High-Accuracy Floating-Point Matrix Multiplication on Low-Precision Floating-Point and Fixed-Point Hardware

Ahmad Abdelfattah, Jack Dongarra, Massimiliano Fasi, Mantas Mikaitis, Françoise Tisseur

Abstract

We have officially entered the exascale era. At the forefront is the Frontier supercomputer, topping the June 2024 Top500 list[1] as the first machine capable of performing over 10^{18} operations per second in binary64 (*double precision*) arithmetic. Modern supercomputers achieve their remarkable speeds by leveraging machine-learning hardware accelerators, which deliver extraordinary throughput by trading off some degree of accuracy. While these accelerators currently support binary64 arithmetic, the field is shifting, and soon many will be optimized exclusively for lower precision.

Today, fully utilizing the potential of these accelerators requires relying on low-precision formats: TensorFloat-32, bfloat16, binary16 (*half precision*), E4M3, E5M2, and even compact integer data types, such as INT8. These reduced-precision formats can have a throughput up to two orders of magnitude higher than binary64, but they lack the precision needed for traditional scientific simulations, which require higher accuracy to yield meaningful results.

To integrate GPUs effectively into scientific computing, we must reimagine high-precision computations by strategically applying lower precision when feasible. Here, we explore techniques to reformulate a high-precision matrix multiplication as a series of low-precision operations, and we outline two strategies for assigning different precision levels across computations. Matrix multiplication is a fundamental kernel in scientific computing, and efficient implementations underpin the performance of many algorithms in numerical linear algebra. The techniques we discuss will enable numerical codes to make better use of current accelerators, where the performance gap between low and high precision is widening, and of future ones, where high precision will be missing altogether.

General scheme for mixed-precision matrix multiplication Let $\mathcal{F}_{\text{low}}$ and $\mathcal{F}_{\text{high}}$ be a low-precision and a high-precision floating-point format, respectively, and let u_{low} and u_{high} be their unit roundoffs. We consider the computation of $C = AB \in \mathcal{F}_{\text{high}}^{m\times n}$, where $A \in \mathcal{F}_{\text{high}}^{m\times p}$ and $B \in \mathcal{F}_{\text{high}}^{p\times n}$. Rows of A and column of B with only zeros do not affect the result, thus we assume that each row of A and column of B contains at least one nonzero element. The high-precision matrices A and B can be written as the unevaluated sum of low-precision matrices

$$A = A^{(1)} + A^{(2)} + \cdots + A^{(s_A)} + \Delta A, \qquad B = B^{(1)} + B^{(2)} + \cdots + B^{(s_B)} + \Delta B, \tag{1}$$

where the entries of $A^{(1)}$, $A^{(2)}$, …, $A^{(s_A)}$, $B^{(1)}$, $B^{(2)}$, …, $B^{(s_B)}$ belong to $\mathcal{F}_{\text{low}}$, while ΔA and ΔB are truncation errors. With the decomposition (1), we can approximate the product as

$$\widetilde{C} \approx \sum_{k=1}^{s_A}\sum_{\ell=1}^{s_B} A^{(k)}B^{(\ell)}. \tag{2}$$

In terms of runtime, (2) will achieve good performance if the low-precision matrix products of the form $A^{(k)}B^{(\ell)}$ are executed on hardware that can efficiently multiply matrices stored in $\mathcal{F}_{\text{low}}$ and accumulate the result in $\mathcal{F}_{\text{high}}$. Two terms contribute to the total error in the approximation $\widetilde{C}$:

- the *truncation error* $\Delta AB + A\Delta B$, which depends on the splitting strategy in (1); and

[1] https://www.top500.org/lists/top500/list/2024/06/

- a *rounding error*, caused by the matrix products and sums in (2)

Matrix multiplication using multi-word arithmetic A natural way to obtain the decomposition (1) is to split A and B as sum of low-precision floating-point matrices [4]. This can be accomplished by applying the splitting algorithm:

$$A^{(k)} = \mathrm{fl}_{\mathrm{low}}\left(A - \sum_{t=1}^{k-1} A^{(t)}\right), \qquad B^{(\ell)} = \mathrm{fl}_{\mathrm{low}}\left(B - \sum_{t=1}^{\ell-1} B^{(t)}\right), \tag{3}$$

where $\mathrm{fl}_{\mathrm{low}}(X)$ rounds the entries of the input matrix X to precision $\mathcal{F}_{\mathrm{low}}$. In this case, we can set $s_A = s_B = s$, as the final accuracy will be limited by the smaller between s_A and s_B.

If the splitting (1) is obtained using (3), and the approximation $\widetilde{C}$ is computed using (2), then [1]

$$\left|\widetilde{C} - C\right| \leq (2u_{\mathrm{low}}^s + u_{\mathrm{low}}^{2s})|A||B| + (n + s^2 - 1)u_{\mathrm{high}} \sum_{k=1}^{s}\sum_{\ell=1}^{s} \left|A^{(\ell)}\right|\left|B^{(\ell)}\right|. \tag{4}$$

For practical choices of u_{low} and u_{high}, a small value of s, 2 or 3 say, is sufficient to make the two terms in (4) of similar size. Furthermore, not all s^2 products in (2) need be computed, since the magnitude of the elements of $A^{(k)}$ and $B^{(\ell)}$ decreases rapidly as k and ℓ increase. Ignoring all products of the form $A^{(k)}B^{(\ell)}$, for $k + \ell > s + 1$, yields a faster algorithm and an error bounded by

$$\left|\widetilde{C} - C\right| \leq \big((s+1)u_{\mathrm{low}}^s + (n + s^2 - 1)u_{\mathrm{high}}\big)|A||B| + \mathcal{O}\big(u_{\mathrm{high}}u_{\mathrm{low}} + u_{\mathrm{low}}^{s+1}\big), \tag{5}$$

which is just slightly weaker than (4). We evaluated this scheme using double-binary16 ($s = 2$ and $u_{\mathrm{low}} = 2^{-11}$) arithmetic to compute binary32 matrix products ($u_{\mathrm{high}} = 2^{-24}$). We run our implementations of the algorithm described above on NVIDIA GPUs equipped with *tensor cores*—mixed-precision units that compute the product of binary16 matrices using binary32 arithmetic. We identified some cases where, surprisingly, double-binary16 fails to achieve binary32 accuracy: this is the case, for example, if the entries of the matrix are drawn from the interval $(0, 1]$. This phenomenon does not contradict the bounds (4) and (5), and with the help of probabilistic rounding error analysis we showed that a possible cause is the fact that the tensor cores use a custom rounding mode that is less accurate than round-to-nearest [2]. To support this conclusion, we used the CPFloat library [3] to simulate a variant of the tensor cores that uses round-to-nearest throughout, and we showed that switching between rounding modes has indeed the expected effect on accuracy.

The Ozaki scheme for matrix multiplication An alternative technique, which goes back to Rump, Ogita, and Oishi [8], uses a fixed-point representation to recast the matrix product as a sequence of error-free transformations. In the case of matrix multiplication [7], this technique is known as the *Ozaki scheme*. The decomposition (1) is computed using the element-wise algorithm

$$\begin{aligned} a_{ij}^{(k)} &= \mathrm{fl}\left(\mathrm{fl}\left(\alpha_i + \left(a_{ij} - \sum_{t=1}^{k-1} a_{ij}^{(t)}\right)\right) - \alpha_i\right), \qquad \alpha_i = 2^{\max_{1\leq j\leq p}\lceil \log_2 |a_{ij}|\rceil + f(a_{ij})}, \\ b_{ij}^{(\ell)} &= \mathrm{fl}\left(\mathrm{fl}\left(\beta_j + \left(b_{ij} - \sum_{t=1}^{\ell-1} b_{ij}^{(t)}\right)\right) - \beta_j\right), \qquad \beta_j = 2^{\max_{1\leq j\leq p}\lceil \log_2 |b_{ij}|\rceil + f(b_{ij})}, \end{aligned} \tag{6}$$

where $f(x)$ returns 1 if x is a power of two, and 0 otherwise. If the routine computing $A^{(k)}B^{(\ell)}$ in (2) takes matrices with elements in $\mathcal{F}_{\mathrm{low}}$ as input but uses precision u_{high} internally, then the intermediate precision used by the fl operator in (6) can have at most

$$q = \left\lceil (\log_2 u_{\mathrm{high}}^{-1} - \log_2 p)/2 \right\rceil$$

bits, where p is the common dimension of A and B. This choice of q ensures that all multiplications of the form $A^{(k)}B^{(\ell)}$ will be exact.

Implicitly, the algorithm (6) performs two actions. First, it scales all entries in the ith row of A by α_i^{-1}, where α_i is the smallest power of two that is strictly larger, in magnitude, than all elements in the ith row of A; this ensures that $\alpha_i^{-1}a_{ij}$ has magnitude in the interval $[0, 1)$. Next, each $\alpha_i^{-1}a_{ij}$ is interpreted as a fixed-point number, and its representation is divided up into q-bit segments, each assigned to a different low-precision slice $A^{(k)}$. The matrix B is sliced analogously, with the proviso that the algorithm operates by columns rather than by rows. This gives the representation

$$A = \Delta A + \operatorname{diag}(\alpha)\sum_{k=1}^{s_A} 2^{-kq}A^{(k)}, \qquad B = \Delta B + \sum_{\ell=1}^{s_B} 2^{-\ell q}B^{(\ell)}\operatorname{diag}(\beta), \tag{7}$$

where $A^{(1)}$, $A^{(2)}$, ..., $A^{(s_A)}$ and $B^{(1)}$, $B^{(2)}$, ..., $B^{(s_B)}$ are slices of a fixed-point representation of the elements in A and B. Since α_i and β_j depend on the magnitude of the largest entry in row i and column j, respectively, the leading matrices may have zeros in positions corresponding to small elements in A and B.

If s_A and s_B are large enough to guarantee that $\Delta A = 0$ and $\Delta B = 0$ in (7), then algorithm (2) will produce an extremely accurate approximation $\widetilde{C}$, where the only rounding errors are due to the $s_A s_B$ floating-point sums. Mukunoki et al. [5] have specialized this algorithm and have implemented it to obtain binary64 accuracy by using binary16 arithmetic on the NVIDIA tensor cores.

The latest NVIDIA GPUs can perform matrix multiplication even more efficiently using integer arithmetic. The tensor cores of NVIDIA H100 cards, for example, can compute the product of matrices stored in INT8 format (an 8-bit signed format) using 32-bit signed integer arithmetic. Exploiting the fixed-point nature of the Ozaki scheme, Ootomo, Ozaki, and Yokota [6] have therefore developed a method that computes the product of two binary64 matrices using only INT8 matrix multiplications. This initial idea was further refined by Uchino, Ozaki, and Imamura [9], who developed a more accurate and efficient variant of this scheme and gave a first error analysis. For $s_A = s_B = s$, they show that

$$\left|\widetilde{C} - C\right| \leq 4(s+1)k2^{-qs}\alpha\beta^T + (s-1)u_{\text{high}}|A||B|,$$

where u_{high} is the unit roundoff of the floating-point arithmetic used to accumulate the partial matrix products in (2). This result suggests that the algorithm can be inaccurate if s is too small, or if the entries of the matrix are large in absolute value, as this will cause the entries of the vectors α and β to be large.

We propose an alternative error analysis that can be used to inform the choice of the parameters s_A and s_B, which we argue need not be equal in the Ozaki scheme. First, we note that the terms in (7) satisfy

$$|\delta a_{ij}| < \alpha_i u_A, \quad u_A := 2^{-s_A q}, \qquad |\delta b_{ij}| < \beta_j u_B, \quad u_B := 2^{-s_B q}. \tag{8}$$

In error analysis, it is often more informative to bound the relative error. Such bounds arise naturally when using floating-point arithmetic, because floating-point numbers have constant precision. In fixed-point arithmetic, precision is tapered, so bounds like those in (8) are more familiar, but it is still possible to bound $|\delta a_{ij}|$ and $|\delta b_{ij}|$ in terms of $|a_{ij}|$ and $|b_{ij}|$, respectively, since

$$|\delta a_{ij}| \leq \kappa_A u_A |a_{ij}|, \qquad \kappa_A := 2\max_{1\leq i\leq m} \frac{\max\{|a_{ij}| : 1 \leq j \leq p\}}{\min_j\{|a_{ij}| : 1 \leq j \leq p \text{ and } a_{ij} \neq 0\}},$$

$$|\delta b_{ij}| \leq \kappa_B u_B |b_{ij}|, \qquad \kappa_B := 2\max_{1\leq j\leq n} \frac{\max\{|b_{ij}| : 1 \leq i \leq p\}}{\min\{|b_{ij}| : 1 \leq i \leq p \text{ and } b_{ij} \neq 0\}}.$$

Our analysis yields the alternative error bound

$$\left|\widetilde{C} - C\right| \leq \kappa_A u_A + \kappa_B u_B + \kappa_A \kappa_B u_A u_B + \gamma_{s_A s_B - 1}(1 + \kappa_A u_A + \kappa_B u_B + \kappa_A \kappa_B u_A u_B))|A||B|.$$

In other words, the overall error can be substantial if either κ_A or κ_B are large. One can counteract the prominence of these two terms by increasing s_A and s_B, but doing so will negatively impact the performance of the algorithm, which needs to perform $\mathcal{O}(s_A s_B)$ integer matrix multiplications.

The integer-based Ozaki scheme can be much faster than traditional high-precision alternatives, but our analysis suggests that it can also be significantly less accurate, depending on the dynamic range of the entries of A and B. The value of the parameters s_A and s_B required to meet a specific accuracy target can be determined by examining κ_A and κ_B, which are inexpensive to compute. For a given choice of s_A and s_B, we can estimate the runtime of the scheme, and we can opt for a traditional high-precision routine when the latter is expected to be faster.

References

[1] M. Fasi, N. J. Higham, F. Lopez, T. Mary, and M. Mikaitis. Matrix multiplication in multiword arithmetic: Error analysis and application to GPU tensor cores. *SIAM J. Sci. Comput.*, 45(1): C1–C19, 2023.

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

[3] M. Fasi and M. Mikaitis. CPFloat: A C library for simulating low-precision arithmetic. *ACM Trans. Math. Software*, 49(2):1–32, 2023.

[4] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter. NVIDIA tensor core programmability, performance & precision. In *Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018.

[5] D. Mukunoki, K. Ozaki, T. Ogita, and T. Imamura. DGEMM using tensor cores, and its accurate and reproducible versions. In *High Performance Computing*, P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, editors, Springer-Verlag, 2020, page 230–248.

[6] H. Ootomo, K. Ozaki, and R. Yokota. DGEMM on integer matrix multiplication unit. *Int. J. High Performance Computing Applications*, 38(4):297–313, 2024.

[7] K. Ozaki, T. Ogita, S. Oishi, and S. M. Rump. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. *Numer. Algorithms*, 59(1):95–118, 2012.

[8] S. M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part I: Faithful rounding. *SIAM J. Sci. Comput.*, 31(1):189–224, 2008.

[9] Y. Uchino, K. Ozaki, and T. Imamura. Performance enhancement of the Ozaki scheme on integer matrix multiplication unit. Technical report, September 2024. arXiv:2409.13313 [cs.DC].