Ozaki Scheme I: Emulation method for Matrix Multiplication

Katsuhisa Ozaki
Department of Mathematical Sciences
Shibaura Institute of Technology, Japan

The 4th FugakuNEXT Application Seminar
Aug. 20, 2025 (online)

Katsuhisa Ozaki

friendly tigers In Thailand

- Professor,
  Department of
  Mathematical Sciences,
  Shibaura Institute of
  Technology, Japan

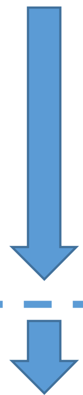
- Research Interests:
  - Reliable computing
  - Rounding error analysis
  - Numerical linear Algebra

Agenda

Today, we will not review accurate dot product algorithms and the DD approach.

- Ozaki Scheme I:
  Error-free Transformation of Matrix Multiplication

- Basics (the original method) 2012
- Using FP16 Tensor Cores 2020
- Using INT8 Tensor Cores and recent progress 2025



- Ozaki Scheme II 2025

Ozaki Scheme I

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
Numerical Algorithms, 59, 95-–118 (2012).

D. Mukunoki, **K. Ozaki**, T. Ogita, T. Imamura:
DGEMM Using **Tensor Cores**, and Its Accurate and
Reproducible Versions, Lecture Notes in Computer
Science, 12151, 2020, 230-248.

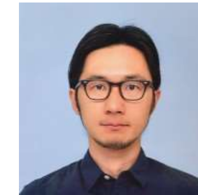
H. Ootomo, **K. Ozaki**, R. Yokota:
DGEMM on integer matrix multiplication unit,
The International Journal of High Performance
Computing Applications,38 (2024), 297--313.

Y. Uchino, **K. Ozaki**, T. Imamura:
Performance Enhancement of the Ozaki Scheme on Integer
Matrix Multiplication Unit,
International Journal of High Performance Computing
Applications, 39:3 (2025), 462--476.

Ogita

Oishi

Rump


Mukunoki

Ogita

Imamura

Ootomo

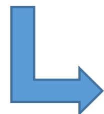
Yokota

Uchino

Imamura

Floating-point Numbers

- IEEE 754 binary formats:
  - FP16 ：1 bit for sign, 5 bit for exponent, **10** bit for significand
  - FP32 ：1 bit for sign, 8 bit for exponent, **23** bit for significand
  - FP64 ：1 bit for sign, 11 bit for exponent, **52** bit for significand
  - **FP128** ：1 bit for sign, 15 bit for exponent, **112** bit for significand
    (**+1 hidden bit for normal numbers**)


No hardware support

- Non IEEE 754 binary format
  - FP4, FP6, FP8, TF32, BF16, …

Error-free Transformation of Mat. Mul.

- For two matrices A and B (ex., represented in FP64),
- We obtain
- $A = A_1 + A_2 + \cdots + A_k$, and
- $B = B_1 + B_2 + \cdots + B_l$.
- All elements in A_i and B_j are represented by FP64.
- We can compute $A_i * B_j$, $1 \le i \le k, 1 \le j \le l$ **without rounding error.**
- $AB = (A_1 + A_2 + \cdots + A_k)(B_1 + B_2 + \cdots + B_l)$

$$= A_1B_1 + A_1B_2 + A_2B_1 + \cdots + A_kB_l$$

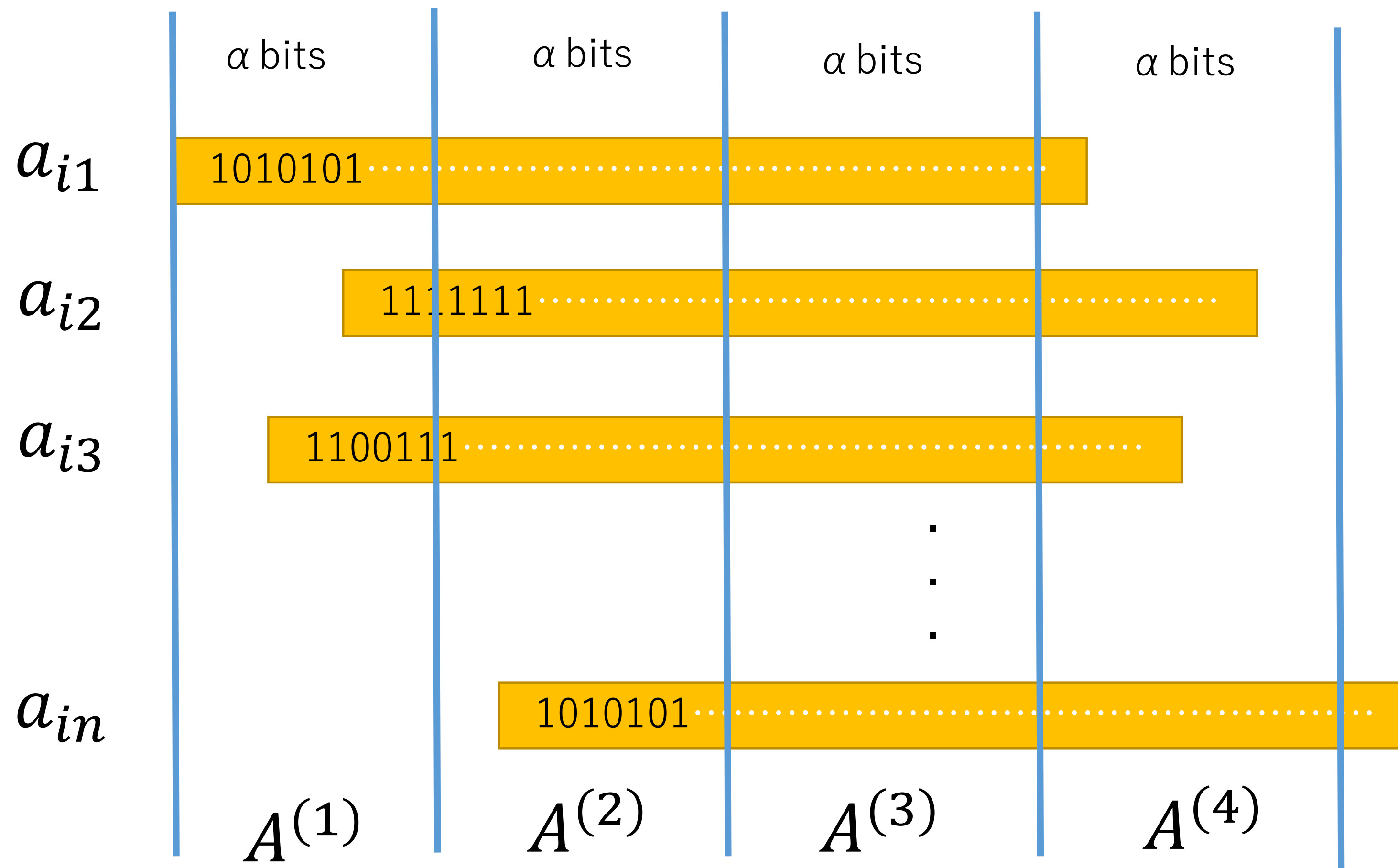
← **gemm**

k slices,
l slices

AB is transformed into kl floating-point matrices.

Assume divide and conquer methods like Strassen's and Winograd's are not applied.

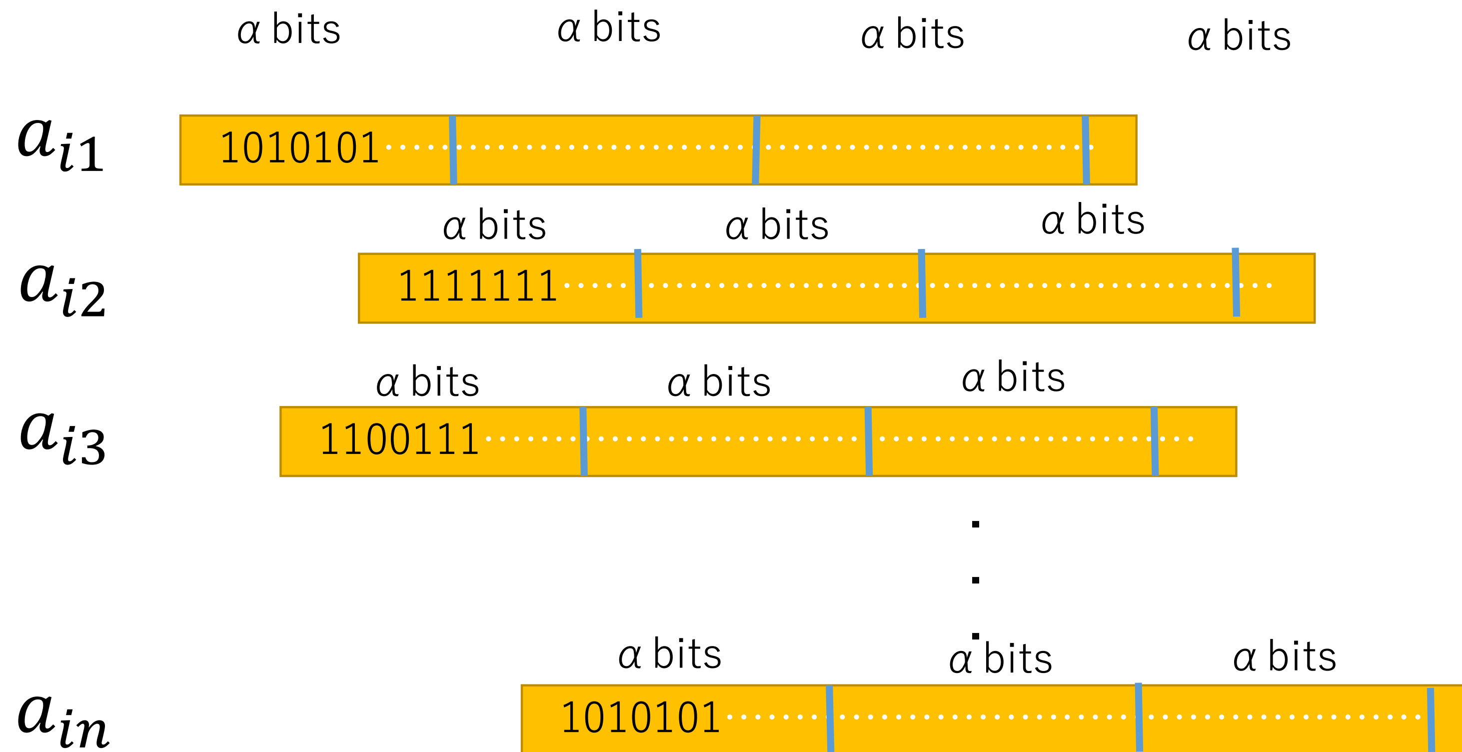
Image



For FP64:
α =floor((53-log2(n))/2) bits

For FP32:
α =floor((24-log2(n))/2) bits

Incorrect Image



For FP64:
α =floor((53-log2(n))/2) bits

For FP32:
α =floor((24-log2(n))/2) bits

Image by bits (Ex. FP32)

- A is an m-by-**n** matrix, and B is an **n**-by-p matrix.
- $a_{ij}^{(k)}$ and $b_{ij}^{(k)}$ have maximally ceil((24-log2(n))/2) bit.

- For example, n = 1,000 ($\leqq 2^{10}$),
- $a_{ik}^{(*)}$ and $b_{kj}^{(*)}$ have maximally 7 bit.
- $a_{ik}^{(*)} * b_{kj}^{(*)}$ has maximally 14 bit.
- $\sum_{k=1}^{n} a_{ik}^{(*)} b_{kj}^{(*)}$ has maximally 24 bit ($\leqq$ **24 bit**).

Image by bits (Ex. FP32)

- A is an m-by-**n** matrix, and B is an **n**-by-p matrix.
- $a_{ij}^{(k)}$ and $b_{ij}^{(k)}$ have maximally ceil((24-log2(n))/2) bit.

- For example, n = 20,000 (≦ 2^{15}),
- $a_{ik}^{(*)}$ and $b_{kj}^{(*)}$ have maximally 4 bit.
- $a_{ik}^{(*)} * b_{kj}^{(*)}$ has maximally 8 bit.
- $\sum_{k=1}^{n} a_{ik}^{(*)} b_{kj}^{(*)}$ has maximally 23 bit (≦ **24 bit**).

Image by bits (Ex. FP64)

- A is an m-by-**n** matrix, and B is an **n**-by-p matrix.
- $a_{ij}^{(*)}$ and $b_{ij}^{(*)}$ have maximally floor((53-log2(n))/2) bit.

- For example, n = 1,000 ($\leqq 2^{10}$),
- $a_{ik}^{(*)}$ and $b_{kj}^{(*)}$ have maximally 21 bit.
- $a_{ik}^{(*)} * b_{kj}^{(*)}$ has maximally 42 bit.
- $\sum_{k=1}^{n} a_{ik}^{(*)} b_{kj}^{(*)}$ has maximally 52 bit ($\leqq$ **53 bit**).

Image by bits (Ex. FP64)

- A is an m-by-**n** matrix, and B is an **n**-by-p matrix.
- $a_{ij}^{(*)}$ and $b_{ij}^{(*)}$ have maximally floor((53-log2(n))/2) bit.

- For example, n = 10,000 ($\leqq 2^{14}$),
- $a_{ik}^{(*)}$ and $b_{kj}^{(*)}$ have maximally 19 bit.
- $a_{ik}^{(*)} * b_{kj}^{(*)}$ has maximally 38 bit.
- $\sum_{k=1}^{n} a_{ik}^{(*)} b_{kj}^{(*)}$ has maximally 52 bit ($\leqq$ **53 bit**).

Drawbacks

We **never** think this technique is **perfectly useful**.

- Memory consumption
- Big difference of the magnitude among elements?

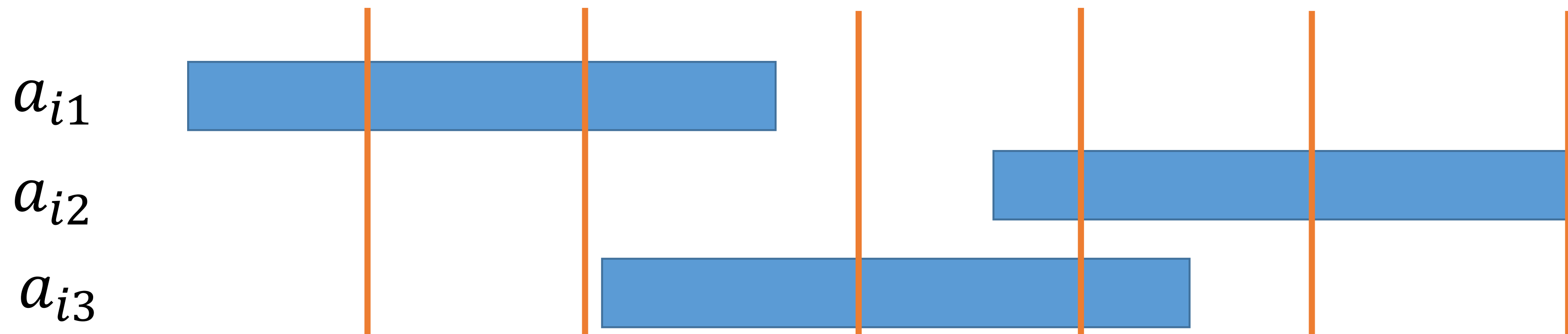
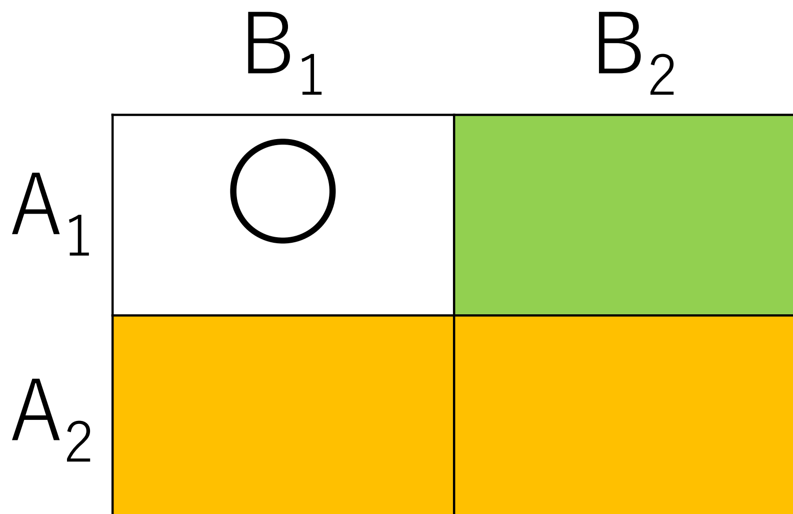
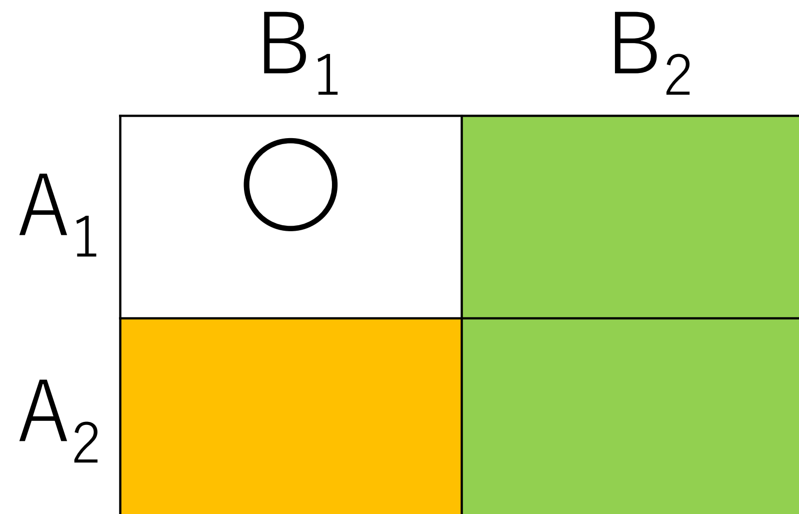


Image of 2 slices





$A_1B_1+A_1B_2+A_2B$

$A_1B_1+A_2B_1+AB_2$

Image of 3 slices

	B_1	B_2	B_3
A_1	○	○	
A_2	○		
A_3			

$A_1B_1+A_1B_2+A_2B_1+A_1B_3+A_2(B_2+B_3)+A_3B$

	B_1	B_2	B_3
A_1	○	○	
A_2	○	○	
A_3			

$A_1B_1+A_1B_2+A_2B_1+A_2B_2+(A_1+A_2)B_3+A_3B$

Demo

Age of low-precision matrix engine

TFLOPS / TOPS for data center class GPUs

GPU	INT8 TC	FP16 TC	FP32	FP64 TC	FP64
B200	4500	2250	75	37	37
GH200	1979	989	67	67	34
A100	624	312	19.5	19.5	9.7

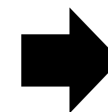
new

TFLOPS / TOPS for consumer grade GPUs

GPU	INT8 TC	FP16 TC	FP32	FP64 TC	FP64
RTX5090	**1674**	419	104.8	—	**1.64**
RTX4090	**660**	165	82.6	—	**1.29**

new

- INT8TC:FP64 = 121:1
- INT8TC:FP64 = 1020:1
- INT8TC:FP64 = 30:1
- INT8TC:FP64 = 512:1



FP64 Emulation

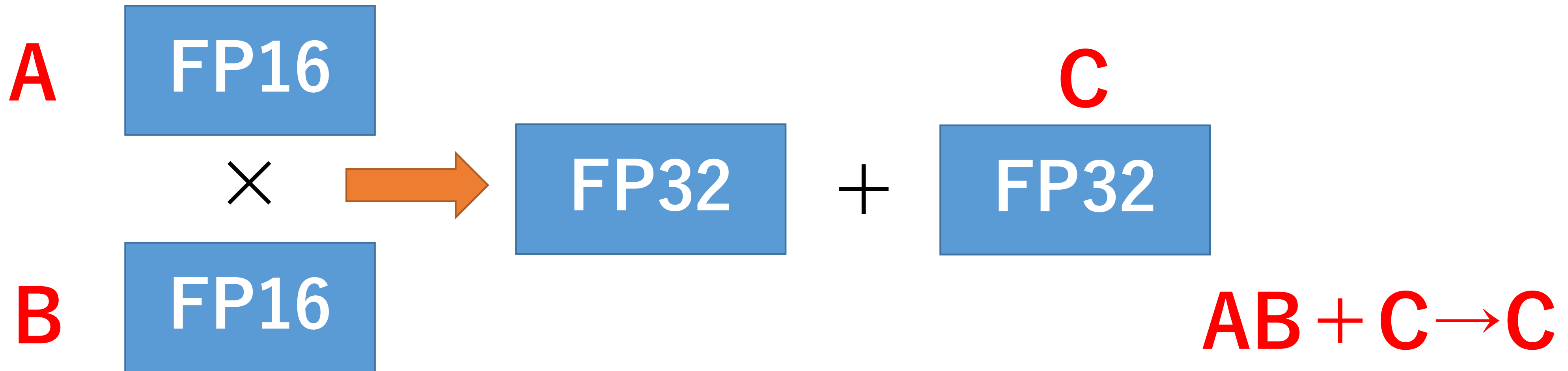
Tensor Cores

- FP16 TC
  - input: FP16 (11 bits significand), output: FP16 (11 bits significand)
  - input: FP16 (11 bits significand), output: FP32 (24 bits significand)

  ※ including the hidden 1 bit

- INT8 TC
  - input : INT8 (like 7 bits significand)
    output: INT32 (like 31 bits significand)

FP16 Tensor Cores



For example, n = 1,000,

$a_{ik}^{(*)}$ and $b_{kj}^{(*)}$ have maximally **7 bits.**

FP16 is sufficient

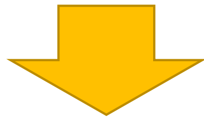
$a_{ik}^{(*)} * b_{kj}^{(*)}$ has maximally 14 bits.

$\sum_{k=1}^{n} a_{ik}^{(*)} b_{kj}^{(*)}$ has maximally **24 bits.**

FP32 is sufficient

INT8 Tensor Cores

- Input: 8 bits integer (like 7 bits significand)



- Output: **32 bits** integer (like 31 bits significand)

$a_{ik}^{(*)}$ and $b_{kj}^{(*)}$ have maximally **7** bit.

$a_{ik}^{(*)} * b_{kj}^{(*)}$ has maximally 14 bit.

$\sum_{k=1}^{n} a_{ik}^{(*)} b_{kj}^{(*)}$ has maximally 14+$\log_2 n$ bit (≦31 bit).

$n = 2^{17} \rightarrow 31$bit

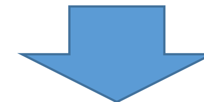
Reproducible

	B_1	B_2	B_3	B_4	B_5	B_6
A_1	○	○	○	○	○	○
A_2	○	○	○	○	○	
A_3	○	○	○	○		
A_4	○	○	○			
A_5	○	○				
A_6	○					

○ : error-free

$$A \fallingdotseq A_1 + \cdots + A_6$$
$$B \fallingdotseq B_1 + \cdots + B_6$$

We fix a computation order of the sum



Reproducible

FP16 or INT8 Tensor Cores

① Splitting

$$A \fallingdotseq A_1 + A_2 + \cdots + A_k, \qquad B \fallingdotseq B_1 + B_2 + \cdots + B_k$$

Scaling and cast are necessary.

② Computing matrix products using **FP16 TC or INT8 TC**

after computing matrix products, inverse scaling and cast are necessary

③ Computing the sum using **FP64**

INT8 TC v.s. FP16 Tensor Cores (N=5,000)

The maximum relative error

Slices	Muls	FP16 TC	INT8 TC
FP32		1.1027e+00	1.1027e+00
2	3	3.1601e+05	5.1525e+03
3	6	9.1655e+03	5.5134e+01
4	10	6.6406e+01	4.3142e-01
5	15	1.7326e+00	3.0162e-03
6	21	1.9342e-02	1.9846e-05
7	28	6.6692e-04	1.2183e-07
8	36	8.1598e-06	6.5806e-10
9	45	9.6758e-08	3.6994e-12
FP64		3.4942e-09	3.4942e-09

INT8 TC works **4 times faster** than FP16 TC on Consumer grade GPU.
INT8 TC works **2 times faster** than FP16 TC on Data center class GPU.

For B300, …

Demo

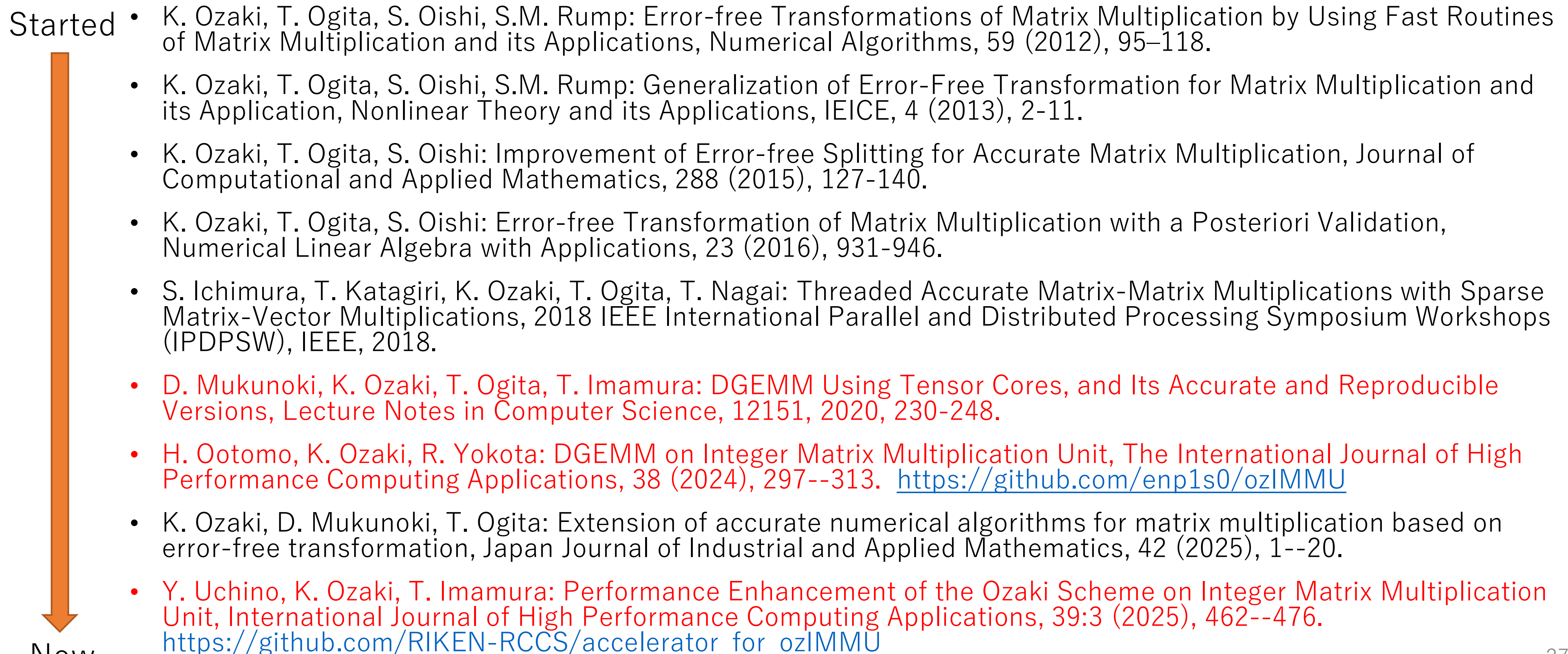
News about Ozaki Scheme I

- S. Jones: CUDA: New Features and Beyond, GTC24
  Cholesky Factorization Performance:
  Based on the method of **Ootomo et. al.** to apply integer tensor cores to DGEMM, L40s achieves a **7x performance increase** over native FP64.
- H. Bayraktar, J. Gunnels: Let's measure what matters when benchmarking supercomputers: **science per Watt,** ISC'24
  - **FP64 Emulation Using INT8 Tensor Cores**
    - **Emulated** vs. Native DGEMM & HPL
    - Perf/Watt Ratios of **Emulated** vs. Native DGEMM & HPL

News about Ozaki Scheme I

- D. Eadline, Have You Heard About the Ozaki Scheme? **You Will**, HPC Wire, 2025.04.17.
- J. Dongarra, J. Gunnels, H. Bayraktar, A. Haidar, D. Ernst, Hardware Trends Impacting Floating-Point Computations In Scientific Applications, arXiv:2411.12090, 2025.
  - High Performance LINPACK (HPL) using the Ozaki Scheme I has been reported to achieve a **2.3× speedup** and a **1.6× improvement** in power efficiency compared to using DGEMM.
- H. Bayraktar:
  Precision Redefined: Unlocking and Delivering the Full Power of Modern GPUs for Scientific Computing,
  The Platform for Advanced Scientific Computing (PASC) Conference, Brugg, Switzerland, June, 2025
  - Double-precision (FP64) with Ozaki-I method **will be released second half of 2025**

Related Papers and Codes



GPU topics

Conclusion

- Overview of Ozaki-scheme I
- Rounding error analysis of Ozaki Scheme I:
  - Y. Uchino, K. Ozaki, T. Imamura:
    Performance Enhancement of the Ozaki Scheme on Integer Matrix Multiplication Unit,
    International Journal of High Performance Computing Applications, 39:3 (2025), 462--476.
  - A. Abdelfattah, J. Dongarra, M. Fasi, M. Mikaitis, F. Tisseur:
    Analysis of Floating-Point Matrix Multiplication Computed via Integer Arithmetic,
    arXiv:2506.11277, 2025.
- Recent topic
  - D. Mukunoki:
    DGEMM without FP64 Arithmetic -- Using FP64 Emulation and FP8 Tensor Cores with Ozaki Scheme,
    arXiv:2508.00441

Hot Topic and Future work

- Ozaki Scheme II based on Chinese Reminder Theorem
  - K. Ozaki, Y. Uchino, T. Imamura:
    Ozaki Scheme II: A GEMM-oriented emulation of floating-point matrix multiplication using an integer modular technique,
    arXiv:2504.08009.
  - Y. Uchino, K. Ozaki, T. Imamura:
    High-Performance and Power-Efficient Emulation of Matrix Multiplication using INT8 Matrix Engines,
    arXiv:2508.03984.
- Applications: Problems in Numerical Linear Algebra

Thank you very much for your attention!