SC26 Paper
Chicago, IL, Nov 18, 2026

Double-Precision Matrix Multiplication Emulation via Ozaki-II Scheme with FP8 Quantization

Yuki Uchino[†]
Katsuhisa Ozaki[⋆]
Toshiyuki Imamura[†]

[†]RIKEN Center for Computational Science
[⋆]Shibaura Institute of Technology

Background

Modern GPUs prioritize low-prec. arithmetic.

Gap

For Scientific computing, we still need FP64.

GPU performance in T(FL)OP/s (dense)

	GH200	GB200	GB300	Rubin
FP64	67	40	1.4	33
FP8	1979	5000	5000	17500
INT8	1979	5000	**166**	**250**

We proposed FP64 emulation using low-precision matrix engines.

- Ozaki-I scheme [Ozaki et al., 2012]
- Ozaki-II scheme [Ozaki et al. 2026]

Both schemes are well suited to INT8.

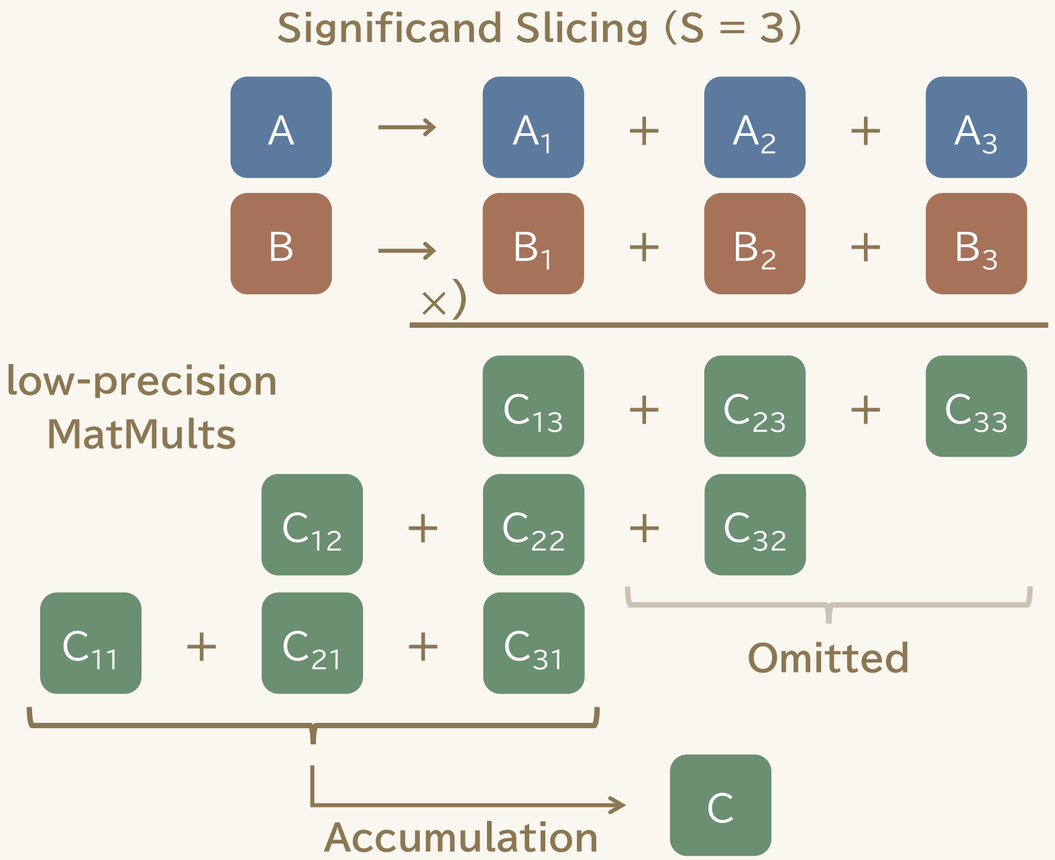
- INT8 is receiving less architectural emphasis in recent NVIDIA GPUs.
- Limited INT8 throughput is not unique to NVIDIA GPUs.
  - E.g., on Tenstorrent Blackhole, FP8 throughput is 4x that of INT8.

→ We propose FP64 emulation on **FP8 matrix engines.**

Ozaki Schemes

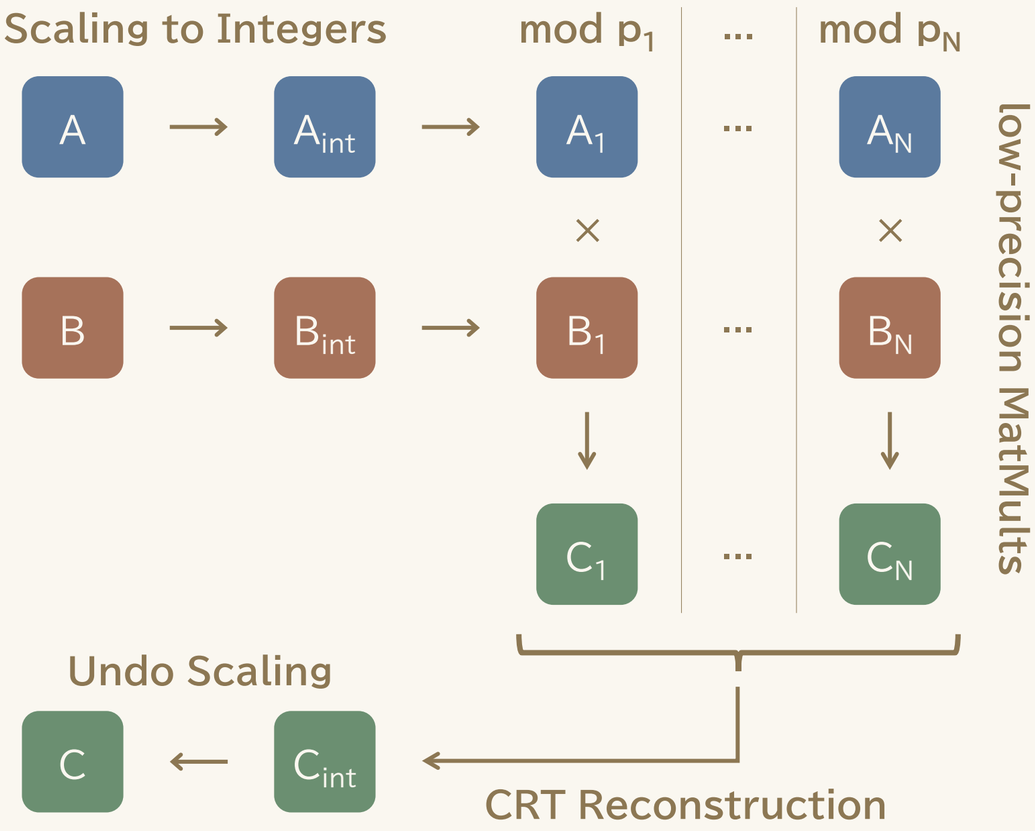
Ozaki-I — long multiplication

$S(S+1)/2$ low-precision matrix multiplications



Ozaki-II — CRT

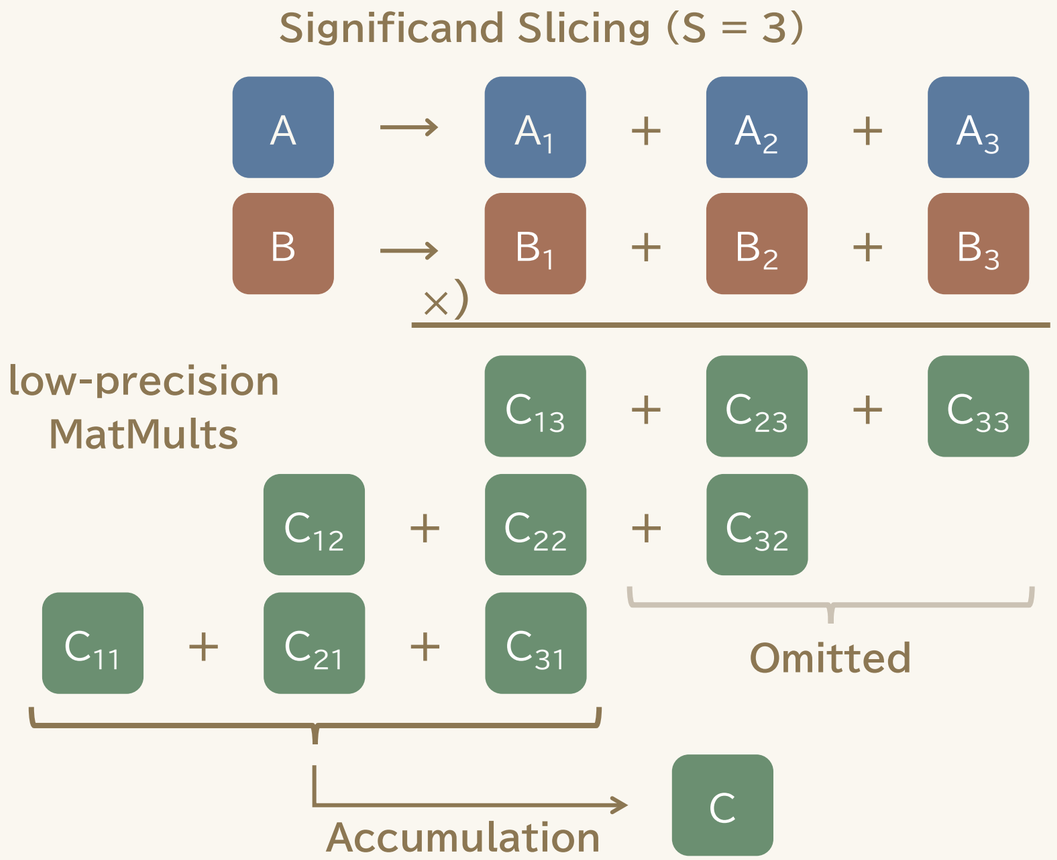
N low-precision matrix multiplications



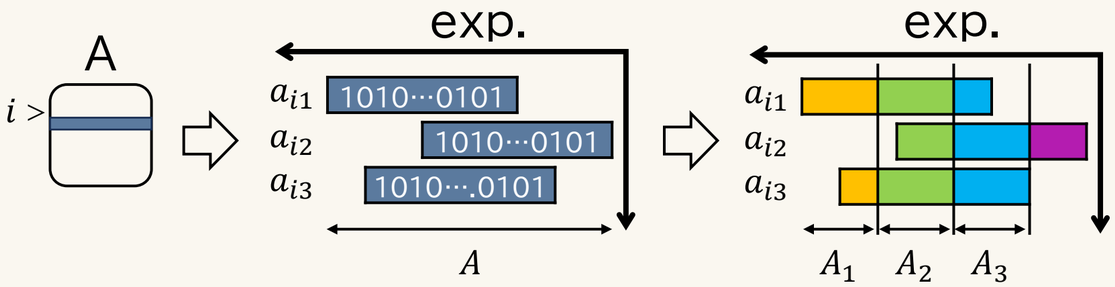
Ozaki-I Scheme

Ozaki-I — long multiplication

$S(S+1)/2$ low-precision matrix multiplications



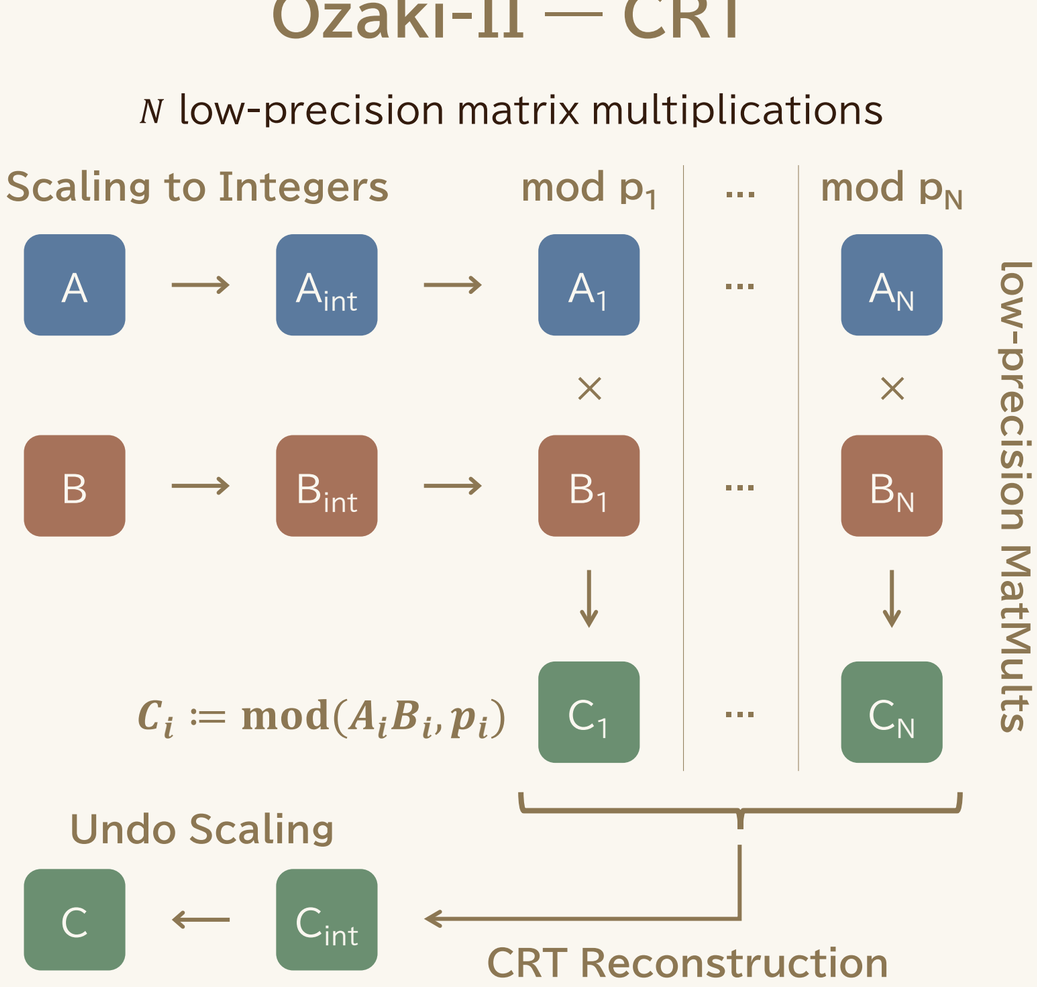
1. Treat the matrices as fixed-point
2. Split into slices using a shared exponent per row/column.



- Any data types for slices
  - FP16, BF16, FP8, INT8, etc.
- The slices do not involve exponent components.
  → Integer types align naturally.

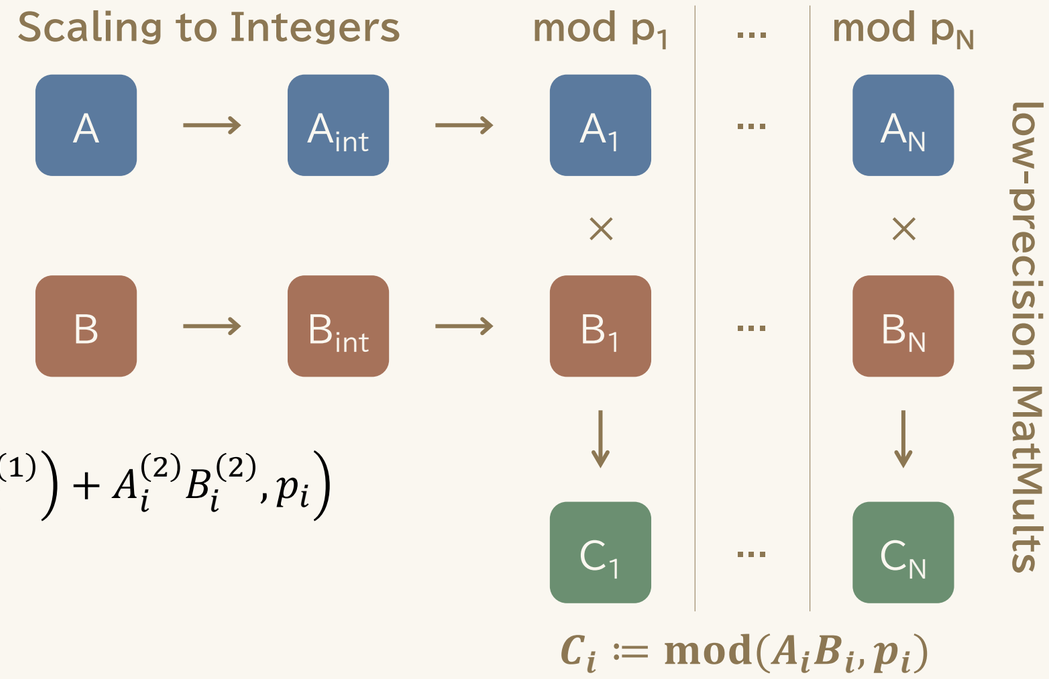
Ozaki-II Scheme

- $p_i \in \mathbb{N}$: pairwise coprime
- Final accuracy $\lesssim \log_2 \sqrt{\prod p_i/2}$ bits
- When using INT8, $p_i \leq 256$
  - many possible values: 256, 255, 253, ⋯
  - Final accuracy
    - All moduli → $\log_2 \sqrt{\prod p_i/2} \approx 170$
    - 14 moduli → $\log_2 \sqrt{\prod p_i/2} \approx 55$
- When using FP8_E4M3, $p_i \leq 77$:
  - limited possible values: 77, 71, 65, ⋯
  - Final accuracy
    - All moduli → $\log_2 \sqrt{\prod p_i/2} \approx 47 < 53$
- We cannot directly exploit FP8.



Proposed Method

At the time the paper was written



- $A_i \coloneqq \mathrm{mod}(A_{\mathrm{int}}, p_i)$, $B_i \coloneqq \mathrm{mod}(B_{\mathrm{int}}, p_i)$
- Represent residues using 2 FP8 matrices:
  - $A_i = sA_i^{(1)} + A_i^{(2)}$, $B_i = sB_i^{(1)} + B_i^{(2)}$
  - $C_i = \mathrm{mod}(A_i B_i, p_i) = \mathrm{mod}\left(s^2 A_i^{(1)} B_i^{(1)} + s\left(A_i^{(1)} B_i^{(2)} + A_i^{(2)} B_i^{(1)}\right) + A_i^{(2)} B_i^{(2)}, p_i\right)$
  - We had assumed $\left|A_i^{(x)}\right|, \left|B_i^{(y)}\right| \leq 16$.

1. Karatsuba-based method ($s = 16$, $p_i \leq 513$)
   - $A_i^{(3)} \coloneqq A_i^{(1)} + A_i^{(2)}$: FP8 & $\left|A_i^{(3)}\right| \leq 16$ if $|A_i| \leq 256$.
   - $C_i = \mathrm{mod}\left(s^2 C_i^{(1,1)} + s\left(C_i^{(3,3)} - C_i^{(1,1)} - C_i^{(2,2)}\right) + C_i^{(2,2)}, p_i\right)$, $C_i^{(x,y)} \coloneqq A_i^{(x)} B_i^{(y)}$ → 3 MatMults
2. Modular reduction without Karatsuba ($s = \sqrt{p_i}$, $p_i \leq 1089 = 33^2$)
   - $\mathrm{mod}(s^2, p_i) = \mathrm{mod}(p_i, p_i) = 0$
   - $C_i = \mathrm{mod}\left(s^2 C_i^{(1,1)} + s\left(C_i^{(1,2)} + C_i^{(2,1)}\right) + C_i^{(2,2)}, p_i\right) = \mathrm{mod}\left(s\left(C_i^{(1,2)} + C_i^{(2,1)}\right) + C_i^{(2,2)}, p_i\right)$ → 3 MatMults

Proposed Method

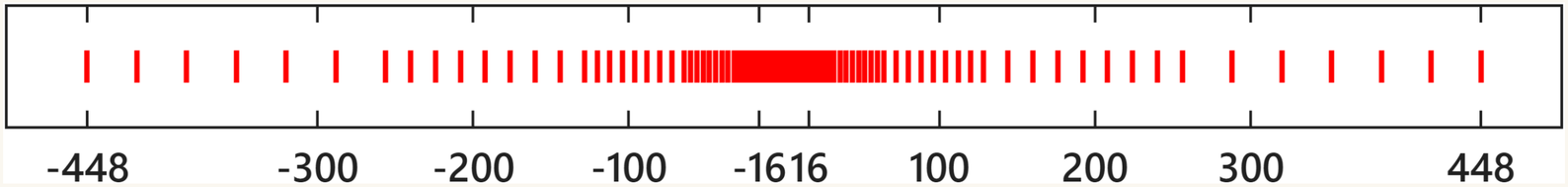
At the time the paper was written

- $A_i = sA_i^{(1)} + A_i^{(2)}$, $B_i = sB_i^{(1)} + B_i^{(2)}$, $\left|A_i^{(x)}\right|, \left|B_i^{(y)}\right| \leq 16$
- A_iB_i is computed using
  1. Karatsuba-based method ($s = 16$, $p_i \leq 513$)
  2. Modular reduction without Karatsuba ($s = \sqrt{p_i}$, $p_i \leq 1089$)
- Many possible values of p_i:
  - 1089, 1024, 961, 841, 625, 529, : Modular reduction
  - 511, 509, 103, 499, 491, 487, ⋯ : Karatsuba
- Final accuracy
  - All moduli → $\log_2 \sqrt{\prod p_i/2} \approx 372$
  - 12 moduli → $\log_2 \sqrt{\prod p_i/2} \approx 55$, 36 MatMults
- We can exploit FP8 to emulate FP64 with limb representation.

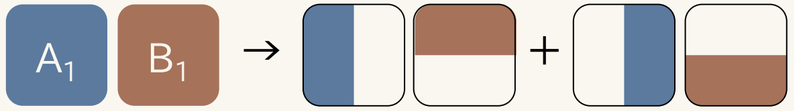
Extending the FP8 Limb Range

Improve the proposed method

- The original method restricts limb magnitudes to 16.
- FP8 E4M3 can represent larger integers exactly.



$$\{0, \pm 1, \ldots, \pm 16\} \cup \{\pm 18, \pm 20, \ldots, \pm 32\} \cup \{\pm 36, \pm 40, \ldots, \pm 64\} \cup \{\pm 72, \pm 80, \ldots, \pm 128\} \cup \{\pm 144, \pm 160, \ldots, \pm 256\} \cup \{\pm 288, \pm 320, \ldots, \pm 448\}$$

- A wider limb range provides more encoding options, at the cost of a shorter exact accumulation length:
  - $kL^2 \leq 2^{24}$, $L =$ maximum absolute limb value
  - $L \leq 16 \rightarrow k \leq 65536$, $L \leq 32 \rightarrow k \leq 16384$, $L \leq 64 \rightarrow k \leq 4096$
  - For large k, partition the inner dimension into blocks.
  - 

Smaller GEMMs may execute less efficiently.
- A custom modular GEMM could perform periodic modular reductions internally.

Improved FP8-Based Emulation (1)

Wider limbs

- Increase the maximum limb magnitude from 16 to 32.
  - A single FP8 GEMM can handle $k \leq 16384$ and moduli as large as 49.
  - Moduli larger than 49 impose substantially tighter bounds on k.
    - $p_i = 53 \rightarrow k \leq 3236, \quad p_i = 55 \rightarrow k \leq 809, \quad p_i = 59 \rightarrow k \leq 169$
- Combine 2 or 3 small moduli via CRT to support larger composite moduli.
  - Combining small-modulus results is more efficient than processing them separately.
  - Example: $p_i = 49 \times 47 = 2303$
    - $A_i^{(q)} = \mathrm{mod}(5A_i, q), \quad B_i^{(q)} = \mathrm{mod}(27B_i, q), \quad A_iB_i \equiv -47\, A_i^{(49)}B_i^{(49)} + 49\, A_i^{(47)}B_i^{(47)} \bmod 2303$
  - **Pre-scaling** reduces the magnitude of the CRT coefficients.
    - $-47\, A_i^{(49)}B_i^{(49)} + 49\, A_i^{(47)}B_i^{(47)}$ can be accumulated using INT32.
    - Without pre-scaling: $A_iB_i \equiv 1128\, A_i^{(49)}B_i^{(49)} - 1027\, A_i^{(47)}B_i^{(47)} \bmod 2303$

Improved FP8-Based Emulation (2)

Alternative representations

- AWE: Adaptive Weight Encoding [Hayashi et al., 2026, arXiv.2609.24519]
  - $A_i \equiv \sum_j w_j A_i^{(j)} \mod p_i, \;\; B_i \equiv \sum_j v_j B_i^{(j)} \mod p_i$
  - $A_i B_i \equiv \sum_t \lambda_t \left(\sum_j \alpha_{t,j} A_i^{(j)}\right)\left(\sum_j \beta_{t,j} B_i^{(j)}\right) \mod p_i$
- Specialize AWE to FP8 with 2 limbs and 3 products:
  - $A_i \equiv h\left(A_i^{(1)} + w A_i^{(2)}\right) \mod p_i, \;\; B \equiv h\left(B_i^{(1)} + w B_i^{(2)}\right) \mod p_i$
  - $A_i B_i \equiv h^2(1-w) A_i^{(1)} B_i^{(1)} + h^2(w^2 - w) A_i^{(2)} B_i^{(2)} + h^2 w \left(A_i^{(1)} + A_i^{(2)}\right)\left(B_i^{(1)} + B_i^{(2)}\right) \mod p_i$
- Design the weights and encodings to satisfy FP8 representability constraints.
- Example: $p_i = 1313$
  - $h = 8,\ w = 349$
  - $A_i B_i \equiv 49\, A_i^{(1)} B_i^{(1)} - 32\, A_i^{(2)} B_i^{(2)} + 15\left(A_i^{(1)} + A_i^{(2)}\right)\left(B_i^{(1)} + B_i^{(2)}\right) \mod 1313$

Improved FP8-Based Emulation (3)

Exploit congruence relations between FP8 components to avoid explicit cross-products.

- Special case for $p_i = 343 = 7^3$
  - $A_i \equiv 7\,A_i^{(1)} - 6\,A_i^{(2)} \mod 343$ with $A_i^{(1)} \equiv A_i^{(2)} \mod 7$
  - $B_i \equiv 7\,B_i^{(1)} - 6\,B_i^{(2)} \mod 343$ with $B_i^{(1)} \equiv B_i^{(2)} \mod 7$
  - $$\begin{aligned} A_iB_i &\equiv \left(7\,A_i^{(1)} - 6\,A_i^{(2)}\right)\left(7\,B_i^{(1)} - 6\,B_i^{(2)}\right) \mod 343 \\ &\equiv 49\,A_i^{(1)}B_i^{(1)} + 36\,A_i^{(2)}B_i^{(2)} - 42\,A_i^{(1)}B_i^{(2)} - 42\,A_i^{(2)}B_i^{(1)} \mod 343 \\ &\equiv 49\,A_i^{(1)}B_i^{(1)} + 36\,A_i^{(2)}B_i^{(2)} - 42\,A_i^{(1)}B_i^{(1)} - 42A_i^{(2)}B_i^{(2)} + 42\left(A_i^{(1)} - A_i^{(2)}\right)\left(B_i^{(1)} - B_i^{(2)}\right) \mod 343 \\ &\equiv 7\,A_i^{(1)}B_i^{(1)} - 6A_i^{(2)}B_i^{(2)} \mod 343 \end{aligned}$$

Multiples of 7

$$42\left(A_i^{(1)} - A_i^{(2)}\right)\left(B_i^{(1)} - B_i^{(2)}\right) \equiv 0 \mod 343$$

- Involves only 2 FP8 GEMMs

Improved FP8-Based Emulation (4)

Exploit congruence relations between FP8 components to avoid explicit cross-products.

- Special cases for $p_i = 19305 = 11 \cdot 13 \cdot 135$ & $p_i = 11583 = 11 \cdot 13 \cdot 81$
  - $135, 81 > 49$
  - $\mathrm{mod}(A_i B_i, p_i)$ is reconstructed using the CRT with 3 moduli: 11, 13, & $3r$ ($r \in \{45, 27\}$)
- Choose $A_i^{(1)}$, $A_i^{(2)}$ and $A_i^{(3)}$ to satisfy
  - $A_i \equiv A_i^{(1)} + r A_i^{(2)} \bmod 3r, \quad A_i^{(3)} \equiv A_i^{(1)} + A_i^{(2)} \bmod 3, \quad A_i \equiv A_i^{(2)} \bmod 11, \quad A_i \equiv A_i^{(3)} \bmod 13$
- Run 3 FP8 GEMMs
  - $C_2 = A_i^{(2)} B_i^{(2)}, \quad C_3 = A_i^{(3)} B_i^{(3)}, \quad C_1 = (1 - r) A_i^{(1)} B_i^{(1)} - r C_2 + r C_3$
- Then,
  - $A_i B_i \equiv C_1 \bmod 3r, \quad A_i B_i \equiv C_2 \bmod 11, \quad A_i B_i \equiv C_3 \bmod 13$
- Reconstruct $\mathrm{mod}(A_i B_i, p_i)$ from C_1, C_2, & C_3 via the CRT:
  - $A_i B_i \equiv 8866\, C_1 - 9360\, C_2 + 6930\, C_3 \bmod 19305$
  - $A_i B_i \equiv -5291\, C_1 - 351\, C_2 - 2079\, C_3 \bmod 11583$

Effect of Improvements

Original Proposed Method

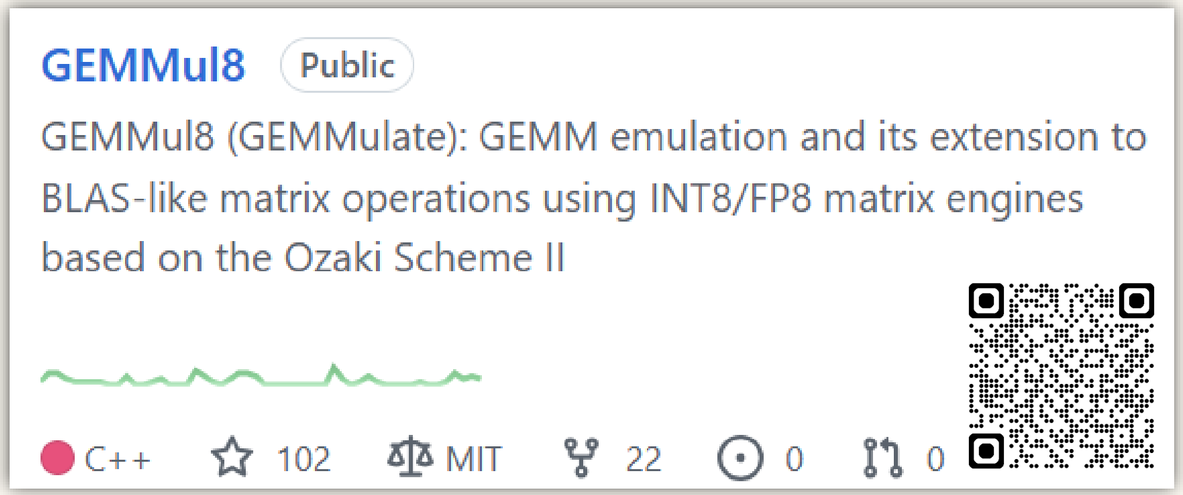
#moduli	Bits	#MatMult
5	24	15
6	29	18
7	33	21
8	38	24
9	42	27
10	46	30
11	51	33
12	**55**	**36**
13	60	39
14	64	42
15	69	45

Improved Proposed Method

#moduli	Bits	#MatMult
5	28	11
6	34	14
7	39	17
8	44	20
9	48	22
10	**53**	**25**
11	58	28
12	63	31
13	68	34
14	73	37
15	78	40

Our Implementation

https://github.com/RIKEN-RCCS/GEMMul8



- Supports NVIDIA and AMD GPUs
- Features:
  - All Level-3 cuBLAS operations
  - A special operation: triangular x triangular
  - MxP execution: FP32 x FP64 = FP64, etc.
  - Memory-saving mode → enables apps. to run with GEMMul8 even on RTX.
  - Hook mode:
    - Intercepts existing cuBLAS/hipBLAS calls and redirects them to GEMMul8
    - Just setting the environment variables
  - etc.
  - For more details, please refer to README.

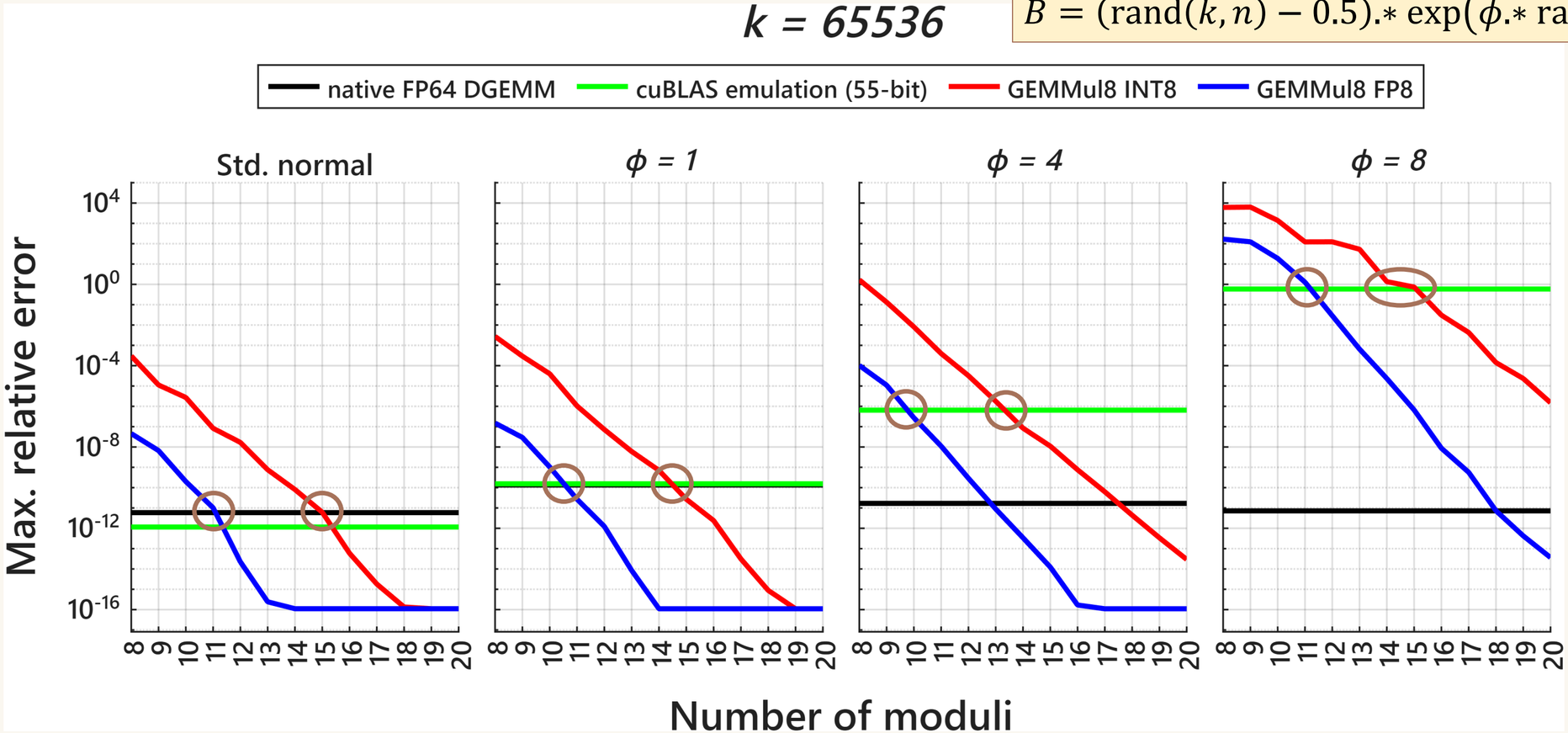
Accuracy

- 55-bit emulation is sufficient for most applications [Bernabeu (NVIDIA), NVIDIA GTC 2025].
  - INT8-basaed Ozaki-II : 13—15 moduli
  - Proposed FP8-based Ozaki-II : 9—12 moduli

$m = n = 128$

$A = (\mathrm{rand}(m,k) - 0.5).* \exp\big(\phi.* \mathrm{randn}(m,k)\big);$

$B = (\mathrm{rand}(k,n) - 0.5).* \exp\big(\phi.* \mathrm{randn}(k,n)\big);$



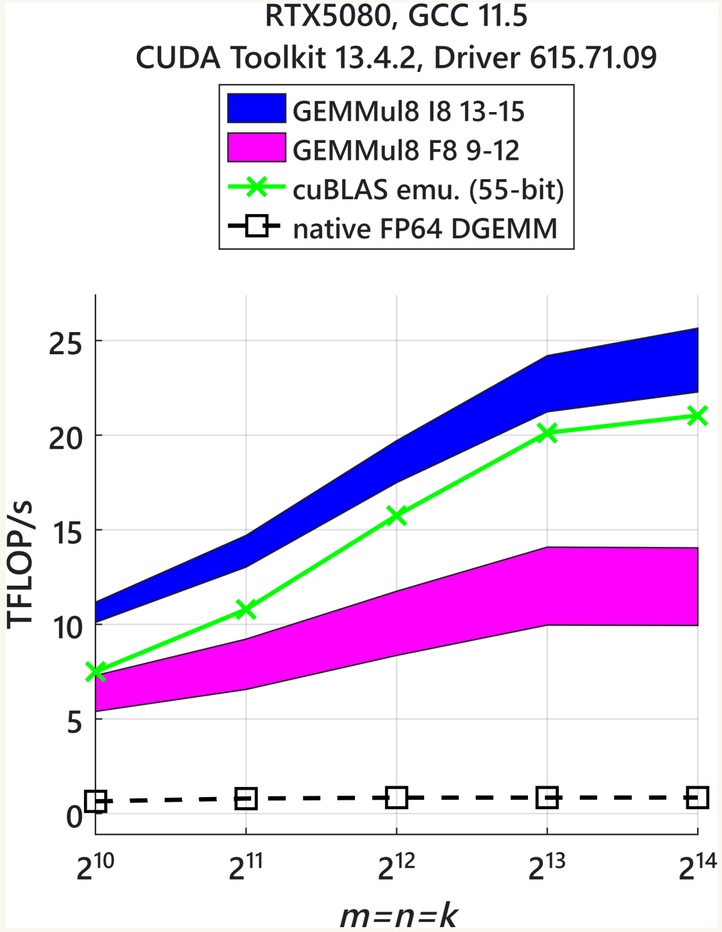
Throughput

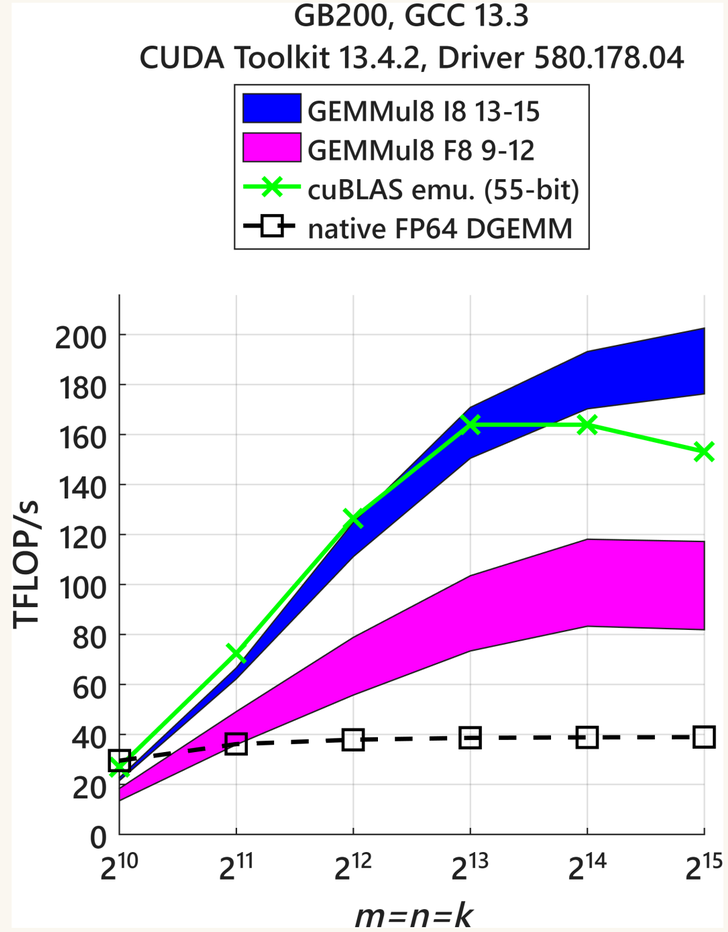
The INT8-based approaches are preferable if INT8 capacity is sufficient.

- I8:F8 = 1:1 on both GPUs.
- I8 emulation is faster.
  - I8 : achieves 200 TFLOP/s
  - F8 : achieves 120 TFLOP/s
- F8 emulation outperforms native FP64.

↓

- F8 emulation can be particularly effective on GPUs with limited I8 throughput.
  - GB300, Rubin, etc.





Further Discussion

	GB200	Rubin	Speedup
FP16	2500 TFLOP/s	4000 TFLOP/s	x1.6
FP8	5000 TFLOP/s	17500 TFLOP/s	**x3.5**

- NVIDIA's new feature: TI16
  - FP16 x FP16 (11-bit int.) → INT32
  - On Rubin, cuBLAS DGEMM emulation achieves up to 212 TFLOP/s (cf. release note)
- Our Method
  - On GB200, achieves up to 120 TFLOP/s (FP8 GEMM: 90%)
  - On Rubin, can achieve up to 333 TFLOP/s (FP8 GEMM throughput x3.5)???
- AMD MI430X
  - Projected peak native FP64 throughput: up to 288 TFLOP/s
  - → What throughput can be achieved in practice?
- Where is the Ozaki scheme effective?
  - NVIDIA: "Real world apps. will yield satisfactory results with emulation." [Bayraktar-Gunnels, ISC '24]
  - AMD: "Ozaki doesn't address 90% of the HPC apps." [HPCwire, March 13, 2026]
  - → Which high-arithmetic-intensity workloads are not well suited to Ozaki schemes?

Conclusion

- We proposed the FP8-based Ozaki-II scheme.
  - Introduced some modular techniques.
- If INT8 capacity is sufficient,
  → the INT8-based emulation is preferable.
  (On GB200 (I8:F8 = 1:1), I8: 200 TFLOP/s, F8: 120 TFLOP/s)
- If INT8 throughput is limited, (GB300, Rubin, etc.)
  → FP8-based emulation can be effective.
- Future works
  - Error analysis of the proposed method
  - Add automatic selection of the required number of moduli to GEMMul8.

Thank you.

yuki.uchino.fe@riken.jp