第１５回計算科学技術特論B(2026)
HPC環境における行列積のエミュレーション（２）

芝浦工業大学
尾崎克久

2026年7月23日（木）13:00－14:30

主催：高度情報科学技術研究機構(RIST)
次世代HPC・AI研究開発支援センター(HAIRDESC)
共催：東京大学物性研究所
後援：理化学研究所計算科学研究センター、
計算物質科学人材育成コンソーシアム(PCoMS)

自己紹介

- 芝浦工業大学
- システム理工学部
- 数理科学課程

- 研究内容:
  - 高信頼数値計算
  - 精度保証付き数値計算
  - 丸め誤差解析
  - 数値線形代数

タイ・バンコク郊外にて
良い子は真似しないでください（本物のトラは重いです）

講義概要（7/16, 7/23）

2回の講義で，低精度アクセラレータ時代における尾崎スキームI / IIの考え方と応用を概観する。

7/16 導入と尾崎スキームI

- 導入的内容
- 尾崎スキームIに関する話題
- CPU版・GPU版の考え方

7/23 尾崎スキームIIと関連話題

- 尾崎スキームIIに関する話題
- CPU版・GPU版の実装と性能
- スキームの性質：精度制御・再現性
- 尾崎スキームの応用
- 現状の課題と今後の方向

全体の流れ：尾崎スキームIで基本的な発想を確認し
尾崎スキームIIでCRTによる新しい展開を見る

導入
尾崎スキームI → 尾崎スキームII
性質・応用・課題

なぜ行列積をエミュレーションするのか？（CPU）

FP64より高い精度が必要になると，通常はソフトウェアによる多倍長精度演算に頼るため，計算コストが急増する。

ハードウェア精度の上限

- CPUで標準的に高速サポートされる最大精度は通常 FP64
- FP64を超える精度は専用ハードウェアで直接高速化されにくい
- そのため，より高い精度では実装方法が問題になる

従来の解決策

- MPFRなどの任意精度演算
- double-double / quad-double などの multi-word arithmetic
- 高精度だが，FP64の数十倍以上のコストになることがある

尾崎スキームの発想

- 行列積を複数の低精度行列積へ分解
- 低精度GEMMの高スループットを利用
- 必要な精度だけをエミュレートして，コストを抑える

FP64を超える精度要求 → **通常はソフトウェア多倍長** → **コストが大きい** → **低精度GEMMを多数回利用** → **高精度行列積をエミュレート**

ポイント：高精度そのものをハードウェアで持たなくても，行列積の構造を使えば，高速な低精度ユニットから高精度計算を構成できる。

なぜ行列積をエミュレーションするのか？

近年のNVIDIAのGPUでは INT8 Tensor Core のピーク性能が急増する一方で，Native FP64 は相対的に小さい。

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

FP64 Emulation
低精度ピーク性能を高精度計算に活用する発想

なぜ行列積をエミュレーションするのか？

背景

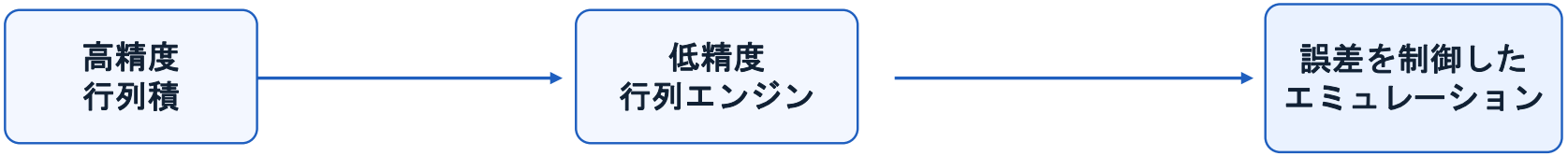
多倍長・多成分演算は高精度だが，行列積ではコストが大きい。
一方，GEMMはBLASの中でも最も最適化が進んだ基本演算である。

狙い

「高精度な結果」を「高速な低精度GEMM」の組合せで得る。
特にTensor CoreやINT8行列エンジンの性能を活用する。

課題

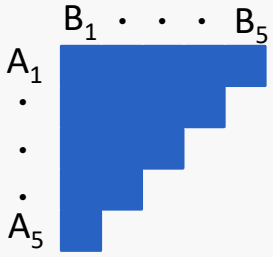
低精度演算を単に使うだけでは丸め誤差が大きい。
誤差なしに計算できる構造を設計する必要がある。



尾崎スキーム I

$$A = A_1 + A_2 + \cdots + A_k, \quad B = B_1 + B_2 + \cdots + B_k$$

- 入力行列を複数の低精度成分に分解する。
- 主要コストは，分解後の成分同士の行列積である。
- 構造に応じて GEMM / TRMM / SYRK などの高性能BLASを使える。



例：三角形状に現れる成分積

必要な行列積数

k slices の場合
k(k+1)/2 回

Level 3 BLAS ルーチンの比較（前回分訂正）

GFLOPS: CUBLAS_EMULATE_DOUBLE_PRECISION = **0**

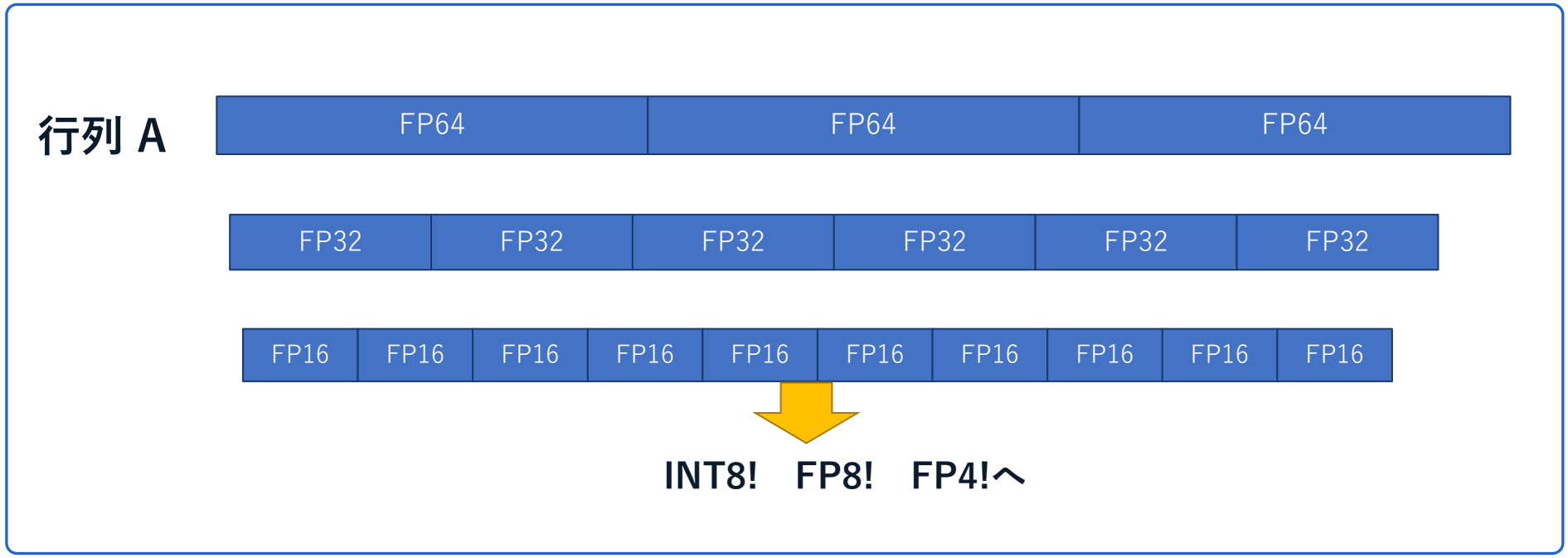
サイズ	DGEMM	DSYMM	DSYRK	DSYR2K	DTRMM	DTRSM
1024	340	282	317	385	196	245
2048	317	280	301	363	280	252
4096	329	311	311	365	307	282
8192	329	322	323	394	342	326

GFLOPS: CUBLAS_EMULATE_DOUBLE_ PRECISION = **1**

サイズ	DGEMM	DSYMM	DSYRK	DSYR2K	DTRMM	DTRSM
1024	910	320	318	385	316	246
2048	1806	326	323	363	324	480
4096	**2446**~~329~~	311	311	365	307	820
8192	2787	327	931	1083	348	1388

RTX5060 Tiを使用した実験結果です

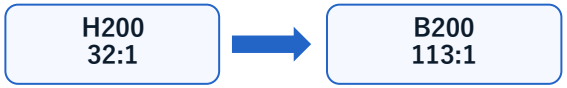
尾崎スキームI


行列 A
FP64
FP64
FP64
FP32
FP32
FP32
FP32
FP32
FP32
FP16
FP16
FP16
FP16
FP16
FP16
FP16
FP16
FP16
FP16
INT8!　FP8!　FP4!へ

尾崎スキームI

1. このままだと尾崎スキームI（GPU版）は
   非効率なアルゴリズムになる可能性
2. FP8とか，そのレベルでFP64の精度を達成しようと思うと，
   もはや多倍長精度演算のような感覚・・・・
3. 多倍長精度演算の勉強をはじめよう
   ⇒ すぐに中国式剰余定理（CRT）がヒット

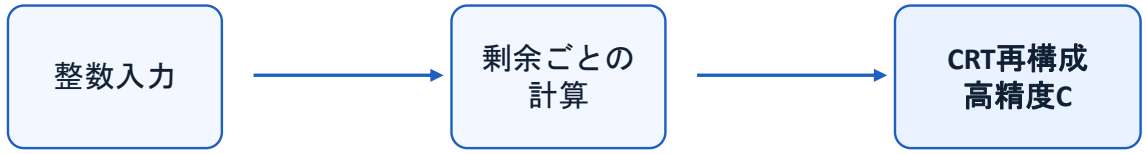
尾崎スキームI



1. 当時H200というGPUがあり，INT8:FP64の性能比は32:1
2. INT8尾崎スキームIでFP64エミュレーションを行うと，
   最低7スライス程度必要 ⇒ 28回の行列積が必要
3. B200ではINT8:FP64の性能比が113:1になり，
   未来では有用と言われていた

でも・・・今勝ちたい

実は整数演算は理論的に成熟している



中国式剰余定理（CRT）を，低精度・高スループットGEMMの利用に結び付ける。

高橋秀俊・石橋善弘，「電子計算機による exact な計算の新方法（modulo p演算の応用）」，『情報処理』，Vol. 1，No. 2，pp. 78–86，1960.

Residue Number System（RNS）と呼ばれる世界：文献多数

FLINT，FFLAS，NTLなどのライブラリがある

中国式剰余定理から尾崎スキームIIへ

中国式剰余定理（CRT）を，低精度・高スループットGEMMの利用に結び付ける。

尾崎スキームIの制約

slices k	2	3	4	5	6	7	8	9	10
muls.	3	6	10	15	21	28	36	45	55

問題点 1

k を 1 増やすと，追加で k+1 回の行列積が必要になる。
精度向上を**細かく制御しにくい**。

問題点 2

$A_1,\ldots,A_k$ と $B_1,\ldots,B_k$ を保持するため，
メモリ消費が大きい。

問題点 3

成分間に絶対値の差があるときに精度を出しにくい。

尾崎スキームIIでは「行列積の回数 s」を直接制御し，CRTで再構成する。

準備：対称剰余

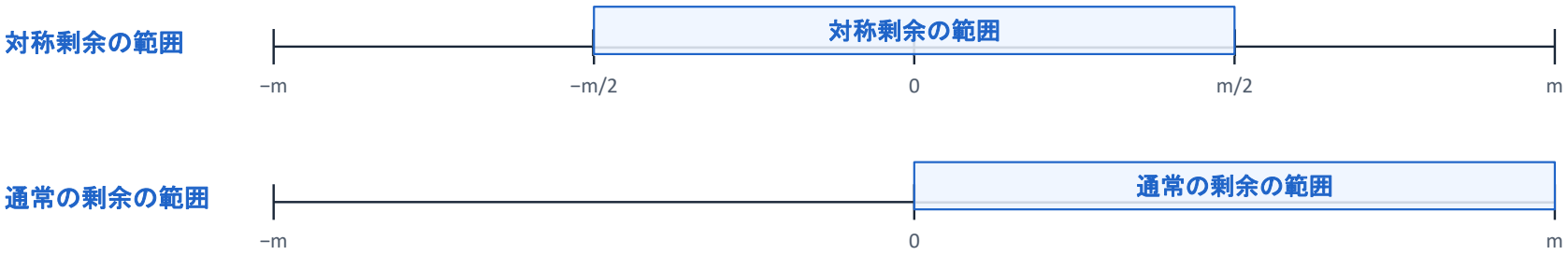
1) 対称剰余

$$r = a \bmod m = a - m \lfloor a/m + 1/2 \rfloor, \quad -m/2 \le r < m/2$$

2) 通常の剰余（非負剰余）

$$r = a \bmod m = a - m \lfloor a/m \rfloor, \quad 0 \le r < m$$

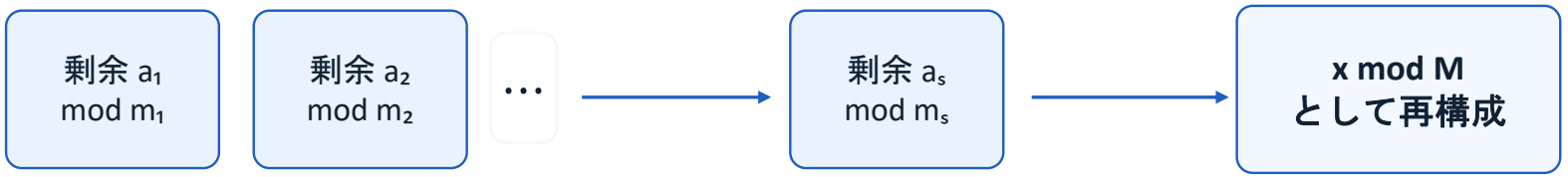
- 対称剰余では，0 を中心とした範囲に剰余を取る。
- 通常の剰余では，$0 \le r < m$ の範囲に剰余を取る。
- 行列に対する mod は，各要素ごとに適用する。



中国式剰余定理（CRT）

$m_1, \ldots, m_s$ は互いに素，　$M = \prod_i m_i$

$x \equiv a_1 \pmod{m_1},\ \ldots,\ x \equiv a_s \pmod{m_s} \Rightarrow x$ は mod M で一意

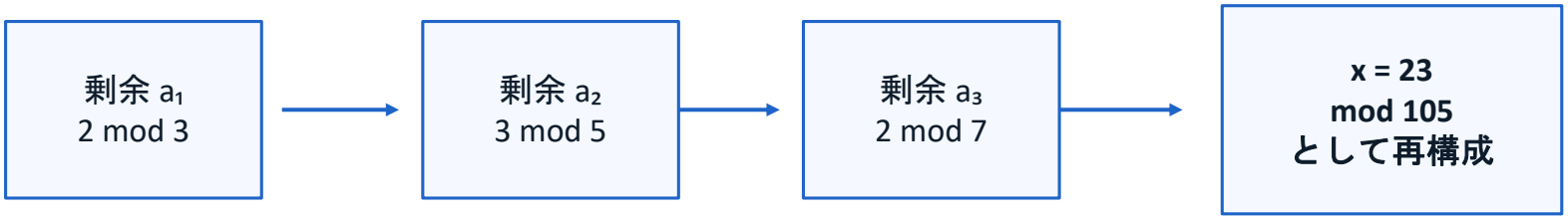


多数の小さな法で計算し，最後に大きな法 M の下で1つに戻す。

中国式剰余定理（CRT）の具体例

$m_1 = 3,\quad m_2 = 5,\quad m_3 = 7$ は互いに素，　$M = 3\times5\times7 = 105$

$x \equiv 2 \pmod 3,\quad x \equiv 3 \pmod 5,\quad x \equiv 2 \pmod 7 \Rightarrow x \equiv 23 \pmod{105}$



確認：23 を 3, 5, 7 で割ると，それぞれ 2, 3, 2 が余る。

CRTの再構成式

$$x \equiv \Sigma_i \, a_i \, M_i \, y_i \pmod{M}$$

定義

$M_i = M / m_i$
y_i は $M_i \, y_i \equiv 1 \pmod{m_i}$ を満たす逆元

行列計算での役割

各法 m_i で得た行列積の結果を，
$M_i y_i$ を重みとして足し合わせる。

- 本論文では直接再構成を用いる。別法として Garner 法も考えられる。

CRTの再構成式の具体例

$m_1=3,\ m_2=5,\ m_3=7,\ M=3\times5\times7=105$

$a_1=2,\ a_2=3,\ a_3=2$ に対して $x \equiv 2 \pmod 3,\ x \equiv 3 \pmod 5,\ x \equiv 2 \pmod 7$

定義

$M_1 = 105/3 = 35$

$M_2 = 105/5 = 21$

$M_3 = 105/7 = 15$

逆元 y_i

$35y_1 \equiv 1 \pmod 3 \Rightarrow y_1 = 2$

$21y_2 \equiv 1 \pmod 5 \Rightarrow y_2 = 1$

$15y_3 \equiv 1 \pmod 7 \Rightarrow y_3 = 1$

再構成

$x \equiv \Sigma\, a_i M_i y_i$

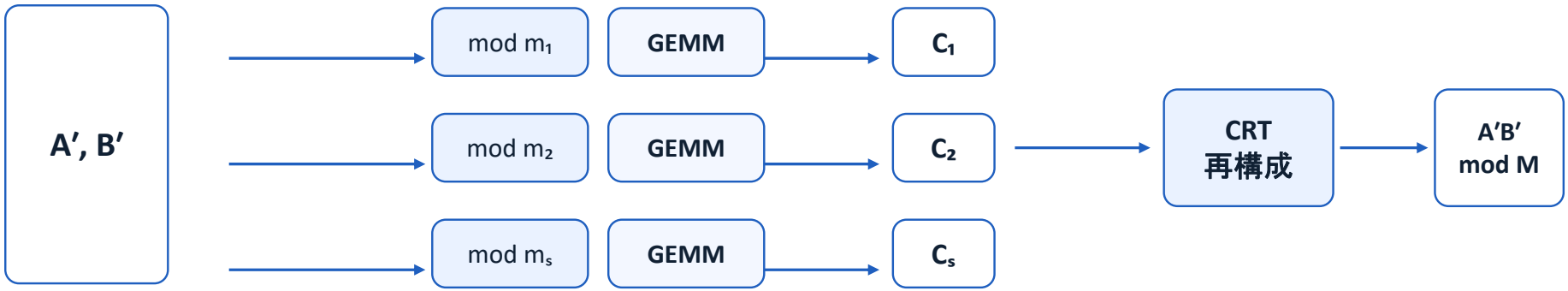
$= 2\cdot35\cdot2 + 3\cdot21\cdot1 + 2\cdot15\cdot1$

$= 233 \equiv 23 \pmod{105}$

確認：23 を 3, 5, 7 で割ると，それぞれ 2, 3, 2 が余る。

CRTを行列積に適用する

$$C_i \equiv A'B' \pmod{m_i}, \quad C \equiv A'B' \equiv \Sigma_i C_i M_i y_i \pmod{M}$$

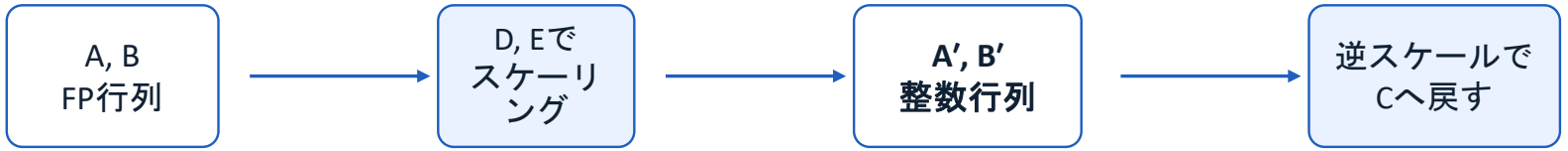


各 C_i の計算は，法ごとの独立した行列積として実行できる。

浮動小数点行列を整数行列に変換する

$$C = AB = D^{-1} D A B E E^{-1} \approx D^{-1} \mathbf{A'B'} E^{-1}$$

$$D A \approx A' \in \mathbb{Z}_{k^A}^{p \times q}, \quad B E \approx B' \in \mathbb{Z}_{k^B}^{q \times r}$$



D, E の対角要素は2のべき乗とし，スケーリング自体の丸め誤差を避ける。

CRT再構成で一意性を保証する条件

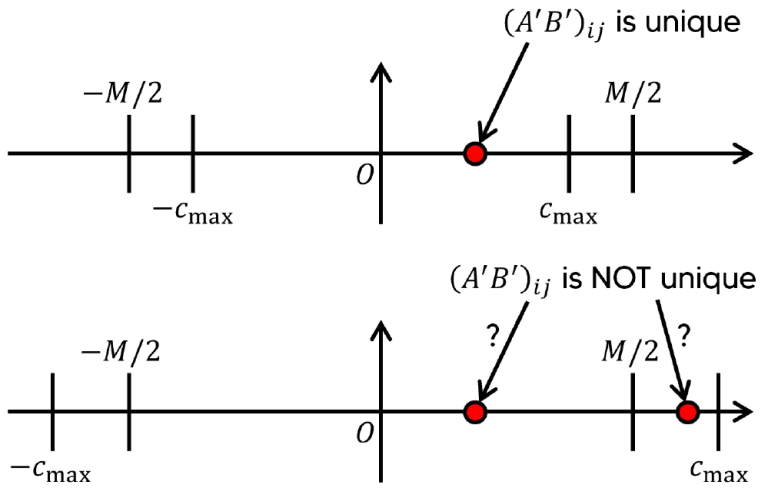
$$c_{\min} \le (A'B')_{ij} \le c_{\max}, \quad c_{\max} - c_{\min} < M$$

簡単化： $c_{\max} = \max_{ij}(|A'||B'|)_{ij}$ として $2c_{\max} < M$

読み方

- CRTの再構成では，値は mod M の範囲でのみ決まる。
- 候補は $y_{ij} + \ell M$ （$\ell \in Z$）の形で複数存在し得る。
- 真の値が入る範囲の幅 $c_{\max} - c_{\min}$ が M 未満なら，一意に選べる。
- $c_{\min} < -M/2$， $c_{\max} > M/2$ では，複数候補が現れ得る。

下段では $c_{\min}$ が $-M/2$ より小さく，$c_{\max}$ が $M/2$ より大きい状況を示している。



Original figure: Ozaki, Uchino, and Imamura, Fig. 2

尾崎スキームII：k, k_A, k_B の意味

k_A と k_B は，A と B を整数化したときに各行列へ割り当てる有効ビット数を表す。

A
入力行列

B
入力行列

D A ≈ A′
A′ ∈ $\mathbb{Z}_{k\_A}$

B E ≈ B′
B′ ∈ $\mathbb{Z}_{k\_B}$

A′ B′
CRTで復元

D^{-1} X E^{-1}
浮動小数点数化

k_A
- A′ 側の整数化精度
- 大きいほどAの量子化誤差が小さい

k_B
- B′ 側の整数化精度
- 混合精度では非対称に設定可能

$$\max_{ij}\left(|A'|\,|B'|\right)_{ij} \le q\,2^{\{k\_A+k\_B\}} < M/2$$

$$k\_A+k\_B := \mathrm{floor}\left(\log_2((M/2-1)/q)\right)$$

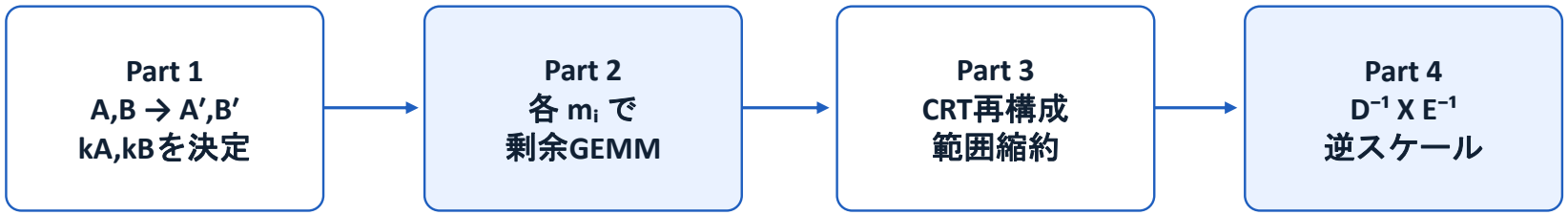
k := k_A = k_B と置く場合
- A と B に同じ精度を割り当てるときの略記
- $k = \mathrm{floor}(1/2\,\log_2((M/2-1)/q))$

補足：$\mathbb{Z}_k$（¥mathbb{Z}_k）は「kビット程度で表現できる整数化後の値の集合」を表す略記。
A′ ∈$\mathbb{Z}_{k\_A}$, B′ ∈$\mathbb{Z}_{k\_B}$ は，各要素がそれぞれ k_A, k_B ビットに収まることを意味する。

s を増やす → M=∏m_i が大きくなる → k_A+k_B を大きくできる

尾崎スキームII：全体像

Four parts: integerization, modular GEMMs, reconstruction, inverse scaling



Ozaki I との違い

成分行列を多数保持するのではなく，
A'_i, B'_i を各法ごとに作り，
C_i を得た後は**すぐ破棄**できる。

重要なパラメータ

s：法の数 = 行列積の回数
M：再構成できる範囲
kA,kB：整数化で保持するビット数

GPU版：INT8 Tensor Coresを使う場合

法の選択

$2 \le m_i \le 256$, かつ互いに素となるように選ぶ
対称剰余は INT8 の範囲 −128,…,127 に収まる

誤差なし行列積

INT8 Tensor Core の積は INT32 に蓄積される
$q < 2^{17}$ なら $A'_iB'_i$ を誤差なく計算できる

例：s = 16 では 256, 255, 253, 251, 247, …, 191 を使用

INT8 Tensor Core版のアルゴリズム

1. FP64 から INT8 剰余へ

1. D,E のシフト値を決定
2. $A' = \mathrm{trunc}(DA)$, $B' = \mathrm{trunc}(BE)$
3. $A'_i = A' \bmod m_i$, $B'_i = B' \bmod m_i$

2. INT8 TC GEMM

4. $C'_i = A'_i B'_i$ を計算
5. C_i を $\bmod m_i$ に戻す

3. CRT再構成と逆スケール

6. $\Sigma C_i M_i y_i$ を高精度に蓄積
7. mod M による範囲縮約(F)
8. $C = D^{-1} F E^{-1}$

s 個の法を使うため，主要な行列積は s 回。s が精度とコストを直接決める。

精度制御：法の数 s が鍵になる

整数化精度

s を増やすと $M=\prod m_i$ が大きくなり，
kA,kB を大きく取れる。
その結果，整数化の打ち切り誤差が減少する。

INT8 TCでの目安

s=16 で k≈53 が期待され，FP64エミュレーションに対応する
Ozaki Iで同程度の精度には7〜8 slices,
すなわち28〜36回の行列積が必要。

行列積回数と精度制御の比較

精度向上の基本関係： s ↑ ⇒ M ↑ ⇒ k ↑
⇒ 誤差 ↓

論文の実験では，ϕ= 0.5 程度の行列で s= 14〜15 がDGEMM レベルの精度の目安

ϕ

GPU版：精度評価

原論文 Fig. 5：GH200上で，法の個数を増やしたときの OS II-fast / OS II-accu の相対誤差を比較する。

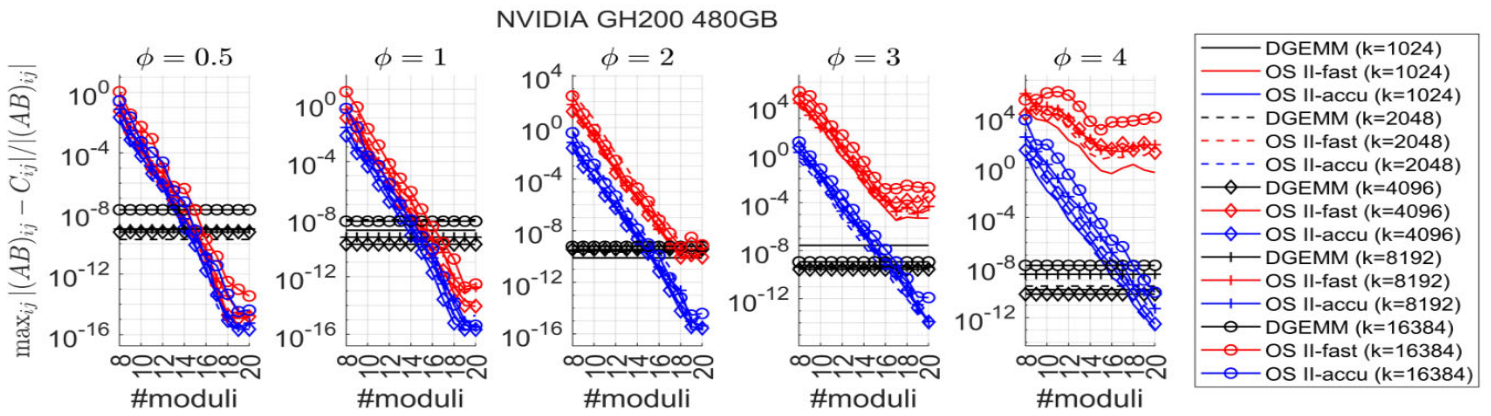


Figure 5. Accuracy comparison for $p = r = 1024$ and $q \in \{1024, 2048, 4096, 8192, 16384\}$ on NVIDIA GH200 Grace Hopper Superchip

横軸

法の個数 #moduli。OS II では法を増やすほど復元可能な有効ビット数が増える。

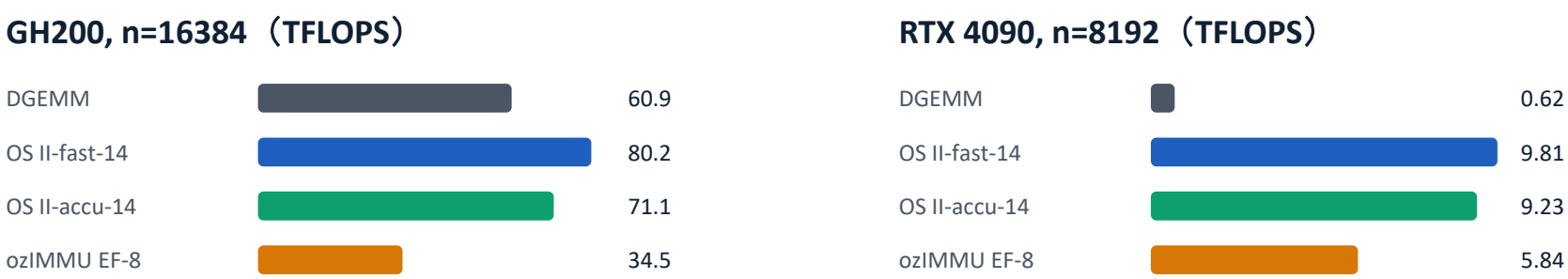
読み取り

ϕ=0.5では14〜15個程度の法でDGEMM水準の精度に到達する。

fast と accu

accu は上界推定を直接計算するため，難しい分布では fast より高精度になりやすい。

GPU実験：FP64エミュレーション



大規模行列では，OS II-fast/accu がDGEMM相当の精度を目指しながら，
DGEMMや従来ozIMMUより高いスループットを示した

CPU版：FP64 GEMMで多成分演算をエミュレート

$$q\, m_i^2 \le 4u^{-1}$$

- この条件により，剰余行列 $A'_iB'_i$ をFP64 GEMMで丸め誤差なく計算できる
- 法 m_i は互いに素ならよいが，CPU版では大きな法を選べるので素数を使う
- 多成分行列 $A=\Sigma A^{(i)}$, $B=\Sigma B^{(i)}$ に対しても同じ考え方で適用できる。

uは単位の相対丸め(unit roundoff): FP64なら2^{-53}
qは行列のサイズ（内積方向）

q=1024ならばm_iは$2^{22.5}$以下

CPU版：精度評価

原論文 Fig. 8：Intel Core i9-10980XE 上で，Ozaki Scheme I / II の最大相対誤差を比較する。

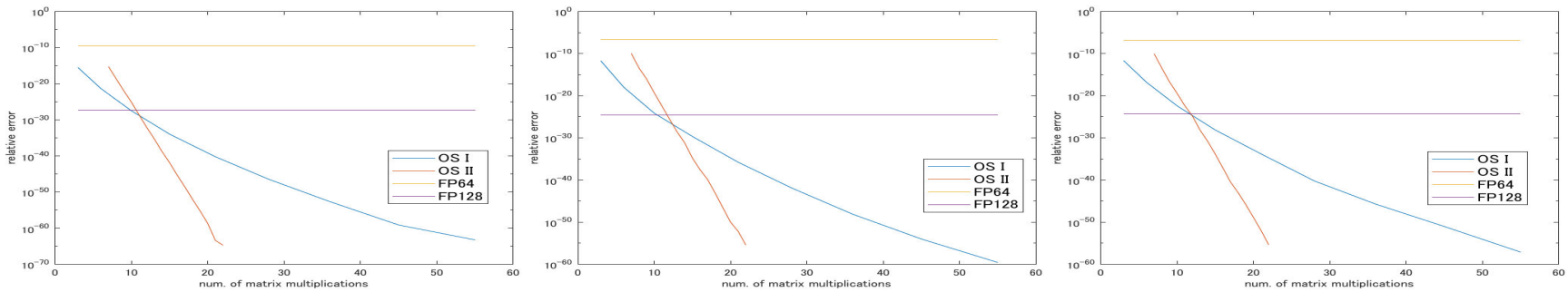


Figure 8. Accuracy comparison for $p = r = q \in \{1000, 8000, 15000\}$ on Intel® Core™ i9-10980XE processor. Horizontal values represent number of moduli.

横軸

- 行列積回数
- OS IIでは法の個数を増やすことで精度が上がる

CPU版の位置づけ

- FP64 arithmeticを基礎演算に利用
- quad-word相当の高精度行列積を狙う

読み取り

- OS IIは法の個数を増やすと誤差が急速に低下
- FP64を超える精度域まで到達できる

ポイント：GPU版だけでなく，CPU上の多倍長精度行列積エミュレーションにも尾崎スキームIIを適用できる。

CPU実験：quad-word相当の行列積

精度傾向

行列積回数が少ない領域ではOzaki Iが有利な場合がある一方，回数を増やすとOzaki IIは対数スケールでほぼ線形に精度が改善する。

性能傾向

nが大きくなるほど Part 2-b（GEMM）の比率比率が高まり，GEMM指向の設計が効く。

CPU	比較	最大速度向上
Core i7-8665U	OS I-10 vs OS II-22	2.29×（n=4000）
Core i9-10980XE	OS I-10 vs OS II-22	2.12×（n=8000）

n=1000 ではオーバーヘッドが相対的に大きく，Ozaki IIが必ずしも速くならない点に注意。

まとめ：尾崎スキームIIの要点

1. 中国式剰余定理により，小さな法で得た行列積の剰余から A′B′ を再構成する。
2. D,E による2のべき乗スケーリングで，浮動小数点行列を整数行列に変換する。
3. 法の数 s が，行列積回数・M・整数化ビット数 k を直接制御する。
4. INT8 Tensor Core版ではFP64エミュレーション，
   CPU版ではquad-word相当のエミュレーションを実証した。

尾崎スキームの特徴

数値再現性

数値再現性とは

同じアルゴリズム・入力データで実行したときに，
常に同じ数値結果が得られる性質のこと。

なぜ重要か

- デバッグや検証のために，同じ計算を繰り返しても結果が一致することが必要
- 小さな差が長い計算で増幅し，分岐や収束性に影響する可能性がある
- 科学計算では，結果の信頼性・比較可能性が求められる

浮動小数点の非結合性

浮動小数点演算は一般に
結合法が成り立たない。

例（丸めは最も近い値）

$(1e16 + 1) - 1e16 = 0$

$1e16 - 1e16 + 1 = 1$

→ **計算順序が変わると結果が異なる**

再現性を損なう主な要因

- 並列計算での加算順序の違い（スレッド数・スケジューリング・ループ順序など）
- FMA の有無や最適化の違い（コンパイラ・CPU/GPU の設定）
- 異なるライブラリやハードウェアでの丸めモード・演算精度の違い

対策の例

- ✓ 演算順序を固定する
- ✓ スレッド数・スケジューリングを固定する
- ✓ 丸めモードを明示的に設定する
- ✓ 同一環境（同一コンパイラ・ライブラリ・ハードウェア）で実行する

→ **数値再現性と数値安定性は異なる概念**

再現性 = 毎回同じ結果が得られること， 安定性 = 小さな誤差が大きく増幅しないこと

数値再現性

固定した順序で行列積を足す

○：無誤差で計算できる部分積

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

○：無誤差で計算

尾崎スキームIでの再現性

$$A := A_1 + \cdots + A_6$$
$$B := B_1 + \cdots + B_6$$

行列積の和を固定の順番で足す

↓

再現性を持つ

加算順序を固定することで，並列実行でも同じ入力に対して同じ数値結果を得る。

数値再現性：尾崎スキームII

同じ整数化・同じ法・同じCRT再構成を用いれば，演算順序に依存しない結果を得やすい

浮動小数点 A, B → 整数化 A', B' → 剰余ごとの整数GEMM → CRT再構成 → 浮動小数点数化 → 再現性を持つ C

$C_i = A_i B_i \pmod{m_i}$ を各 i で独立に計算し，$C = \mathrm{CRT}(C_1, \ldots, C_s)$ を固定式で再構成

通常の浮動小数点GEMM

加算の順序・並列スケジューリング・FMAの有無などで丸めの入り方が変わり，bitwiseに一致しない場合

尾崎スキームII

剰余ごとの計算は整数演算として扱い，最後にCRTを固定順序で再構成するため，同じ設定なら結果が固定

注意：再現性は「同じ設定で同じ結果」という性質であり，精度や数値安定性とは別の概念。

精度のコントロール

● 尾崎スキーム I と II は，精度を自由にコントロール可能（FP32 以上 FP64 以下）

- 内部の整数演算は誤差なく実行されるため，最終精度は「整数化のスケール」や
  「使用する法の数（スライス数と窓・素数個数）」で決まる。
- 要求する有効ビット数（小数部の桁数）に合わせてパラメータを設定できる。

尾崎スキーム I（スライス法）

スライス数 s を増減

少ない → 多い

保持できる有効ビット数 k が増加

k は s にほぼ比例して増加

尾崎スキーム II（CRT 法）

使用する互いに素な法の個数 s を増減

少ない → 多い

保持できる有効ビット数 k が増加

k は使用する法（moduli）の積の対数に比例

精度のコントロール範囲（目安）

ポイント

- ✓ 必要な精度に合わせて s（スライス数）または t（法の個数）を選ぶだけ
- ✓ 同じ設定なら，実行環境が変わっても同一の結果を再現できる

→ **尾崎スキーム I と II は，FP32 以上 FP64 以下の任意の精度を，数値再現性を保ったまま実現可能**

用途に応じて，必要な精度と計算コストのバランスを自由に調整できる

精度のコントロール

尾崎スキーム I / II では，入力行列 A, B の型が異なっていても柔軟に対応できる

例：A が FP32，B が FP64 の場合でも，必要な精度に応じて内部パラメータを調整できる。

入力型の例

A: FP32，B: FP32，C: FP64
A: FP32，B: FP64，C: FP64
A: FP64，B: FP32，C: FP64
A: FP64，B: FP64，C: FP64

各行列のスケールや分割・整数化を個別に設定できる。

どう精度を決めるか

- 欲しい最終精度（例：FP32以上 FP64以下）に応じて設定する
- 尾崎スキーム I：スライス数 s を調整
- 尾崎スキーム II：法の個数 t や法の積 M を調整
- 入力型が異なっても，同じ考え方で必要ビット数を確保する

ポイント

- 混合精度入力にも対応可能
- 必要な精度と計算コストのバランスを調整可能
- 用途に応じて FP32 超〜FP64 級の精度を狙える

精度のコントロール（Cholesky-QR法）

A = QR
Q: 直交行列
R: 上三角行列

Cholesky-QR法は，通常のHouseholder QRの代わりに A^TA の Cholesky 分解を使って QR 分解を行う方法である。

CholeskyQR

```
function [Q, R] = cholqr(A)
    S = Aᵀ A;
    R = chol(S);
    Q = A / R;
end
```

① A^TAを作る → ② Cholesky → ③ 三角行列方程式（TRSM）

※ 行列積エミュレーションの主な対象

CholeskyQR2

```
function [Q, R] = cholqr2(A)
    [Q₁, R₁] = cholqr(A);
    [Q,  R₂] = cholqr(Q₁);
    R = R₂ R₁;
end
```

CholeskyQRを2回適用し，Qの直交性を改善する

Cholesky QRは加速部分がかなりある

精度のコントロール（Cholesky-QR法）

A = QR
Q: 直交行列
R: 上三角行列

Cholesky-QR法は，通常のHouseholder QRの代わりに A^TA の Cholesky 分解を使って QR 分解を行う方法である。

CholeskyQR

```
function [Q, R] = cholqr_(A)
    S = double(single(Aᵀ))*
        double(single(A));
    R = chol(S);
    Q = A / R;
end
```

① A^TAを作る → ② Cholesky → ③ 三角行列方程式（TRSM）

※ 行列積エミュレーションの主な対象

CholeskyQR2

```
function [Q, R] = cholqr2(A)
    [Q₁, R₁] = cholqr_(A);
    [Q,  R₂] = cholqr(Q₁);
    R = R₂ R₁;
end
```

CholeskyQRを2回適用し，Qの直交性を改善する

変更前の残差ノルム： 4.3534e-15
変更後の残差ノルム： 4.7279e-15

Cholesky QRは加速部分がかなりある

尾崎スキームの応用編

アムダールの法則

$$S(p) = 1 / ((1 - \alpha) + \alpha / p)$$

α：並列化可能な割合，　$1 - \alpha$：逐次実行部分，　p：プロセッサ数

要点

- 全体の一部しか並列化できないと，プロセッサ数を増やしても速度向上には上限がある。
- 逐次部分 $1 - \alpha$ が大きいほど，高速化は頭打ちになりやすい。
- $p \to \infty$ の極限では，最大速度向上は $1/(1 - \alpha)$ となる。

例　$\alpha = 0.9$ のとき

$p = 10$ なら　$S(10) = 1 / (0.1 + 0.9/10) \approx 5.26$

$p \to \infty$ でも 最大 10 倍

意味

- 並列化率 α が高いほど有利
- しかし逐次部分が少しでも残ると限界がある
- アルゴリズム設計では逐次部分の削減が重要

→　並列部分を高速化しても，逐次部分が全体性能の限界を決める

アルゴリズム全体で見た行列積エミュレーションの効果

アルゴリズム中に k 個の行列積があり，それぞれのエミュレーション速度向上率が異なる場合

一般モデル

元の総実行時間

$$T = T\_other + T_1 + T_2 + \cdots + T_k$$

各行列積 j の加速率を s_j とすると

$$T' = T\_other + T_1/s_1 + T_2/s_2 + \cdots + T_k/s_k$$

したがって，全体の速度向上率は

$$S = T / T'$$

アムダール型の正規化

$f_j = T_j / T$，　$f\_other = T\_other / T$ とおくと

$$S = 1 / (f\_other + \Sigma_j f_j / s_j)$$

ここで

$$f\_other + f_1 + f_2 + \cdots + f_k = 1$$

すべての行列積が同じ加速率 s を持つなら

$$S = 1 / ((1-\alpha) + \alpha/s)$$

解釈

- 寄与率 f_j が大きい行列積ほど，全体性能への影響が大きい。
- 同じ行列積でも，加速率 s_j が高いほど効果が大きい。
- 一部だけ非常に速くなっても，他の部分が律速になる。

線形代数への応用（Cholesky分解）

Cholesky分解は，対称正定値行列 A を $A = R^{T}R$ または $A = LL^{T}$ と分解する方法

2×2ブロックで見た計算の流れ

ブロック化すると，最後の更新に大きな行列積が現れる

各ステップの役割

1. **小さいブロックを分解**
   $A_{11} = R_{11}{}^{T}R_{11}$ を計算する。
2. **三角行列を係数とする連立方程式**
   $R_{11}{}^{T} R_{12} = A_{12}$ を解く。
3. **更新：行列積が支配的**
   $A_{22} \leftarrow A_{22} - R_{12}{}^{T}R_{12}$。
   ここが GEMM/SYRK 型の高コスト部分。

尾崎スキームの狙い：分解全体ではなく，支配的な行列積・ランク更新をエミュレーションで高速化する

線形代数への応用（Cholesky分解）

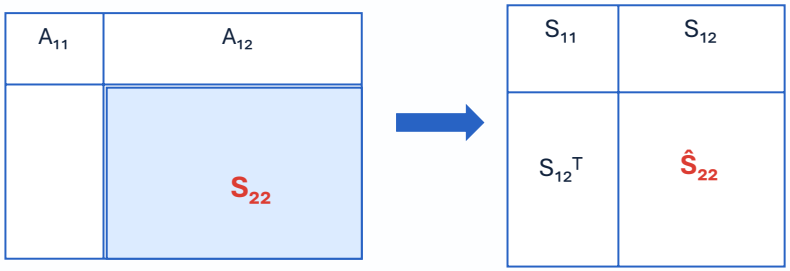
対称正定値行列 A に対して $A = R^T R$ を満たす上三角行列 R を求める。
ブロック版では Schur complement S_{22} を再び同様に分割して処理する。

全体のブロック Cholesky

① $A_{11} = R_{11}^T R_{11}$	②
	③ $S_{22} = A_{22} - R_{12}^T R_{12}$

① Cholesky → ② 三角行列方程式
→ ③ 行列積で Schur complement を更新

S_{22} をさらに分割して再帰的に進める



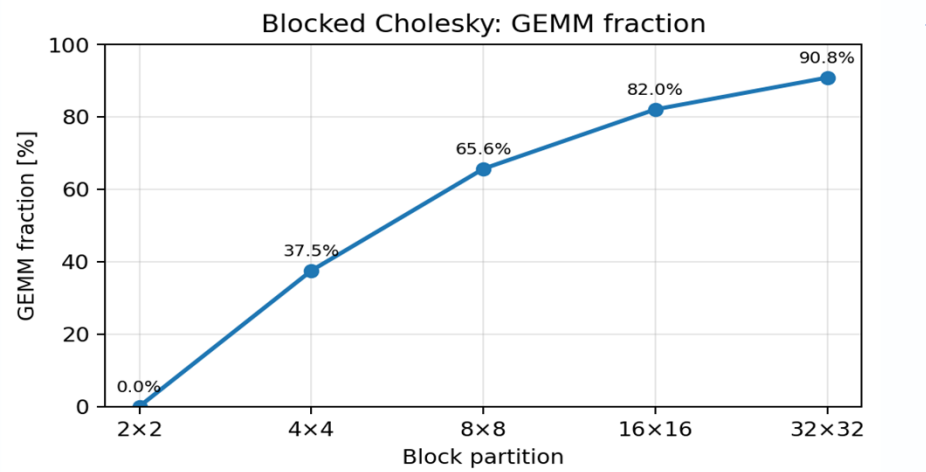
④ $S_{11} = R_{22,11}^T R_{22,11}$
⑤ $S_{12} = R_{22,11}^T R_{22,12}$
⑥ $T_{22} = \hat{S}_{22} - R_{22,12}^T R_{22,12}$

大きいブロックを行列積で更新しながら，
$S_{22} \to \hat{S}_{22} \to \cdots$ と同じ操作を繰り返す。

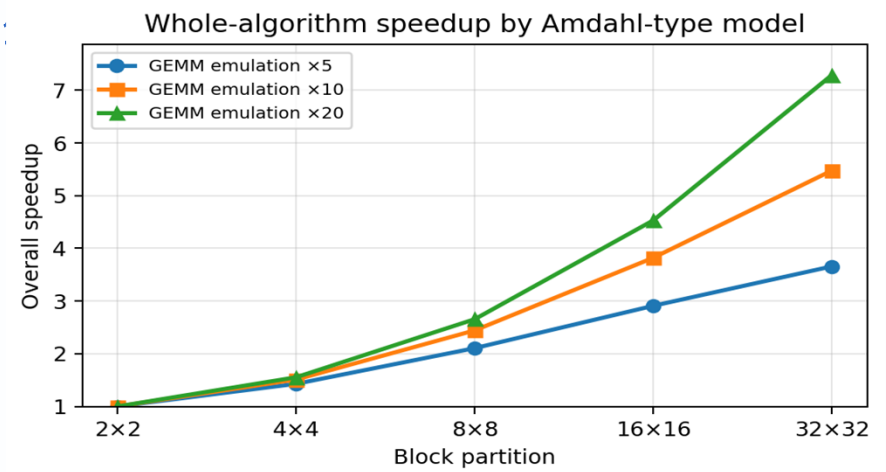
Cholesky分解におけるGEMM fraction と全体加速

仮定：行列を q×q の等サイズブロックに分割し，オフ対角の trailing update に現れる GEMM のみがエミュレーションで一律に加速されるとする

左：GEMM fraction



右：GEMM が一律 ×5, ×10, ×20 のときの



注：TRSM も加速対象に含めれば，実際の全体速度向上はこの簡易モデルより大きくなる可能性がある。

線形代数への応用（Cholesky分解）

n = 8192 のブロック Cholesky 分解について，性能（GFLOPS）と残差を比較する。

性能比較（GFLOPS）

ブロック サイズ	FP32	5 Slices	6 Slices	7 Slices	Performant	FP64
512	5595	687	608	545	410	254
1024		802	715	632	487	
2048		830	748	667	525	
4096		756	663	578	435	

残差比較（相対残差ノルム）

手法	FP32	5 Slices	6 Slices	7 Slices	Performant	FP64
残差	2.630e-07	3.658e-12	1.595e-14	4.264e-16	4.191e-16	4.511e-16

相対残差ノルム：$\|A - R^T R\|_\infty / \|A\|_\infty$

線形代数への応用（Cholesky分解）

行列サイズによる性能の変化と，条件数による残差の変化を比較する

計算速度（GFLOPS）

行列サイズ	FP32	5 slices	6 slices	7 slices	performant	FP64
1024	789	69	68	66	59	112
2048	2373	227	217	216	188	216
4096	3859	461	442	422	395	247
8192	5318	833	749	673	534	254

相対残差ノルム，n=8192

条件数	FP32	5 slices	6 slices	7 slices	performant	FP64
10^3	5.417e-07	3.979e-11	1.781e-13	1.850e-15	1.693e-15	3.103e-16
10^6	7.335e-07	3.590e-11	1.563e-13	1.546e-15	1.363e-15	2.779e-16
10^9	failed	2.830e-11	1.221e-13	2.072e-15	2.063e-15	3.056e-16
10^{12}	failed	3.098e-11	1.331e-13	2.153e-15	2.094e-15	2.973e-16

cuSOLVER における行列積エミュレーション

- cuSOLVER は，NVIDIA GPU上で LU分解，QR分解，Cholesky分解，固有値問題などの密行列・疎行列の数値線形代数計算を高速に実行するための CUDA ライブラリである。

エミュレーションの有効化

1. `cusolverDnSetMathMode` 関数で計算精度のモードを `CUSOLVER_FP64_EMULATED_FIXEDPOINT_MATH` に設定する。
2. `cusolverDnSetEmulationStrategy` 関数でエミュレーションの活用方法を設定する。
3. これにより，行列積のエミュレーションを有効にできる。

cuSOLVER ルーチンの概要

Routine	Description
`cusolverDnXpotrf`	Cholesky 分解 $A = LL^T$ または $A = U^TU$ を計算する.
`cusolverDnDpotri`	Cholesky 分解済みの対称正定値行列から逆行列を計算する.
`cusolverDnXpotrs`	Cholesky 分解済みの対称正定値連立一次方程式を解く.
`cusolverDnXgetrs`	LU 分解済みの一般行列に対する連立一次方程式を解く.
`cusolverDnXgeqrf`	一般行列の QR 分解を Householder 変換に基づいて計算する.
`cusolverDnXgetrf`	部分ピボット付き LU 分解を計算する.
`cusolverDnXtrtri`	三角行列の逆行列を計算する.
`cusolverDnDorgqr`	QR 分解で得られた Householder ベクトルから直交行列 Q を明示的に生成する.
`cusolverDnDormqr`	QR 分解で得られた直交行列 Q または Q^T を行列に作用させる.

※ 表記は cuSOLVER Dense API 名に基づく.

cuSOLVER(+行列積エミュレーション)

RTX5060Ti

各ルーチンについて，サイズごとの性能比を比較する。1より大きい値は高速化，1より小さい値は低速化を表す。

	1024	2048	4096	8192	16384
cusolverDnXpotrf	1.00	1.04	0.99	1.00	1.81
cusolverDnDpotri	0.42	0.65	1.02	1.48	1.99
cusolverDnXpotrs	0.75	2.08	3.43	4.54	5.57
cusolverDnXgetrs	0.76	2.40	3.70	5.21	6.68
cusolverDnXgeqrf	0.78	0.86	1.10	1.32	1.19
cusolverDnXgetrf	0.68	1.03	1.68	2.65	4.37
cusolverDnXtrtri	0.32	0.48	0.71	1.01	1.35
cusolverDnDorgqr	0.30	0.60	0.97	1.20	1.27
cusolverDnDormqr	0.85	2.00	2.79	3.29	3.74

Thanks to NVIDIA's great work

大きいサイズでは potrs / getrs / getrf / ormqr などで効果が大きい。

cuEST：尾崎スキームIIを使用したライブラリ

cuEST は，量子化学・電子状態計算の重い行列計算を GPU 上で高速化する CUDA-X ライブラリ。

cuESTとは

- 量子化学・電子状態計算向け CUDA-X
- DFT/SCF の重い行列計算を GPU 化
- 部品単位 API で既存コードへ統合

尾崎スキームとの関係

- hybrid DFT の DF-K 交換行列が対象
- Ozaki I/II の固定小数点 GEMM を利用
- FP64相当の精度を低精度ユニットで実現

精度制御

- Ozaki I： slice count
- Ozaki II： modulus count
- 性能と精度のバランスを調整

GEMMul8：尾崎スキームIIを実現するライブラリ

GEMMul8 は，Ozaki Scheme II と CRT に基づき，
INT8/FP8 matrix engine で高精度GEMMをエミュレートするライブラリ（理研：内野氏）

何をするライブラリか

- SGEMM/DGEMM/CGEMM/ZGEMM のエミュレーション
- 低精度ユニットを使い，高スループット化を狙う
- CRTの法の数で精度とコストを調整

Level-3 BLAS への拡張

- GEMMだけでなく，SYRK/SYR2K, HERK/HER2K なども対象
- TRMM/TRSM など三角行列系ルーチンにも拡張
- 複素演算では Hermitian 系ルーチンも扱う

実行形態

- CUDA と HIP バックエンドをサポート
- 直接API利用に加え，Hook mode で cuBLAS/hipBLAS 呼び出しを置換可能
- INT8 / FP8 backend を選択可能

戻り値として scaling / low-precision GEMM / re-quantization / CRT reduction などの内部フェーズ時間を取得でき，性能分析にも使いやすい。

GEMMul8：Level-3 BLAS 数値実験結果

代表値：results_08/GB200 の square case 最大サイズ n=32768。比較対象は OS2-i8-accu-14 に固定

実数倍精度ルーチン

Routine	代表条件	Native FP64 TFLOPS	OSII-i8-acc14 TFLOPS	Speedup
DGEMM	n=n=k	39.0	183.1	4.7
DSYMM	left/lower	38.0	183.5	4.8
DSYRK	lower, N	39.1	171.4	4.4
DSYR2K	lower, N	28.2	181.1	6.4
DSYRKX	lower, N	28.3	159.8	5.7
DTRMM	left/lower/N/nonunit	38.0	166.1	4.4
DTRSM	left/lower/N/nonunit	35.7	79.6	2.2
DTRTRMM（特注）	lower/lower/N,N/nonunit	6.5	117.0	18.0

注：DTRTRMM の native は結果ファイル上の “DTRTRMM (GEMM)” 行。Speedup = OSII-i8-acc14 / Native FP64

GEMMul8：Hermitian 系 Level-3 BLAS

複素 Hermitian 系ルーチンでも，OSII-i8-acc14 による大きな高速化が得られる。

複素倍精度 Hermitian ルーチン

Routine	代表条件	Native FP64 TFLOPS	OSII-i8-acc14 TFLOPS	Speedup
ZHEMM	left/lower	35.9	251.3	7.0
ZHERK	lower, N	35.4	232.7	6.6
ZHER2K	lower, N	28.2	247.0	8.8
ZHERKX	lower, N	28.2	219.4	7.8

ポイント

ZHER2K, ZHERKX では native FP64 に対し約 8 倍前後。

読み方

高速版ではなく，精度重視の acc14 設定で比較。

議論・論争？中

現在論争中

FP64専用ハードウェアを増やすべきか
低精度ユニット＋Ozaki emulationで補うべきか？

背景・導入

2025/04 **Have You Heard About the Ozaki Scheme? You Will**

尾崎スキームの考え方をHPC一般読者向けに紹介する導入記事。

2025/12 **Nvidia Says It's Not Abandoning 64-Bit Computing**

NVIDIA側の説明：FP64を捨てるのではなく，cuBLAS等のemulationも使うという立場。

2026/02 **Genesis Mission Will Lean Heavily on Ozaki Scheme**

AI for Science / Genesis Mission とOzaki emulationの関係を報じた記事。

論争・展望

2026/03 **AMD Hints at Big FP64 Increases in MI430X GPU** as Ozaki Underwhelms

AMD側のFP64重視の見方と，Ozaki emulationへの慎重な立場を示す記事。

2026/06 **HPC Precision Wars: Satoshi Matsuoka Plants the Ozaki Flag**

FP8/Ozaki IIをHPCの将来像として位置づける議論を紹介。

2026/07 **In Search of Better Mixed Precision Strategies for HPC**

混合精度戦略全体の文脈でOzaki schemeが注目されていることを紹介。

講義での位置づけ：Ozaki Schemeは単なる高速化手法ではなく
GPU設計・FP64性能・HPCアプリの将来をめぐる論点になっている

NVIDIA路線・AMD路線

科学技術計算でFP64性能をどう確保するか：低精度Tensor Core＋emulationか，Native FP64ハードウェア強化か?

相対的に：Native FP64を大幅強化しない路線か？

NVIDIA：emulation重視

Low precision engines + Ozaki Scheme

- Native FP64 は維持するが，大幅増強よりも
低精度Tensor Coreとemulationを重視
- cuBLASではFP64 GEMM emulationとして Ozaki Scheme を提供
- Automatic Dynamic Precision により，
安全に使える場合だけemulationを選択
- 考え方：多くの科学計算は raw FP64 compute ではなく，
HBM・cache・register 側で律速しやすい

相対的に：Native FP64を強化する路線

AMD：Native FP64強化

Hardware FP64 + AI precision support

- MI430XをAIとHPCの両方に向けたGPUとして位置づけ
- AMDは true hardware-based FP64 precision を明示
- 432GB HBM4，19.6TB/s bandwidth とともにFP64を重視
- DiscoveryやAlice Recoqueなど，AI Factory / exascale級
システムでの利用を想定

**論点：アプリが memory-bound ならemulation路線が有利になりやすく
compute-boundでFP64密度が重要ならNative FP64強化が有利になりやすい**

INT8TCが・・・・

B200世代では INT8 Tensor Core が FP64 emulation の強力な土台だったが,
B300以降では重心が NVFP4 / FP8 側へ移っている。

B200まで

INT8TC が非常に強い

- Dense換算で約 4.5 POPS/GPU
- FP64 emulation では INT8TC:FP64 が大きい
- Ozaki Scheme II の INT8版に追い風

B300

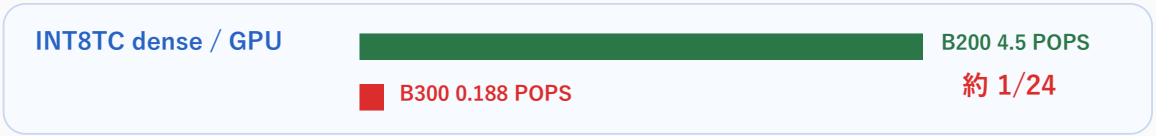
公式HGX仕様で INT8TC が大幅低下

- HGX B200: 72 POPS（8GPU, sparse）
- HGX B300: 3 POPS（8GPU, sparse）
- Dense/GPU換算: 4500 → 188 TOPS

Rubin

NVFP4 / FP8 を前面に出す世代

- 公式資料は 50 PFLOPS NVFP4 を強調
- Tensor CoreはNVFP4/FP8最適化へ
- INT8前提だけでは将来性にリスク



尾崎スキームIIへの意味

INT8版だけでなく，FP8/FP4量子化版への移行が重要。

注意：B300の数値はNVIDIA HGX仕様表のsparse値からdense/GPUに換算。Rubinについては公開資料上，NVFP4/FP8最適化が強調されている。

参考文献・スライド・動画

深く知りたい人のために

尾崎スキームIIに関する論文

- **[原論文]**
  K. Ozaki, Y. Uchino, and T. Imamura,
  “**Ozaki Scheme II**: A GEMM-Oriented Emulation of Floating-Point Matrix Multiplication Using an Integer Modular Technique,”
  arXiv:2504.08009, 2025.
  DOI: 10.48550/arXiv.2504.08009（論文採択された）.

- **[FP32・FP64・電力]**
  Y. Uchino, K. Ozaki, and T. Imamura,
  “High-Performance and Power-Efficient Emulation of Matrix Multiplication Using INT8 Matrix Engines,” *Proceedings of the SC ’25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1824–1831, 2025.
  DOI: 10.1145/3731599.3767539. arXiv:2508.03984.

尾崎スキームIIに関する論文

- **[誤差解析]**
  Y. Uchino, K. Ozaki, and T. Imamura,
  “**Error Analysis** of Matrix Multiplication Emulation Using Ozaki-II Scheme,”
  arXiv:2602.02549, 2026.
  DOI: 10.48550/arXiv.2602.02549.

- **[改良]**
  S. Kawakami and D. Takahashi,
  “**Improved Scaling** for Fast Mode of Ozaki Scheme II,”
  arXiv:2606.29129, 2026.
  DOI: 10.48550/arXiv.2606.29129.

尾崎スキームIIに関する論文

- **[FP8をベースにした尾崎スキームII]**
  Y. Uchino, K. Ozaki, and T. Imamura,
  “Double-Precision Matrix Multiplication Emulation via Ozaki-II Scheme with **FP8 Quantization**,”
  arXiv:2603.10634, 2026.
  DOI: 10.48550/arXiv.2603.10634.

- **[複素行列積]**
  Y. Uchino, Q. Ma, T. Imamura, K. Ozaki, and P. L. Gutsche,
  “Emulation of **Complex Matrix Multiplication** Based on the Chinese Remainder Theorem,” *ISC High Performance 2026 Research Paper Proceedings*, pp. 1–12, 2026.
  DOI: 10.23919/ISC.2026.11520500.

紹介・応用

- **A. Kashi, H. Lu, W. Brewer, D. Rogers, M. Matheson, M. Shankar, and F. Wang**,
“Mixed-Precision Numerics in Scientific Applications: Survey and Perspectives,” *The Journal of Supercomputing*, Vol. 82, No. 5, Article 287, 2026.
DOI: 10.1007/s11227-026-08264-4.

- **H. Liu, J. Li, Y. Wang, N. K. Nepal, and Y. Wang**,
“A Precision Emulation Approach to the GPU Acceleration of Ab Initio Electronic Structure Calculations,” arXiv:2603.29975, 2026.
DOI: 10.48550/arXiv.2603.29975.

- **S. Matsuoka**,
“FP8 Is All You Need (Part 1): Debunking Hardware FP64 as the HPC Holy Grail,” arXiv:2606.06510, 2026.
DOI: 10.48550/arXiv.2606.06510.

尾崎スキームIIに関するライブラリ

- **GEMMul8, 理研・内野氏によるライブラリ**

- **D. Lu, A. Maeder, M. Luisier, and A. N. Ziogas**,
  “EmuGEMM: Fused Tensor Core Kernels for Precision Emulation in Matrix Multiplication,” arXiv:2606.25453, 2026.
  DOI: 10.48550/arXiv.2606.25453.

- NVIDIAが今年中に尾崎スキームIIをリリース予定
  （情報元：GTC26）

NVIDIA GTC 2026

- Katsuhisa Ozaki, NVIDIA GTC 2026：
  Ozaki Scheme: Addressing the Accuracy Challenge in the Era of Low-Precision Accelerators

- Harun Bayraktar, NVIDIA GTC 2026：
  All the Precision, None of the Transistors

- Robert Parrish, NVIDIA GTC 2026：
  cuEST: Accelerating Quantum Chemistry on GPUs

学会発表

- First Workshop on Mixed- and Adaptive-Precision Numerics for Scientific Computing (MxP4S), May 25, 2026, at IPDPS, New Orleans, LA, USA
  （講演者のスライド有）
- SIAM Conference on Parallel Processing for Scientific Computing (PP26)
- (MS38) すべての講演がOzaki Scheme関連（スライド公開なし）
  - **Architecture-Aware Optimization of the Ozaki Scheme on AMD Datacenter GPUs**
  - **Analysis of Floating-Point Matrix Multiplication Computed via Integer Arithmetic**
  - **Leveraging Low-Precision Tensor Cores for Mixed-Precision Computing and Floating-Point Emulation**
  - **Accelerating and emulating GEMM operations with devices and circuits in the age of AI**

終わりに

- 本日は尾崎スキームIIについて解説した.
- 尾崎スキームIとIIは「適切に使うことが重要」
  - 薬と一緒？用法・用量は正しく，みたいな
- 尾崎スキームをどう使うか？も今後の研究テーマ
- 常に最新情報をチェックしてください

ご清聴ありがとうございました