Emulation of Complex Matrix Multiplication based on the Chinese Remainder Theorem

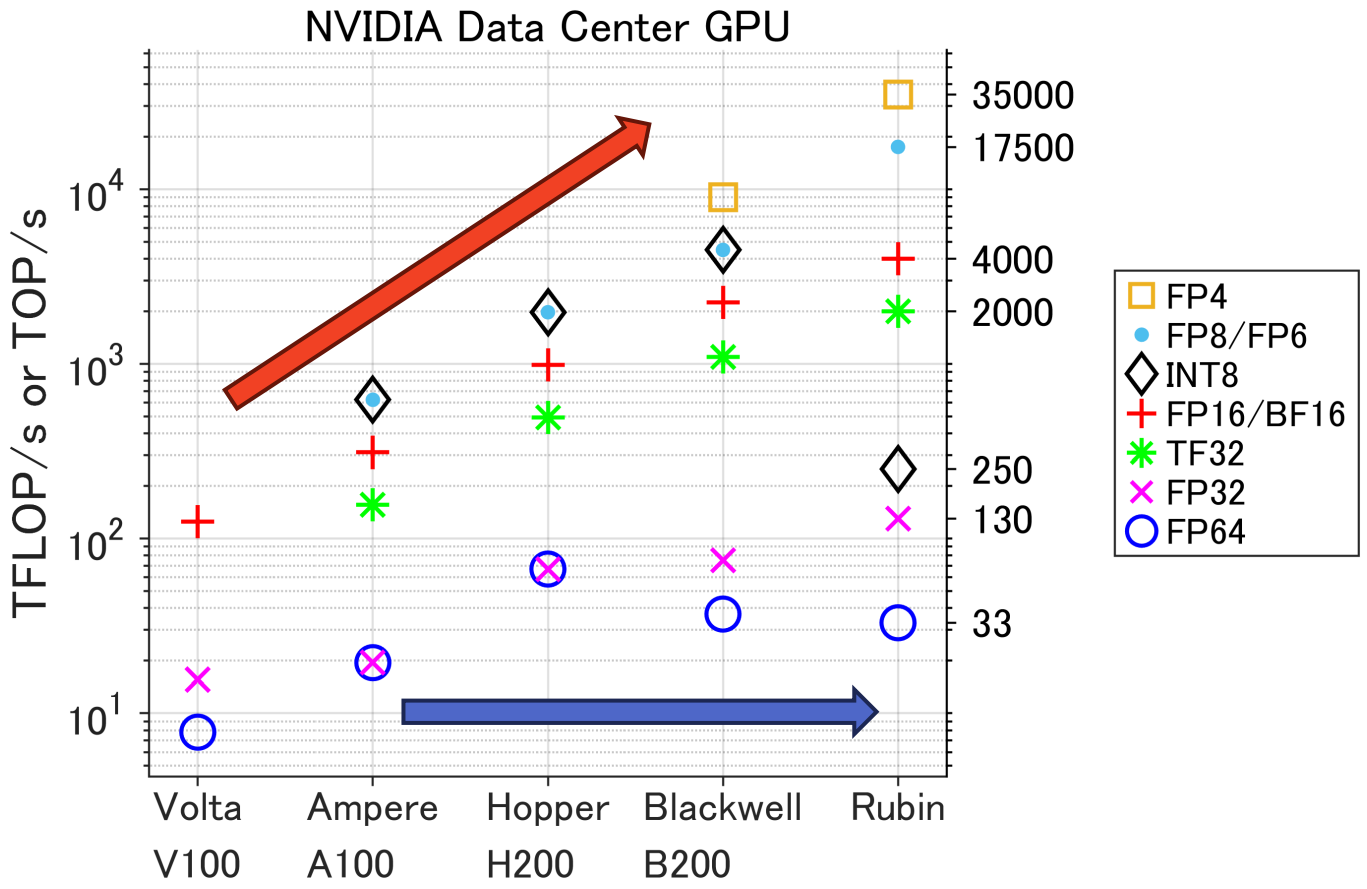
Yuki Uchino[†], Qianxiang Ma[†], Toshiyuki Imamura[†]
Katsuhisa Ozaki[‡], Patrick Lars Gutsche[*]

[†]RIKEN R-CCS, [‡]Shibaura Institute of Technology, [*]Ecole Normale Superieure de Lyon

yuki.uchino.fe@riken.jp

Motivation

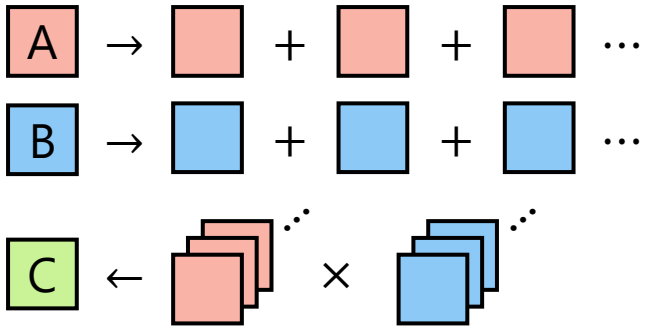
- Low-precision performance has been improving very rapidly.
- High-precision performance has improved much more slowly.
  - **After Hopper, FP64 performance has continued to decline.**
- This trend can contribute to performance stagnation of traditional HPC workloads.
- To use low precision has been becoming a common strategy.
- This talk focuses on **FP32/FP64 GEMM emulation** using low-precision matrix engines.



GEMM Emulation (1)

- cuMpSGEMM
  - FP16/TF32 [Ootomo et al. ‘22/’23]
  - https://github.com/enp1s0/cuMpSGEMM
- BF16x9
  - BF16 [Bayraktar et al. ‘26]
  - cuBLAS 12.9+
- Ozaki-I scheme
  - INT8 [Ootomo et al. ‘24] [U. et al. ‘24] [Schwarz et al. ‘26]
  - FP8 [Mukunoki ‘26], FP16 [Mukunoki et al. ‘20] [Mukunoki ‘26]
  - F64 → fixed-point (integer) → multi-I8/F8/F16
  - $s(s+1)/2$ matrix multiplications for s slices
  - cuBLAS 13.0u2+
  - https://github.com/enp1s0/ozIMMU
  - https://github.com/RIKEN-RCCS/accelerator_for_ozIMMU

Long-multiplication-based methods

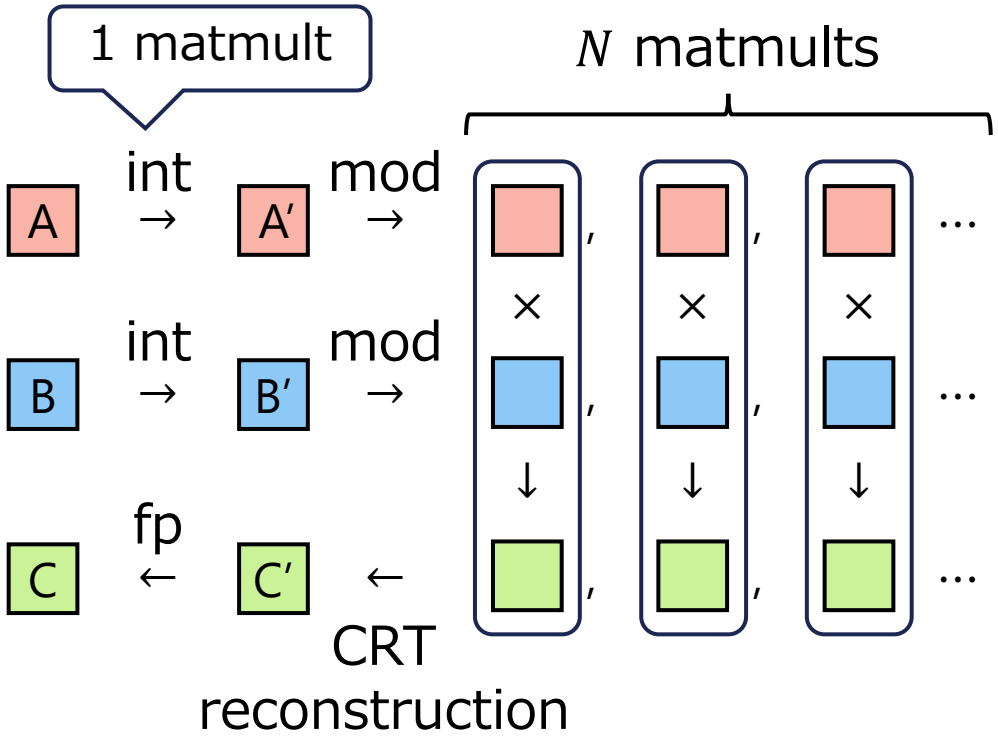


		A_1	A_2	A_3
	$\times)$	B_1	B_2	B_3
		A_1B_3	A_2B_3	A_3B_3
	A_1B_2	A_2B_2	A_3B_2	
A_1B_1	A_2B_1	A_3B_1		

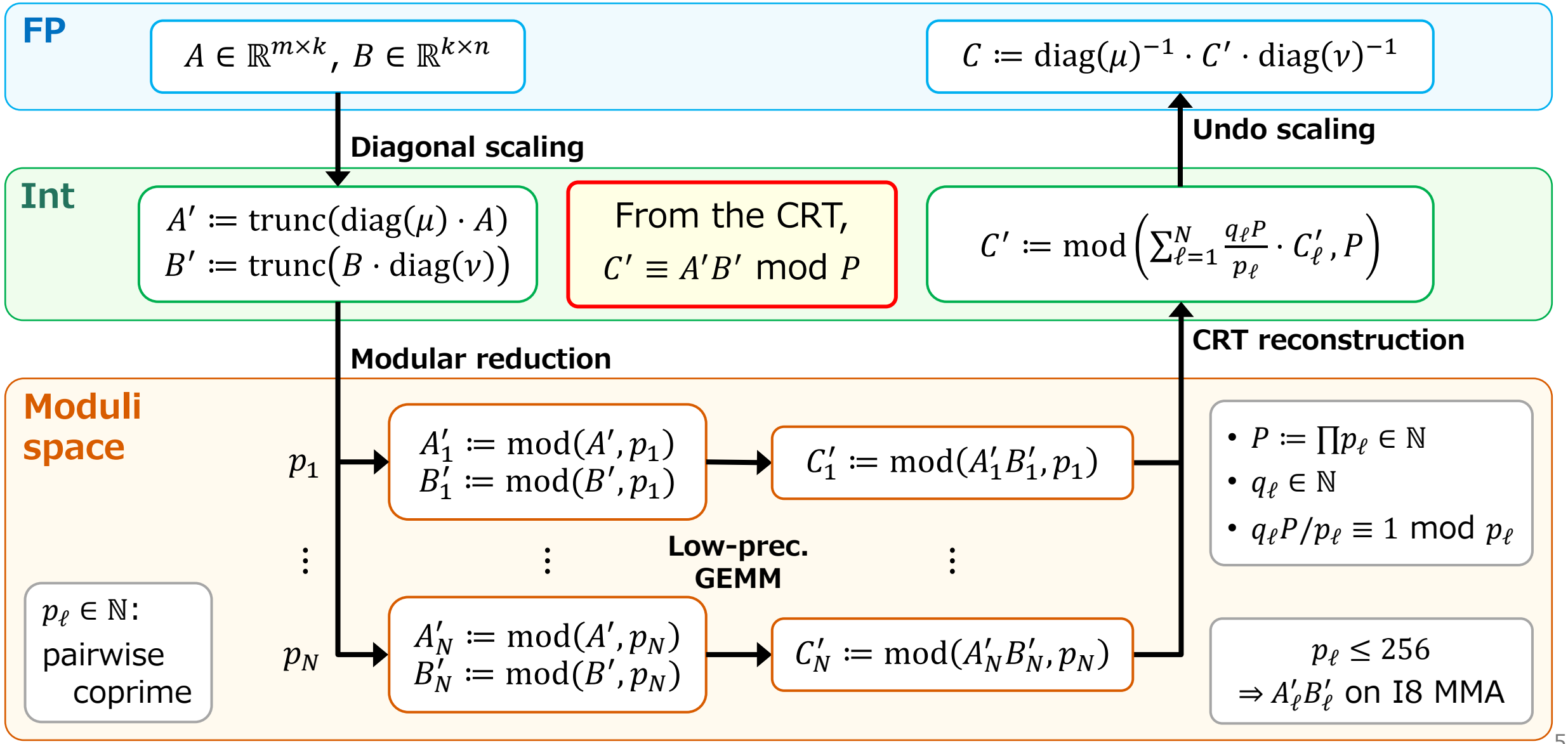
GEMM Emulation (2)

- Ozaki-II scheme
  - Arbitrary-precision emulation [Ozaki et al. '25]
  - INT8 [U. et al. '25] etc.
  - FP8 [U. et al. '26]
  - F32/F64 → fixed-point (integer) → multi-I8/F8
  - $N+1$ matrix multiplications for N CRT moduli
- Accuracy:
  - cuMpSGEMM ⋯ full FP32 precision
  - BF16x9 ⋯ full FP32 precision
  - Ozaki-I ⋯ Depending on the # of slices
  - Ozaki-II ⋯ Depending on the # of moduli
  - NVIDIA [Bernabeu GTC25] reported that they can't find applications that require $s > 7$.
  - $N = 15$ achieves Ozaki-I-7-level accuracy.

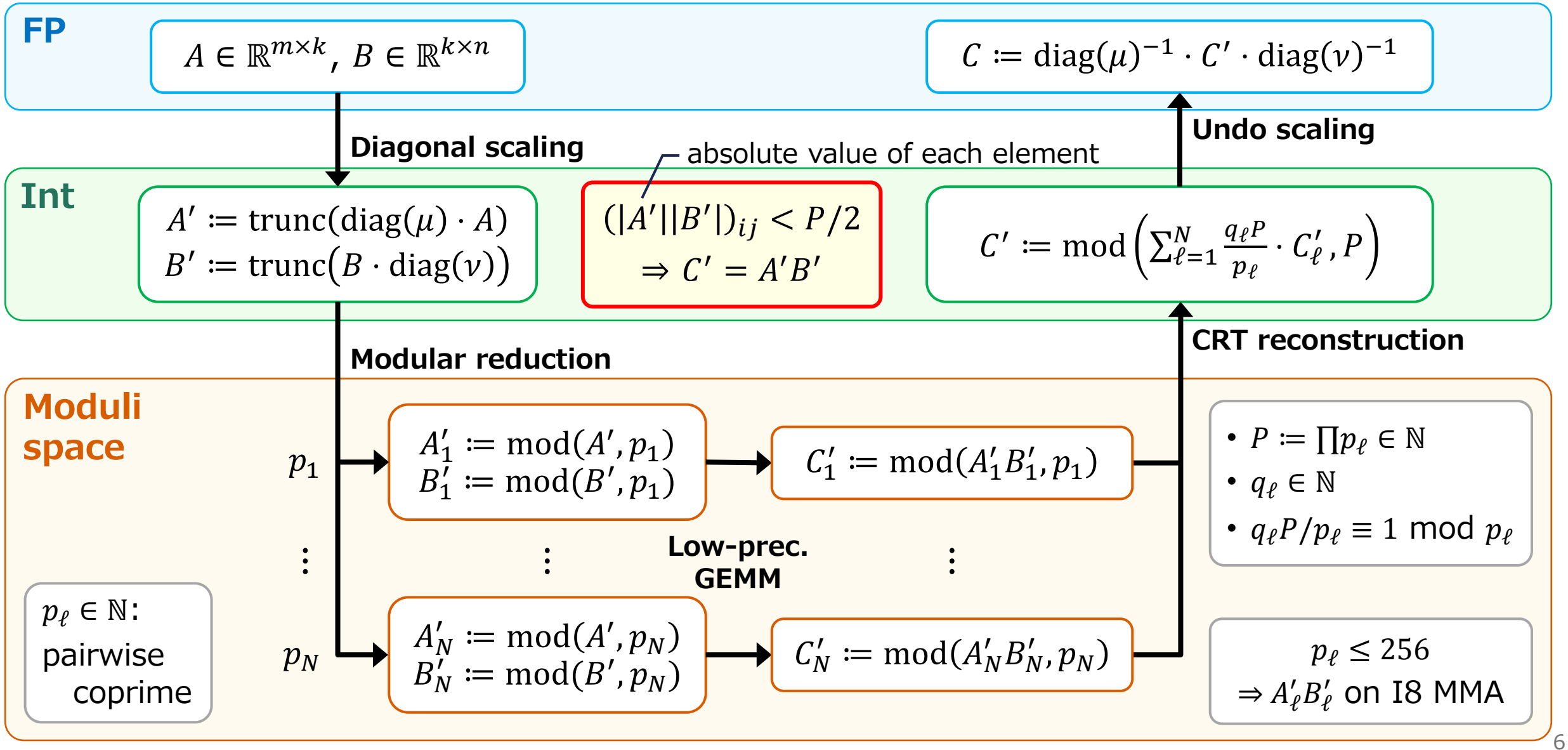
CRT-based method



A Closer Look at Ozaki-II



A Closer Look at Ozaki-II



Extension to Complex-valued GEMM

- This study extends the Ozaki-II to C/ZGEMM: $(A_R + iA_I)(B_R + iB_I)$
- Simple approach:
  - To decompose C/ZGEMM into multi-S/DGEMMs: $(A_R B_R - A_I B_I) + i(A_R B_I + A_I B_R)$
    1. Convert a complex matrix (AoS) to two real matrices (SoA)
    2. Call 4 real-GEMMs
    3. Convert multiple real-GEMM results (SoA) back to a complex matrix (AoS)
  - Some drawbacks:
    - Requires extra buffers for the S/DGEMM results.
    - Final reconstruction involves the rounding error.
      - If $A_R B_R \approx A_I B_I$, cancelation occurs in $A_R B_R - A_I B_I$ and the accuracy becomes poor.
- Employed strategy:
  - **Keep the complex structure** throughout Ozaki-II workflow
  - Apply **Karatsuba algorithm** inside each modular complex matrix multiplication

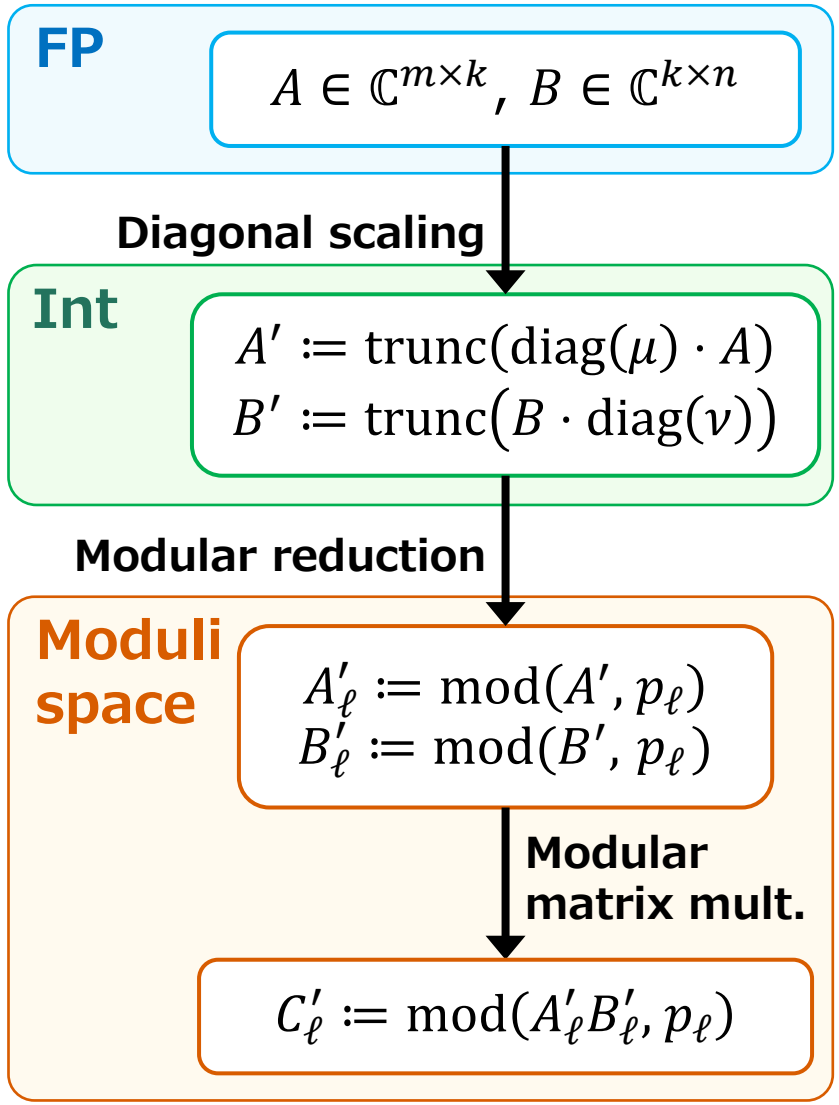
Diagonal scaling for Complex GEMM

- Uniqueness condition for Real: $(|A'||B'|)_{ij} < P/2$
- Uniqueness condition for Complex:
  - $(|A'_R||B'_R| + |A'_I||B'_I|)_{ij} < P/2$, $(|A'_R||B'_I| + |A'_I||B'_R|)_{ij} < P/2$

1. Apply diagonal scaling to fit in INT8:
   - $S := \lceil \mathrm{diag}(x) \cdot A \rceil$, $T := \lceil B \cdot \mathrm{diag}(y) \rceil$, S & T: INT8
2. Compute $U := S \cdot T$ via a modified Karatsuba:
   - $U_I := S_R T_I + S_I T_R$, $U_R := U_I + (S_R - S_I)(T_R - T_I)$

- Then,
  - $|A'_R||B'_R| + |A'_I||B'_I| \le \mathrm{diag}(\mu)\, \mathrm{diag}(x)^{-1} U_R \mathrm{diag}(y)^{-1}\, \mathrm{diag}(\nu)$
  - $|A'_R||B'_I| + |A'_I||B'_R| \le \mathrm{diag}(\mu)\, \mathrm{diag}(x)^{-1} U_I \mathrm{diag}(y)^{-1}\, \mathrm{diag}(\nu)$

  known part
- Choose μ & ν backward so that both bounds are $< P/2$ elementwise.

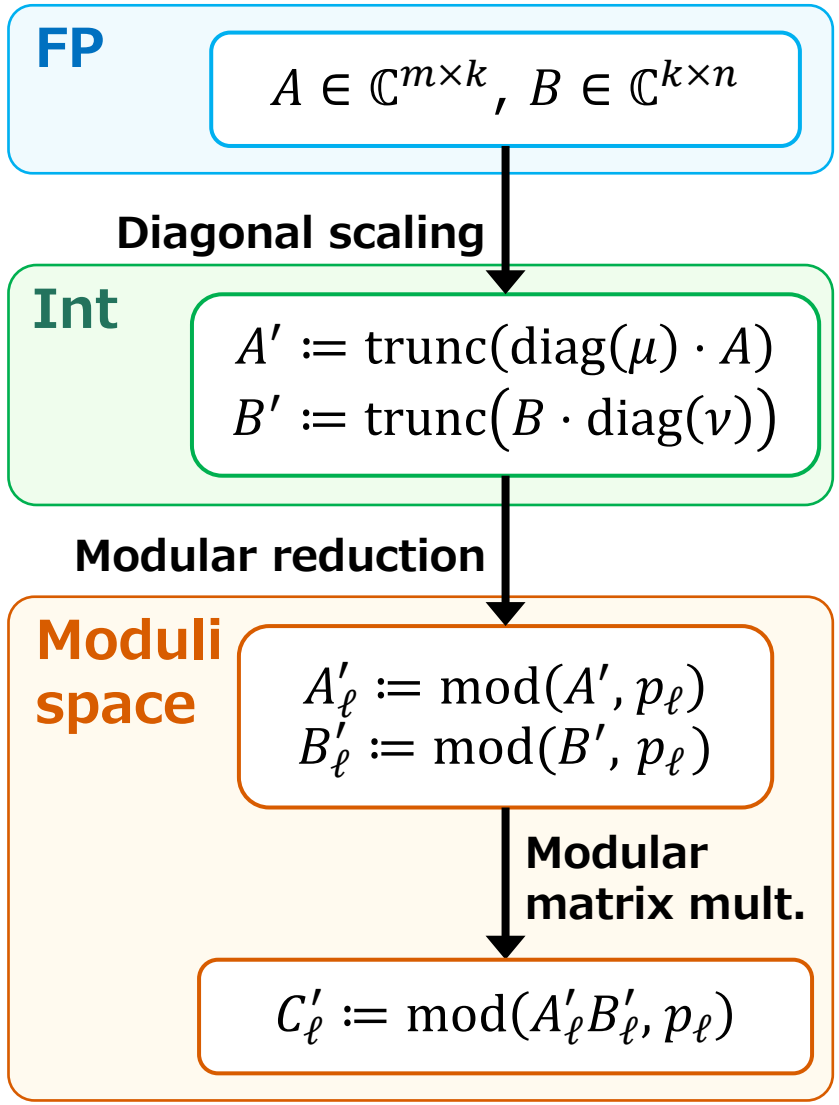


Error-free Karatsuba multiplication

- Conventional Karatsuba:
  - $D = (A'_\ell)_R (B'_\ell)_R$
  - $E = (A'_\ell)_I (B'_\ell)_I$
  - $F = \big((A'_\ell)_R + (A'_\ell)_I\big)\big((B'_\ell)_R + (B'_\ell)_I\big)$
  - $A'_\ell B'_\ell = \big((A'_\ell)_R + i(A'_\ell)_I\big)\big((B'_\ell)_R + i(B'_\ell)_I\big)$
    $\rightarrow A'_\ell B'_\ell = (D - E) + i \cdot (F - D - E)$

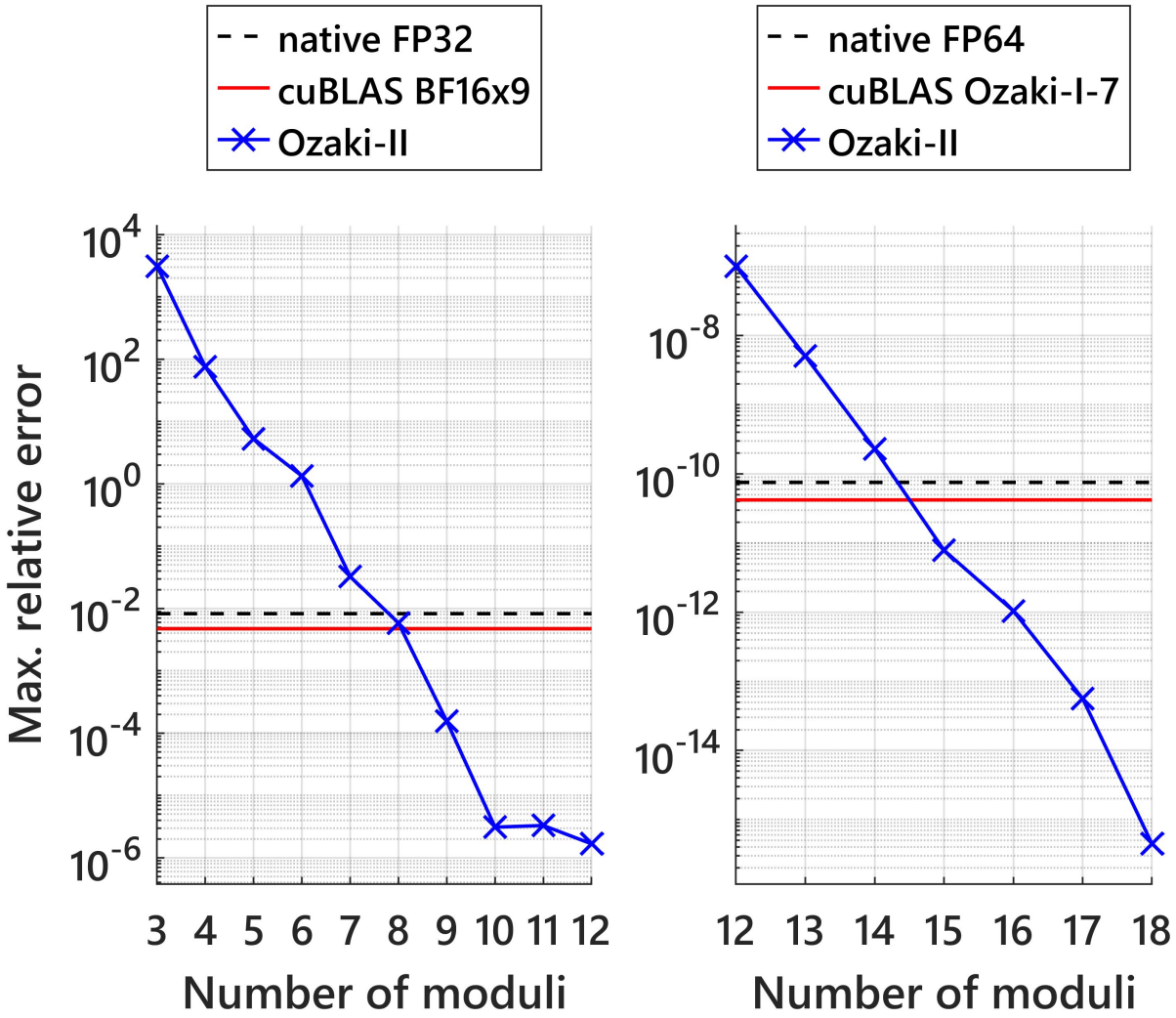
may exceed INT8 range

- Karatsuba in the Moduli Space:
  - $D = \mathrm{mod}\big((A'_\ell)_R (B'_\ell)_R, p_\ell\big)$
  - $E = \mathrm{mod}\big((A'_\ell)_I (B'_\ell)_I, p_\ell\big)$
  - $F = \mathrm{mod}\big(\mathrm{mod}\big((A'_\ell)_R + (A'_\ell)_I, p_\ell\big)\mathrm{mod}\big((B'_\ell)_R + (B'_\ell)_I, p_\ell\big), p_\ell\big)$
  - $C'_\ell = \mathrm{mod}(D - E, p_\ell) + i \cdot \mathrm{mod}(F - D - E, p_\ell)$
    - blue mod performs just a reduction
  - All ops can be performed using INT32



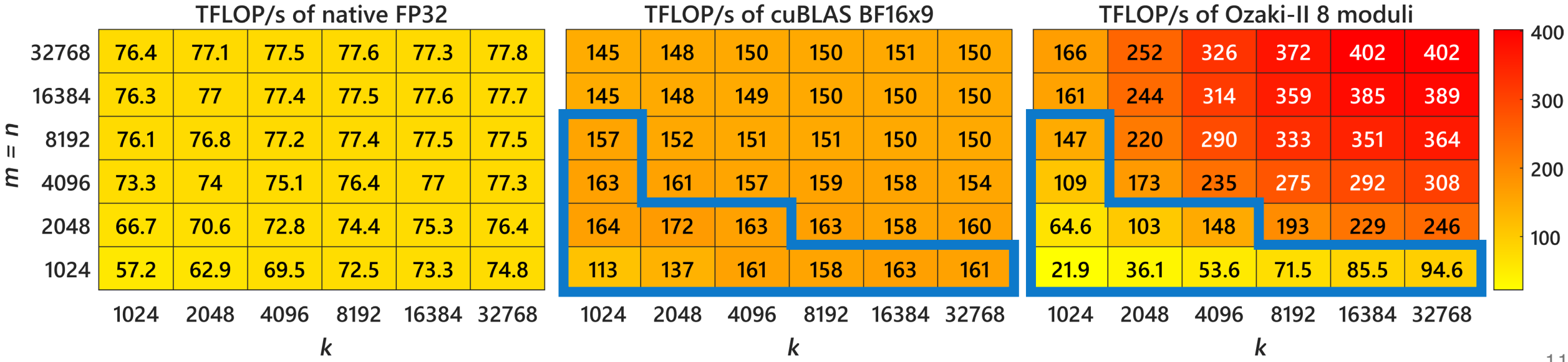
Accuracy of C/ZGEMM

- Test matrices:
  - standard normal random matrices
  - k = 65536, m = n = 128
  - The accuracy of matmult depends on k.
- CGEMM:
  - Ozaki-II with 8 moduli achieves accuracy close to that of BF16x9.
  - 7 moduli for small k
- ZGEMM:
  - Recall that 7 slices are sufficient.
  - Ozaki-II with 14-15 moduli achieves accuracy close to that of Ozaki-I.
  - 14 moduli for small k
- the accuracy is tunable.



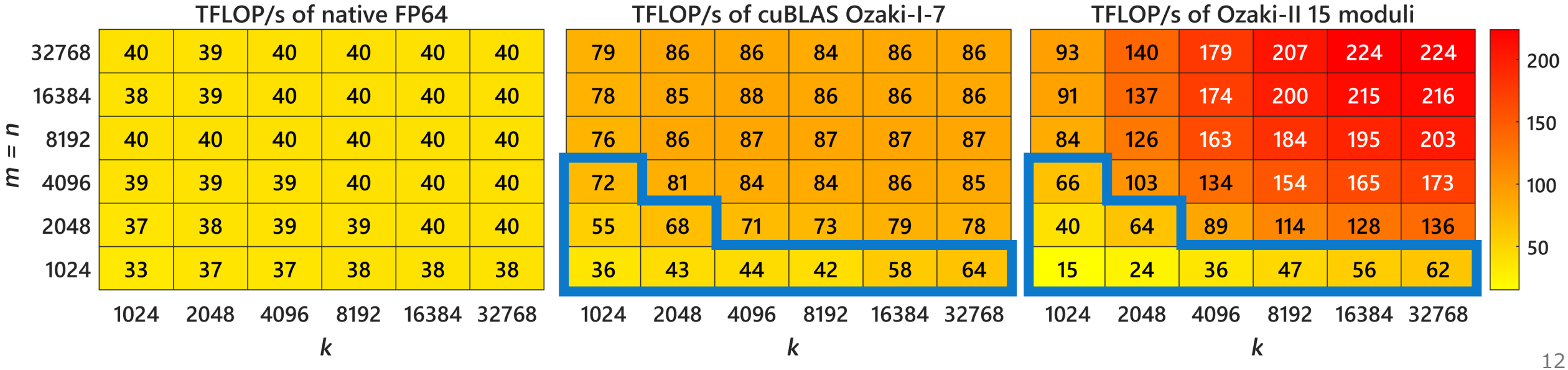
CGEMM Throughput on GB200

- BF16x9 CGEMM : Up to 2.5x speedup over native FP32
- Ozaki-II CGEMM : Up to 5.2x speedup over native FP32
  - For small matrices, the overhead is still visible.
  - Can become a viable alternative to BF16x9
    - when the user knows the required number of moduli, or
    - when the problem size is sufficiently large.



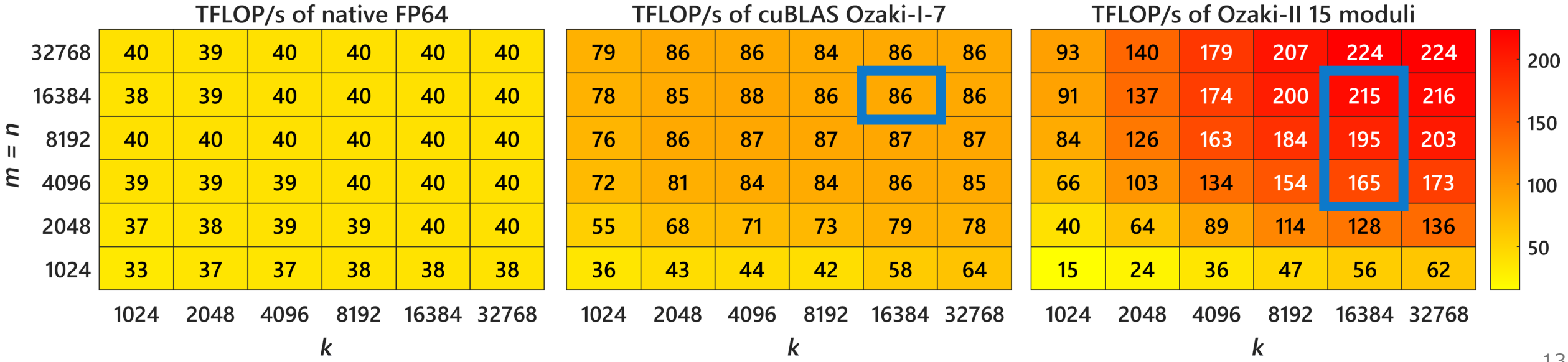
ZGEMM Throughput on GB200

- Ozaki-I ZGEMM ：Up to 2.2x speedup over native FP64
- Ozaki-II ZGEMM：Up to 5.6x speedup over native FP64
- For small-scale problems (blue box), Ozaki-I is faster than Ozaki-II.
- Outside that region, Ozaki-II is faster.
- For sufficiently large problems, the throughput exceeds 200 TFLOP/s.

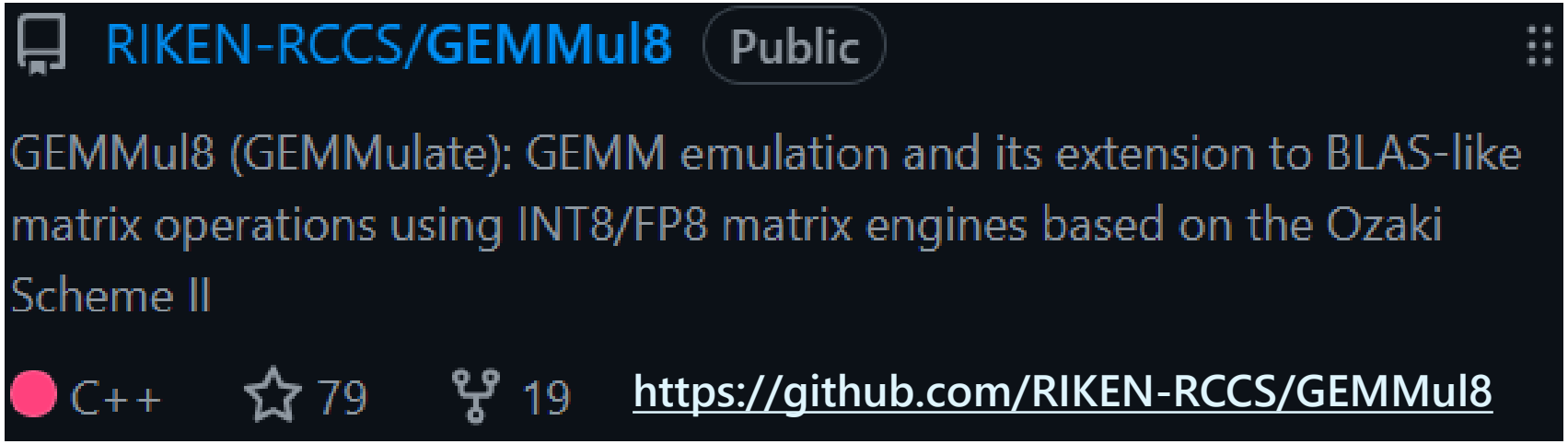


Working Memory Footprint

- The working-memory requirement of the current implementation
  - $\approx 3N(mk + kn) + (2N + 12)mn$ bytes
  - (m,n,k) = (16384, 16384, 16384) : 35 GB
  - (m,n,k) = (8192, 8192, 16384) : 15 GB
  - (m,n,k) = (4096, 4096, 16384) : 8 GB
- Can be reduced by applying blocking in the m- and n-dimensions.



Open-Source Library



- Open-source software is publicly available.
- GEMMul8 v3.0.4 was used for the numerical experiments in this talk.
- Supported platforms: CUDA, HIP
- Supported features:
  - Emulation of all Level-3 BLAS operations
  - Mixed-precision execution (e.g., FP32 x FP32 -> FP64)
  - Overloading of existing cuBLAS/hipBLAS routines
- For further details, please refer to README.md.

Conclusion

- Presented in this talk
  - An extension of the Ozaki-II scheme to complex GEMM.
  - For larger problems, Ozaki-II outperforms native GEMM & cuBLAS emulation.
  - Achieved 200+ TFLOP/s on a current-generation (Blackwell-class) GPU.
    - NVIDIA's published FP64 GEMM emulation performance is 150 TFLOP/s.
- Not covered in this talk
  - FP8-based emulation (see arXiv:2603.10634)
  - Extensions to Level-3 BLAS-like operations (EASIAM2026)
- Future work
  - Incorporate memory-reduction functionality into GEMMul8.
  - Simple blocking in the m/n-dimensions.
  - Alternative algorithm-level strategies for reducing workspace requirements.

GEMMul8

Thank you.

Contact: yuki.uchino.fe@riken.jp

Working Memory Footprint

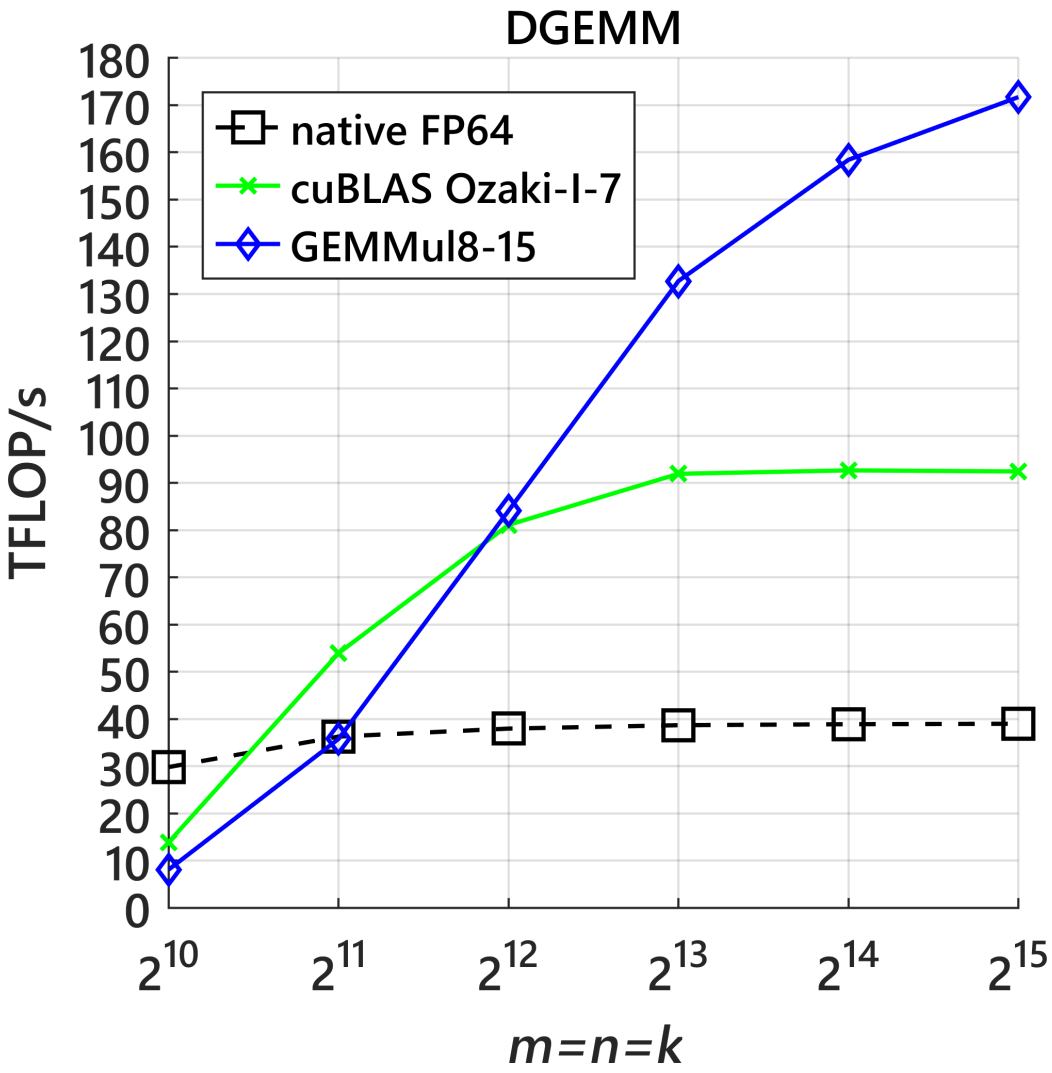
- The working-memory requirement of the current implementation
  - $\approx 3N(mk + kn) + (2N + 12)mn$ bytes

m=n | k	4096	8192	16384
4096	2 GB	4 GB	7 GB
8192	6 GB	9 GB	15 GB
16384	17 GB	23 GB	35 GB

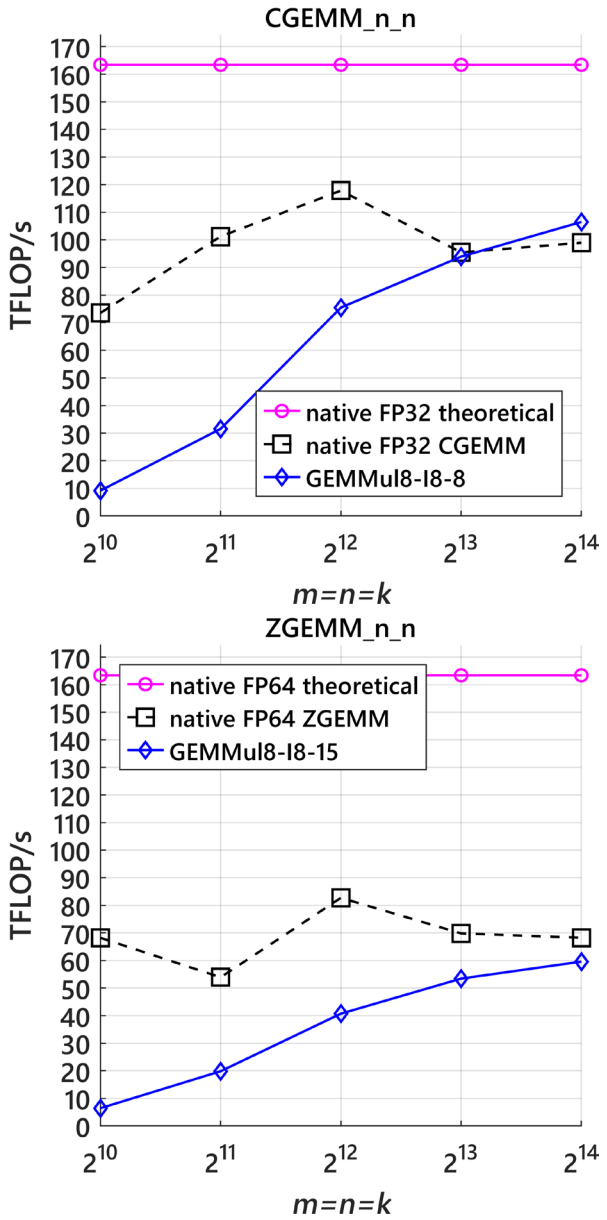
- k-dimension blocking is less attractive.
  - It requires FP32/FP64 accumulation of partial block products.
  - This introduces additional overhead and further degrades performance.

DGEMM Throughput on GB200

- Ozaki-I DGEMM:
  - Up to 93 TFLOP/s
  - Up to 2.4x speedups over native FP64
- Ozaki-II DGEMM:
  - Up to 172 TFLOP/s
  - Up to 4.4x speedups over native FP64
- For small-scale problems, Ozaki-II is slower than the other methods.
  - due to the overheads of diagonal scaling and modular reduction
- For $m, n, k > 2048$, Ozaki-II becomes the fastest.



Throughput on MI300X



- Theoretical peak throughput
  - INT8: 2600 TOP/s
  - FP32: 163.4 TFLOP/s (FP32:INT8 = 1:16)
  - FP64: 163.4 TFLOP/s (FP64:INT8 = 1:16)
- Measured throughput
  - INT8: around 800 TOP/s
  - FP32: around 100 TFLOP/s (FP32:INT8 = 1:8)
  - FP64: around 70 TFLOP/s (FP64:INT8 = 1:11)
- On MI300X, low-precision throughput is not sufficiently faster than FP32/FP64.
  - the performance headroom for the Ozaki scheme is limited.
- MI430X is projected to deliver 200 TFLOP/s of native FP64 performance.

Experimental Setup on a GB200

- Software
  - CUDA version: 13.2
  - NVIDIA driver: 595.45.04
  - Host compiler: g++ (GCC) 11.5.0
  - GEMMul8 v3.0.4
- Measurement protocol
  - Workspace was preallocated before timing.
  - Each benchmark was warmed up for at least 3 seconds.
  - Timed measurements were then collected for at least 12 seconds.
  - The median execution time was used for reporting performance.
  - TFLOP/s = 1e-12 * 8mnk/sec
- Implementations compared
  - Native FP32/FP64 GEMM, Ozaki-I, BF16x9: cuBLAS
  - Ozaki-II: GEMMul8

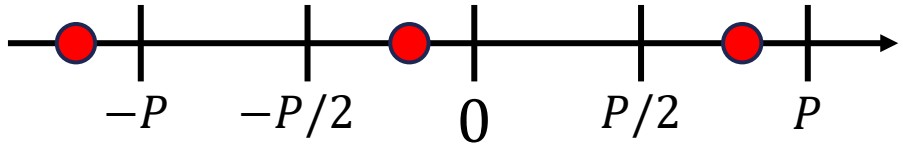
Cost for low-precision matmults.

Method	Real GEMM work	Real Example	Complex GEMM work	Complex Example
Ozaki-II	2(N+1)mnk	32mnk (N=15)	6(N+1)mnk	96mnk (N=15)
Ozaki-I	s(s+1)mnk	56mnk (s=7)	4s(s+1)mnk	224mnk (s=7)
Ozaki-I Karatsuba			3s(s+1)mnk	216mnk (s=8)

- Karatsuba for low-precision matrix multiplications $A_i B_j$:
  - $\mathrm{Re}(A_i B_j) = \mathrm{Re}(A_i)\mathrm{Re}(B_j) - \mathrm{Im}(A_i)\mathrm{Im}(B_j)$
  - $\mathrm{Im}(A_i B_j) = \big(\mathrm{Re}(A_i) + \mathrm{Im}(A_i)\big)\big(\mathrm{Re}(B_i) + \mathrm{Im}(B_i)\big) - \mathrm{Re}(A_i)\mathrm{Im}(B_j) - \mathrm{Im}(A_i)\mathrm{Re}(B_j)$
- Ozaki-II: applies Karatsuba inside the modular space.
  - $\mathrm{Re}(A_i) + \mathrm{Im}(A_i)$ is reduced not to exceed the I8 range as $\mathrm{mod}(\mathrm{Re}(A_i) + \mathrm{Im}(A_i), p_\ell)$.
- Ozaki-I Karatsuba:
  - $\mathrm{Re}(A_i) + \mathrm{Im}(A_i)$ can exceed the I8 range if each slice uses 8 bits.
  - each slice must be reduced from 8 bits to 7 bits. Need 8 slices with 7-bit slicing.

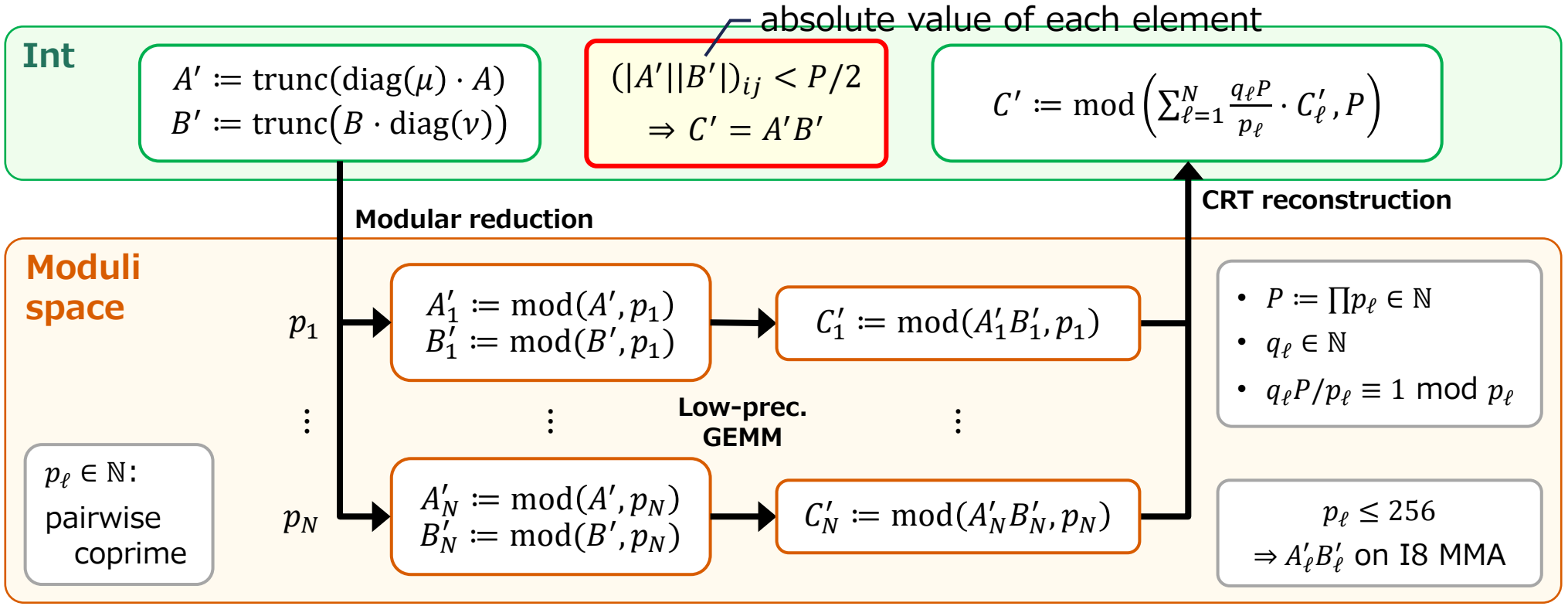
Uniqueness condition

- From the CRT, $C' \equiv A'B' \bmod P \Leftrightarrow C' = A'B' + zP,\ z \in \mathbb{Z}$.



: Possible values of C'

- If $(|A'||B'|)_{ij} < P/2$, the possible value is only one.



Current Limitations

- Small-scale problems
  - Ozaki-I is often faster for small matrices.
  - Ozaki-I mainly relies on simple mantissa slicing, such as bit masking.
  - Ozaki-II requires additional non-GEMM operations:
    - modular reduction,
    - CRT reconstruction,
    - adjustment of scaling vectors to satisfy the uniqueness condition.
  - These overheads are not fully amortized for small matrices.
- Accuracy selection
  - Ozaki-I/II requires an appropriate number of slices/moduli.
  - Too few slices/moduli may violate the required accuracy.
  - Too many slices/moduli increase runtime and workspace.
  - Automatic selection of the number of moduli is future work.
  - For the fields, such as Reliable Computing and Verified Numerical Computations, native FP64 may remain preferable.