-existing rounding of the floating-point inputs, which is independent of the dot product algorithm and reflects the numeric conditioning of the dot product itself.

For clarity, we collect proofs for all the results of this section in Section 7.

2.1 Definitions

Floating-Point Numbers Fix a floating point domain F with M mantissa bits, including the leading 1. For a nonzero $a \in F$, let $\epsilon(a)$ be the maximum distance a number can be from a and still round to a. We will assume round-to-nearest throughout, so $\epsilon(a) = 2^{\lfloor \log_2 |a| \rfloor - M}$ for normal numbers, with no assumptions on the tie-breaking rule. For all a, we have $\frac{1}{2}|a| \cdot \epsilon(1) < \epsilon(a) \leq |a| \cdot \epsilon(1)$. Finite normal floating-point numbers are arranged in *binades* $[2^e, 2^{e+1})$, where e is an integer exponent.

Fixed-Point Numbers A fixed-point number with $\hat{M}$ mantissa bits relative to a maximum value A is a number of the form $\pm 2^{\lfloor \log_2 |A| \rfloor - \hat{M} + 1} \cdot m$, where m is an integer in $[-2^{\hat{M}}, 2^{\hat{M}}]$. For any $a \in [-|A|, |A|]$, the distance between a and the nearest fixed-point number is at most $\hat{\epsilon}(A) = 2^{\lfloor \log_2 |A| \rfloor - \hat{M}}$. (For $A \in F$ and $\hat{M} \geq M$, the values $m = \pm 2^{\hat{M}}$ are never needed.) We will often think of $\hat{M}$ as a sum $M + E$, where E is a nonnegative number of additional fixed-point mantissa bits.

We will use the following variables throughout:

- x, $y \in F^k$: input floating-point vectors, of length k.
- $z = x \cdot y \in \mathbb{R}$: mathematically exact dot product.
- $w = x \circ y \in \mathbb{R}^k$: Hadamard product, $w = (w_i) = (x_i y_i)$.
- $\hat{x}$, $\hat{y} \in \mathbb{R}^k$: fixed-point approximations to x, y, relative to $\|x\|_\infty$ and $\|y\|_\infty$, respectively.
- $\hat{z} = \hat{x} \cdot \hat{y}$: mathematically exact dot product of $\hat{x}$ and $\hat{y}$.

For simplicity, we will assume throughout that the inputs x, y and intermediate floating-point values are finite, normal, nonzero numbers. Denormal numbers, underflow, and overflow will reduce accuracy of floating-point algorithms, but not fixed-point algorithms, so this simplification is reasonable. Without loss of generality, we can drop zero inputs without affecting any algorithm's accuracy.

For average-case analysis, we need to choose probability distributions from which to draw inputs. Typically we will characterize rounding and discretization errors as belonging to uniform distributions, and let $\mathcal{U}(a, b)$ denote the uniform distribution over the interval $[a, b]$.

2.2 Input Rounding Error Bounds

In general, the floating-point inputs x, y originate from a previous floating-point operation, where they were rounded from some exact real values $\overline{x}$, $\overline{y} \in \mathbb{R}^k$. Because we do not know the true values $\overline{x}$, $\overline{y}$, while we can compute the exact product $z \in \mathbb{R}$, there is a fundamental uncertainty that limits the useful precision of z.

While it is well-known how input rounding affects the exact product z, it will be useful to compare these bounds (which represent an "error floor" for other algorithms) against our later fixed- and floating-point error bounds.

Proposition 2.1. *The error due to input rounding is*

$$|\overline{x} \cdot \overline{y} - z| \leq 2\epsilon(1) \cdot \|w\|_1 + O(\epsilon(1)^2).$$

To quantify average-case error, replace $\overline{x}$, $\overline{y}$ by vector-valued random variables $X = (X_i)$, $Y = (Y_i)$ with entries drawn from independent uniform distributions:

$$X_i \sim \mathcal{U}(x_i - \epsilon(x_i), x_i + \epsilon(x_i)), \qquad Y_i \sim \mathcal{U}(y_i - \epsilon(y_i), y_i + \epsilon(y_i)).$$

We can then examine the standard deviation of $X \cdot Y$:

Proposition 2.2. *The standard deviation due to input rounding is on the order of $\epsilon(1) \cdot \|w\|_2$. More specifically,*

$$\frac{1}{\sqrt{6}} \cdot \epsilon(1) \cdot \|w\|_2 \leq \sigma(X \cdot Y - z) \leq \frac{2}{\sqrt{6}} \cdot \epsilon(1) \cdot \|w\|_2 + O(\epsilon(1)^2).$$

Straightforward proofs of both results may be found in Section 7.1.

2.3 Fixed-Point Error Bounds

The analysis for fixed-point error is similar to input rounding, except that we replace input rounding errors $\epsilon(x_i)$, $\epsilon(y_i)$ with discretization errors $\hat{\epsilon}(\|x\|_\infty)$, $\hat{\epsilon}(\|y\|_\infty)$, which depend only on the magnitude of the largest elements in x and y. In addition, when $\hat{M} \geq M$, the largest elements in x, y can be exactly represented in fixed point, and do not contribute any error to $\hat{z}$, allowing tighter bounds.

Proposition 2.3. *The error of the fixed-point dot product $\hat{z}$ is bounded by*

$$\begin{aligned} |\hat{z} - z| &\leq \hat{\epsilon}(1) \cdot \left[\|x\|_\infty \|y\|_1 + \|y\|_\infty \|x\|_1\right] + O(\hat{\epsilon}(1)^2) \\ &\leq 2\hat{\epsilon}(1) \cdot k\|x\|_\infty \|y\|_\infty + O(\hat{\epsilon}(1)^2). \end{aligned}$$

Furthermore, if $E = \hat{M} - M > 0$, the bound can be made twice as tight:

$$|\hat{z} - z| \leq \hat{\epsilon}(1) \cdot k\|x\|_\infty \|y\|_\infty + O(\hat{\epsilon}(1)^2).$$

For average-case error, we model the discretization errors as independent uniform random variables:

$$\hat{x}_i - x_i \sim \mathcal{U}(-\hat{\epsilon}(\|x\|_\infty), \hat{\epsilon}(\|x\|_\infty)), \qquad \hat{y}_i - y_i \sim \mathcal{U}(-\hat{\epsilon}(\|y\|_\infty), \hat{\epsilon}(\|y\|_\infty)). \tag{1}$$

Note that this probabilistic model, while a relatively mild assumption on the input vectors, may not be sufficiently representative of all applications.

Proposition 2.4. *Under the independent rounding model* (1)*, $\mathbb{E}[\hat{z}] = z$, and the standard deviation of $\hat{z}$ is on the order of $\hat{\epsilon}(1)\sqrt{\|x\|_2^2\|y\|_\infty^2 + \|y\|_2^2\|x\|_\infty^2}$. More precisely,*

$$\frac{\hat{\epsilon}(1)}{2\sqrt{3}}\sqrt{\|x\|_2^2\|y\|_\infty^2 + \|y\|_2^2\|x\|_\infty^2} \leq \sigma(\hat{z}) \leq \frac{\hat{\epsilon}(1)}{\sqrt{3}}\sqrt{\|x\|_2^2\|y\|_\infty^2 + \|y\|_2^2\|x\|_\infty^2} + O(\hat{\epsilon}(1)^2). \tag{2}$$

Furthermore,

$$\sigma(\hat{z}) \leq \sqrt{\frac{2}{3}}\hat{\epsilon}(1) \cdot \sqrt{k}\|x\|_\infty \|y\|_\infty + O(\hat{\epsilon}(1)^2). \tag{3}$$

Proofs may be found in Section 7.2.

2.4 Floating-Point Error Bounds

The study of floating-point dot product accuracy has a long history in the literature. A straightforward application of the IEEE-754 rounding rules leads to a $O(k/(1-k\epsilon(1)))$ worst-case bound [6], which was sharpened to $O(k)$ by Jeannerod and Rump [9]. However, it is well-recognized that the worst-case bounds significantly overestimate the average error. Higham and Mary [7] prove that average error, under a randomized rounding error model, grows as $O(\sqrt{k \log k})$, much slower than the worst-case $O(k)$ bound. In this section, we pursue a different approach to average error, calculating variances instead of using a concentration inequality. Furthermore, we will allow accumulation order to vary, rather than analyzing a specific order of floating-point operations.

We start with a basic sequential dot product implementation, assuming hardware with fused multiply-add (FMA):

$$z \approx d(x,y) = \mathrm{fma}(\cdots \mathrm{fma}(\mathrm{fma}(0, x_1, y_1), x_2, y_2) \cdots, x_k, y_k).$$

Here $\mathrm{fma}(a,b,c)$ computes $a+bc$ correctly rounded to an element of F. It is not hard to see that the rounding error experienced by the i^{th} FMA in this implementation is bounded (to first order) by $\epsilon(s_i)$, where $s_i = \sum_{j=1}^{i} x_j y_j = \sum_{j=1}^{i} w_j$ is the i^{th} partial sum. More precisely, if e_i is the rounding error incurred by the i^{th} FMA, then

$$d(x,y) = x_1y_1 + e_1 + x_2y_2 + e_2 + \ldots + x_ky_k + e_k = z + \sum_{i=1}^{k} e_i,$$

with errors bounded by

$$|e_i| \le \epsilon\left(\sum_{j=1}^{i} x_j y_j + \sum_{j=1}^{i-1} e_j\right) = \epsilon\left(s_i + \sum_{j=1}^{i-1} e_j\right).$$

In almost all cases, the accumulated error $\sum_{j=1}^{i-1} e_j$ does not push s_i into a higher binade, in which case we may simplify the right-hand side to $|e_i| \le \epsilon(s_i)$.

We first recall Jeannerod and Rump's worst-case error bound, which in our setting reads

$$|d(x,y) - z| \le \epsilon(1) \cdot k \|w\|_1. \tag{4}$$

Note that this bound does not assume the use of FMA.

Now consider the variance under the model that each rounding error is drawn independently: $e_i \sim \mathcal{U}(-\epsilon(s_i), \epsilon(s_i))$. As Higham and Mary [7] discuss at length, it is *not* necessarily safe to assume that rounding errors are independent, but nevertheless such a model is expected to provide a reasonably accurate proxy for the correct distribution.

Under this model, we have

$$\mathrm{Var}(d(x,y)) = \sum_{i=1}^{k} \mathrm{Var}(e_i) = \frac{1}{3}\sum_{i=1}^{k} \epsilon(s_i)^2 \approx \frac{\epsilon(1)^2}{3}\|s\|_2^2.$$

Using the fact that $\frac{1}{2} < \epsilon(s_i)/|s_i|\epsilon(1) \le 1$, we conclude that

$$\mathrm{Var}(d(x,y)) = \frac{c\epsilon(1)^2}{3}\|s\|_2^2, \qquad \text{for some } c \in \left(\tfrac{1}{4}, 1\right]. \tag{5}$$

Not surprisingly, the 2-norm of the partial sums s_i is the determining factor: the larger the accumulator values over the course of the computation, the more error can be expected.

So far, we have accumulated the terms x_iy_i in the order $1, \ldots, k$, but dot product implementations in practice do not always follows this ordering, and it seems reasonable to allow a dot product to accumulate in any order desired. Given a permutation $\pi \in S_k$, define the permuted-sequential FMA dot product

$$d_\pi(x,y) = \mathrm{fma}(\cdots \mathrm{fma}(\mathrm{fma}(0, x_{\pi(1)}, y_{\pi(1)}), x_{\pi(2)}, y_{\pi(2)}) \cdots, x_{\pi(k)}, y_{\pi(k)}).$$

Table 1: Error bounds, in terms of the norms of x, y, and $w = x \circ y$, to first order in $\epsilon(1)$ and $\hat{\epsilon}(1)$. For brevity, $C_1, C_2, \ldots$ represent numbers that may depend on x, y but lie within a range bounded independently of x, y. Variances assume the uniform independent rounding models of this section.

Error source	Worst case	Variance
Input rounding	$C_1\epsilon(1) \cdot \|w\|_1$	$C_2\epsilon(1)^2 \cdot \|w\|_2^2$
Fixed point	$C_3\hat{\epsilon}(1) \cdot k\|x\|_\infty\|y\|_\infty$	$C_4\hat{\epsilon}(1)^2 \cdot k\|x\|_\infty^2\|y\|_\infty^2$
Floating point, averaged over all orders	$\epsilon(1) \cdot k\|w\|_1$	$C_5\epsilon(1)^2 \cdot (k+1)(2z^2 + \|w\|_2^2)$
Floating point, worst order	$\epsilon(1) \cdot k\|w\|_1$	$C_6\epsilon(1)^2 \cdot k\|w\|_1^2$

	C_1	C_2	C_3	C_4	C_5	C_6
Minimum value	2	$\frac{1}{6}$	-	-	$\frac{1}{72}$	$\frac{1}{384}$
Maximum value	2	$\frac{2}{3}$	1 ($E > 0$), 2 ($E \leq 0$)	$\frac{2}{3}$	$\frac{1}{18}$	$\frac{1}{96}$

As the permutation varies, so does the partial-sum norm $\|s\|_2$: we can make $\|s\|_2$ larger by moving large entries of w having the same sign to the beginning of the dot product, or smaller, by moving large entries of w to the end and arranging for cancellation between neighboring terms when possible.

Using these ideas, we can derive a lower bound for the worst-case variance $\mathrm{Var}(d_\pi(x, y))$ over all possible permutations π, and also characterize the average variance over all permutations.

Proposition 2.5. *For any x, y, under the independent rounding error model,*

$$\max_{\pi \in S_k} \mathrm{Var}(d_\pi(x, y)) \geq \frac{c\epsilon(1)^2}{96} k\|w\|_1^2$$

for some $c \in (\frac{1}{4}, 1]$.

Proposition 2.6. *Under the independent rounding error model, the average variance over all permutations is given by*

$$\frac{1}{k!} \sum_\pi \mathrm{Var}(d_\pi(x, y)) = \frac{c\epsilon(1)^2}{18} \cdot (k+1)\left(2z^2 + \|w\|_2^2\right),$$

for some $c \in (\frac{1}{4}, 1]$.

Proofs may be found in Section 7.3.

2.5 Error Bound Summary

Table 1 collects the bounds of this section in their simplified forms. Effectively, the error bounds split into two families: the fixed-point bounds depending on $\|x\|_\infty\|y\|_\infty$, and the other bounds depending on $\|w\|_1$ and $\|w\|_2$. Later, we will define a family of *ESC* (exponent span capacity) functions that capture the gaps between these vector norms. Note that none of the bounds are affected by cancellation in the summation, with the exception of the z^2 term in the floating-point average order bound. In the absence of cancellation, $|z| = \|w\|_1$.

The other major difference between bounds are the factors of k found in the fixed- and floating-point bounds, but absent from the input rounding bounds, which represent an effective accuracy limit for any dot product. This will imply that fixed-point methods allow us to achieve the best possible error bounds, at a cost of $\Theta(\log_2 k)$ extra mantissa bits.

3 Achieving Floating-Point-Like Accuracy

Equipped with the dot product accuracy bounds of Section 2, our goal is to choose a fixed-point mantissa count $\hat{M}$ given M, x, and y so that fixed-point methods provide comparable accuracy to a standard floating-point implementation. Fasi, Mary, and Mikaitis [4] recently performed a similar analysis, using worst-case bounds. We will extend their analysis to average-case bounds and sharpen their results for worst-case bounds. In the process, it will be natural to introduce a new family of p-exponent span capacity (esc_p) functions, which given x, y will determine the necessary number of extra mantissa bits $E = \hat{M} - M$. In particular, esc_∞ is identical to the original ESC function, as defined in [18], up to a small constant number of bits. By taking a maximum over the dot products within a GEMM, these results immediately generalize to full matrix multiplications.

3.1 Matching Fixed- and Floating-Point Bounds

Matching Worst-Case Error Bounds Referring to Table 1 and recalling that $\epsilon(1) = 2^{-M}$, $\hat{\epsilon}(1) = 2^{-\hat{M}}$, the fixed-point worst-case error is no larger than the floating-point worst-case error when

$$C_3 2^{-\hat{M}} \|x\|_\infty \|y\|_\infty \leq 2^{-M} \|w\|_1,$$

or equivalently,

$$E \geq \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_1} + \log_2(C_3).$$

Recalling that we may take $C_3 = 1$ if $E > 0$ and $C_3 = 2$ otherwise, we may weaken this to

$$E \geq E_{\text{worst}} = \max\left(1, \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_1}\right). \tag{6}$$

Equation (6) can be thought of as a sharpened version of Fasi et al.'s ESC bound [4].

We may also wish to pick E large enough to achieve a stronger bound, where the fixed-point error is no worse than the error due to input rounding, in the worst case. This scenario requires

$$C_3 2^{-\hat{M}} k \|x\|_\infty \|y\|_\infty \leq C_1 2^{-M} \|w\|_1.$$

By similar reasoning as above, and recalling $C_1 = 2$, it suffices to guarantee

$$E \geq \overline{E}_{\text{worst}} = \max\left(1, \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_1} + \log_2 k - 1\right). \tag{7}$$

In other words, we may pay $\log_2 k$ extra mantissa bits in order to achieve optimal fixed-point accuracy (*optimal* in the sense of matching the inevitable worst-case error due to input rounding, which captures the ill-conditioning of the dot product).

Matching Average-Case Error Bounds If we average over all floating-point accumulation orders, fixed-point variance will be no larger than floating-point variance when

$$C_4 2^{-2\hat{M}} \|x\|_\infty^2 \|y\|_\infty^2 \leq C_5 \left(1 + \frac{1}{k}\right) 2^{-2M} (2z^2 + \|w\|_2^2).$$

In the presence of cancellation, the term $2z^2$ could be, in the worst case, zero, so we may focus on the 2-norm of w. Dropping the z term, we are assured fixed-point variance is no greater than floating-point variance if

$$E \geq \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_2} + \frac{1}{2} \log_2 \frac{C_4 k}{C_5 (k+1)}.$$

Weakening this slightly to account for the bounds on C_4, C_5, it suffices to ensure

$$E \geq E_{\text{avg}} = \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_2} + 2 + \log_2(\sqrt{3}). \tag{8}$$

On the other hand, if there is no cancellation, $|z| = \|w\|_1$, and it suffices to ensure the weaker inequality

$$E \geq E'_{\text{avg}} = \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_1} + 2 + \log_2(\sqrt{3/2}). \tag{9}$$

Finally, if we wish to match the noise floor given by input rounding, instead of matching floating-point accuracy, we have

$$C_4 2^{-2\hat{M}} k \|x\|_\infty^2 \|y\|_\infty^2 \leq C_2 2^{-2M} \|w\|_2^2$$

and given $C_4 \leq \frac{2}{3}$, $C_2 \geq \frac{1}{6}$, it suffices to ensure

$$E \geq \overline{E}_{\text{avg}} = \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_2} + \frac{1}{2} \log_2 k + 1. \tag{10}$$

3.2 The p-Exponent Span Capacity

Examining the five options (6–10) for choosing the fixed-point mantissa width, it is natural to introduce the *p-exponent span capacity*

$$\text{esc}_p(x, y) = \log_2 \frac{\|x\|_\infty \|y\|_\infty}{\|w\|_p}. \qquad (1 \leq p \leq \infty)$$

Evidently $E_{\text{worst}} = \max(1, \text{esc}_1(x, y))$ and $E'_{\text{avg}} = \text{esc}_1(x, y) + O(1)$, while $E_{\text{avg}} = \text{esc}_2(x, y) + O(1)$. The ideal bounds $\overline{E}_{\text{worst}}$, $\overline{E}_{\text{avg}}$ similarly can be represented in terms of esc_1, esc_2, plus an additional $\Theta(\log_2 k)$ overhead. Note that the original ESC function of Schwarz et al. [18] is given by

$$\begin{aligned} \text{esc}(x, y) &= \lfloor \log_2 \|x\|_\infty \rfloor + \lfloor \log_2 \|y\|_\infty \rfloor - \lfloor \log_2 \|w\|_\infty \rfloor \\ &\in [\text{esc}_\infty(x, y) - 2, \text{esc}_\infty(x, y) + 1]. \end{aligned}$$

Accordingly, we can often use the original esc and the new esc_∞ interchangeably, with the proviso that they may differ by up to 2 bits.

Since $\|w\|_p \geq \|w\|_{p'}$ whenever $p \leq p'$, we immediately have

$$\text{esc}_1(x, y) \leq \text{esc}_2(x, y) \leq \text{esc}_\infty(x, y),$$

and we conclude that the original ESC is overly conservative for matching either worst- or average-case error, where esc_1 and esc_2 respectively suffice. This has also been observed experimentally, cf. [2].

The p-ESC extends readily to matrix multiplications. Given A, B, let a_i denote the i^{th} row of A and b_j the j^{th} column of B. Since all fixed-point accuracy bounds are monotonic with respect to $\hat{M}$, it suffices to ensure that the worst dot product $a_i \cdot b_j$ is calculated in sufficient accuracy. Accordingly, define

$$\text{esc}_p(A, B) = \max_{i,j} \text{esc}_p(a_i, b_j).$$

4 Computing p-Exponent Span Capacity

Having defined p-exponent span capacity and shown how it can be used to derive fixed point mantissa bit widths, we now present two new algorithms for efficiently approximating esc_p. Both algorithms are built on low-precision (bfloat16) matrix multiplication, which is very fast on modern hardware.

4.1 Computing ∞-Exponent Span Capacity

We begin by recalling Schwarz et al.'s *coarsened ESC* algorithm [18], which provides an upper bound for $\mathrm{esc}_\infty(x, y)$, and discussing its weaknesses. We then present a new *binned ESC* algorithm that avoids these weaknesses while remaining efficient to compute on modern hardware.

The algorithms we discuss here will in fact calculate upper bounds for Schwarz et al's original ESC function, $\mathrm{esc}(x, y)$. As mentioned in Section 3.2, $\mathrm{esc}(x, y)$ and $\mathrm{esc}_\infty(x, y)$ differ by at most 2 bits. We will let $e(a) = \lfloor \log_2 |a| \rfloor$ denote the exponent of a nonzero floating-point number $a \in F$, and write $e_x = \max_i e(x_i)$ and $e_y = \max_i e(y_i)$.

4.1.1 Coarsened ESC

In [18], the authors observe that it is computationally expensive to calculate $\mathrm{esc}(x, y)$ directly. To reduce runtime, they *coarsen* the data in the k dimension. Specifically, for a coarsening factor c and $x, y \in F^k$, partition the indices into blocks $B_j = \{(j-1)c + 1, \ldots, \min(jc, k)\}$ $(j = 1, \ldots, \lceil k/c \rceil)$, and keep only the extreme exponents of each block,

$$e_j^{(\max)}(x) = \max_{i \in B_j} e(x_i), \qquad e_j^{(\min)}(x) = \min_{i \in B_j} e(x_i),$$

and likewise for y. Now define a coarsened ESC

$$E_\infty^{(c)}(x, y) = e_x + e_y - \max_j \max\left(e_j^{(\max)}(x) + e_j^{(\min)}(y),\ e_j^{(\min)}(x) + e_j^{(\max)}(y)\right) \tag{11}$$

Within a block, the entry attaining $e_j^{(\max)}(x)$ is paired with an entry of y of exponent at least $e_j^{(\min)}(y)$, so $E_\infty^{(c)} \geq E_\infty^{(1)} \geq \mathrm{esc}$, and since $\|x\|_\infty \|y\|_\infty < 2^{e_x + e_y + 2}$,

$$\mathrm{esc}_\infty < E_\infty^{(c)} + 2.$$

Zero inputs represent a special case in the coarsened ESC algorithm, because the exponent is effectively $-\infty$. The challenge is that once we compress the inputs to a minimum and maximum exponent per block, we cannot know how the zero values of this block would interact with other blocks it multiplies against: the dot product of two sparse blocks may be zero (no contribution to ESC) or nonzero (in which case coarsened ESC provides no information).

Accordingly, Schwarz et al. [18] propose representing zeros with two sentinels: $e(0) =$ `ZERO_SENTINEL` and e_x, $e_y =$ `ZERO_MAX_SENTINEL` if x, y are all zero, respectively. The sentinel values are both far below the exponent range of F and are chosen such that (11) produces a zero value for $E_\infty^{(c)}$ when either x or y contain all zeros, and otherwise a very large sentinel value when every block contains zeros, indicating a failure to bound E_∞. Their ADP framework flags these large sentinel ESC values and dispatches to native floating-point computation.

4.1.2 Binned ESC

While coarsened ESC provides a safe bound for esc_∞, scientific applications often multiply sufficiently sparse matrices to trigger this ambiguous path, as we will show later. We now propose an alternate computational method for bounding esc_∞ using a floating-point binning algorithm. Our new algorithm is designed to be efficiently computed using fast, low-precision bfloat16 tensor cores.

The core idea is to let a lower precision floating-point dot product handle the exponent bookkeeping. Let $F_{\mathtt{bf16}}$ be the set of bfloat16 values. We quantize elements $x \in F$ into nonnegative bin representatives $b(x) \in F_{\mathtt{bf16}}$ based on their exponent distance from the maximum exponent in their vector. Each bin covers a fixed range of exponent space; the first holds elements within a few binades of the maximum, and each subsequent bin holds the next range below it. Unlike the coarsened ESC algorithm, we are no longer able to calculate ESC values across the full double precision exponent range, due to the much smaller exponent range of bfloat16. As a result, we use the final bin for non-zero elements that are too small to fit into

Algorithm 1 `coarsened_quantize`(x)

Require: $x \in F^k$ finite; coarsening factor c; `ZERO_SENTINEL` $= -16384$, `ZERO_MAX_SENTINEL` $= -32768$
Ensure: $e^{(\max)}(x), e^{(\min)}(x) \in \mathbb{Z}^{\lceil k/c \rceil}$, the largest and smallest exponent of each block; e_x, the largest exponent of x

$e(x_i) \leftarrow \lfloor \log_2 |x_i| \rfloor$ if $x_i \neq 0$, else `ZERO_SENTINEL`
for $j = 1, \ldots, \lceil k/c \rceil$ **do**
  $B_j \leftarrow \{(j-1)c + 1, \ldots, \min(jc,\ k)\}$
  $e_j^{(\max)}(x) \leftarrow \max_{i \in B_j} e(x_i)$
  $e_j^{(\min)}(x) \leftarrow \min_{i \in B_j} e(x_i)$
end for
$e_x \leftarrow \max_j e_j^{(\max)}(x)$
if $e_x =$ `ZERO_SENTINEL` **then**
  $e_x \leftarrow$ `ZERO_MAX_SENTINEL`
end if
return $e^{(\max)}(x), e^{(\min)}(x), e_x$

Algorithm 2 Coarsened esc_∞ calculation for $x, y \in F^k$ with coarsening factor c

Require: $x, y \in F^k$ finite; coarsen factor c; `ZERO_SENTINEL` $= -16384$, `ZERO_MAX_SENTINEL` $= -32768$
Ensure: the count $\hat{M} = 53 + E_\infty^{(c)} + 1$ with $E_\infty^{(c)} \geq E_\infty(x, y)$, or the fallback count

$e^{(\max)}(x), e^{(\min)}(x), e_x \leftarrow$ `coarsened_quantize`(x)
$e^{(\max)}(y), e^{(\min)}(y), e_y \leftarrow$ `coarsened_quantize`(y)
`prod` $\leftarrow \max_j \max\left(e_j^{(\max)}(x) + e_j^{(\min)}(y),\ e_j^{(\min)}(x) + e_j^{(\max)}(y)\right)$
$E_\infty^{(c)} \leftarrow \max(e_x + e_y - \texttt{prod},\ 0)$ // *0 if x or y is all zeros*
if $53 + E_\infty^{(c)} + 1 > 4000$ **then**
  return fallback
end if
return $53 + E_\infty^{(c)} + 1$

the compressed exponent space. During quantization, each non-zero element is replaced with a canonical representative for the bin, typically the smallest value in the range covered by the bin. Zero inputs are mapped to zero bfloat16 representatives. Algorithm 3 describes this mapping in detail.

The binned ESC algorithm is parameterized by three meta-parameters `in_width`, `in_top`, and `out_width`:

- **`in_width`** is the number of binades in F the are represented within a bin. Larger values increase the maximum ESC value that can be successfully bounded but also increase the overprediction over the exact value of $\text{esc}(x, y)$ (see equation (12)).
- **`in_top`** is a special value for the number of binades in F that are represented within the most significant bin. A larger `in_top` also influences overprediction and the maximum ESC value that can be calculated.
- **`out_width`** is the number of binades separating bins in $F_{\texttt{bf16}}$. A smaller value increases the number of bins but can diminish the final algorithm's performance due to more frequent exponent update steps (smaller `k_max` in Algorithm 4).

For Ozaki-II with double-precision inputs, the configuration `in_width` $= 4$, `in_top` $= 2$, `out_width` $= 8$ maximizes ESC resolution while supporting a maximum ESC of 100 or more. To achieve good performance with `out_width` to 8, we implemented a specialized binned ESC kernel which uses bfloat16 tensor cores and periodically updates maximum exponents and resets the accumulator within the kernel.

Unlike the coarsened ESC algorithm, the overprediction in the binned ESC algorithm can be bounded by a constant. The binning provides an exact exponent span capacity after rounding into a bin, so the only error comes from combining multiple exponents into a single bin. The maximum exponent adjustment is

Algorithm 3 `binned_quantize`(x)

Require: $x \in F^k$; `in_width`, `in_top`, `out_width`, `top`, `nbins`
Ensure: `q` $\in F^k$, powers of two (or zero)
1: $\texttt{exp}_i \leftarrow \lfloor \log_2 |x_i| \rfloor$
2: $\texttt{loss}_i \leftarrow \max_j \texttt{exp}_j - \texttt{exp}_i$
3: $\texttt{bin}_i \leftarrow \min\big(\lceil \max(\texttt{loss}_i - \texttt{in_top}, 0)/\texttt{in_width} \rceil, \texttt{nbins}\big)$
4: $\texttt{q}_i \leftarrow 2^{\texttt{top}-\texttt{out_width}\cdot\texttt{bin}_i}$ if $x_i \neq 0$, else 0
5: **return** `q`

Algorithm 4 Binned ESC estimate for $x, y \in F^k$

Require: $x, y \in F^k$; `in_width`, `in_top`, `out_width`
Ensure: $E \geq \text{esc}_{\text{inf}}(x, y)$
1: $\texttt{top} \leftarrow \lfloor (127 - \texttt{out_width})/2 \rfloor$
2: $\texttt{nbins} \leftarrow \lfloor (\texttt{top} + 75)/\texttt{out_width} \rfloor$
3: $\texttt{k_max} \leftarrow 2^{\texttt{out_width}} - 1$
4: $\texttt{q_x} \leftarrow \texttt{binned_quantize}(x)$
5: $\texttt{q_y} \leftarrow \texttt{binned_quantize}(y)$
6: $\texttt{dot} \leftarrow 0$
7: **for** $\texttt{start} = 1, 1 + \texttt{k_max}, 1 + 2\,\texttt{k_max}, \ldots, k$ **do**
8: $\quad \texttt{chunk} \leftarrow \texttt{start}, \ldots, \min(\texttt{start} + \texttt{k_max} - 1,\ k)$
9: $\quad \texttt{dot} \leftarrow \max\big(\texttt{dot},\ \sum_{i \in \texttt{chunk}} \texttt{q_x}_i\, \texttt{q_y}_i\big)$
10: **end for**
11: **if** $\texttt{dot} = 0$ **then**
12: $\quad$ **return** 0
13: **end if**
14: $\texttt{b} \leftarrow \max\big(0,\ \lceil (2\,\texttt{top} - \lfloor \log_2 \texttt{dot} \rfloor)/\texttt{out_width} \rceil\big)$
15: **if** $\texttt{b} \geq \texttt{nbins} - 1$ **then**
16: $\quad$ **return** fallback
17: **end if**
18: **return** $2\,\texttt{in_top} + \texttt{in_width} \cdot \texttt{b}$

`in_width` $- 1$ per element or rounding up a trivial ESC value, leading to a worst-case overprediction

$$\text{esc}_{\text{binned}}(x, y) - \text{esc}(x, y) \leq 2 \max(\texttt{in_top}, \texttt{in_width} - 1) + 1. \tag{12}$$

4.2 Computing p-Exponent Span Capacity

To compute a p-norm based exponent span capacity for finite $p \geq 1$, we notice that $|x^p| \cdot |y^p| = \|w\|_p^p$, where $|x^p|$ is shorthand for the vector $(|x_1|^p, \ldots, |x_k|^p)$. Therefore, as long we can get an approximation of $\tilde{z}^p \in F_{\texttt{bf16}}$ such that $\tilde{z} \leq |x^p| \cdot |y^p|$, then we know our calculated esc_p is a safe over-estimation. Similar to the binned infinity-norm exponent span capacity, we need to carefully handle the reduced range. Both of these considerations can be handled through quantization. In practice, $p = 1$ or $p = 2$ are the most relevant.

We reuse binned ESC's `exponent_top` and `exponent_bottom` constants, which are derived from the underlying datatype, and we align our elements to the `element_top` and the maximum exponent for each vector. Any non-zero element that is too small for this compressed range is stored with the minimum exponent `exponent_bottom` and requires an extra step in our algorithm to ensure that the output is not influenced by these small elements. If they are, we dispatch to the same sentinel as in the binned ESC algorithm. The detailed steps can be found in Algorithm 6.

There are three sources of error in Algorithm 6:

(a) rounding error in the quantization step $\tilde{x}_i \leftarrow \text{rd}_{\texttt{bf16}}(|x_i|^p)$;

Algorithm 5 `esc_p_quantize`(x)

Require: $x \in F^k$; e_x, `exponent_top`, `exponent_bottom`, `rounding_function`

if $x = 0$ **then**
  return 0
end if
$\tilde{x} \leftarrow$ `rounding_function`$\left(|x|^p \, 2^{\texttt{exponent_top} - p e_x}\right)$
return $\max(\tilde{x}, 2^{\texttt{exponent_bottom}})$

Algorithm 6 esc_p calculation for $x, y \in F^k$, $1 \le p < \infty$

Require: $x, y \in F^k$ finite; `exponent_top`, `exponent_bottom`
Ensure: $\hat{M}$ with $E_p \ge \mathrm{esc}_p(x, y)$, or fall back

$e_x \leftarrow \max_i \lceil \log_2 |x_i| \rceil$
$e_y \leftarrow \max_i \lceil \log_2 |y_i| \rceil$
  // special care is needed to avoid underflow and overflow
$\tilde{x}_i \leftarrow$ `esc_p_quantize`$(x_i, e_x, \mathrm{rd}_{\mathtt{bf16}})$
$\tilde{y}_i \leftarrow$ `esc_p_quantize`$(y_i, e_y, \mathrm{rd}_{\mathtt{bf16}})$
$\tilde{z} \leftarrow \sum_i \tilde{x}_i \, \tilde{y}_i$ *// calculated in single precision*
if $\tilde{z} = 0$ **then**
  return 53
end if
$\ell \leftarrow \lfloor \log_2 \tilde{z} \rfloor$
  // entries clamped at $2^{\texttt{exponent_bottom}}$ *may dominate* $\tilde{z}$
if $\ell \le$ `exponent_top` $+$ `exponent_bottom` $+ \lceil \log_2 k \rceil$ **then**
  return fallback
end if
$E_p \leftarrow \max\left(\lceil (2 \, \texttt{exponent_top} - \ell + 1)/p \rceil, \, 0\right)$ *// +1: ru*
return $53 + E_p + 1$

(b) rounding error from finite precision accumulation in the bfloat16 multiplication $\tilde{z} = \tilde{x} \cdot \tilde{y}$.

(c) limited exponent range, causing entries of x, y that are very small relative to the maximum to become inflated.

For (a), the use of rounded-down arithmetic ensures that $\tilde{x}_i$, $\tilde{y}_i$ are underestimates of their exact values, and hence E_p is an overestimate. In theory (b) can be handled similarly, but hardware low-precision matrix multiplication accelerators typically do not allow for configurable rounding modes. However, bfloat16 matrix multiplication units typically accumulate in IEEE-754 single precision, so that we can bound the accumulator relative error by $2^{-24}k$, and multiply $\tilde{z}$ by $1 - 2^{-24}k$ to ensure that $\tilde{z}$ is an underestimate of its exact value. To handle (c), the final check on ℓ ensures that there are enough contributions from non-underflowed values to deduce a valid bound on esc_p.

In the special case $p = 1$, we can also calculate an upper bound on the number of integer bits (`c_bits`) required to represent the product z exactly (Algorithm 7). This is useful for Ozaki-II, where this bound can help us choose an appropriate number of moduli— see Section 5 for more details. Note that to avoid underestimating `c_bits`, which could lead to catastrophic accuracy loss, Algorithm 7 rounds upwards, which can result in slightly underestimating esc_1. It is possible to avoid underestimation by carefully evaluating the effects of the rounding error and correcting for them after forming the dot product $\tilde{z}$.

When calculating `c_bits` for a full matrix multiplication, care must be taken when reducing results across dot products. Rather than element-wise calculating $\hat{M}$ and `c_bits`, then maximizing across the dot products, it is necessary to first find the maximum $\hat{M}$ and `extra_c_bits` values from Algorithm 7 for each dot product, then calculate `c_bits` $= 2\hat{M} +$ `extra_c_bits`.

Algorithm 7 esc_1 and accumulator bit count for $x, y \in F^k$

Require: $x, y \in F^k$ finite; `exponent_top`, `exponent_bottom`
Ensure: $\hat{M}$ and `c_bits`, or fallback
$e_x \leftarrow \max_i \lceil \log_2 |x_i| \rceil$
$e_y \leftarrow \max_i \lceil \log_2 |y_i| \rceil$
$\tilde{x}_i \leftarrow \texttt{esc_p_quantize}(x_i, e_x, \text{ru}_{\texttt{bf16}})$
$\tilde{y}_i \leftarrow \texttt{esc_p_quantize}(y_i, e_y, \text{ru}_{\texttt{bf16}})$
$\tilde{z} \leftarrow \sum_i \tilde{x}_i \tilde{y}_i$
if $\tilde{z} = 0$ **then**
  return $\hat{M} = 53$, `c_bits` $= 2\hat{M}$
end if
$\ell \leftarrow \lfloor \log_2 \tilde{z} \rfloor$
if $\ell \leq$ `exponent_top` + `exponent_bottom` $+ \lceil \log_2 k \rceil$ **then**
  return fallback
end if
$E_1 \leftarrow \max(2\,\texttt{exponent_top} - \ell,\ 0)$ *// may add 1 for safety to compensate for rounding inputs up*
$\hat{M} \leftarrow 53 + E_1 + 1$
`extra_c_bits` $\leftarrow \max\big(\lceil \log_2 \tilde{z} \rceil - 2\,\texttt{top} + 1,\ 0\big)$ *// +1: sum rounding*
return $\hat{M}$, `c_bits` $= 2\hat{M}$ + `extra_c_bits`

5 ESC in Ozaki-II

In this section, we briefly note how ESC-derived mantissa bit counts can be integrated into the Ozaki-II algorithm. For more detailed background on the algorithm, see [16, 2]. In Ozaki et al's original description [16], accuracy is selected via scaling factors, and the authors present several methods ("fast" and "accurate" scaling) that aim to maximize input integer precision subject to a fixed accumulator precision, parameterized by modulus count.

Our viewpoint is the opposite: instead of maximizing accuracy subject to a given performance target (modulus count), we wish to maximize performance subject to the accuracy requirement embodied in the mantissa bit count $\hat{M}$. An additional benefit of viewing scaling this way is that as the underlying datatypes change [21, 5] we can maintain comparable, even bitwise equivalent, accuracy.

To achieve this, we first determine Ozaki-II scaling factors with a "fixed scaling" method. Letting a_i denote the i^{th} row of A, the scaling factor for that row is given by $\sigma_i = 2^{\hat{e}(\|a_i\|_\infty)}$. Similarly the j^{th} column b_j of B will be scaled by $\tau_j = 2^{\hat{e}(\|b_j\|_\infty)}$. Note that $\hat{e}$ captures the mantissa bit requirement $\hat{M}$. This matches the fixed-point discretization scheme of Section 2.1.

After scaling and discretization, Ozaki-II multiplies its integer inputs in modular arithmetic, modulo a set of small coprime integers $m_1, \ldots, m_N$, then applies the Chinese Remainder Theorem and lifts the result from the modular domain to an integer in the range $(-M_N/2, M_N/2)$, where $M_N = \prod_{i=1}^{N} m_i$. To avoid catastrophic wraparound when lifting, it is therefore necessary to choose N (and therefore M_N) large enough that the integer product's elements lie within $(-M_N/2, M_N/2)$. That is, we require an upper bound on the elements of the product matrix $C = AB = (c_{ij})$; roughly speaking[1], we need $|c_{ij}| \lesssim \sigma_i \tau_j M_N/2$. Subject to this requirement, we wish to minimize N to minimize computation time, which means bounding C from above as tightly as possible.

In the worst case, we have the simple upper bound $|c_{ij}| \leq \sigma_i \tau_j 2^{2\hat{M}} k$. A tighter bound on C can be found by precalculating the 2-norms of each row of A and column of B and applying the Cauchy-Schwarz inequality, similar to the fast scaling method. Alternatively, if we have already calculated $\text{esc}_1(A, B)$, the algorithm in Section 4.2 provides this upper bound directly via its `c_bits` output.

[1] This inequality is not exact due to the fixed-point discretization.

6 Accuracy Studies

Let us begin by providing a name to an accuracy definition that was first used in (4) and described as Jeannerod and Rump's "worst-case error bound" (the entry $\epsilon(1) \cdot k \|w\|_1$ in the row "Floating point, worst order" of Table 1, and the bound that Section 3.1 asks the fixed-point worst-case error not to exceed). **Criterion 1** (the classical bound). $|\hat{z} - z| \le k\,\epsilon(1)\|w\|_1$, where $\|w\|_1 = \sum_l |x_l y_l|$. We use it for two results: the fixed-point dot product $\hat{z}$ and, with $\mathrm{fl}(\hat{z})$ in place of $\hat{z}$, the FP64 number that is returned ("after fl" in Table 2). This is the classical worst-case bound for a floating-point dot product in any summation order, Higham's inner-product bound [6], $\gamma_k \|w\|_1$ with $\gamma_k = ku/(1-ku)$ and unit roundoff $u = \epsilon(1)$, with $k\epsilon(1)$ in place of γ_k. The two agree to first order in $\epsilon(1)$. It is the requirement of Fasi, Mary, and Mikaitis [4] that the emulated product satisfy this same bound (their Eq. (3.3) is a mantissa bit count that guarantees it). With $k = n$, the second form is also the threshold of the published grade-A tests of Demmel et al. [3] ($f_A(n) < n$ with $\epsilon = 2^{-53} = \epsilon(1)$, where $f_A(n)$ denotes the largest ratio $|\hat{C}_{ij} - C_{ij}|/(\epsilon\,(|A|\,|B|)_{ij})$ over the entries of the computed $\hat{C}$). Grade A as defined admits any polynomial $f_A(n)$. Criterion 1 measures the fixed-point error against the worst case of floating point. It does not cover the average-case comparison of Section 3.1, in which the fixed-point variance is to be no larger than the floating-point variance.

A *rule* returns a mantissa bit count, which we also call its *width*. The rules compared below are those of Section 4; except for binned, a rule's width is 53 plus its exponent term plus a constant C. The Criterion 1 ratios of Table 2 come from 463 constructed and random instances, k from 2 to 2049, every rule at its prescribed width under round to nearest, exact arithmetic. The two rows of Table 2 for the rules built on the 1-norm and on the 2-norm of w (Section 4.2) are from a second run, of 12 November, which adds three constructed instances built against those rules' base width of 56 bits and draws the random families anew. All measurements use a developer build of cuBLAS, planned for an upcoming release, with test code for the Grading the BLAS from Demmel et al. [3] (test code from [17]). Table 2, the summary of Criterion 1 over these instances, is placed in Section 6.3, beside the inputs that attain its worst ratios.

6.1 "Grading the BLAS"

The Grading the BLAS tests grade BLAS implementations through tests designed to be "progressively less gameable." Where the goal is to achieve an "A" grade, where the published grade-A threshold is Criterion 1 with $k = n$ (the start of Section 6), so every rule that meets Criterion 1 after the final rounding to FP64 meets this bar. Most relevant to the study of exponent span capacity is Test 2, which iteratively increases the dynamic range under permutation and sign changes and was designed specifically to be adversarial for fixed-point methods. Test 2 consists of three increasingly more challenging subtests 2a, 2b, and 2c of which our measurements in Figures 1, 2, 3, and 4 use test 2c.

Figures 1 and 2 demonstrate that a fixed 53 mantissa bit Ozaki-II implementation is not sufficient to achieve a grade A. The high dynamic range of the inputs along with the narrow dynamic range of the fixed-point representation leads to many elements rounding away many or even all of their significant bits. This leads to much higher max componentwise error than native FP64. All of the exponent span capacity methods dynamically adapt the number of mantissa bits and consistently handle the increased dynamic range from the tests. The plots also show that the automatic dynamic precision framework dispatches to native FP64 once the dynamic range is sufficiently large (in this case roughly 100 mantissa bits, indicated by a hollow symbol in the plots) and obtain a grade A. When the Ozaki-II scheme is used, the max componentwise error is often more than an order of magnitude smaller than the native FP64.

One may wonder if we can get away with just increasing the number of fixed mantissa bits past 53 and start obtaining a grade A. While possible, to do this robustly you would need to maintain a potentially impractical number (in excess of 2000) mantissa bits. Finally, Figure 3 shows several fixed mantissa bit counts running on test 2c and that the dynamic range of test 2c eventually becomes too large for them to be graded an A.

CEST denotes the rule built on the 1-norm of w (Section 4.2), with exponent term $E_1 = e_x + e_y - \lfloor \log_2 \|w\|_1 \rfloor$ (e_x and e_y as in Section 4.1), the integer form of $\mathrm{esc}_1(x, y)$ (the exponent of each norm in place of its logarithm). Its width is $53 + E_1 + 3$ with a floor of 54 bits, the form of the widths the library logged. Algorithm 7 takes e_x and e_y rounded up, as $\lceil \log_2 \|x\|_\infty \rceil$ and $\lceil \log_2 \|y\|_\infty \rceil$: where no entry is clamped at $2^{\texttt{exponent_bottom}}$, its E_1 is never more than one less than this E_1, and it is this E_1 plus 1 or 2 when neither maximum is a power of two

and the $\max(\cdot, 0)$ of its line 13 does not bind, so that its width, 53 plus its own E_1 plus 1, is then the width above or one bit less.

The runs. Nine runs of Test 2c of the "Grading the BLAS" harness [3] (BLAS, the Basic Linear Algebra Subprograms) against cuBLAS on an NVIDIA B200 are behind Figures 1 to 3: native FP64 without emulation, the three width rules the library implements (binned, CEST, and the coarsened ESC), and fixed widths of 53, 55, 69, 85, and 101 bits (the figures draw all but the 55-bit run), each at the six sizes $m = n = k = 32, 64, \ldots, 1024$ and at every exponent range from ±0 to ±40, the entries of A and B spanning 2^{-s} to 2^{s} at range $\pm s$. These runs logged no width. Under a rule the library either emulates the GEMM (the fixed-point kernels run) or takes the FP64 path (its fallback to FP64 arithmetic). Which of the two happened is read from the result: for $n \leq 128$ the FP64 path returns the native run's product bit for bit, and for $n \geq 256$, where its product differs from the native run's, its error $f_A(n)$ (the start of Section 6) is above 7 and the emulated error is below 3. The widths shown with the runs (Figure 1) are those of the three rules, in the forms given in the caption of Figure 5, evaluated on matrices of the test's construction; in separate runs with the library's log on, that evaluation agreed with the log in all 306 cases (283 widths, and 23 cases in which binned logged none).

The width rules. The three rules earn grade A at every size and range. While they emulate, $f_A(n)$ stays below 3.1 (the largest value, 3.01, is CEST at $n = 64$, ±16; at $n = 1024$ it is at most 2.7). Native FP64 earns grade A throughout as well, with f_A from 3.6 to 53 across the sizes, growing with n as a sum of n roundings should (its largest value is 8.2 at $n = 32$ and 53 at $n = 1024$). Fixed 53 bits does not: it is better than native at no range beyond ±5 to ±9, depending on the size, first loses grade A between ±6 and ±10, and from ±31 its error is of the order of $|A|\,|B|$ itself, with f_A between 1.9×10^{15} and 5.7×10^{15} (Figures 1 and 2).

The hand-off to FP64. Each rule emulates up to a range and runs every wider range on the FP64 path. The binned rule emulates through ±24 (±25 at $n = 32$), the coarsened ESC through ±25 or ±26, and CEST through ±27 to ±30, depending on the size. The library's nominal cap on the emulated width is 100 bits (the 100-bit limit of Figure 1) and can be extended upward; in these runs, by the evaluated widths, the hand-off came a few bits past it. On the FP64 path the three rules return the same product at every size.

Fixed mantissa widths. The runs at 53, 69, 85, and 101 bits, with a run at 55 bits that the figures do not draw, measure what a fixed request delivers as the range grows, the inverse of the question the rules answer. Each fixed width keeps the emulated error level, f_A below 3.1, up to a range that grows with the width, and beyond it the error climbs, though not monotonically from range to range. The error first passes 3.1 at ±4 to ±6 for 53 bits, depending on the size, at ±6 or ±7 for 55, at ±12 or ±14 for 69, at ±20 to ±22 for 85, and at ±29 or ±30 for 101. Grade A, $f_A < n$, is first lost one to six ranges later, the three largest sizes holding it longer than the three smallest: at ±6 to ±10 for 53 bits, ±7 to ±12 for 55, ±15 to ±19 for 69, ±24 to ±27 for 85, and ±32 to ±35 for 101, each further 16 bits of width moving the loss out by about eight ranges (Figure 3 shows $n = 1024$). At 101 bits the errors stay at the emulated level through ±28 at every size, where native FP64 has f_A from 39 to 53 at $n = 1024$, so the fixed-point kernels do run at a 101-bit request.

Fixed widths and grade A. Demmel et al.'s [3] remark that with enough slices or moduli nothing prevents an A is borne out and bounded. A fixed width earns grade A on Test 2c only as far as the exponent range lets it (the thresholds above and Figure 3), while the width rules earn it at every size and range by choosing the width per call and taking the FP64 path past the cap.

GEMMul8 on the same matrices. Figure 4 shows GEMMul8 with the binned rule on the same Test 2c matrices at $n = 1024$, run on an NVIDIA B200, beside two of the runs of Figure 1, cuBLAS with the same rule and the native FP64 GEMM of cuBLAS. GEMMul8 emulates through ±10, with f_A at most 1.91, and runs in FP64 from ±11, where binned requests 78 bits (74 at ±10); cuBLAS with the same rule emulates through ±24, with f_A at most 2.39, and runs in FP64 from ±25, where binned requests 106 bits (102 at ±24).

6.2 Random data

The *exponent rules* are the exponent-sum ESC (uncoarsened, $c = 1$ in (11)), with exponent term $E_\infty = E_\infty^{(1)}$, and the coarsened ESC; the *norm forms* are CEST and CEST′, which is CEST with the 2-norm of w in place of the 1-norm, so that its exponent term is $E_2 = e_x + e_y - \lfloor \log_2 \|w\|_2 \rfloor$. At the proved widths (see Section 7.4) the random families (fine mantissa bits, flat, lognormal, random signs) stay inside the Criterion 1 bound, below a ratio of 0.4 (the worst, 0.39, from the fine-mantissa-bit family at 54 bits, the width of the exponent rules there), where the constructed adversary reaches 0.999; we therefore lead with constructed inputs and

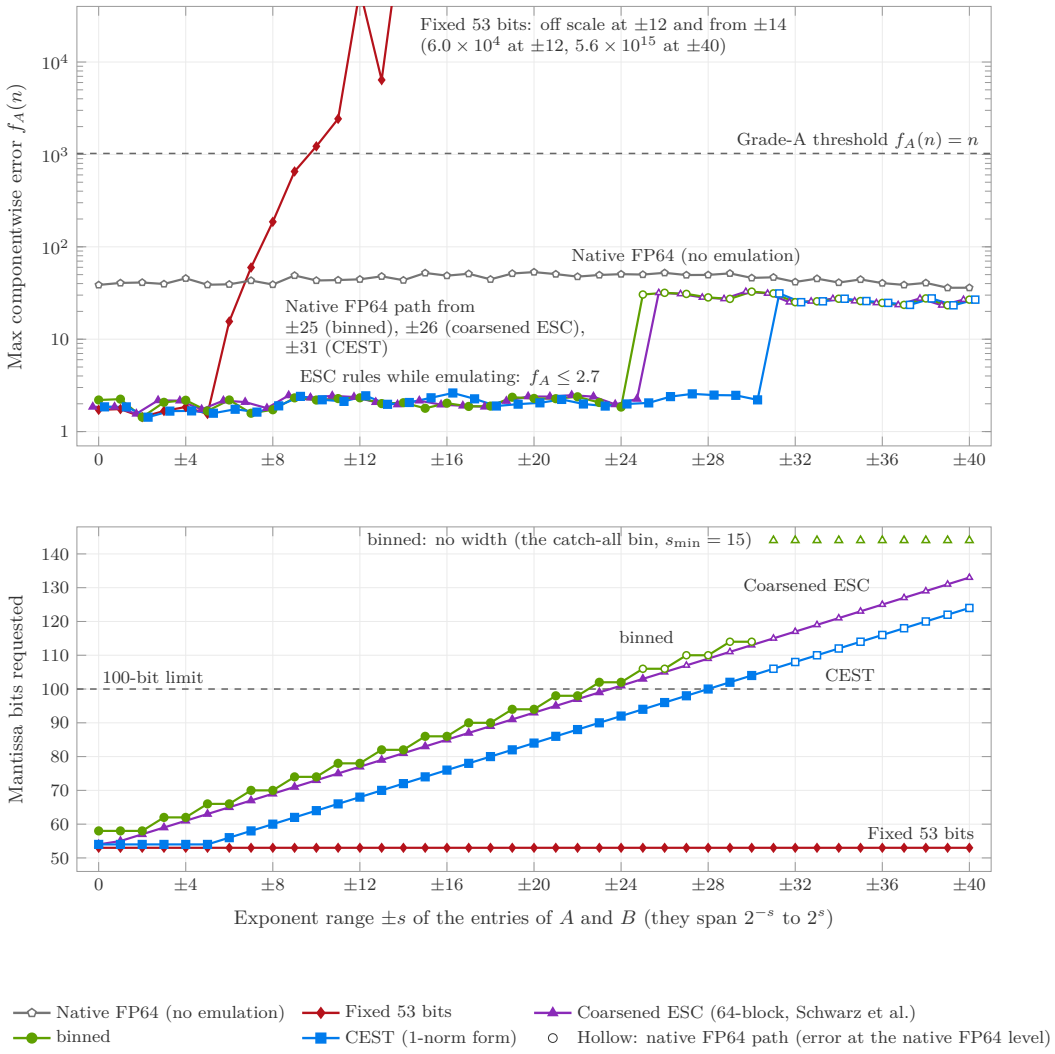


Figure 1: cuBLAS on "Grading the BLAS" [3] Test 2c, $m = n = k = 1024$: on an NVIDIA B200, at every exponent range from ±0 to ±40, the maximum componentwise error $f_A(n)$ of five modes, with the grade-A threshold $f_A(n) = n$, and the mantissa bits each rule requests, with the library's nominal 100-bit limit and binned's no-width cells. By these widths the hand-off came a few bits above that limit. The widths are those of the rules, evaluated on matrices of the test's construction, not measured: these runs logged none. Every width the library logged in separate runs equals the evaluated width (Section 6.1). Hollow markers: the GEMM ran in FP64, the native run throughout and a rule's run where it took the FP64 path (from ±25 for binned, ±26 for the coarsened ESC, and ±31 for CEST), which its error level shows (at least 23 there, at most 2.7 while the fixed-point kernels ran). Every error is the harness's exact value.

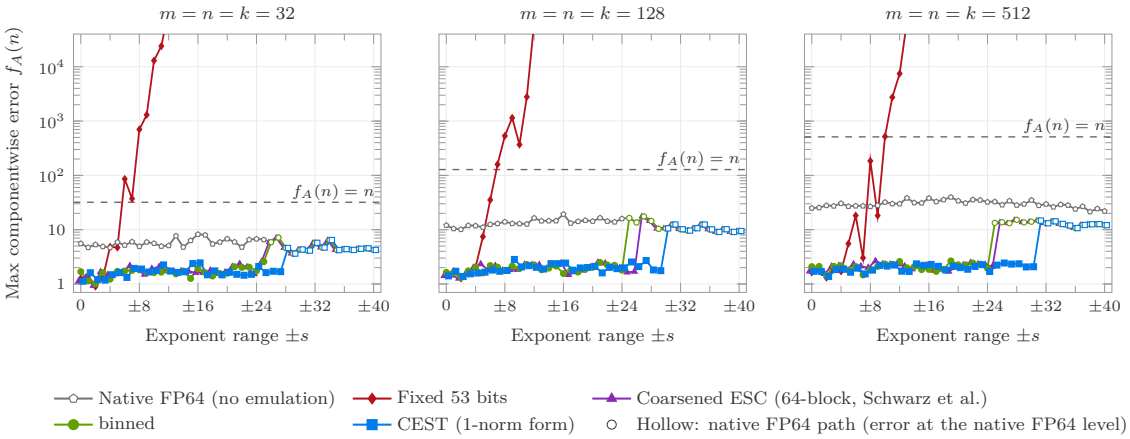


Figure 2: cuBLAS on "Grading the BLAS" [3] Test 2c, maximum componentwise error against the exponent range, by matrix size: the error panel of Figure 1 for three more of the six sizes, $n = 32$, 128, and 512, each against its own grade-A line $f_A(n) = n$. At these three sizes the three rules stay below 2.9 while they emulate (the largest value, 2.83, is CEST at $n = 128$, ±9) and at or above 3.6 on the FP64 path; binned hands off at ±25 (±26 at $n = 32$), the coarsened ESC at ±26 (±27 at $n = 128$), and CEST at ±28, ±30, and ±31 as n grows. For $n \le 128$ the FP64 path returns the native run's product, so from the hand-off the native curve lies under the rules' hollow markers.

report random results as the typical case.

Bits requested on synthetic matrices. Figure 5 shows the mantissa bits the five rules request for a GEMM with $m = n = k = 1024$ on nine kinds of synthetic matrices. The bars are calculated from the rules of Section 4 on the matrices themselves, in the forms the caption gives; none is a width measured with cuBLAS. On standard normal entries the rules ask 54 to 63 bits: CEST 54, CEST' 56, the exponent-sum ESC 57, binned 58, and the coarsened ESC 62 or 63, since it bounds the largest product of a block of 64 from below, by the largest exponent of one vector in the block plus the smallest of the other. One percent of zeros moves only the coarsened ESC, to 63 or 64 bits. With ten percent the coarsened ESC has no width and the other four are unchanged; at this size the coarsened ESC loses its width between one and two percent of zeros. On Test 2c the two exponent rules agree (54, 93, and 133 bits at ±0, ±20, and ±40); binned asks one bit more at ±20 and has no width at ±40, and at both CEST asks nine bits fewer and CEST' four. On matrices of the form that Ozaki, Uchino, and Imamura use as tests [16] (they take ϕ from 0.5 to 4) the rules ask 58 to 69 bits at $\phi = 1$; at $\phi = 5$ four of them ask 92 to 94 bits and the coarsened ESC 109 or 110; at $\phi = 10$ binned has no width, the coarsened ESC asks 162 to 175 bits, and the other three 127 to 142.

6.3 Adversarial details

Each Criterion 1 constant of the exponent rules comes with an input that attains it; the norm forms are at the +3 of the library, two bits above what Criterion 1 needs before the final rounding (Section 7.4).

At a width of $\hat{M}$ bits the entries of x are rounded to the grid of step $2^{e_x+1-\hat{M}} = 2\hat{\epsilon}(\|x\|_\infty)$, and those of y likewise with e_y. The FP64 unit in the last place (ulp) of a nonzero real a is $2\epsilon(a)$, and y_l is the *partner* of x_l.

The midpoint adversary: an input on which rounding the converted operands to nearest moves every fixed-point term up by nearly half a grid step, so that the error is the most the width allows. Here $x_1 = y_1 = 1$, a pair in the top binade of both vectors, so $E_\infty = 0$ and every rule sits at its base width. The other $k-1$ entries of x lie one FP64 ulp above a midpoint $(m_l + \frac{1}{2})\,g$ of the $\hat{M}$-bit grid, $g = 2^{1-\hat{M}}$ (since $e_x = 0$), between $2^{20-\hat{M}}$ and $2^{21-\hat{M}}$ (33 to 34 binades below the anchor $x_1 = y_1 = 1$ at $\hat{M} = 54$). Their partners in y are $2 - 4 \cdot 2^{-52}$, just below 2, so that each term's error is multiplied by nearly 2. Nearest rounding at $\hat{M}$ bits moves every small entry up by almost half a step, so $\hat{z}$ lands $(k-1)\,2^{53-\hat{M}}$ ulps of z above z, $(k-1)/2$ ulps at $\hat{M} = 54$,

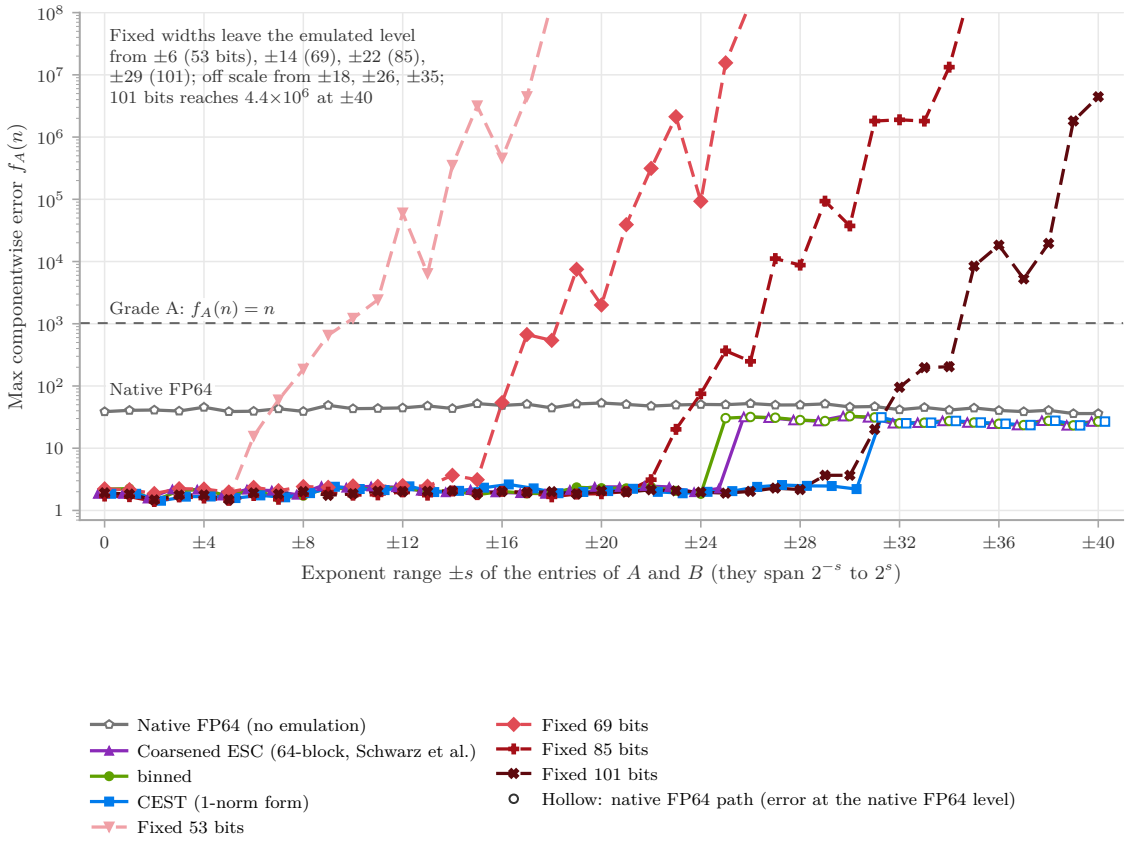


Figure 3: cuBLAS on "Grading the BLAS" [3] Test 2c, $m = n = k = 1024$, fixed widths against the width rules: on an NVIDIA B200, the maximum componentwise error of the runs at fixed mantissa widths of 53, 69, 85, and 101 bits (dashed) against native FP64 and the three width rules of Figure 1. Each fixed width leaves the emulated level where the exponent range outgrows it, first at ±6, ±14, ±22, and ±29, and first loses grade A at ±10, ±19, ±27, and ±35 (the climb is not monotone from range to range); 53 bits is off the scale from ±18 (5.6×10^{15} at ±40), 69 from ±26, 85 from ±35, and 101 bits reaches 4.4×10^6 at ±40.

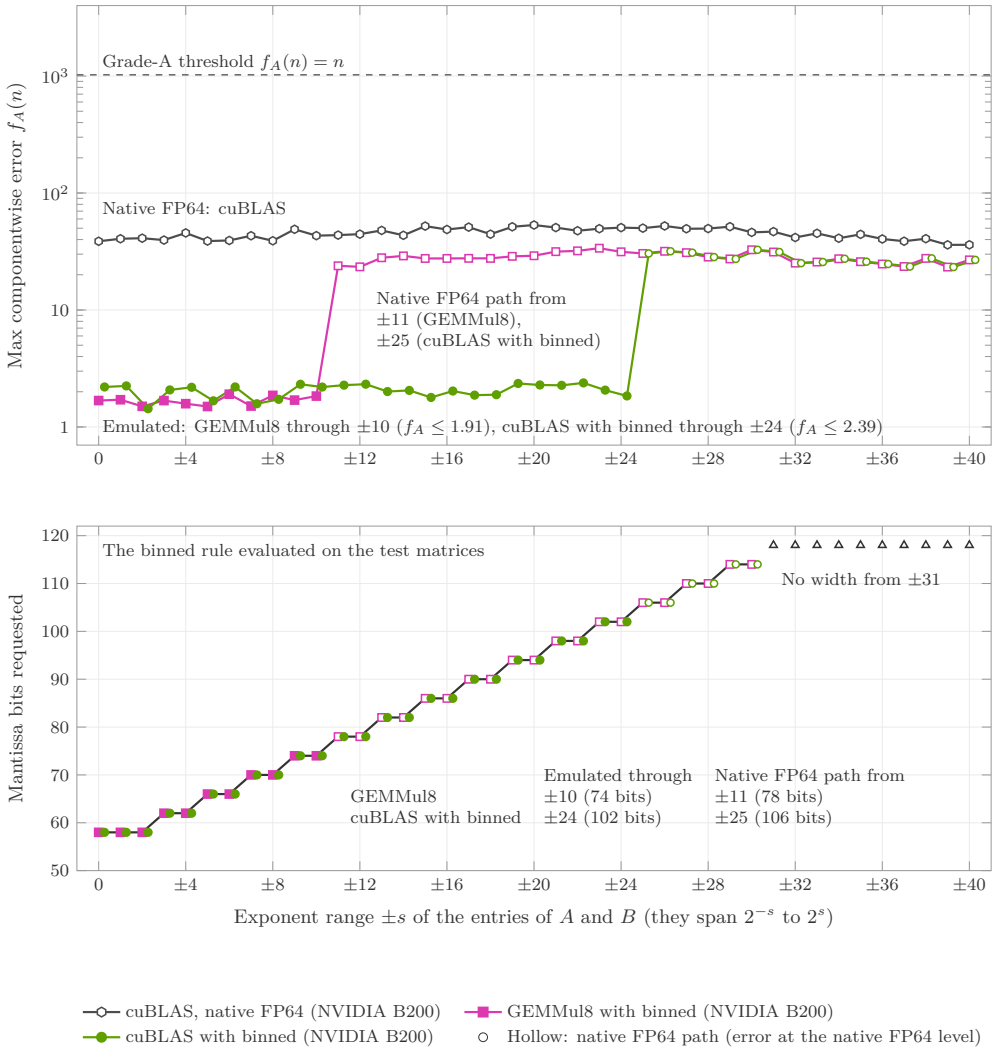


Figure 4: GEMMul8 on an NVIDIA B200, "Grading the BLAS" [3] Test 2c, $m = n = k = 1024$: GEMMul8 with the binned rule, with two of the cuBLAS runs of Figure 1 (cuBLAS with the same rule and its native FP64 GEMM); the maximum componentwise error $f_A(n)$ at every exponent range from ±0 to ±40, with the grade-A threshold $f_A(n) = n$, and the mantissa bits binned requests, each width carrying the marker of GEMMul8 and that of cuBLAS with binned (triangles: the ranges, from ±31, at which the rule gives no width). GEMMul8 logs no width, and these cuBLAS runs logged none: the widths are those of the rule, evaluated on matrices of the test's construction, the same widths as binned's in Figure 1. Hollow markers: the GEMM ran in FP64, the native run throughout, GEMMul8 from ±11, and cuBLAS with binned from ±25, which the error level of each of the two shows (for GEMMul8 at least 23 from ±11 and at most 1.91 through ±10; for cuBLAS with binned at least 23 from ±25 and at most 2.39 through ±24). From ±25 GEMMul8 and cuBLAS with binned return the same product; the markers of cuBLAS with binned are set a little to the right of their ranges so that both runs show.

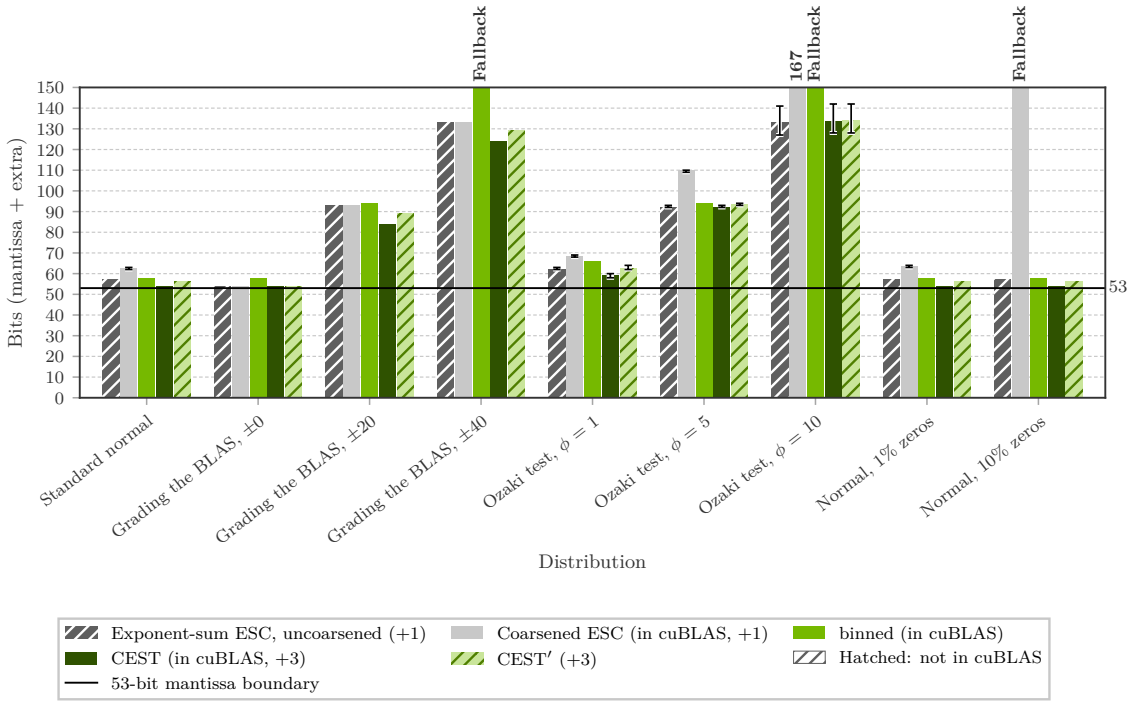


Figure 5: Synthetic distributions ($m = n = k = 1024$), all values in bits: mantissa bits requested by the five rules for a GEMM of that size on synthetic matrices, calculated from the rules of this paper (Section 4) on the matrices themselves, not with cuBLAS: no bar is a width measured in the library. One width per GEMM, the largest over its n^2 dot products. Bars: the mean over five draws of A and B; whiskers: the smallest and largest draw. Grading the BLAS: Test 2c [3] at exponent range $\pm s$, one construction each. Ozaki test: entries $(\text{rand} - 0.5)\exp(\phi\,\text{randn})$ [16]. Normal, 1% and 10% zeros: standard normal entries, each set to zero with that probability. The three rules that are in cuBLAS take the form of the widths it logged (Section 6.1): the coarsened ESC at $53 + E_\infty^{(c)} + 1$, binned at $58 + 4s_{\min}$ ($s_{\min}$ is the bin $\mathtt{b}$ of Algorithm 4), and CEST at $53 + E_1 + 3$ with a floor of 54. CEST′, which is not in cuBLAS, takes the same form with E_2. The exponent-sum ESC is $53 + E_\infty + 1$. These five forms are the rules of Table 2. Fallback: the rule gives no width. The bar marked 167 is off the scale (162 to 175 bits over the draws). The chart shows requests only: in the runs of Section 6.1 cuBLAS ran in FP64 every request that, by the evaluated widths, was more than a few bits above its nominal 100-bit cap.

against an allowance $k\,\epsilon(1)\,\|w\|_1$ of about $k/2$ ulps. It is generated by construction, not by search: for the target count $\hat{M}$, draw an integer m_l for each small entry uniformly from $[2^{19}, 2^{20}]$ (in steps of $2^{\hat{M}-54}$ when $\hat{M} > 54$, so that each midpoint lies $2^{-\hat{M}}$ above a point of the 54-bit grid), set x_l to the FP64 number one ulp above $(m_l + \frac{1}{2})\,g$ and $y_l = 2 - 4 \cdot 2^{-52}$, and verify the claim by exact rational arithmetic, computing z and the fixed point at every width. The random m_l only avoid a special pattern, and any values in the range give the same numbers.

Criterion 1 holds at every rule's own width, with a worst ratio of 0.999 over the instances, that is $(k-1)/k$, to the allowance at $k = 1024$, attained by the midpoint adversary above on the exponent rules at their width of 54 bits (Table 2). For binned the worst ratio is 0.06, one sixteenth of that, as its width is four bits more. For the norm forms, at +3 with a floor of 54, it is 0.25, from the same adversary built against their 56 bits. For the exponent rules $(k-1)/k$ is also the worst case over all inputs, and for the norm forms the ratio is below $1/4$ on every input (Section 7.4). After the fixed-point result is rounded to FP64 the worst ratio over the instances is unchanged for the exponent rules, 0.25 for the norm forms, and 0.16 for binned, the one cell of Table 2 set by a random input, from the fine-mantissa-bit family at $k = 7$, where half an ulp of final rounding weighs against an allowance of only a few ulps. After the rounding Criterion 1 holds on every input for the exponent rules and for the norm forms at +3; at +1 the norm forms would exceed the allowance on inputs built for the purpose (ratios of 1.25 and 1.50 at $k = 2$), which is why they are not used at +1 (Section 7.4).

Mixed signs inherit every single-sign case: no mixed-sign cell exceeds its single-sign value.

Table 2: Criterion 1 under round to nearest: every rule at its own width, $53 + C$ plus the row's own exponent term (E_∞, E_1, or E_2), with C its constant. Green, safe: the criterion holds at the rule's own width on every instance, and the number in parentheses is the worst ratio $|\hat{z} - z|/(k\,\epsilon(1)\,\|w\|_1)$ over the instances, the error against the Criterion 1 allowance, which is $k/2$ to k ulps of $\|w\|_1$; 1 would be the bound itself. The norm forms are at +3 with a floor of 54 bits, the form of the widths cuBLAS logged for CEST (Section 6.1); at +1 the rounding to FP64 could take their result past the allowance (Section 7.4). For binned, $s_{\min}$ is the bin b of Algorithm 4. "After fl" is the same ratio after the fixed-point result is rounded to FP64. Mixed-sign inputs inherit every cell. Every cell by exact rational arithmetic.

Rule (constant under round to nearest)	Criterion 1	Criterion 1 after fl
Exponent-sum ESC, uncoarsened (+1)	Safe (0.999)	Safe (0.999)
Coarsened ESC, 64-block (+1)	Safe (0.999)	Safe (0.999)
binned ($58 + 4s_{\min}$)	Safe (0.06)	Safe (0.16)
1-norm form, CEST (+3, floor 54)	Safe (0.25)	Safe (0.25)
2-norm form, CEST′ (+3, floor 54)	Safe (0.25)	Safe (0.25)

The *exponent span* of the data is the number of binades between the smallest and the largest nonzero entry exponents.

The wasted maxima (any rule, any span), the range family parametrized by s: entries confined to $[2^{-s}, 2^{s+1})$, a span of $2s$ binades, the maximum of x paired with the minimum of y and vice versa, the remaining entries of x at the floor of the span, one FP64 ulp above a midpoint of the $\hat{M}$-bit grid so that nearest rounding moves each by almost half a step, and the remaining entries of y at the top of the span. This is within one bit of the worst Criterion 1 input for a given exponent span: it needs $2s + 43$ bits at $k = 1024$ (no input of the span needs more than $2s + 44$, which an input with both entries of each remaining index at the floor of the span needs for $s \le 8$), against which the exponent-sum ESC and the coarsened ESC provide $2s + 54$, binned $2s + 58$ ($2s + 56$ at odd s; no width from $s = 30$, as the rule falls back from bin 15, line 15 of Algorithm 4), CEST (the 1-norm form) $2s + 46$, and CEST′ (the 2-norm form) $2s + 51$ (each raised to 54 where it is less). Under the library's nominal 100-bit cap (Section 6.1) the exponent rules request more than the cap from $s = 24$ and binned from $s = 22$, while at $s = 24$ CEST and CEST′ still request less (94 and 99 bits against a true need of 91); on Test 2c, by the evaluated widths, the hand-off came a few bits past the cap. Test 2 of Demmel et al. [3] (Section 6.1) has the same structure, but its y entries are not all at the top, so it needs less than this family at the same s, by 6 to 12 bits for s up to 40 and by 10 to 12 at $s = 100, 200, \ldots, 500$, the last being the range of the published Tests 2b and 2c ($2^{\pm 500}$); the counts are for one draw of the test's

vector and move by a few bits with the draw.

The shortest instances, in decimal. Each pattern above is already present in a vector of three or four entries, and the entries are ordinary FP64 numbers, so the instances can be printed (Tables 3 and 4) and typed into any implementation. Every decimal in the two tables is the shortest string that reads back to the same FP64 number under round to nearest (Python's `repr`), so the vectors reproduce bit for bit, and the form columns give the construction.

The midpoint adversary at $k = 4$ (Table 3; built against $\hat{M} = 54$). The three small entries lie one FP64 ulp above a midpoint of the 54-bit grid, $g = 2^{-53}$, and their partners $2 - 2^{-50}$ are just below 2, so the fixed point lands 1.5 ulps above z against an allowance of 2 ulps, a ratio of $0.75 = (k-1)/k$, and Criterion 1 holds at 54 bits. At binned's 58 bits the midpoints of the 54-bit grid are grid points and the error is negligible, about 3.5×10^{-10} ulps of z (the entries lie one FP64 ulp above those points). Appending a negligible negative product, $x_5 = 2^{-90}$ and $y_5 = -1$, changes neither the widths nor the fixed point, so the ratio is $0.6 = (k-2)/k$ at $k = 5$, the single-sign inheritance in its smallest form.

The wasted maxima at $k = 3$, $s = 4$ (Table 4). The maximum of x meets the minimum of y and vice versa, and the third pair is the floor of the span against its top: the largest and the smallest product are one binade apart, yet $E_\infty = 2s = 8$ and the rules ask 62 (the exponent-sum ESC and its block coarsening), 66 (binned), 62 (CEST), and 63 (CEST′) bits. Every entry is represented exactly from 61 bits on.

Table 3: The midpoint adversary under round to nearest, built against $\hat{M} = 54$, at $k = 4$: the entries by their form and as the shortest decimals that read back to the same FP64 numbers.

l	x_l (form)	y_l (form)	x_l	y_l
1	1	1	1	1
2	$(m+\frac{1}{2})2^{-53} + 2^{-86}$, $m = 754715$	$2 - 2^{-50}$	8.379025251414875e-11	1.9999999999999991
3	$(m+\frac{1}{2})2^{-53} + 2^{-86}$, $m = 874262$	$2 - 2^{-50}$	9.70626357066351e-11	1.9999999999999991
4	$(m+\frac{1}{2})2^{-53} + 2^{-86}$, $m = 956121$	$2 - 2^{-50}$	1.0615081036391417e-10	1.9999999999999991

Table 4: The wasted maxima at $s = 4$, $k = 3$: the entries by their form and as the shortest decimals that read back to the same FP64 numbers.

l	x_l (form)	y_l (form)	x_l	y_l
1	2^4	2^{-4}	16	0.0625
2	2^{-4}	2^4	0.0625	16
3	$2^{-4} + 2^{-56}$	$2^5 - 2^{-48}$	0.06250000000000001	31.999999999999996

How the requirement scales with k. Criterion 1 does not scale. The proved (see Section 7.4) constants hold for every k, and for the exponent rules the worst case approaches the bound from below as $(k-1)/k$ (0.75, 0.857, 0.984, 0.999 at $k = 4, 7, 64, 1024$, respectively), because the adversary's $k-1$ half-steps and the allowance $k\,\epsilon(1)\,\|w\|_1$ grow together; for the norm forms at +3 the ratio stays below 1/4. Two k dependences run the other way: the flat-data saving of the norm forms, about $\log_2 k$ and $\frac{1}{2}\log_2 k$ bits (Section 3), which the floor of 54 bits caps, and the range stress, where the need falls with k while the maxima-tied widths do not. The wasted maxima need about $2s + 53 - \log_2 k$ (the $2s + 43$ above at $k = 1024$) while the exponent rules' $2s + 54$ does not move. On Test 2 at ±0, ±10, and ±29 the need falls by one to two bits per doubling of n (46 to 37, 63 to 55, and 99 to 91, for $n = 32$ to 1024) while every maxima-tied width moves by at most a few bits with n, so the rules' margin there is about $\log_2 n$ bits plus the structure's own slack. Finally, k enters the cost: the integer accumulator of an Ozaki-II product at $\hat{M}$ bits needs $2\hat{M} + \lceil \log_2 k \rceil + 1$ bits, one more per doubling of k.

7 Proofs

In this section, we provide proofs of the error bounds in Section 2.

7.1 Fundamental Uncertainty Bounds

Proof of Proposition 2.1. Applying the inequalities $|\overline{x}_i - x_i| \le \epsilon(x_i)$, $|\overline{y}_i - y_i| \le \epsilon(y_i)$,

$$\begin{aligned}
|\overline{x} \cdot \overline{y} - x \cdot y| &\le \sum_{i=1}^{k} |\overline{x}_i \overline{y}_i - x_i y_i| \\
&\le \sum_{i=1}^{k} \left|(\overline{x}_i - x_i) y_i + x_i(\overline{y}_i - y_i) + (\overline{x}_i - x_i)(\overline{y}_i - y_i)\right| \\
&\le \sum_{i=1}^{k} \left[\epsilon(x_i)|y_i| + |x_i|\epsilon(y_i) + \epsilon(x_i)\epsilon(y_i)\right] \\
&\le \sum_{i=1}^{k} 2\epsilon(1)|x_i||y_i| + O(\epsilon(1)^2) \\
&= 2\epsilon(1)\|w\|_1 + O(\epsilon(1)^2).
\end{aligned}$$

□

Proof of Proposition 2.2. By direct calculation, using the independence of the X_i, Y_i:

$$\begin{aligned}
\operatorname{Var}(X \cdot Y) = \sum_{i=1}^{k} \operatorname{Var}(X_i Y_i) &= \sum \left[x_i^2 \operatorname{Var}(Y_i) + y_i^2 \operatorname{Var}(X_i) + \operatorname{Var}(X_i)\operatorname{Var}(Y_i)\right] \\
&= \sum \left[\frac{1}{3}\left(x_i^2 \epsilon(y_i)^2 + y_i^2 \epsilon(x_i)^2\right) + \frac{1}{9}\epsilon(x_i)^2 \epsilon(y_i)^2\right].
\end{aligned}$$

Since $\frac{1}{2}\epsilon(1) \cdot |a| < \epsilon(a) \le \epsilon(1) \cdot |a|$,

$$\begin{aligned}
\operatorname{Var}(X \cdot Y) &\ge \sum \left[\frac{1}{6} x_i^2 y_i^2 \left(\epsilon(1)^2 + \frac{\epsilon(1)^4}{24}\right)\right] \\
&\ge \frac{1}{6}\epsilon(1)^2 \cdot \|w\|_2^2.
\end{aligned}$$

Taking square roots, the standard deviation is bounded below by

$$\sigma(X \cdot Y) \ge \frac{1}{\sqrt{6}} \cdot \epsilon(1) \cdot \|w\|_2.$$

The upper bound can be derived similarly:

$$\sigma(X \cdot Y) \le \frac{2}{\sqrt{6}} \sqrt{1 + \frac{\epsilon(1)^2}{6}} \cdot \epsilon(1) \cdot \|w\|_2.$$

□

7.2 Fixed-Point Error Bounds

Proof of Proposition 2.3. The fixed-point discretization errors can be expressed in terms of the round-off error $\hat{\epsilon}(a) := 2^{\lfloor \log_2 a \rfloor - \hat{M}}$, applied to the largest elements $\|x\|_\infty$, $\|y\|_\infty$:

$$|x_i - \hat{x}_i| \le \hat{\epsilon}(\|x\|_\infty), \qquad |y_i - \hat{y}_i| \le \hat{\epsilon}(\|y\|_\infty).$$

Plugging these errors into the dot product:

$$\begin{aligned}
|\hat{z} - z| &\le \sum_{i=1}^{k} |\hat{x}_i \hat{y}_i - x_i y_i| \\
&\le \sum \left[\hat{\epsilon}(\|x\|_\infty) \cdot |y_i| + \hat{\epsilon}(\|y\|_\infty) \cdot |x_i| + \hat{\epsilon}(\|x\|_\infty)\hat{\epsilon}(\|y\|_\infty)\right] \\
&= \hat{\epsilon}(\|x\|_\infty) \cdot \|y\|_1 + \hat{\epsilon}(\|y\|_\infty) \cdot \|x\|_1 + k\hat{\epsilon}(\|x\|_\infty)\hat{\epsilon}(\|y\|_\infty).
\end{aligned}$$

Both inequalities above, while quite pessimistic in the average case, can be made tight with an appropriate choice of input data x, y, if $E < 0$. Weakening the bound slightly further, we can pull out $\hat{\epsilon}(1) = 2^{-\hat{M}}$:

$$\begin{aligned} |\hat{z} - z| &\leq \hat{\epsilon}(\|x\|_\infty) \cdot \|y\|_1 + \hat{\epsilon}(\|y\|_\infty) \cdot \|x\|_1 + O(\hat{\epsilon}(1)^2) \\ &\leq \hat{\epsilon}(1) \cdot \left[\|x\|_\infty \|y\|_1 + \|y\|_\infty \|x\|_1\right] + O(\hat{\epsilon}(1)^2). \end{aligned} \tag{13}$$

We can fine-tune this bound slightly if $E > 0$, i.e., the fixed-point representation has more mantissa bits as the floating-point representation. In this case, the largest elements in x, y can be exactly represented in fixed point, and do not contribute any error to $\hat{z}$. That is, $\hat{x}_i = x_i$ if $|x_i| \geq 2^{-E}\|x\|_\infty$, and similarly for y.

Let $I_x \coloneqq \{i \in \{1, \ldots, k\} : |x_i| \geq 2^{-E}\|x\|_\infty\}$ be the set of error-free indices in x, and define I_y similarly. For each i, consider the error term r_i associated with discretizing x_i, y_i:

$$r_i = (\hat{x}_i - x_i) \cdot y_i + (\hat{y}_i - y_i) \cdot x_i + O(\hat{\epsilon}(1)^2).$$

If $i \in I_x$, then the first term of r_i vanishes; similarly, if $i \in I_y$ the second term vanishes. In either case, $|r_i|$ can be bounded to first order by $\hat{\epsilon}(1)\|x\|_\infty\|y\|_\infty$. Finally, if $i \notin I_x \cup I_y$, then $|x_i| \leq \frac{1}{2}\|x\|_\infty$, $|y_i| \leq \frac{1}{2}\|y\|_\infty$, and again to first order $|r_i|$ is bounded by $\hat{\epsilon}(1)\|x\|_\infty\|y\|_\infty$. Summing over i, we conclude

$$|\hat{z} - z| \leq \hat{\epsilon}(1) \cdot k\|x\|_\infty\|y\|_\infty + O(\hat{\epsilon}(1)^2),$$

completing the proof. □

Proof of Proposition 2.4. The proof is very similar to that of Proposition 2.2. By direct calculation,

$$\begin{aligned} \operatorname{Var}(\hat{x} \cdot \hat{y}) = \sum_{i=1}^{k} \operatorname{Var}(\hat{x}_i \hat{y}_i) &= \sum \left[x_i^2 \operatorname{Var}(\hat{y}_i) + y_i^2 \operatorname{Var}(\hat{x}_i) + \operatorname{Var}(\hat{x}_i)\operatorname{Var}(\hat{y}_i)\right] \\ &= \sum \left[\frac{1}{3}\left(x_i^2 \hat{\epsilon}(\|y\|_\infty)^2 + y_i^2 \hat{\epsilon}(\|x\|_\infty)^2\right) + \frac{1}{9}\hat{\epsilon}(\|x\|_\infty)^2 \hat{\epsilon}(\|y\|_\infty)^2\right] \\ &= \frac{1}{3}\left[\|x\|_2^2 \hat{\epsilon}(\|y\|_\infty)^2 + \|y\|_2^2 \hat{\epsilon}(\|x\|_\infty)^2\right] + \frac{k}{9}\hat{\epsilon}(\|x\|_\infty)^2 \hat{\epsilon}(\|y\|_\infty)^2 \\ &\leq \frac{\hat{\epsilon}(1)^2}{3}\left[\|x\|_2^2 \|y\|_\infty^2 + \|y\|_2^2 \|x\|_\infty^2\right] + O(\hat{\epsilon}(1)^4). \end{aligned}$$

Taking square roots, the standard deviation is bounded above by

$$\sigma(\hat{x} \cdot \hat{y}) \leq \frac{\hat{\epsilon}(1)}{\sqrt{3}} \sqrt{\|x\|_2^2 \|y\|_\infty^2 + \|y\|_2^2 \|x\|_\infty^2} + O(\hat{\epsilon}(1)^2). \tag{14}$$

A lower bound can be extracted similarly, leading to the same right-hand side as (14), but multiplied by $\frac{1}{2}$. This establishes (2), and (3) is an immediate consequence.

Note that, just as in the worst-case analysis, we may fine-tune this bound when $E > 0$ by noting that the largest terms of x and y have no discretization error. □

7.3 Floating-Point Error Bounds

Proof of Proposition 2.5. Without loss of generality, assume $z \geq 0$. (If $z < 0$, we may replace x by $-x$ and negate at the end without changing the result.) Choose a permutation π such that $w_{\pi(1)} \geq w_{\pi(2)} \geq \cdots \geq w_{\pi(k)}$.

For brevity, let us reorder the input vectors x, y by π so that $w_1 \geq w_2 \geq \ldots \geq w_k$. Let $N_+ = |\{i : w_i > 0\}|$ count the number of positive w_i. Note that by our assumptions on x, y, the w_i are all nonzero.

Let $s_i = \sum_{j=1}^{i} w_j$ be the vector of partial sums. The s_i form a sequence that is first increasing ($i \leq N_+$), then potentially decreasing ($i > N_+$). Let us count how many s_i exceed $\|w\|_1/4$.

Note that $s_{N_+} \geq \|w\|_1/2$, as

$$\|w\|_1 = \sum_{i \leq N_+} w_i + \sum_{i > N_+} -w_i = 2s_{N_+} - z \leq 2s_{N_+}.$$

By a similar argument, $s_{\lceil N_+/2\rceil} \geq \|w\|_1/4$ and $s_{k-\lceil (k-N_+)/2\rceil} \geq \|w\|_1/4$. Thus $s_i \geq \|w\|_1/4$ for all $i \in [\lceil N_+/2\rceil, k - \lceil (k-N_+)/2\rceil]$, which contains at least $k/2$ indices. We conclude that

$$\|s\|_2^2 \geq \frac{k}{32}\|w\|_1^2,$$

finishing the proof. □

Proof of Proposition 2.6. Averaging the variance (5) over all permutations,

$$\frac{1}{k!}\sum_{\pi} \mathrm{Var}(d_\pi(x,y)) = \frac{c\epsilon(1)^2}{3k!}\sum_{\pi}\sum_{i=1}^{k}\left(\sum_{j=1}^{i} w_{\pi(j)}\right)^2, \tag{15}$$

for some $c \in (\frac{1}{4}, 1]$.

We will simplify equation (15) in two steps. First, let us count how many times a particular instance of the innermost sum occurs. Each instance of the innermost sum is equal to $\sum_{i\in I} w_i$ for some non-empty index set $I \subseteq [k]$. Conversely, given a subset I, we encounter the sum $\sum_{i\in I} w_i$ in (15) exactly $|I|! \cdot (k-|I|)!$ times, so

$$\begin{aligned}\frac{1}{k!}\sum_{\pi} \mathrm{Var}(d_\pi(x,y)) &= \frac{c\epsilon(1)^2}{3k!}\sum_{I\subseteq[k]}\left(\sum_{i\in I} w_i\right)^2 \cdot |I|! \cdot (k-|I|)! \\ &= \frac{c\epsilon(1)^2}{3}\sum_{I\subseteq[k]}\left(\sum_{i\in I} w_i\right)^2 \cdot \binom{k}{|I|}^{-1}. \end{aligned} \tag{16}$$

After expanding the square, we will get a linear combination of binomials $w_i w_j$. Separate these into two classes: cross-terms ($w_i w_j$, $i \neq j$) and squares (w_i^2). Even without knowing the coefficients of each binomial, we know that by symmetry all the cross-terms have the same coefficient, say Q_1, and all the squares have identical coefficients, say Q_2:

$$\frac{1}{k!}\sum_{\pi} \mathrm{Var}(d_\pi(x,y)) = c\epsilon(1)^2\left[\sum_{1\leq i<j\leq k} Q_1 w_i w_j + \sum_{i=1}^{k} Q_2 w_i^2\right].$$

Changing variables to $Q_1 = 2N_1$, $Q_2 = N_1 + N_2$, this simplifies to

$$\frac{1}{k!}\sum_{\pi} \mathrm{Var}(d_\pi(x,y)) = c\epsilon(1)^2\left[N_1(\textstyle\sum w_i)^2 + N_2\|w\|_2^2\right] = c\epsilon(1)^2\left[N_1 z^2 + N_2\|w\|_2^2\right].$$

All that remains is to calculate the coefficients N_1, N_2. Letting $\ell = |I|$ and counting the number of times a cross-term $w_i w_j$ appears in (16),

$$\begin{aligned} 2N_1 = \frac{2}{3}\sum_{\ell=2}^{k}\binom{k}{\ell}^{-1}\binom{k-2}{\ell-2} = \frac{2}{3}\sum_{\ell=2}^{k}\frac{\ell(\ell-1)}{k(k-1)} &= \frac{2}{3k(k-1)}\sum_{\ell=2}^{k}[\ell(\ell-1)] \\ &= \frac{2}{3k(k-1)}\frac{k(k+1)(k-1)}{3} \\ &= \frac{2}{9}(k+1). \end{aligned}$$

The square terms are simpler:

$$N_1 + N_2 = \frac{1}{3}\sum_{\ell=1}^{k}\binom{k}{\ell}^{-1}\binom{k-1}{\ell-1} = \frac{1}{3}\sum_{\ell=1}^{k}\frac{\ell}{k} = \frac{1}{6}(k+1),$$

and we conclude $N_1 = (k+1)/9$, $N_2 = (k+1)/18$. Therefore the average variance over all permutations is

$$\frac{1}{k!}\sum_{\pi} \mathrm{Var}(d_\pi(x,y)) = \frac{(k+1)c\epsilon(1)^2}{18}\cdot\left[2z^2 + \|w\|_2^2\right],$$

completing the proof. □

7.4 The Criterion 1 constants

For Criterion 1, the first line of (13) (in the proof of Proposition 2.3), with $\|y\|_1 < k\,2^{e_y+1}$, gives +2 for every rule whose exponent term satisfies $2^{e_x+e_y-E} \le \|w\|_1$, which all four do (the +3 of Fasi, Mary, and Mikaitis [4] is for a full-step conversion error). The +1 of this paper holds by a sharper argument, index by index. In this paragraph let E be the exponent term of a rule (elsewhere $E = \hat{M} - M$) with $2^{e_x+e_y-E} \le \|w\|_1$, let its width satisfy $\hat{M} \ge 54 + E$ and $\hat{M} \ge 53$, and put $B = 2^{e_x+e_y+1-\hat{M}}$, so that $B \le \epsilon(1)\,\|w\|_1$. An FP64 entry x_i is a multiple of $2^{e(x_i)-52}$, so on the grid of step $2^{e_x+1-\hat{M}}$ it converts exactly unless it lies at least $\hat{M} - 52$ binades below its vector's maximum, and its conversion error is at most $2^{e_x-\hat{M}}$. An index with one inexact entry therefore errs by less than $2^{e_x-\hat{M}}\,2^{e_y+1} = B$, and an index with two inexact entries, both that far below their maxima, by less than B as well, the step that needs $\hat{M} \ge 53$. The k indices together err by less than $kB \le k\,\epsilon(1)\,\|w\|_1$, which is Criterion 1 for the fixed-point result at +1. For the exponent rules ($E \ge E_\infty$) the index that attains the largest exponent sum converts exactly, so the error is below $(k-1)\,\epsilon(1)\,\|w\|_\infty$; the midpoint adversary reaches $(k-1)/k$ to within 10^{-6} (Section 6.3), and the constant is tight for $k \ge 3$: built against one bit fewer, the same adversary has the ratio $2(k-1)/k$. At +1 the norm forms would be covered as well, since E_1 satisfies the inequality through the floor of $\log_2 \|w\|_1$ and E_2 through $\|w\|_2 \le \|w\|_1$ (with widths raised to 53 where the formula gives less); but for them every index can err, and the ratio comes within 10^{-12} of 1 for CEST (at $k = 2$, 4, and 1024) and within 10^{-6} of 1 for CEST′ (at $k = 2$, 4, and 64). The final rounding of $\hat{z}$ to FP64 adds up to $\epsilon(1)\,|\hat{z}|$. The exponent rules keep Criterion 1 after it, the exact index leaving room for that half ulp. The norm forms at +1 do not: on $x = (2 - 2^{-50},\ a)$, $y = (a,\ 2 - 5 \cdot 2^{-52})$ with $a = 2^{-8}(1 + 3 \cdot 2^{-52})$, CEST at +1 asks 60 bits, and the ratio is just below 1 before that rounding and 1.25 after it (1.50 on an input on which CEST′ at +1 asks 60 bits as well). The norm forms are therefore used at +3 with a floor of 54, the form of the widths the library logged for CEST (Section 6.1), here extended to CEST′: then $B \le \epsilon(1)\,\|w\|_1/4$, so the ratio is below $1/4$ before the final rounding and below $1/4 + 1/k$ after it for $k \ge 2$ (at $k = 1$ both entries convert exactly), and Table 2 shows them at that width. The tightened bound of Proposition 2.3 does not supply the +1: with $\|x\|_\infty \|y\|_\infty < 2^{e_x+e_y+2}$ it is below $k\,2^{e_x+e_y+2-\hat{M}}$ to first order, the limit of the first line of (13), so it gives +2 as well. Kept in the norms, it is the condition (6) of Section 3.1, $\hat{M} - 53 \ge \max(1, \mathrm{esc}_1(x, y))$; since $E_1 - 1 < \mathrm{esc}_1(x, y) < E_1 + 2$, the smallest width that meets it is $53 + E_1$, $53 + E_1 + 1$, or $53 + E_1 + 2$ where $E_1 \ge 1$, and, as that bound is stated, it gives Criterion 1 to first order in $\hat{\epsilon}(1)$.

8 Related Work

Two papers derive the bit count for FP64 emulation on integer tensor cores from the exponent spread. Schwarz et al. [18] give the exponent-sum ESC with a +1 margin for the mantissa-product carry, coarsened over blocks along k. Their paper is generic in the block length, while the library's coarsened ESC uses 64. Fasi, Mary, and Mikaitis [4] supply the rigorous derivation, whose sharpest bound, their Eq. (3.3), is the 1-norm form with +1 on exact logarithms, and whose max-form bound carries +3. Their separate widths for x and y are a possible further saving not evaluated here. Of the roughly twenty papers citing the two papers, none replaces "ESC plus a constant". Abdelfattah, Dongarra, Fasi, Mikaitis, and Tisseur [1] analyze the error from the input side with a per-row factor and note that it may fail on badly scaled rows. The Ozaki-II line [16] fixes $\hat{M} = 53$ (Kouya [10] adds 16 guard bits) and adapts the modulus count instead, with at least 14 INT8 moduli for FP64 (Uchino [21]). Mazumder, Calotoiu, and Hoefler [13] keep truncation and observe that the vendor's path sets its width per call, which a fixed count cannot. Matsuoka [12] proposes an ESC-adaptive modulus count and notes that truncation does not inherit the round-to-nearest bound, so it must be validated separately. The density-fitting paper [8] picks its slice count from the self-consistent-field (SCF) energy change with no formal bound, and the other application papers (device simulation [11], eigenvector refinement [20], mixed-precision singular value decomposition (SVD) [19]) adopt the method without a rule of their own. Caday [2] observes that the Fasi, Mary, and Mikaitis constant on the max form (ESC +3, his q^*) consistently overpredicts, which agrees with our +2/+1 for that form, that nearest rounding is worth a bit, and that ESC +2 suffices for his 2M complex GEMM. Demmel et al. [3] make the bit count an explicit open problem. This paper adds proved (in Section 7.4) and attained constants under round to nearest, coarsening safety, and the zero and range conventions.

9 Conclusion

We have analyzed the worst-case and average-case error of fixed-point and floating-point dot products and, by matching the fixed-point bounds to the floating-point bounds, obtained the p-exponent span capacities. Of these, esc_1 suffices to match the worst-case error of floating point and esc_2 its average-case error, so the original ESC is conservative for both. The capacities can be approximated with a low-precision (bfloat16) GEMM, and the binned ESC algorithm approximates the original ESC more robustly, and often more tightly, than the coarsened ESC does. In Ozaki-II, the mantissa bit count fixes the scaling factors, and the esc_1 algorithm also returns a bound on the product from which the number of moduli can be chosen (Section 5).

For the exponent rules and the norm forms, this paper proves that choosing the mantissa bit count automatically is numerically safe in the sense of Criterion 1, the classical worst-case bound for a floating-point dot product, and for these rules and binned it demonstrates that safety in software. With the constants of Section 7.4, +1 for the exponent rules and +3 with a floor of 54 bits for the norm forms, these rules keep Criterion 1 on every input, before and after the final rounding to FP64, and the constant of the exponent rules is tight for $k \geq 3$. In exact arithmetic, each of the five rules of Table 2 kept the bound at its own width on every instance, constructed or random, the midpoint adversary reaching $(k-1)/k$ of the allowance on the exponent rules. In a developer build of cuBLAS on an NVIDIA B200, the three rules that the library implements earned grade A on Test 2c of "Grading the BLAS" [3] at all six sizes and at every exponent range from ±0 to ±40. A fixed count of 53 bits first lost the grade between ±6 and ±10, depending on the size. Each rule emulated up to a range, beyond which the library ran the GEMM in FP64, and the grade was kept in both cases.

One possible further saving remains to be evaluated: the separate widths of Fasi, Mary, and Mikaitis [4] for x and y. The cuBLAS measurements used a developer build, which is planned for an upcoming release.

References

[1] Ahmad Abdelfattah, Jack Dongarra, Massimiliano Fasi, Mantas Mikaitis, and Françoise Tisseur. Analysis of floating-point matrix multiplication computed via integer arithmetic. arXiv:2506.11277, 2025.

[2] Peter Caday. The 2M multiplication algorithm for complex matrices. *ACM Transactions on Mathematical Software*, October 2026. arXiv:2609.05419.

[3] James Demmel, Greg Henry, Igor Kozachenko, Julien Langou, Xiaoye Sherry Li, Jason Riedy, and Jackson Vanover. How to grade the accuracy of the BLAS. arXiv:2609.12307, 2026. Test code at `github.com/Reference-LAPACK/grading-the-BLAS`.

[4] Massimiliano Fasi, Theo Mary, and Mantas Mikaitis. Sharper bounds on the required fixed-point precision for emulating floating-point matrix multiplication. *ACM Transactions on Mathematical Software*, 2026. Preprint hal-05658479.

[5] Shun-ichiro Hayashi, Daichi Mukunoki, Tetsuya Hoshino, and Takahiro Katagiri. DGEMM with Ozaki scheme I/II on FP4 tensor cores: A base-13 E2M1 limb representation. arXiv:2608.06812, 2026.

[6] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, second edition, 2002.

[7] Nicholas J Higham and Theo Mary. A new approach to probabilistic rounding error analysis. *SIAM journal on scientific computing*, 41(5):A2815–A2835, 2019.

[8] Hua Huang, Wenkai Shao, and Jeff Hammond. Accelerating density fitting with adaptive precision and 8-bit integer on AI accelerators. *The Journal of Physical Chemistry A*, 130(17):3483–3493, 2026. arXiv:2601.08077.

[9] Claude-Pierre Jeannerod and Siegfried M Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM Journal on Matrix Analysis and Applications*, 34(2):338–344, 2013.

[10] Tomonori Kouya. Ozaki scheme II is fast on CPUs too: Multiple-precision matrix multiplication on Intel AMX-INT8 and Arm SVE2-i8mm. arXiv:2609.27831, 2026.

[11] Alexander Maeder, Denghui Lu, Nicolas Vetsch, Vincent Maillou, Anders Winka, Jiang Cao, Mauro Dossena, Alexandros Nikolaos Ziogas, and Mathieu Luisier. Optimizing semiconductor device simulations through low-precision arithmetic. arXiv:2606.25595, 2026.

[12] Satoshi Matsuoka. Ozaki 2.5: Engineering the deconstruction path of fp64-emulated dense matrix multiplication on FP8 tensor cores. arXiv:2609.09095, 2026.

[13] Pratyai Mazumder, Alexandru Calotoiu, and Torsten Hoefler. FP64 is all you want, INT8 is all you need, FP4/6/8 is all you have. arXiv:2609.37693, 2026.

[14] Hiroyuki Ootomo, Katsuhisa Ozaki, and Rio Yokota. DGEMM on integer matrix multiplication unit. *The International Journal of High Performance Computing Applications*, 38(4):297–313, 2024. arXiv:2306.11975.

[15] Katsuhisa Ozaki, Takeshi Ogita, Shin'ichi Oishi, and Siegfried M. Rump. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. *Numerical Algorithms*, 59(1):95–118, 2012.

[16] Katsuhisa Ozaki, Yuki Uchino, and Toshiyuki Imamura. Ozaki scheme II: A GEMM-oriented emulation of floating-point matrix multiplication using an integer modular technique. *The International Journal of High Performance Computing Applications*, 2026. arXiv:2504.08009.

[17] Reference-LAPACK. grading-the-BLAS: a floating-point accuracy benchmarking harness for DGEMM implementations. GitHub repository, `https://github.com/Reference-LAPACK/grading-the-BLAS`, 2026.

[18] Angelika Schwarz, Anton Anders, Cole Brower, Harun Bayraktar, John Gunnels, Kate Clark, RuQing G. Xu, Samuel Rodriguez, Sebastien Cayrols, Paweł Tabaszewski, and Victor Podlozhnyuk. Guaranteed DGEMM accuracy while using reduced precision tensor cores through extensions of the Ozaki scheme. In *SCA/HPCAsia 2026: Supercomputing Asia and International Conference on High Performance Computing in Asia Pacific Region*, pages 91–101, Osaka, Japan, 2026. ACM. arXiv:2511.13778.

[19] Angelika Schwarz, Rasmus Munk Larsen, Samuel Rodriguez, Christopher Baker, and Lev Kruglyak. Mixed-precision SVD on GPUs via Ogita–Aishima iterative refinement. NVIDIA Research publication, June 2026.

[20] Takeshi Terao and Katsuhisa Ozaki. Forward error-oriented iterative refinement for eigenvectors of a real symmetric matrix. arXiv:2602.19090, 2026.

[21] Yuki Uchino, Katsuhisa Ozaki, and Toshiyuki Imamura. Double-precision matrix multiplication emulation via Ozaki-II scheme with FP8 quantization. arXiv:2603.10634, 2026.