Juliaと高精度計算

芝浦工業大学　システム理工学部
数理科学科　尾崎　克久

JuliaLang Japan 2025
東京科学大学 大岡山キャンパス
2025年12月13日(土)

概要

- 講演者は高信頼数値計算の研究をしている.
- ちょっとしたことで，約３か月前にはじめてJuliaを使用してみた.

- 普段はMATLABのヘビーユーザ（その次によく使うのはC言語）

- その観点からJuliaにおける高信頼数値計算の調査・感覚を報告
- 特に行列積のエミュレーションを扱う

本講演での実験環境
CPU: Core i7-240H
GPU: RTX5060
MATLAB R2025b
Julia 1.12.2

浮動小数点数

±１．（仮数部・バイナリ）＊2指数部

正規化数です

	符号部のビット数	仮数部のビット数	指数部のビット数
FP8 (E5M2)	1	2	5
FP8 (E4M3)	1	3	4
FP16	1	10	5
BF16	1	7	8
FP32	1	23	8
FP64	1	52	11
FP128	1	112	15

正規化数ならば＋１ビットあるのと同じ

高精度計算

- MATLAB
  - Advanpix（有料）が有名（優秀）
    https://www.advanpix.com/
  - VPAは遅い（正確にはSymbolic Math Toolboxに属する）

- Julia
  - MultiFloats （浮動小数点数をつなげた数）
    2~8までの数をつなげられる，例：Float64x5など
  - BigFloat (多倍長精度計算)

高精度計算と線形代数

MATLAB+advanpix

- mp.Digits(34);
- n = 100;
- A = rand(n, ‘mp’);
- B = rand(n, ‘mp’);
- C = A * B;

Juila＋MultiFloats

- using MultiFloats
- const T2 = Float64x2
- n = 100;
- A = rand(T2, n, n);
- B = rand(T2, n, n);
- C = A * B;

MATLABと同じくらいの使いやすさ

高精度計算と線形代数

MATLAB+advanpix

- mp.Digits(34);
- n = 100;
- A = rand(n, ‘mp’);
- B = rand(n, ‘mp’);
- C = A * B;

JuilaのBigFloat(MPFR)

```
setprecision(113);
n = 100;
A = rand(BigFloat, n, n);
B = rand(BigFloat, n, n);
C = A * B;
```

こちらもMATLABと同じくらいの使いやすさ

高精度計算と線形代数

MATLAB+advanpix

- n = 100;
- mp.Digits(34)
- A = rand(n, ‘mp’);
- [L,U,p] = lu(A,’vector’);

Juila+MultiFloats

```
using LinearAlgebra
using MultiFloats
const T2 = Float64x2
n = 100
A = rand(T2, n, n)
F = lu(A)
```

```
L = F.L
U = F.U
p = F.p
```

高精度計算と線形代数

MATLAB+advanpix

- n = 100;
- mp.Digits(34)
- A = rand(n, ‘mp’);
- [L,U,p] = lu(A,’vector’);

MATLABでもdecomposition関数がある

JuilaのBigFloat(MPFR)

```
using LinearAlgebra
setprecision(BigFloat, 113)
n = 100
A = rand(BigFloat, n, n)
F = lu(A)
```

```
L = F.L
U = F.U
p = F.p
```

こちらもMATLABと同じくらいの使いやすさ

実際の性能は？（行列積）

計算時間（秒），正方行列の積

行列サイズ	1024	2048	4096
MATLAB Advanpix(34)	5.01	40.0	318
Julia (float64*2)	6.81	54.8	441
Julia (MPFR 113)	105	1163	？？？

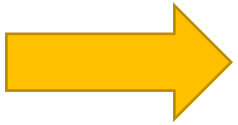
行列サイズ	1024	2048	4096
MATLAB Advanpix(48)	12.3	102	805
Julia (float64*3)	19.7	405	4504
Julia (MPFR 159)	112	1066	？？？

行列サイズ	1024	2048	4096
MATLAB Advanpix(64)	15.9	119	876
Julia (float64*4)	62.0	937	9382
Julia (MPFR 212)	116	1147	？？？

TIPS:自分で並列化

```
function matmul_triple_threads!(C::Matrix{T2}, A::Matrix{T2}, B::Matrix{T2})
    n = size(A, 1)
    @threads for j in 1:n
        for k in 1:n
            @inbounds for i in 1:n
            C[i, j] += A[i, k] * B[k, j]
            end
        end
    end
    return C
end
```

並列化なし
40.1秒



using Base.Threads

7.78秒

実際の性能は？（高速化済の行列積）

計算時間（秒），正方行列の積

	1024	2048	4096
MATLAB Advanpix(34)	5.01	40.0	318
Julia (float64*2)	0.92	7.76	62.5
Julia (MPFR 113)	39.6	350	3165

	1024	2048	4096
MATLAB Advanpix(48)	12.3	102	805
Julia (float64*3)	2.29	19.4	156
Julia (MPFR 159)	58.3	435	3841

	1024	2048	4096
MATLAB Advanpix(64)	15.9	119	876
Julia (float64*4)	4.79	38.3	307
Julia (MPFR 212)	59.0	433	4199

自分でメモリ連続アクセス・並列化をした場合

MultiFloats+Tullioという実行方法もありえる

尾崎スキームについて

Doug Eadline:
Have You Heard About the Ozaki Scheme? You Will,
HPC Wire, April 17, 2025

尾崎スキームとは？

- 行列積のエミュレーション
- 低精度演算を活用して，行列積の結果を得るスキーム
- 諸事情で今では尾崎スキームIと呼ぶ（英語ではOzaki-I scheme?）

原点:

K. Ozaki, T. Ogita, S. Oishi, S. M. Rump:

Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications,

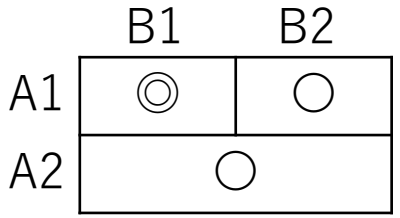
Numerical Algorithms, Vol. 59:1 (2012), 95-118.

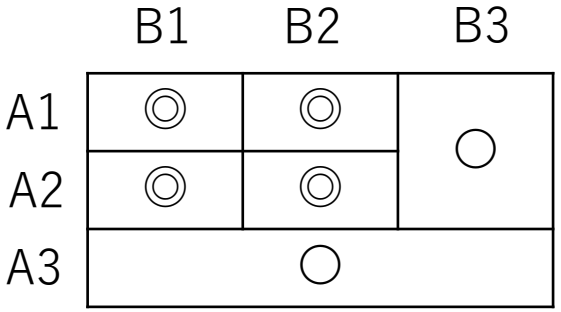
その後いくつかの論文はあるが，　GPU活用型を提案

- Mukunoki, Ozaki, Ogita, Imamura(2020)：FP16TC活用
- Ootomo, Ozaki, Yokota(2024)：INT8TCを活用
- Uchino, Ozaki, Imamura(2025) ：INT8TCを活用

尾崎スキームとは？

- 簡単な計算例
- A=A1+A2, B=B1+B2
- AB = A1*B1+A1*B2+A2*B





- A=A1+A2+A3, B=B1+B2+B3
- AB = A1*B1+A1*B2+A2*B1+A2*B2+(A1+A2)*B3+A3*B

- 赤字の計算には誤差が発生しない

通常精度の行列積のルーチンを呼ぶだけ→高速

尾崎スキーム（２分割版）

```
function C = Mul3(A,B)
n = size(A,2);
beta = ceil((53 + log2(n))/2);
%% Split A
mu = abs(max(A, [], 2, “ComparisonMethod”,”abs”));
w = 0.75*2.^(ceil(log2(mu)) + beta);
A1 = (A + w) - w;
A2 = A - A1;
%% Split B
[mi,ma] = bounds(B);
mu = max(ma,-mi);
w = 0.75*2.^(ceil(log2(mu)) + beta);
B1 = (B + w) - w;
B2 = (B - B1);
%% Compute matrix products and the sum
C = A1 * B1 + (A1 * B2 + A2 * B);
end
```

```
function Mul3(A::Matrix{Float64}, B::Matrix{Float64})
    n = size(A, 2)
    beta = ceil(Int, (53 + log2(n)) / 2)
    # ==== Split A (行方向：dims=2) ====
    mu = vec(maximum(abs, A; dims=2))
    wA  = 0.75 .* 2 .^ (ceil.(log2.(muA) .+ beta))
    A1 = (A .+ wA) .- wA
    A2 = A .- A1
    # ==== Split B (列方向：dims=1) ====
    mu = vec(maximum(abs, B; dims=1))
    wB  = 0.75 .* 2 .^ (ceil.(log2.(muB) .+ beta))
    B1 = (B .+ wB) .- wB
    B2 = B .- B1
    # compute matrix products and the sum
    return A1 * B1 + (A1 * B2 + A2 * B)
end
```

A1*B1+A1*B2+A2*B

数値実験（入力FP64・内部DD・出力FP64）

n=2048, 最大相対誤差

	1 (FP64)	3	4	5	6	8	9	10
Ozaki Scheme	4.34e+05	9.76e-02	3.62e-04	3.78e-04	1.74e-07	4.57e-10	**2.37e-12**	8.48e-14
Julia MF2				**6.94e-12**				
Advanpix				1.11e-16				

n=2048, 計算時間（秒）

	1 (FP64)	3	4	5	6	8	9	10
Ozaki Scheme	M：0.06 J：0.09	M：0.21 J：0.36	M：0.31 J：0.46	M：0.37 J：0.53	M：0.45 J：0.60	M：0.57 J：0.82	**M：0.66 J：0.95**	M：0.71 J：0.91
Julia MF2				**7.26**				
Advanpix				**40.0**				

MF2はMultiFloatsでFloat64x2の意味

行列積の問題はあえて悪条件問題にしています

数値実験（入力・出力ともにDD相当）

Float64x2

n=2048, 最大相対誤差

	1	3	4	5	6	8	9	10
Ozaki Scheme	1.24e-09	1.16e-15	1.10e-16	9.04e-18	5.56e-22	1.33e-24	**4.79e-26**	5.99e-28
Julia MF2	**2.10e-25**							
Advanpix	7.81e-27							

n=2048, 計算時間（秒）

	1	3	4	5	6	8	9	10
Ozaki Scheme	M：0.062 J：0.074	M：0.31 J：0.38	M：0.41 J：0.51	M：0.54 J：0.68	M：0.60 J：0.81	M：0.80 J：1.01	M：0.94 J：1.23	M：0.98 J：1.27
Julia MF2	**7.26**							
Advanpix	**40.0**							

MF2はMultiFloatsでFloat64x2の意味（並列化済）　　疑似乱数行列に対する行列積（良条件）

NVIDIA GPUの性能

- アクセラレータ（特にGPU）業界で現在注目されている.

NVIDIAのGPUのピークパフォーマンス（テンソルコアがあればそれを使用）

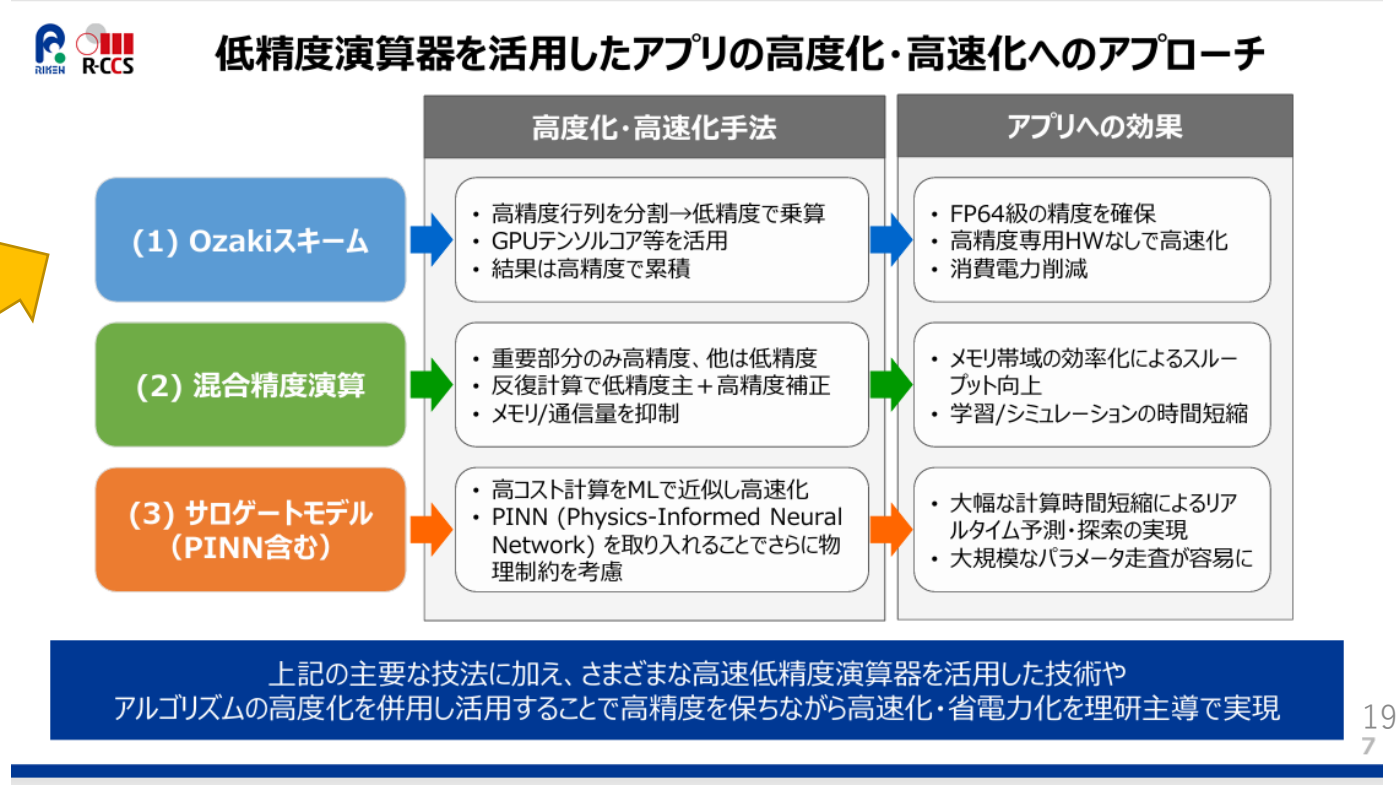
	RTX4090	RTX5090	A100	GH200	B200
INT8	**660**	**1674**	**624**	**1979**	**4500**
FP16	165	419	312	989	2250
BF16	165	419	312	989	2250
FP32	82.6	104.8	156	67	80
FP64	**1.29**	**1.64**	**19.5**	**67**	**40**

INT8は出力がINT32
FP8, FP16・BF16・TF32は出力がFP32

GPU関連の尾崎スキーム

- 2025年8月22日 HPCI計画推進委員会
- 「理化学研究所を中核とした国際連携による「富岳NEXT」開発体制とその意義」から抜粋

https://www.mext.go.jp/content/20250822-mxt-jyohoka01-000044312_41.pdf



GPU関連の尾崎スキーム

- CUDA Toolkit 13.0 Update 2　（2025年10月）

2.3. New Features

CUDA 13.0 is a new major version. CUDA follows semantic versioning, and guarantees ABI stability within the major version series. CUDA Toolkit releases in the 13.x series are ABI-compatible with drivers corresponding to the same series (r580 and newer). Note that CUDA's API can evolve over the course of a major release to include new functionality, and not all new functionality may be supported on older drivers.

2.3.1. General CUDA

- Enabled opt-in fixed-point emulation for FP64 matmuls (D/ZGEMM) which improves performance and power-efficiency. The implementation follows the Ozaki-1 Scheme and leverages an automatic dynamic precision framework to ensure FP64-level accuracy. See here for more details on fixed-point emulation along with the table of supported compute-capabilities and the CUDA library samples for example usages.

参照：
https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

NVIDIA GPUによる行列積

MATLAB

- n = 1024
- A = rand(n, ‘gpuArray’);
- B = rand(n, ‘gpuArray’);
- C = A * B

```
時間計測
gd = gpuDevice;
tic; C = A * B; wait(gd), toc
```

Julia

- using CUDA
- n = 1024;
- A = CUDA.rand(Float64, n, n);
- B = CUDA.rand(Float64, n, n);
- C = A * B;

```
時間計測
t = @elapsed begin
    C = A * B;
    CUDA.synchronize()
end
```

シンプルで使いやすい

GPUによる行列積

MATLAB

- n = 1024
- A = rand(n, ‘gpuArray’);
- B = rand(n, ‘gpuArray’);
- C = A * B

MATLAB R2025bまで確認：
single (FP32)とdouble(FP64)のみ

Julia

- using CUDA, LinearAlgebra
- n = 1024;
- A = CUDA.rand(Float16, n, n);
- B = CUDA.rand(Float16, n, n);
- C = CUDA.zeros(Float32, n, n);
- mul!(C,A,B);

2025年現在：
Float16, BFloat16, Float32, Float64が対応
デフォルトではないが
using BFloat16sで使用可

Julia+GPU+Ozaki Scheme

export CUBLAS_EMULATE_DOUBLE_PRECISION=1
export CUBLAS_EMULATION_STRATEGY= performant

- using CUDA
- n = 4096;
- A = CUDA.rand(Float64, n, n);
- B = CUDA.rand(Float64, n, n);
- C = A*B;

0.554 秒→231 GFLOPS

- using CUDA ← V13.0.2以降
- n = 4096;
- A = CUDA.rand(Float64, n, n);
- B = CUDA.rand(Float64, n, n);
- C = A*B;

0.0745秒→**1718 GFLOPS**

eager: 常にOzaki Scheme を使用
performant: 速くなる場合にのみ
Ozaki Schemeを使用

Juliaの今後に期待する点

- A*Bと書いて，自動的に並列化されたものが実行されると良い
  （例：MultiFloats, BigFloats）
  行列積だけではなく，LU分解やCholesky分解なども・・・
- GPU上でINT8TC, FP8TCのサポート（今はない？）
- GPU上での構造を持った行列積の扱い
  - 三角行列や結果の対称性など
- GPU上でのOzaki-I Schemeの柔軟な運用
  - ビット数の設定やEmulationのオン・オフなど

まとめ

- Juliaは優秀で尾崎スキームをJuliaで実装することに苦はない（CPU/GPU両方）

今後の予定（希望も）

- Julia上でどのように尾崎スキームを展開するか？
- Juliaを使用しているアプリの方と連携したい（私は数値線形代数の基本アルゴリズムに適用済）
- Ozaki Scheme II（中国式剰余定理活用版）の実装
  - これはCPU版もGPU版も，現在デバッグ中

宣伝

- 尾崎スキームIに関してセミナーで講演を行います.
  日本応用数理学会・三部会連携応用数理セミナー
  2025年12月26日（金）：東京都市大学（ハイブリッド開催）

- SCA / HPC Asia@大阪
  2026年1月26日から29日
  尾崎スキームI and IIについて私も講演します.

  Juliaのセッションあり
  Tutorial: Julia for performance-portable High-Performance Computing via JACC