Analysis of Floating-Point Matrix Multiplication Computed via Integer Arithmetic

Mantas Mikaitis

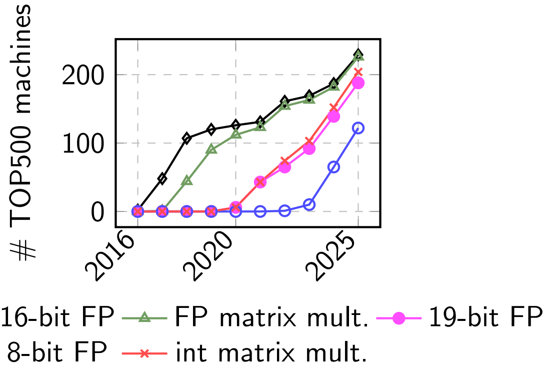
Joint work with A. Abdelfattah, J. Dongarra, M. Fasi, and F. Tisseur

School of Computer Science, University of Leeds, Leeds, UK

SIAM Conference on Parallel Processing for Scientific Computing (PP26)

4 March, 2026

Fast low-precision matrix mult on the TOP500



Devices counted: P100, V100, A100, H100, MI210, MI250X, MI300X, Intel Data Center GPU, from `https://www.top500.org`.

NVIDIA HGX B200 throughputs (OPS/s)
int8 (4.5×10^{15}) fp16 (2.2×10^{15}) fp64 (0.04×10^{15}).

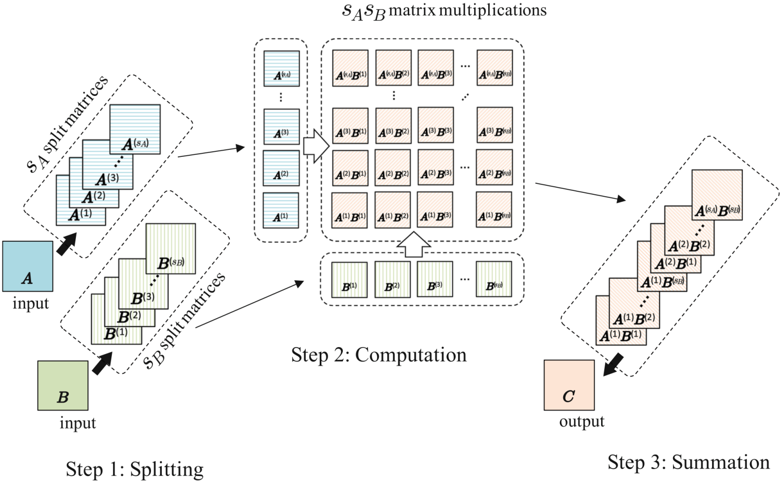
What is the "Ozaki scheme"?

In a nutshell

Split matrices into several "slices", compute many exact matrix multiplies, sum the products → more accurate than one $m \times k \times n$ product.

- [Ozaki, Ogita, Oishi, Rump, 2012] accurate fp64 mat mul by splitting the inputs and performing several exact fp64 mat mul + sum: $AB = \sum \mathrm{fl}(A^{(r)}B^{(s)})$.
- [Ozaki, Ogita, Oishi, Rump, 2013] improve the algorithm by exploiting *last non-zero bit* to reduce the number of splits/products.
- [Mukunoki, Ozaki, Ogita, Imamura, 2020] used 16-bit input, 32-bit output FP tensor cores for the above.
- [Ootomo, Ozaki, Yokota, 2024] used 8-bit input, 32-bit output **integer** tensor cores.
- [Mukunoki, 2026] uses an 8-bit FP tensor core.
- [Schwarz et al., 2026] tackles optimal slicing to 8 bit INT.

Mukunoki et al. 2020 illustration of the Ozaki scheme



FP64 $\to$ int8-int32 tensor core $\to$ FP64

Using integer tensor cores

A hardware FP64 FMA is multiplying 53-bit integers and adding 106-bit integers—FP64 fundamentally is computed through integers.

Similarly, **we can use integer tensor cores for FP64 mat-mul computations**.

Take $A \in \mathbb{F}^{n \times k}$ and $B \in \mathbb{F}^{k \times n}$. Goal: approximate AB by integer tensor cores. Typically $\mathbb{F}$ is double prec.

Assume A and B have no infinities, NaNs, or -0 and the computation cannot produce inf and NaN.

FP64 $\rightarrow$ int8-int32 tensor core $\rightarrow$ FP64

Step 1: Scale factors

1. For each row i of A, compute a scale factor

$$\alpha_i = 2^{\lfloor \log_2 M_i \rfloor} \times 2, \qquad M_i = \max_{1 \le j \le k} |a_{ij}|, \qquad 1 \le i \le m,$$

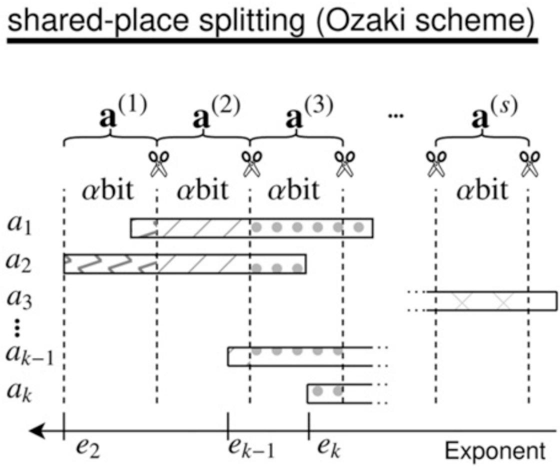
2. This guarantees each element scaled down by α_i is in $[0, 1)$.
3. And $0.5 \le M_i / \alpha_i < 1$.
4. Similar for B, **column-wise**.

Block floating-point arithmetic

Note that this is block floating-point representation of [Wilkinson, 1963]. **Ozaki scheme: each row of A and colum of B is a block with its own scale factor.**

Utilising 8-bit integer tensor cores [Ootomo et al. 2024]

Step 2: Slicing



Scaling and slicing: example on small vectors

$$A = \begin{bmatrix} 1.5625 & 8 & -3.6875 \end{bmatrix}, \qquad B = \begin{bmatrix} 1.3828125 \\ -7.625 \\ 3.625 \end{bmatrix}$$

Scale factors $\alpha = 2^4$ and $\beta = 2^3$.

Example set up: FP precision 8 bits, 4 slices, integer: 3 bits and a sign

$$\underbrace{\begin{bmatrix} 2^0 \cdot 1.1001000 \\ 2^3 \cdot 1.0000000 \\ -2^1 \cdot 1.1101100 \end{bmatrix}}_{A^T} \Rightarrow 2^4 \cdot \underbrace{\begin{bmatrix} \not{0}.\underline{000}\ \underline{110}\ \underline{010}\ \underline{000} \\ \not{0}.\underline{100}\ \underline{000}\ \underline{000}\ \underline{000} \\ -\not{0}.\underline{001}\ \underline{110}\ \underline{110}\ \underline{000} \end{bmatrix}}_{\text{Block fixed-point}} \Rightarrow 2^1 \cdot \underbrace{\begin{bmatrix} 000 \\ 100 \\ -001 \end{bmatrix}}_{A^T_{(1)}} + 2^{-2} \cdot \underbrace{\begin{bmatrix} 110 \\ 000 \\ -110 \end{bmatrix}}_{A^T_{(2)}} + 2^{-5} \cdot \underbrace{\begin{bmatrix} 010 \\ 000 \\ -110 \end{bmatrix}}_{A^T_{(3)}} + 2^{-8} \cdot \underbrace{\begin{bmatrix} 000 \\ 000 \\ 000 \end{bmatrix}}_{A^T_{(4)}}$$

$$\underbrace{\begin{bmatrix} 2^0 \cdot 1.0110001 \\ -2^2 \cdot 1.1110100 \\ 2^1 \cdot 1.1101000 \end{bmatrix}}_{B} \Rightarrow 2^3 \cdot \underbrace{\begin{bmatrix} \not{0}.\underline{001}\ \underline{011}\ \underline{000}\ \underline{100} \\ -\not{0}.\underline{111}\ \underline{101}\ \underline{000}\ \underline{000} \\ \not{0}.\underline{011}\ \underline{101}\ \underline{000}\ \underline{000} \end{bmatrix}}_{\text{Block fixed-point}} \Rightarrow 2^0 \cdot \underbrace{\begin{bmatrix} 001 \\ -111 \\ 011 \end{bmatrix}}_{B^{(1)}} + 2^{-3} \cdot \underbrace{\begin{bmatrix} 011 \\ -101 \\ 101 \end{bmatrix}}_{B^{(2)}} + 2^{-6} \cdot \underbrace{\begin{bmatrix} 000 \\ 000 \\ 000 \end{bmatrix}}_{B^{(3)}} + 2^{-9} \cdot \underbrace{\begin{bmatrix} 100 \\ 000 \\ 000 \end{bmatrix}}_{B^{(4)}}$$

FP64 $\to$ int8-int32 tensor core $\to$ FP64

We now have:

$$A \approx \mathrm{diag}(\alpha) \sum_{\ell=1}^{s_A} 2^{-\ell t} A_{(\ell)}, \qquad B \approx \sum_{h=1}^{s_B} 2^{-ht} B^{(h)} \,\mathrm{diag}(\beta).$$

Here:

- s_A and s_B are the number of slices of BFP A and B.
- $A_{(\ell)}$ is the ℓ-th slices of A.
- $B^{(h)}$ is the h-th slices of B.
- t is the precision of integers in $A_{(\ell)}$ and $B^{(h)}$.

FP64 $\rightarrow$ int8-int32 tensor core $\rightarrow$ FP64

Step 3: Integer products

$$C = AB \approx \left(\mathrm{diag}(\alpha)\sum_{\ell=1}^{s_A} 2^{-\ell t} A_{(\ell)}\right)\left(\sum_{h=1}^{s_B} 2^{-ht} B^{(h)}\,\mathrm{diag}(\beta)\right)$$

$$= \alpha\beta^T \circ \sum_{\ell=1}^{s_A}\sum_{h=1}^{s_B} 2^{-(\ell+h)t}\,\underline{A_{(\ell)}B^{(h)}}.$$

Step 4: Summation and rescaling (back in FP64)

$$C \approx \underline{\alpha\beta^T \circ \sum_{\ell=1}^{s_A}\sum_{h=1}^{s_B} 2^{-(\ell+h)t}} A_{(\ell)}B^{(h)}.$$

Example: products and their positions in BFP

$A_{(1)}B^{(1)}$	$2^{1}\ \cdot$	-00011111
$A_{(1)}B^{(2)}$	$2^{-2}\ \cdot$	-00011001
$A_{(2)}B^{(1)}$	$2^{-2}\ \cdot$	-00001100
$A_{(1)}B^{(3)}$	$2^{-5}\ \cdot$	00000000
$A_{(2)}B^{(2)}$	$2^{-5}\ \cdot$	-00001100
$A_{(3)}B^{(1)}$	$2^{-5}\ \cdot$	-00010000
$A_{(1)}B^{(4)}$	$2^{-8}\ \cdot$	00000000
$A_{(2)}B^{(3)}$	$2^{-8}\ \cdot$	00000000
$A_{(3)}B^{(2)}$	$2^{-8}\ \cdot$	-00011000
$A_{(4)}B^{(1)}$	$2^{-8}\ \cdot$	00000000
$A_{(2)}B^{(4)}$	$2^{-11}\ \cdot$	00011000
$A_{(3)}B^{(3)}$	$2^{-11}\ \cdot$	00000000
$A_{(4)}B^{(2)}$	$2^{-11}\ \cdot$	00000000
$A_{(3)}B^{(4)}$	$2^{-14}\ \cdot$	00001000
$A_{(4)}B^{(3)}$	$2^{-14}\ \cdot$	00000000
$A_{(4)}B^{(4)}$	$2^{-17}\ \cdot$	00000000
AB	$2^{-17}\ \cdot$	$-00100100000110100111000000$

Rounding error analysis 1: slicing

Error induced in slicing

Slicing causes the error in the virtual block floating-point precision, by taking a set number of slices s_A and s_B. The precision to the right of the binary point in BFP is $s_A t$ bits—**the more slices we take, the higher the precision of the input is**. However, BFP causes errors relative to the max value in a block.

For slicing A (analogous for B) we have the **absolute error**:

$$A = \Delta A + \operatorname{diag}(\alpha) \sum_{\ell=1}^{s_A} 2^{-\ell t} A_{(\ell)}, \qquad |\delta a_{ij}| < \alpha_i 2^{-s_A t}, \tag{1}$$

and the relative error (component wise)

$$\frac{|\delta a_{ij}|}{|a_{ij}|} < \frac{\alpha_i}{|a_{ij}|} 2^{-s_A t} \le \frac{2 \max_j |a_{ij}|}{\min_j |a_{ij}|} 2^{-s_A t} \le \kappa_A 2^{-s_A t}, \quad \kappa_A := 2 \max_i \frac{\max_j |a_{ij}|}{\min_j |a_{ij}|}$$

Rounding error analysis 2: product summation

Let $\widehat{C}$ be an FP approximation for $\alpha\beta^T \circ \sum_{\ell=1}^{s_A} \sum_{h=1}^{s_B} 2^{-(\ell+h)t} A_{(\ell)} B^{(h)}$.

We arrived at the bound for $\widehat{C}$:

$$|\widehat{C} - C| \lesssim \left(2^{-s_A t}\kappa_A + 2^{-s_B t}\kappa_B + (s_A s_B - 1)2^{-p}\right)|A||B|.$$

where p is the precision of the FP format used for accumulation (e.g. FP64).

Key take aways

- Large error if κ get large (large ranges of A and/or B).
- Taking more slices can help: inc. s_A or s_B. But more expensive.
- If $s_A t$ and $s_B t$ are large enough, 2^{-p} (accumulation error) dominates.

Experiment 1: 2-element vectors

As a minimal example, we consider the computation of the inner product $a^T b$, where

$$a = \begin{bmatrix} 2^{-\varphi} x \\ 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 2^{\varphi} y \\ 1 \end{bmatrix}, \qquad x, y \sim \mathcal{N}(0, 1).$$

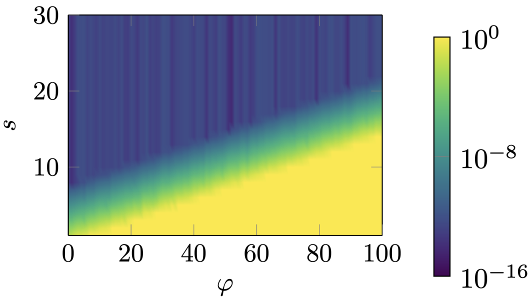
We plot:

$$\frac{|\widehat{c} - c|}{|c|},$$

where

- $\widehat{c}$ is computed with Ozaki scheme on 8-bit to 32-bit int tensor core.
- c is a reference solution computed using the MATLAB Symbolic Toolbox with 32 decimal digits of accuracy.

Experiment 1: 2-element vectors



The x-axis denotes the number of slices and the y-axis controls the wideness of the gap between the min and max exponents.

Key takeaways

- For $\varphi = 0$ about 7 slices are sufficient for FP64 accur.
- For $\varphi = 100$ about 20 slices are needed.

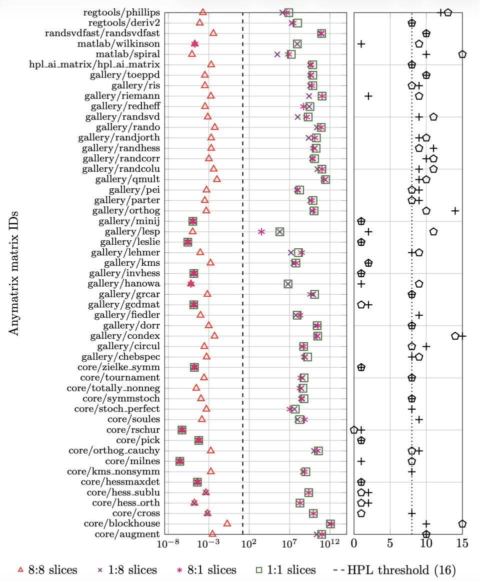
Experiment 2: Solving $Ax = b$ with block LU

- Take $A \in \mathbb{R}^{500 \times 500}$, one of the matrices from the `Anymatrix` collection [Higham and Mikaitis, 2021].
- $b \in \mathbb{R}^{500}$ with entries $[0, 1)$ from unif. distr.
- Use block LU with partial pivoting and block size 10.
- All mat-mul computed with the Ozaki scheme inside the LU iteration.

HPL error measure (used for TOP500)

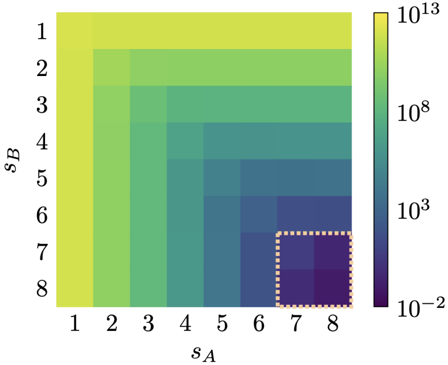
Report: $\frac{\|A\widehat{x}-b\|_\infty}{2u(\|A\|_\infty\|\widehat{x}\|_\infty+\|b\|_\infty)n}$ (HPL pass threshold < 16). Here $u = 2^{-53}$ for FP64.

Experiment 2: Solving $Ax = b$ with block LU



Experiment 2: Solving $Ax = b$ with block LU

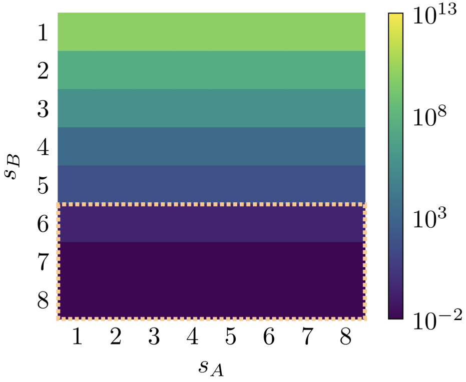
Error $\frac{\|A\widehat{x}-b\|_\infty}{2u(\|A\|_\infty\|\widehat{x}\|_\infty+\|b\|_\infty)n}$ in solving $Ax = b$ for A an Anymatrix matrix `core/blockhouse` with $n = 500$.



HPL threshold 16 passed for configs in dotted square.

Experiment 2: Solving $Ax = b$ with block LU

Error $\frac{\|A\widehat{x}-b\|_\infty}{2u(\|A\|_\infty\|\widehat{x}\|_\infty+\|b\|_\infty)n}$ in solving $Ax = b$ for A an Anymatrix matrix `core/cross` with $n = 500$.



HPL threshold 16 passed for configs in dotted square.

Summary

- Low-precision matrix multipliers are being used for general-purpose computation.
- We have
  - Built error analysis to check some intuitive predictions about the Ozaki scheme.
  - Experimentation to show where the scheme works or not.
  - Unbalanced slicing: future work.

Preprint

A. bdelfattah, J. Dongarra, M. Fasi, M. Mikaitis, and F. Tisseur. *Analysis of Floating-Point Matrix Multiplication Computed via Integer Arithmetic* . **Preprint, arXiv:2506.11277 [math.NA]**. Jun. 2025.

Slides at `http://mmikaitis.github.io/talks`

References I

J. Wilkinson
Rounding Errors in Algebraic Processes
Her Majesty's Stationery Office, 1963 (Reprinted by SIAM in 2023)

K. Ozaki, T. Ogita, S. Oishi, S. M. Rump
Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications
Numer. Alg., 59. 2012.

D. Mukunoki, K. Ozaki, T. Ogita, T. Imamura
DGEMM Using Tensor Cores, and Its Accurate and Reproducible Versions
LNCS 12151. 2020.

N. J. Higham and M. Mikaitis
Anymatrix: An Extensible MATLAB Matrix Collection
Numer. Algorithms, 90. 2021.

References II

H. Ootomo, K. Ozaki and R. Yokota
DGEMM on integer matrix multiplication unit
Int. J. High Perf. Comput. Appl., 38. 2024.

D. Mukunoki
DGEMM using FP64 Arithmetic Emulation and FP8 Tensor Cores with Ozaki Scheme
SCA/HPCAsia 2026 Workshops, Osaka, Japan

A. Schwarz, A. Anders, C. Brower, H. Bayraktar, J. Gunnels, and K. Clark
Guaranteed DGEMM Accuracy While Using Reduced Precision Tensor Cores Through Extensions of the Ozaki Scheme
SCA/HPCAsia 2026 Workshops, Osaka, Japan

(Extra slides) Experiment 3: matrices with increasing dyn. range

We take the matrices $A \in \mathbb{R}^{10\times k}$ and $B \in \mathbb{R}^{k\times 10}$ with

$$A_{ij} = a_{ij}e^{\varphi x_{ij}}, \quad B_{ij} = b_{ij}e^{\varphi y_{ij}}, \quad a_{ij}, b_{ij} \sim \mathcal{U}(-0.5, 0.5), \quad x_{ij}, y_{ij} \sim \mathcal{N}(0, 1),$$

The parameter φ allows us to control the exponent range of the elements in A and B.

We plot the max forw. rel. error (following [Ootomo, Ozaki, Yokota, 2024]):

$$\max_{i,j} \frac{|\widehat{c}_{ij} - c_{ij}|}{|c_{ij}|},$$

(Extra slides) Experiment 3: matrices with increasing dyn. range