芝浦工業大學
SHIBAURA INSTITUTE OF TECHNOLOGY

RIKEN R-CCS

Ozaki Schemes and Applications to Numerical Linear Algebra: Benchmark Results

Katsuhisa Ozaki

Joint work with

Yuki Uchino and Toshiyuki Imamura

First Workshop on Mixed- and Adaptive-Precision Numerics for Scientific Computing (MxP4S)

May 25, 2026, at IPDPS, New Orleans, LA, USA

Today's Topics

FROM HARDWARE TREND TO NUMERICAL METHODS

01 **Background**
Why emulation becomes attractive on modern GPUs
TOPIC

02 **Ozaki-I and II Scheme and its Performance**
Slice-based and CRT-based high-precision matrix multiplication
CORE

03 **Direct Application of Ozaki Schemes**
Applying emulated GEMM directly to numerical algorithms
TOPIC

04 **Refinement (Preconditioning)**
Improving accuracy and efficiency through refinement
ACCURACY

05 **Conclusion**
Summary and future directions
TOPIC

Main flow: motivation → schemes → applications → refinement → conclusion

Ozaki-I / Ozaki-II

Background: NVIDIA GPUs

TFLOPS or TOPS for NVIDIA GPUs

Tensor Core / floating-point throughput by generation

GPU	FP4 TC	INT8 TC	FP8 TC	FP16 TC	FP32	FP64 TC	FP64
Feynman	??	??	??	??	??	??	??
Rubin	35,000	250	17,500	4,000	130	33	33
Blackwell	9,000	4,500	4,500	2,250	80	40	40
Hopper	---	1,978	1,978	989	67	67	34
Ampere	---	624	---	312	20	19	10
Volta	---	---	---	125	16	---	8

Matrix Multiplication Emulation

Low-precision Tensor Core throughput is rising much faster than native FP64 performance. This trend makes matrix-multiplication emulation increasingly attractive.

low precision × many calls → high precision

FP8 / INT8 → FP32-like / FP64-like

400
Rubin single-precision emulation

200
Rubin double-precision emulation

Example emulation performance targets

Key message

Low-precision Tensor Core performance is expanding much faster than native FP64. This strongly motivates matrix-multiplication emulation.

Note: FP32 emulation uses NVIDIA's BFx9 method (not the Ozaki Scheme). On NVIDIA GPUs, the future of INT8 Tensor Cores is uncertain; therefore, a shift toward FP8 is being actively pursued, led primarily by Uchino.

Motivation

DATA-CENTER GPUs

H200
INT8TC : FP64TC
30 : 1

B200
INT8TC : FP64TC
121 : 1

CONSUMER GPUs

RTX4090
INT8TC : FP64
512 : 1

RTX5090
INT8TC : FP64
1020 : 1

FP64

FP64 Emulation

Comparable to FP64 accuracy

INT8 or FP8

VS.

FP64

Advantages of Emulation (Example)

If low-precision arithmetic is **100×** faster than FP64,
then even when emulation incurs a **25×** overhead relative to FP64,
the overall computation remains **4×** faster.

Concept of the Ozaki Scheme

Achieving high-precision matrix multiplication by repeatedly leveraging existing high-speed matrix multiplication

1. Input

Matrix multiplication to be computed at high precision

A (m×k) × B (k×n)

2. Decomposition (into multiple subproblems)

Split the matrices into multiple matrix multiplications

3. Repeatedly call the high-speed matrix multiplication engine

Use an existing highly optimized high-speed matrix multiplication black box

High-Speed Matrix Multiplication Engine (Fast GEMM)

Highly optimized · GPU-accelerated · Cache-optimized · Parallel execution

4. Combination (addition / reconstruction)

Add and reconstruct the computed results to obtain a high-precision result

5. Output

High-precision matrix multiplication result

C = AB (m×n)

Core idea: Realize challenging high-precision matrix multiplication through the "repeated use" of an optimized high-speed matrix multiplication engine

Emulation of DGEMM

GFLOPS

Ozaki-I Scheme: 7 slices

m = k \ n →	1024	2048	4096	8192	10240	12288
1024	3,210	3,320	3,410	3,520	3,580	3,640
2048	3,280	3,420	3,560	3,720	3,820	3,880
4096	3,330	3,520	3,760	3,980	4,080	4,140
8192	3,340	3,580	3,870	4,090	4,180	4,220
10240	3,310	3,560	3,860	4,070	4,160	4,200
12288	3,300	3,540	3,820	4,030	4,120	4,180

2,000 3,000 4,000 5,000

Ozaki-II Scheme: 14 moduli

1024	2048	4096	8192	10240	12288
2,560	3,120	3,820	4,520	4,980	5,280
2,820	3,760	4,880	5,980	6,520	6,920
3,260	4,520	5,960	7,120	7,780	8,220
3,820	5,320	6,880	8,410	8,760	8,874
4,060	5,680	7,280	8,720	8,860	8,870
4,180	5,920	7,540	8,820	8,870	8,874

2,000 4,000 6,000 8,000 10,000

Ozaki-II Scheme reaches up to **8,874 GFLOPS**

Ozaki-I Scheme is stable around **3,300–4,200 GFLOPS**

Ozaki-I Scheme delivers consistent performance, while **Ozaki-II Scheme** scales better and becomes faster as n increases.

SGEMM peak: **18,000 GFLOPS**

DGEMM peak: **360 GFLOPS**

RTX5060Ti

Ozaki-II Scheme: up to **25×** DGEMM

Ozaki-I Scheme: up to **11×** DGEMM

Application to

Matrix Decomposition

High-performance matrix multiplication for factorizations

Ozaki Schemes × Numerical Linear Algebra × GPU Tensor Cores

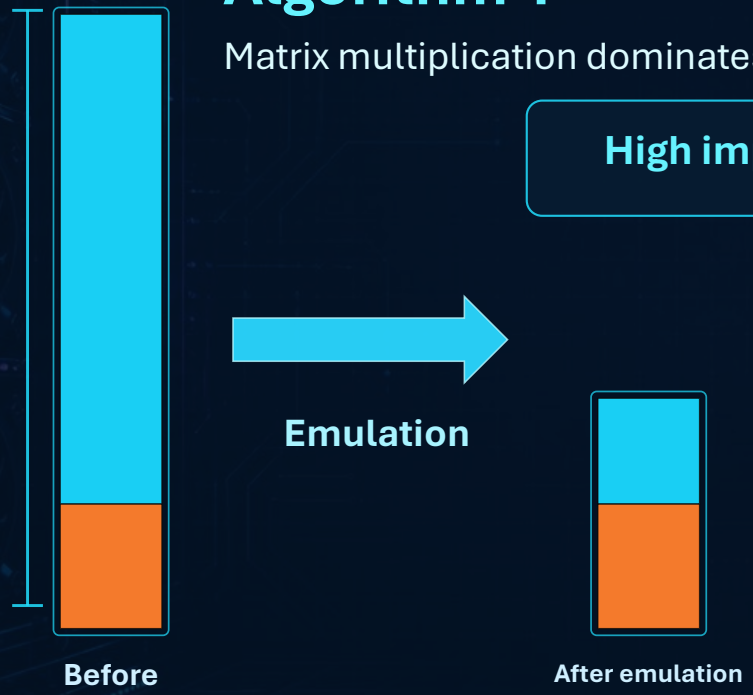
Straightforward Application

Emulation helps only when matrix multiplication dominates the total runtime.

Algorithm 1

Matrix multiplication dominates

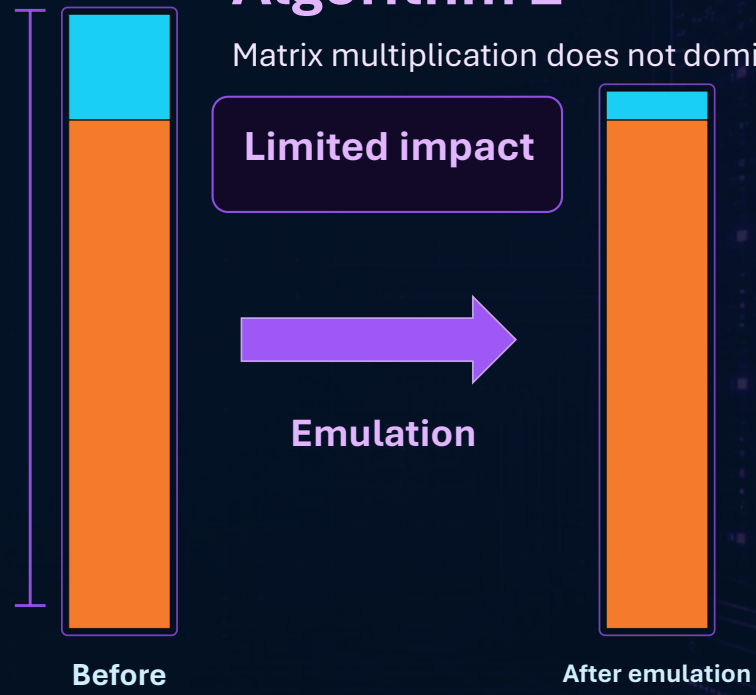
High impact



Algorithm 2

Matrix multiplication does not dominate

Limited impact



Key message: end-to-end speedup is governed by the fraction of time spent in matrix multiplication.

cuSOLVER: Speedup by Emulation

Each entry shows emulated GFLOPS / baseline GFLOPS converted from the original “before → after” measurements

Speedup factors by routine and matrix size

RTX5060Ti · cuSOLVER · larger is better

routine	1024	2048	4096	8192	16384	row max
Xpotrf	1.00×	1.04×	0.99×	1.00×	1.81×	1.81×
Dpotri	0.42×	0.65×	1.02×	1.48×	1.99×	1.99×
Xpotrs	0.75×	2.08×	3.43×	4.54×	5.57×	5.57×
Xgetrs	0.76×	2.40×	3.70×	5.21×	6.68×	6.68×
Xgeqrf	0.78×	0.86×	1.10×	1.32×	1.19×	1.32×
Xgetrf	0.68×	1.03×	1.68×	2.65×	4.37×	4.37×
Xtrtri	0.32×	0.48×	0.71×	1.01×	1.35×	1.35×
Dorgqr	0.30×	0.60×	0.97×	1.20×	1.27×	1.27×
Dormqr	0.85×	2.00×	2.79×	3.29×	3.74×	3.74×

orange outline = maximum in the row

Interpretation

Largest overall speedup

6.68×

Xgetrs
n = 16384

Baseline → Emulation

native GFLOPS → emu. GFLOPS

Key point

Some factorizations and solves benefit strongly when internal matrix-multiplication kernels are accelerated.

Speedup legend

ACCELERATION RATIO

The overall acceleration from emulation depends on how much of the total algorithm is occupied by **matrix multiplication**.

1 Amdahl's law

- Parallelizable fraction: p
- Non-parallelizable fraction: $1 - p$
- Number of processors: N

$$S = \frac{1}{(1-p) + \frac{p}{N}}$$

2 Simple acceleration model using emulation

- Fraction of mat. muls.: p
- Fraction of non-mat. muls.: $1 - p$
- Speedup by the Ozaki Scheme: N

$$S = \frac{1}{(1-p) + \frac{p}{N}}$$

3 Acceleration model using emulation

- Fraction of each mat. mul.: p_i
- Fraction of non-mat.-mul.: $1 - \Sigma p_i$
- Speedup by the Ozaki Scheme for each mat. mul.: N_i

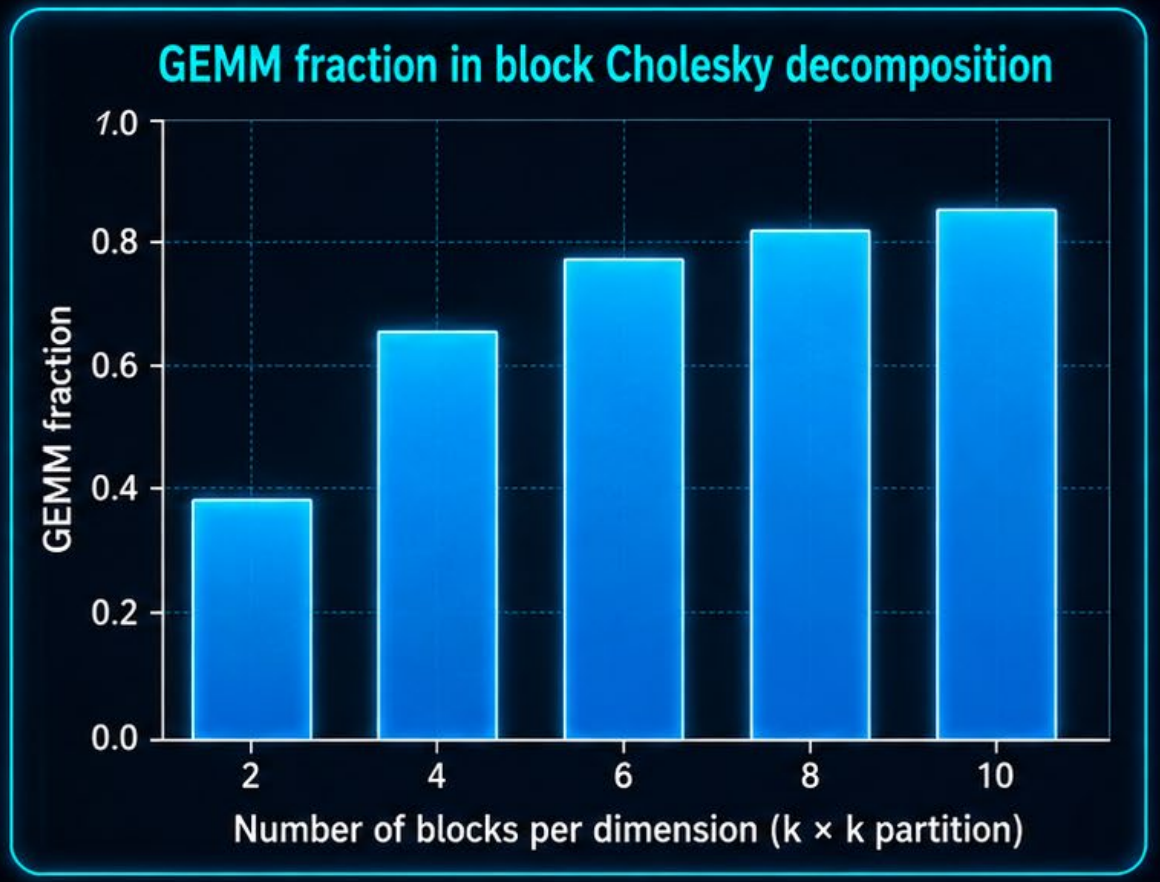
$$S = \frac{1}{(1-\sum p_i) + \sum \frac{p_i}{N_i}}$$

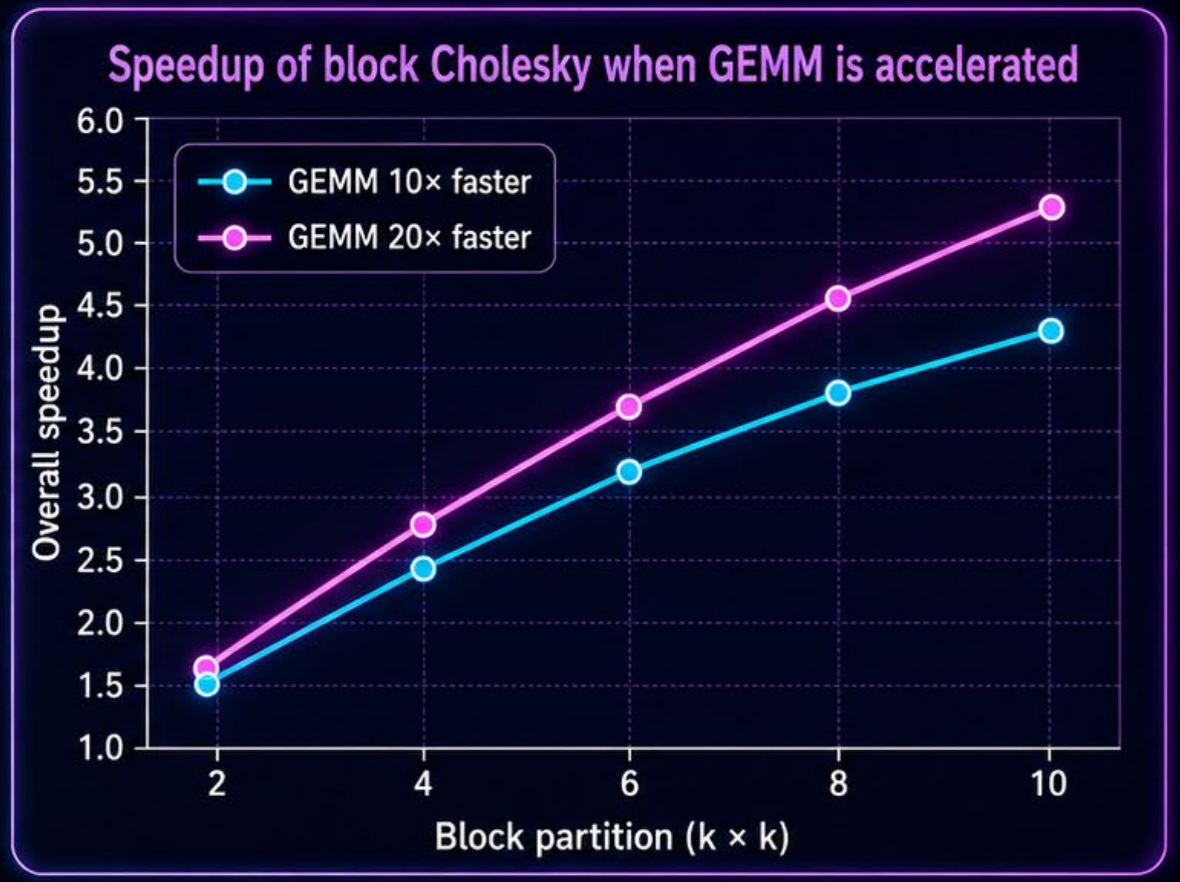
KEY MESSAGE

The speedup achievable by emulation is fundamentally limited by the fraction of the algorithm spent in **matrix multiplication**. The more matrix work, the greater the acceleration.

Simple acceleration model using emulation

- **Ex.:** Block Cholesky Decomposition





Higher GEMM dominance leads to larger overall acceleration from emulation.

Why full performance is not always obtained in block factorizations

GPU DATACF 0.3.12

1 Block LU decomposition image

$$\begin{bmatrix} L_{21} \\ L_{31} \end{bmatrix} \times \begin{bmatrix} U_{12} & U_{13} \end{bmatrix} \Longrightarrow \begin{bmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{bmatrix} - \begin{bmatrix} A_{22} \\ A_{33} \end{bmatrix} - \begin{bmatrix} L_{21} \\ L_{31} \end{bmatrix} - \begin{bmatrix} U_{12} & U_{13} \end{bmatrix}$$

(tall & skinny)

Update of the trailing matrix

2 Full-size GEMM vs. block update GEMM

Full-size GEMM

Square × Square → Square

Good for peak emulation performance

Block update GEMM

Tall-and-skinny × Short-and-wide → $m \times n$

Not a full-size GEMM
Emulation performance may be lower

3 Key messages

- Block LU / Cholesky do not always call full-size square GEMMs.
- The trailing matrix becomes smaller during the factorization.
- To maximize acceleration, we want to use matrix multiplications as large as possible.

Large GEMMs are the most valuable target for matrix-multiplication emulation.

PRECONDITIONING METHOD (AM^{-1})

1 Cholesky Decomposition

- $A = (R_2R_1)^T (R_2R_1)$
- R_1 is preconditioner
- R_1, R_2: FP32 matrices

$$A = (R_2R_1)^T (R_2R_1)$$

2 LU Decomposition

- $PA = L(U_2U_1)$
- U_1 is preconditioner
- L, U_1, U_2: FP32 matrices

$$PA = L(U_2U_1)$$

3 QR Decomposition

- $A = Q(R_2R_1)$
- R_1 is preconditioner
- Q, R_1, R_2: FP32 matrices

$$A = Q(R_2R_1)$$

GOOD RESEARCH BY | Rump, Ogita, Carson–Higham, Terao

Benchmark

RTX5060Ti

GFLOPS for QR decomposition

Decomposition	Method	1024	2048	4096	8192	10240
QR	Native FP32	444	2016	3790	5968	7889
	Native FP64	101	221	273	308	312
	FP64 Emulation	97	263	539	845	1099
	Precond(LU)	327	984	2106	3393	3583
	Precond(QR)	252	813	1882	2943	3141

For the QR-based preconditioning method, we first compute only R in single precision, then compute R^{-1} in single precision, next compute AR^{-1} using GEMMul8, and finally perform a single-precision QR decomposition.

Emulation: 7 slices

Conclusion

Ozaki Schemes and applications to numerical linear algebra

SUMMARY

Benchmarks reported

- Matrix multiplication using Ozaki Schemes
- cuSOLVER routines accelerated by emulation

thanks to NVIDIA's great efforts

Preconditioning

- Useful for ill-conditioned matrices
- Has the potential to accelerate matrix decompositions

effective when GEMM dominates the algorithm

PLEASE NOTE

Emulation may lose accuracy when the magnitudes of matrix entries differ significantly.

The method requires a relatively large amount of working memory.

It may not be very fast for small-sized problems.

Thank you very much for your attention!

Matrix multiplication emulation × preconditioning × high-performance numerical linear algebra