LAPACK/BLAS入門

静岡理工科大学
情報学部　コンピュータシステム学科
幸谷 智紀
kouya.tomonori@sist.ac.jp

本日のメニュー

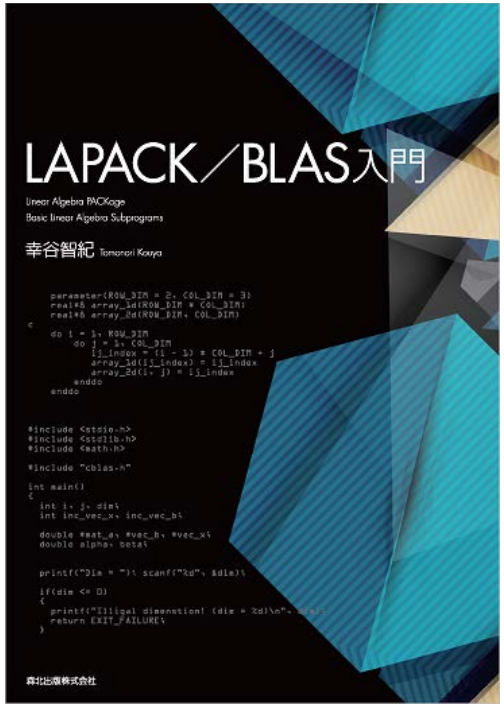
1. 「LAPACK/BLAS入門」について
2. LAPACK/BLASとは？
   - 2.1 LAPACK/BLASの概略
   - 2.2 LAPACKE/CBLAS
   - 2.3 ベクトル，行列のデータ型
   - 2.4 LAPACK/BLASを使うメリット・デメリット
3. BLASの機能と例題
   - 3.1 BLAS Level 1, 2, 3
   - 3.2 GEMVベンチマーク
   - 3.3 GEMMベンチマーク
   - 3.4 [例題] べき乗法
4. LAPACKの機能と例題
   - 4.1 連立一次方程式の直接解法
   - 4.2 行列の固有値問題
5. [応用] 積分方程式

1.「LAPACK/BLAS入門」について

- 幸谷著「LAPACK/BLAS入門」森北出版, 2016年12月刊行
  - BLAS, 連立一次方程式, 標準固有値問題の解説に限定
  - LAPACK/BLAS関連の知識を広く浅く実例（プログラム）と実行結果で示す
  - サンプルCプログラムは別配布

  → [サポートページ] https://na-inet.jp/lapack/
- 反省点
  - 解説が浅すぎる→A5版・135ページ
- 目次

  第1章 LAPACK/BLASって何？
  第2章 LAPACK/BLAS, 最初の一歩
  第3章 BLASを極める
  第4章 LAPACKドライバルーチンひとめぐり
  第5章 疎行列用の線型計算ライブラリ
  第6章 並列化の方法 (Pthread, OpenMPのみ)
  第7章 GPU上のLAPACK/BLAS--- cuBLASとMAGMA, cuSPARSE
  第8章 非線型問題にもチャレンジ！

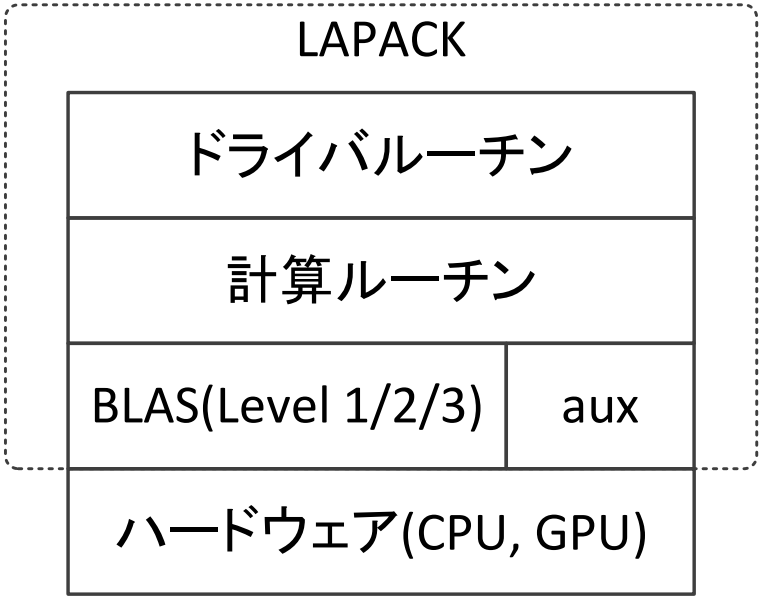


2. LAPACK/BLASとは？

2.1 LAPACK/BLASの概略
2.2 ベクトル，行列のデータ型
2.3 LAPACK/BLASでは出来ないこと

2.1 LAPACK/BLASの概略(1/4)

- Fortranで記述された，線型計算ライブラリ(Cライブラリも同梱)
- IEEE754単精度，倍精度の実数，複素数を要素とするベクトル，行列をサポート
- Version 3.8.0(2017年11月リリース)が最新版

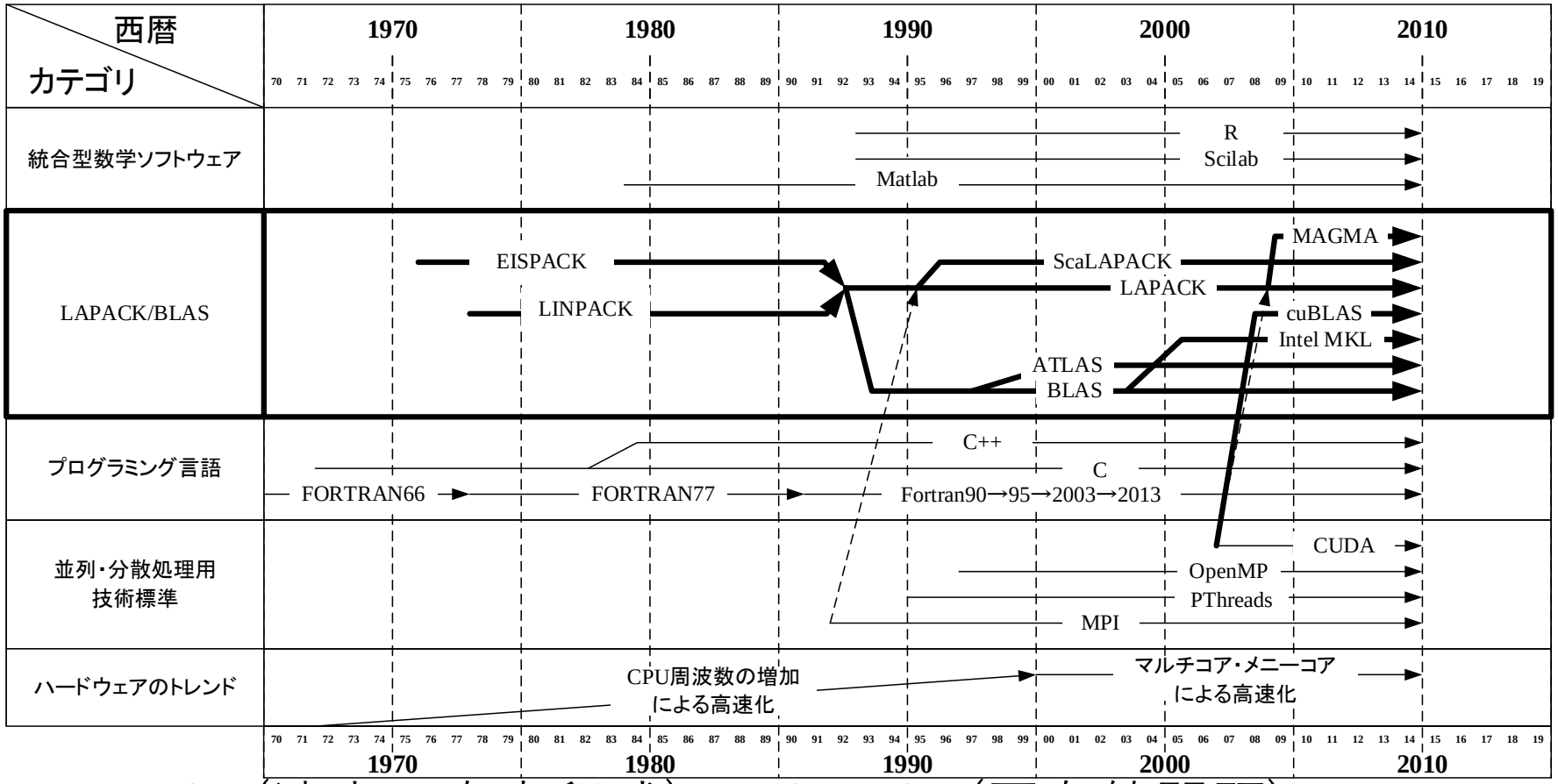


- BLAS（Basic Linear Algebra Subprograms）・・・ベクトル，行列の基本演算を担当
- LAPACK（Linear Algebra PACKage）・・・BLASを基盤とし，より複雑な線型計算を担当

http://www.netlib.org/lapack/

- テネシー大学，カリフォルニア大学バークレイ校，コロラド大学デンバー校　が著作権保持
- Julie&Julian Langou, Dimmel, Dongarraらによるサポート・メンテナンス

2.1 LAPACK/BLASの概略(2/4)



- LINPACK(連立一次方程式)+EISPACK(固有値問題) → LAPACK/BLAS
- 様々な派生形ライブラリも登場→APIとして現役
- SLATE, MAGMA等が後継っぽい・・・が，LAPACK/BLASのバージョンアップも継続中

2.1 LAPACK/BLASの概略(3/4)

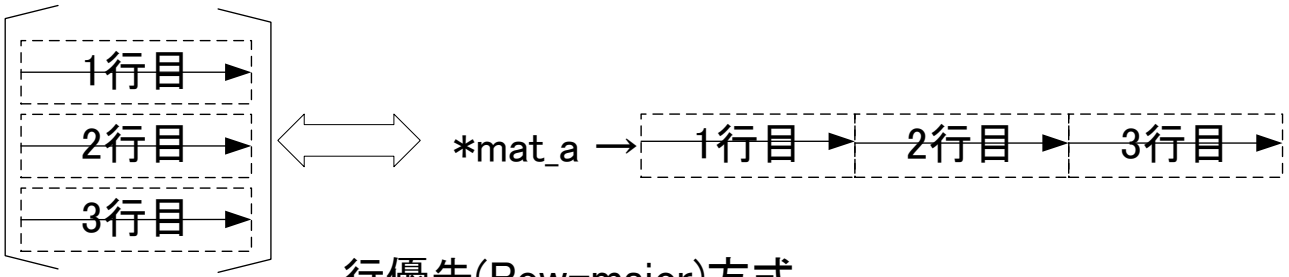
- BLAS ・・・ Basic Linear Algebra Subprograms
- CBLAS ・・・ BLASのCバージョン

- Level 1・・・ベクトル同士の演算（スカラー倍，内積など）
- Level 2・・・行列＆ベクトル演算（行列・ベクトル積など）
- Level 3・・・行列演算

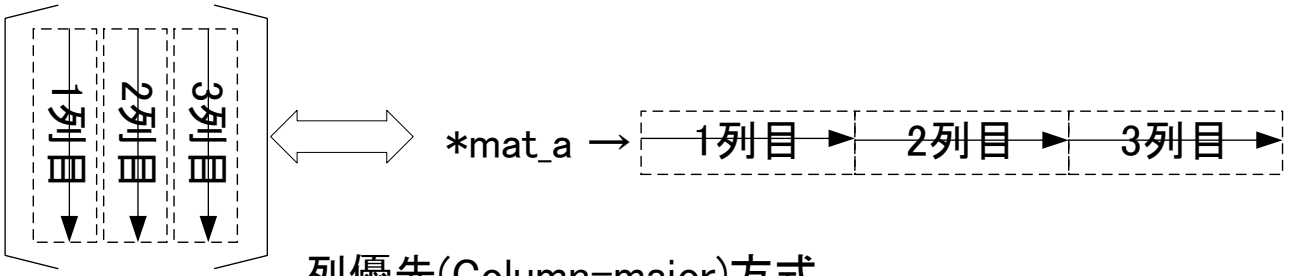
2.1 LAPACK/BLASの概略(4/4)

- LAPACK・・・Linear Algebra PACKage
- LAPACKE・・・LAPACKのCバージョン（≠CLAPACK)
- ドライバルーチン ・・・問題のタイプ毎に存在
  - 連立一次方程式
  - 線型最小二乗(LLS, Linear Least Squares)問題
  - 一般化線型最小二乗問題
  - 標準固有値問題
    - 対称行列の固有値・固有ベクトル計算
    - 非対称行列の固有値・固有ベクトル計算
    - 特異値分解
  - 一般化固有値問題および特異値問題
    - 一般化対称固有値問題
    - 一般化非対称固有値問題
    - 一般化特異値分解
- 計算ルーチン・・・ドライバルーチンの下支え
- アクセサリ(aux)

2.2 ベクトル，行列のデータ型(1/3)



行優先(Row-major)方式



列優先(Column-major)方式

- LAPACK/BLAS ・・・ 全てFortran90で記述
  - 列優先(Column-major)方式による密行列格納
  - サブルーチン内の一時変数は全て引数に指定
- LAPACKE/CBLAS・・・C/C++用LAPACK/BLAS IF + α
  - 行優先(Row-major)か列優先を選択利用可
  - 関数内の一時変数を引数に指定しない方式も採用

API(関数名)の基本命名法

昔のFORTRANの6文字制限

→　xyyzzz

例)
倍精度行列ベクトル積
→DGEMV (BLAS)
→cblas_dgemv
倍精度一般行列の連立一次方程式
→ DGESV (LAPACK)
→ LAPACKE_dgesv

x … 計算精度と実数，虚数の指定。
- S … 単精度実数
- D … 倍精度実数
- C … 単精度複素数
- Z … 倍精度複素数

yy … 使用する行列のタイプの指定。
　　一般の密行列はxGEzzz（一般行列）を使用

zzz … 実行される計算の内容を示す文字列。

ドライバルーチン
- xyySV(連立一次方程式)
- xyyE, xyyEV(固有値・固有ベクトル)

計算ルーチン
- xyyTRF(LU分解),
- xyyTRS(前進，後退代入)
- xyyQRF(QR 分解)

LAPACK/BLASで扱える行列タイプ

文字列 (xYYzzz)	内容
xBDzzz	2 重対角行列
xDIzzz	対角行列
xGBzzz	一般帯行列
xGEzzz	非対称一般行列 (上下三角行列も可)
xGGzzz	一般行列の対
xGTzzz	一般三重対角行列
xHBzzz	複素エルミート帯行列
xHEzzz	複素エルミート行列
xHGzzz	一般化上ヘッセンベルグ行列
xHPzzz	複素エルミート行列の圧縮格納 (packed storage)
xHSzzz	上ヘッセンベルグ行列
xOPzzz	実直交行列の圧縮格納
xORzzz	実直交行列
xPBzzz	対称もしくはエルミート正定値帯行列
xPOzzz	対称もしくはエルミート正定値行列
xPPzzz	対称もしくはエルミート正定値行列の圧縮格納
xPTzzz	対称もしくはエルミート正定値三重対角行列
xSBzzz	実対称帯行列
xSPzzz	対称行列の圧縮格納
xSTzzz	実対称三重対角行列
xSYzzz	対称行列
xTBzzz	上下三角帯行列
xTGzzz	帯行列の対
xTPzzz	上下三角行列の圧縮形式
xTRzzz	上下三角行列 (準三角行列も可)
xTZzzz	台形 (trapezoidal) 行列
xUNzzz	複素ユニタリ行列
xUPzzz	複素ユニタリ行列の圧縮格納

- 一般の非対称・密行列はGE (GEneral)で指定
- 行列のタイプに合わせて処理時間，使用メモリの削減ができるものを選択

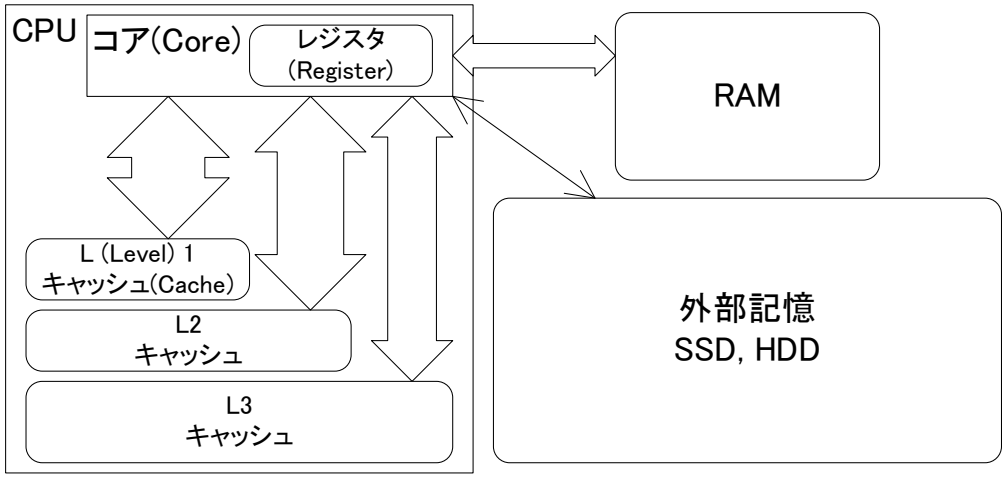
例)

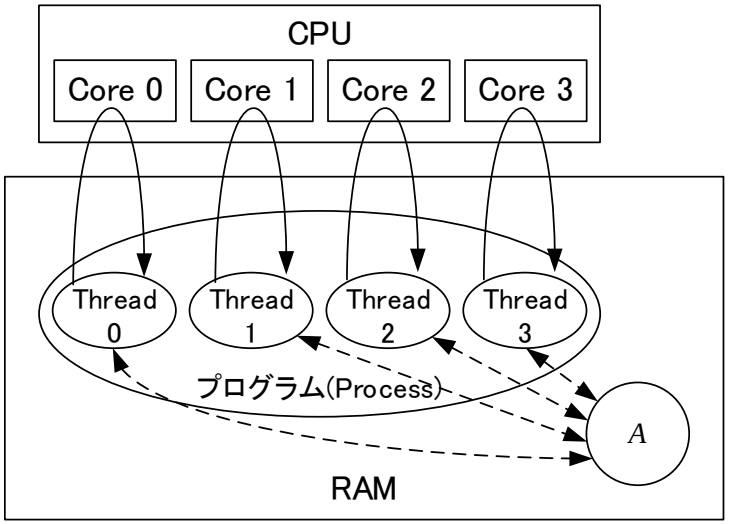
実対称密行列：SY

→メモリ量半減

→固有値・固有ベクトルは実数

派生ライブラリ：Intel Math Kernel

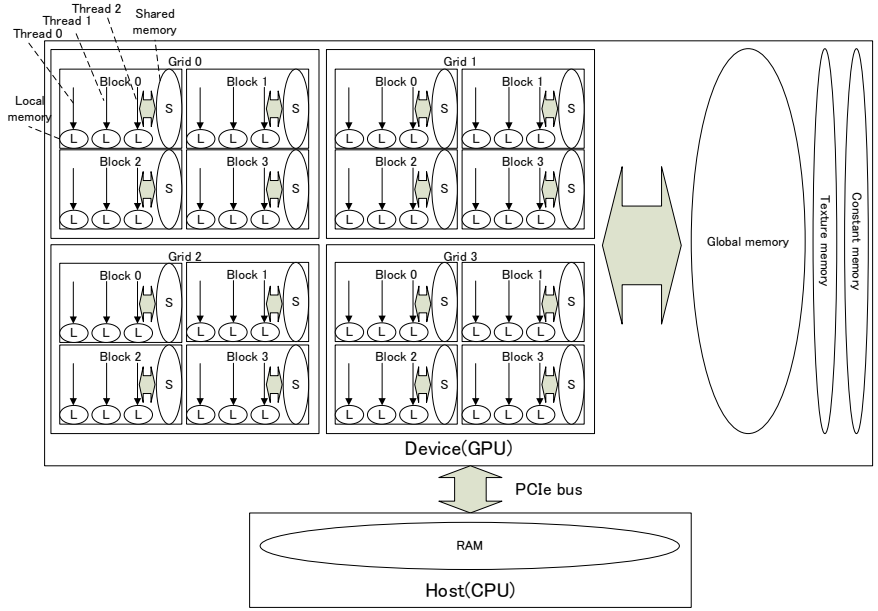


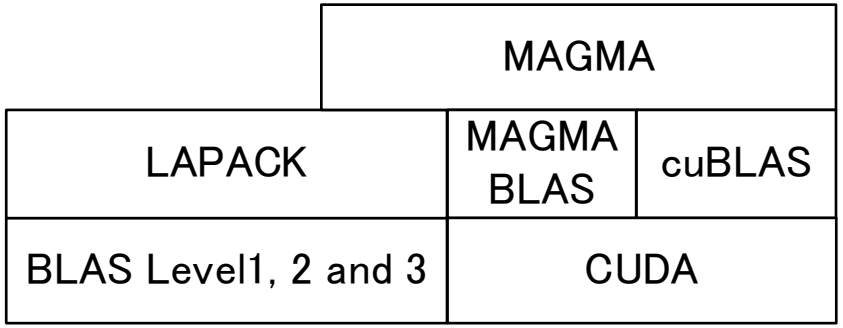


- Intelによる商用高性能計算ライブラリ
- LAPACK/BLAS と LAPACKE/CBLAS IF利用可
- 疎行列計算機能もサポート
- キャッシュメモリチューニング，SIMD命令利用による高速化
- マルチコア環境を利用した並列計算機能

```
// LAPACK(Row-Major or Column-Major)
info = LAPACKE_dgeev(LAPACK_COL_MAJOR, 'N', 'V', dim,
ma, dim, re_eig, im_eig, NULL, dim, revec, dim);
```

派生ライブラリ：cuBLASとMAGMA





- NVIDIA社提供GPGPU開発環境：CUDA
  - cuBLAS・・・BLAS互換ライブラリ
  - cuSPARSE・・・疎行列計算ライブラリ
- MAGMA ・・・ CPU & GPU用 高性能LAPACK/BLAS
  - MAGMA BLAS
  - MAGMA

http://icl.cs.utk.edu/magma/

```
// MAGMA(Column-Major)
magma_dgeev('N', 'V', dim, magma_ma, dim, re_eig, im_eig, NULL,
dim, magma_revec, dim, h_work, (magma_int_t)lwork_num, &info);
```

2.3 LAPACK/BLASでは出来ないこと

- メモリに入り切れないサイズの行列・ベクトルを扱う問題
  - オンメモリが基本

- 単精度計算で10進約7桁，倍精度計算で10進約16桁以上の精度を求める問題
  - 連立一次方程式・・・条件数の大きな問題
  - 固有値問題・・・
    - 条件数の大きな問題
    - 2次以上のJordanブロックを持つ対角化不可能な行列

2.4 LAPACK/BLASを使うメリット・デメリット

- 線型計算をしたいだけなら統合型数値計算環境や動的言語を使うのがbetter
  - 試行錯誤が楽
  - 昔ほど処理が遅くならない
  - グラフィックスや他のパッケージと連携しやすい
- Fortran, C/C++から直接LAPACK/BLASを使わざるを得ないケースは有効
  - C/C++が好き（年寄り？），ブラックボックス嫌い
  - できうる限り高速化したい(MPI, GPUの利用)
  - LAPACK/BLAS APIを使う他のライブラリを使いたい
  - 線型計算を利用する組み込み機器を開発したい

3. BLASの機能と例題

3.1 BLAS Level 1, 2, 3
3.2 GEMVベンチマーク
3.3 GEMMベンチマーク
3.4 [例題] べき乗法

BLAS Level 1

表 3.1: BLAS Level 1 計算

関数名	内容
xROTG	平面上の回転の生成
xROTMG	平面上の回転の生成 (改良版)
xROT	ベクトルの回転
xROTM	ベクトルの回転 (改良版)
xSWAP	ベクトルの入れ替え ($\mathbf{x} \leftrightarrow \mathbf{y}$)
xSCAL	ベクトルのスカラー倍 ($\mathbf{x} := \alpha\mathbf{x}$)
xCOPY	ベクトルのコピー ($\mathbf{y} := \mathbf{x}$)
xAXPY	ベクトルの定数倍と和 ($\mathbf{y} := \alpha\mathbf{x} + \mathbf{y}$)
xDOT	実ベクトル (S, D, DS) 同士の内積 ($(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$)
xDOTU	複素ベクトル (C, Z) 同士の演算 ($\mathbf{x}^T\mathbf{y}$)
xDOTC	複素ベクトル (C, Z) 同士の内積 ($(\mathbf{x}, \mathbf{y}) = \bar{\mathbf{x}}^T\mathbf{y}$)
xxDOT	実ベクトルの内積とスカラー和 ($\alpha + \mathbf{x}^T\mathbf{y}$)
xNRM2	ベクトルのユークリッドノルム ($\|\mathbf{x}\|_2$)
xASUM	実部，虚部の 1 ノルムの和 ($\|\mathrm{Re}(\mathbf{x})\|_1 + \|\mathrm{Im}(\mathbf{x})\|_1$)
IxAMAX	$\max_i(\|\mathrm{Re}(x_i)\| + \|\mathrm{Im}(x_i)\|)$ となる最初の $k(1 \le k \le n)$

- ベクトル（行列）のコピー
- 内積
- スカラー
- 各種ノルムの計算

BLAS Level 2, Level 3

表 3.6: BLAS Level 2 計算

関数名	内容
xyyMV	行列ベクトル積 ($\mathbf{y} := \alpha\mathbf{op}(A)\mathbf{x} + \beta\mathbf{y}, \mathbf{op}(A) = A, A^T, \bar{A}^T$)
xTyMV	行列ベクトル積 ($\mathbf{y} := \mathbf{op}(A)\mathbf{x}$)
xTySV	行列ベクトル積 ($\mathbf{y} := \mathbf{op}(A^{-1})\mathbf{x}$)
xyyR	実数行列 (S, D) 演算 ($A := \alpha\mathbf{x}\mathbf{y}^T + A$)
xyyRU	複素行列 (C, Z) 演算 ($A := \alpha\mathbf{x}\mathbf{y}^T + A$)
xyyRC	複素行列 (C, Z) 演算 ($A := \alpha\mathbf{x}\bar{\mathbf{y}}^T + A$
xyyR2	行列演算 ($A := \alpha\mathbf{x}\bar{\mathbf{y}}^T + \mathbf{y}(\overline{\alpha\mathbf{x}})^T$)

表 3.10: BLAS Level 3 計算

関数名	内容
xyyMM	行列積 ($C := \alpha\mathbf{op}(A)\mathbf{op}(B) + \beta C, \mathbf{op}(X) = X, X^T, \bar{X}^T$)
xyyRK	行列積 ($C := \alpha A\bar{A}^T + \beta C$)
xyyR2K	行列積 ($C := \alpha A\bar{B}^T + \bar{\alpha}B\bar{A}^T$)
xTRSM	三角行列積 ($B := \alpha\mathbf{op}(A^{-1})B$, または $\alpha B\mathbf{op}(A^{-1})$)

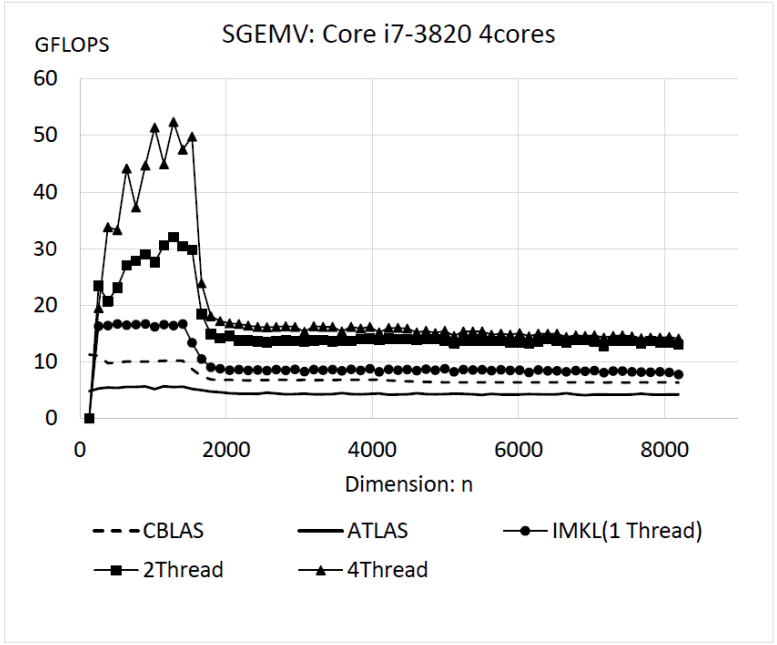
Level 2

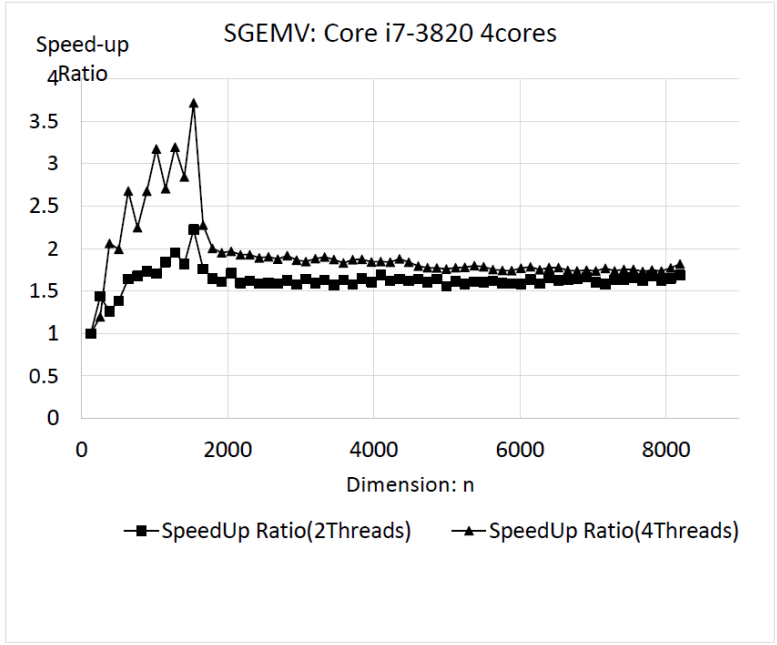
- 行列とベクトルの演算
- 演算結果が行列になる場合もこちら

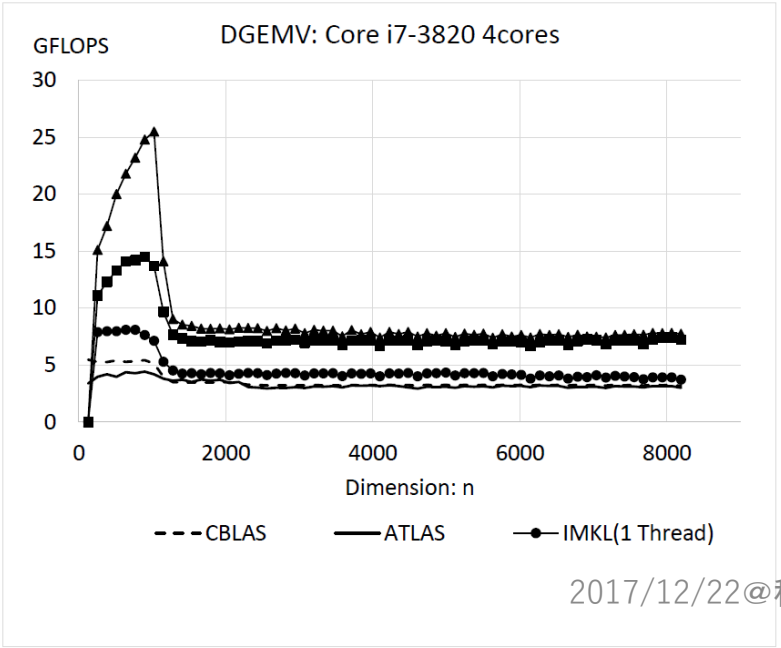
Level 3

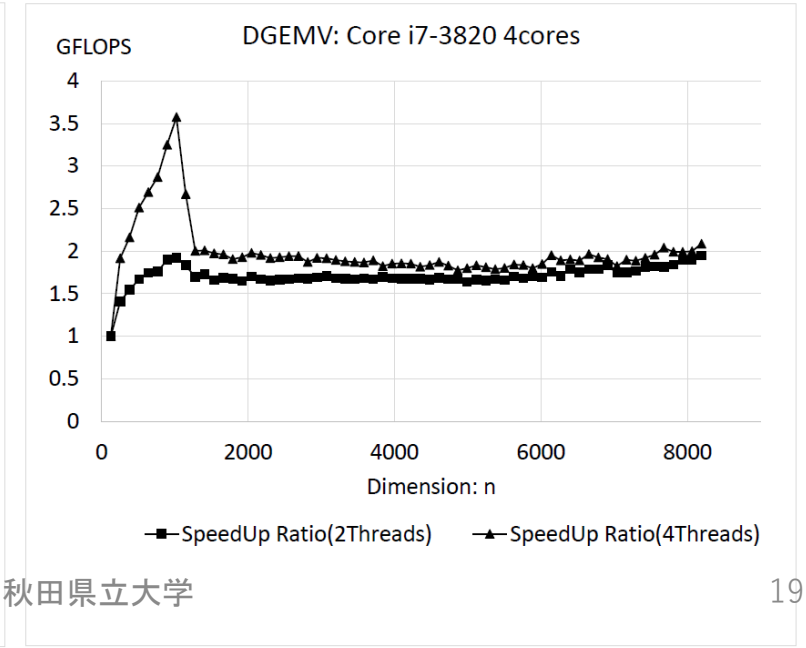
- 行列積
- 和・差・スカラー倍はLevel 1で十分

BLAS Level2(xGEMV) ベンチマーク

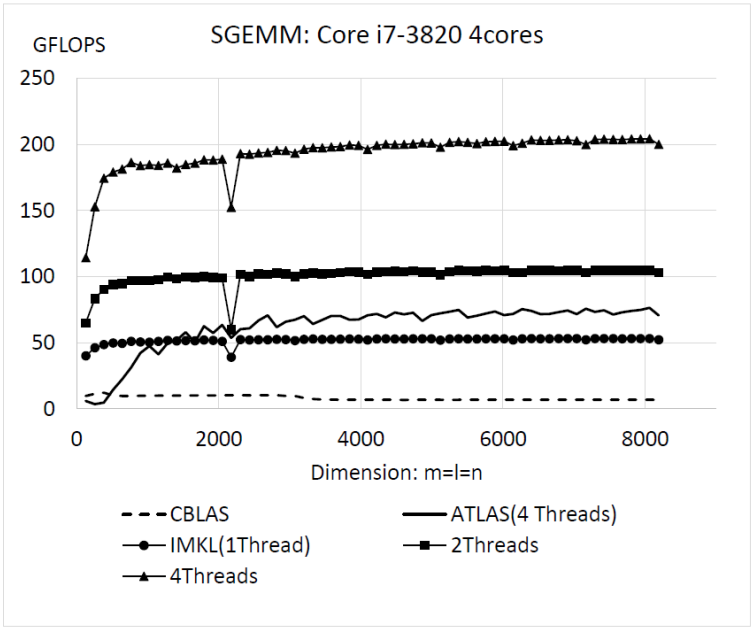


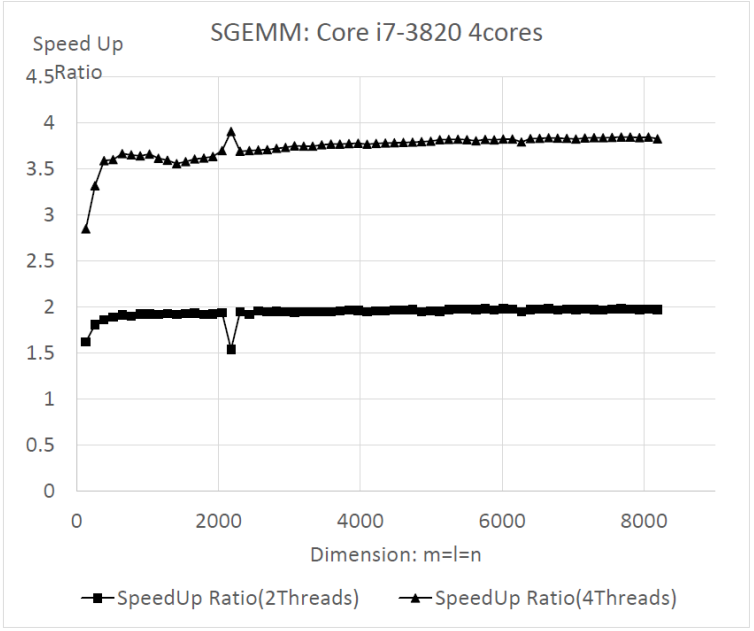


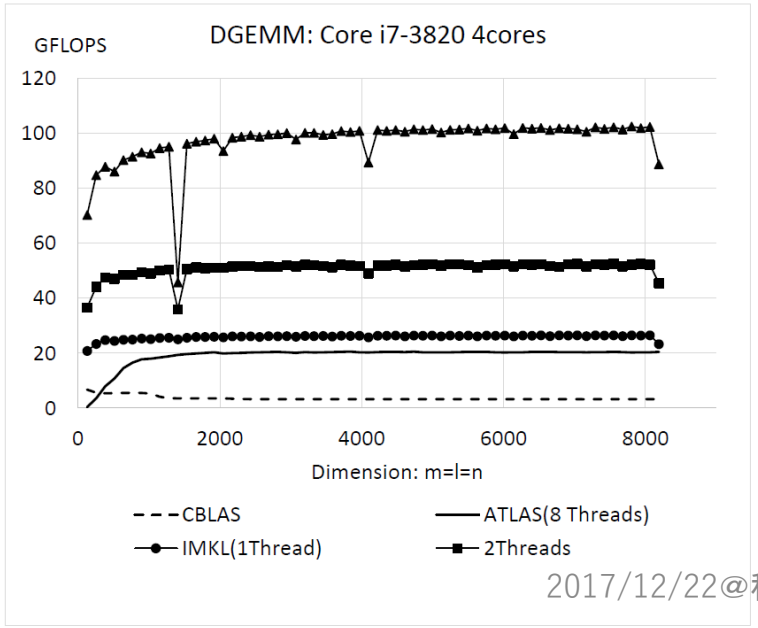


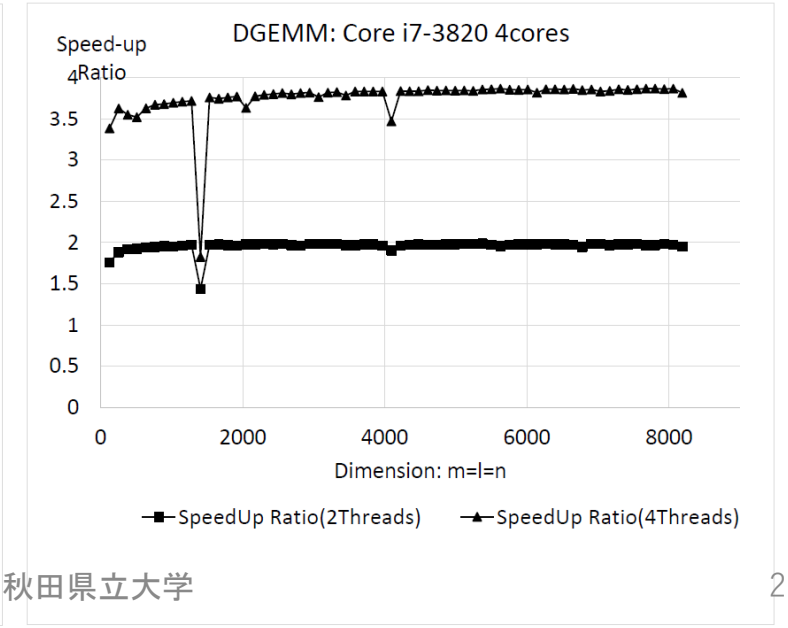


BLAS Level 3(xGEMM)ベンチマーク

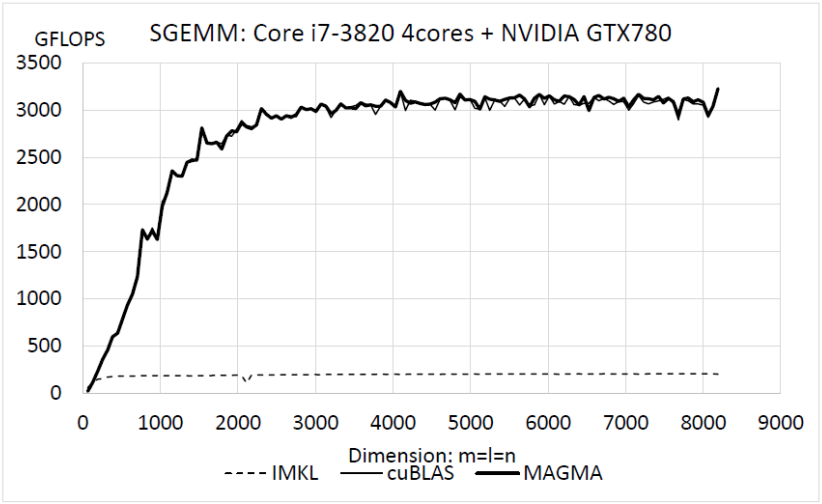


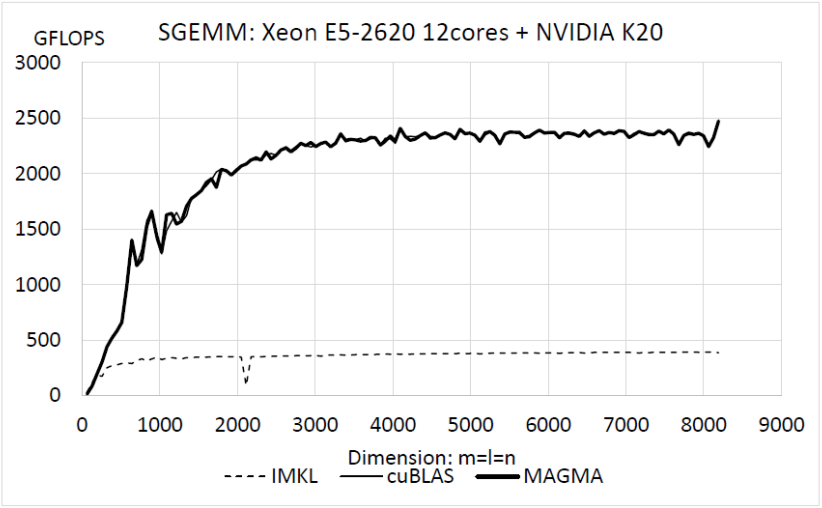


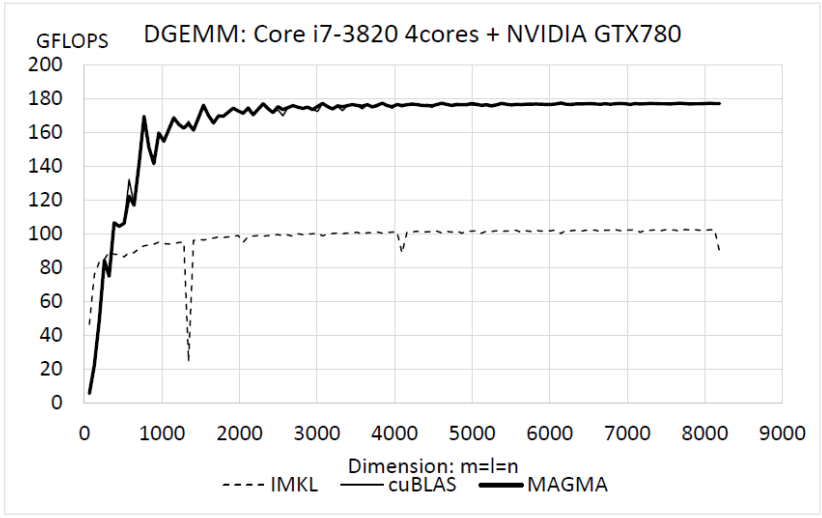




xGEMMの比較(cuBLAS, MAGMA BLAS, IMKL)







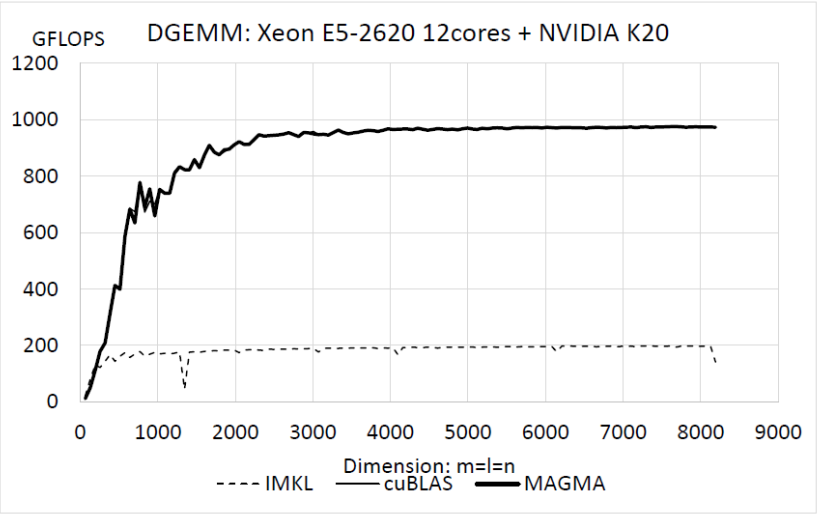


図 7.3: xGEMM: NVIDIA GTX780(左) と Tesla K20(右) の比較

3.4 [例題] べき乗法: power_eig.c

$$\mathbf{x}_0 = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n$$

$$\mathbf{x}_k := A^k\mathbf{x}_0 = (\lambda_1)^k \left\{ c_1\mathbf{v}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k \mathbf{v}_2 + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k \mathbf{v}_n \right\}$$

1. 初期ベクトル $\mathbf{x}_0$(ここで $\|\mathbf{x}_0\| = 1$) を決める。
2. for $k = 0, 1, 2, \ldots$
   - (a) $\mathbf{y}_{k+1} := A\mathbf{x}_k$ ↔ `cblas_dgemv`
   - (b) $\gamma_{k+1} := (\mathbf{y}_{k+1}, \mathbf{x}_k)/(\mathbf{x}_k, \mathbf{x}_k)$ ↔ `cblas_ddot`
   - (c) 収束判定
   - (d) $\mathbf{x}_{k+1} := \mathbf{y}_{k+1}/\|\mathbf{y}_{k+1}\|$ ↔ `cblas_dnrm2, cblas_dscal`

[余談] Top500.orgとLAPACK/BLAS

TOP 10 Sites for November 2017

For more information about the sites and systems in the list, click on the links or view the complete list.

1-100 | 101-200 | 201-300 | 301-400 | 401-500

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
2	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P , NUDT National Super Computer Center in Guangzhou China	3,120,000	33,862.7	54,902.4	17,808
3	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	361,760	19,590.0	25,326.3	2,272
4	**Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler Japan Agency for Marine-Earth Science and Technology Japan	19,860,000	19,135.8	28,192.0	1,350
5	**Titan** - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x , Cray Inc. DOE/SC/Oak Ridge National Laboratory United States	560,640	17,590.0	27,112.5	8,209
6	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom , IBM DOE/NNSA/LLNL United States	1,572,864	17,173.2	20,132.7	7,890
7	**Trinity** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,968	14,137.3	43,902.6	3,844
8	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939
9	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu Joint Center for Advanced High Performance Computing Japan	556,104	13,554.6	24,913.5	2,719
10	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu RIKEN Advanced Institute for Computational Science (AICS) Japan	705,024	10,510.0	11,280.4	12,660

4 **Gyoukou** - ZettaScaler-2.2 HPC system, Xeon D-1571 16C 1.3GHz, Infiniband EDR, PEZY-SC2 700Mhz , ExaScaler
Japan Agency for Marine-Earth Science and Technology
Japan

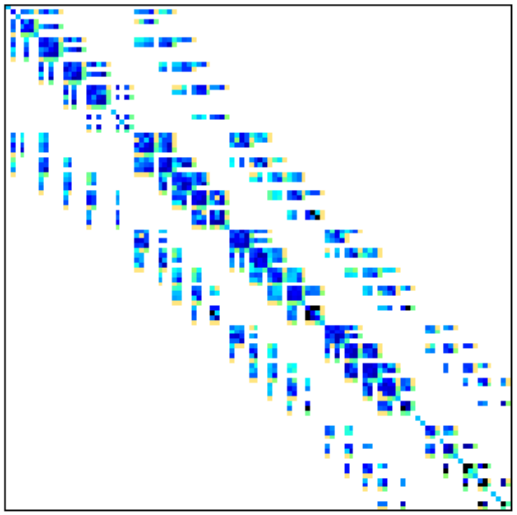
9 **Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu
Joint Center for Advanced High Performance Computing
Japan

10 K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu
RIKEN Advanced Institute for Computational Science (AICS)
Japan

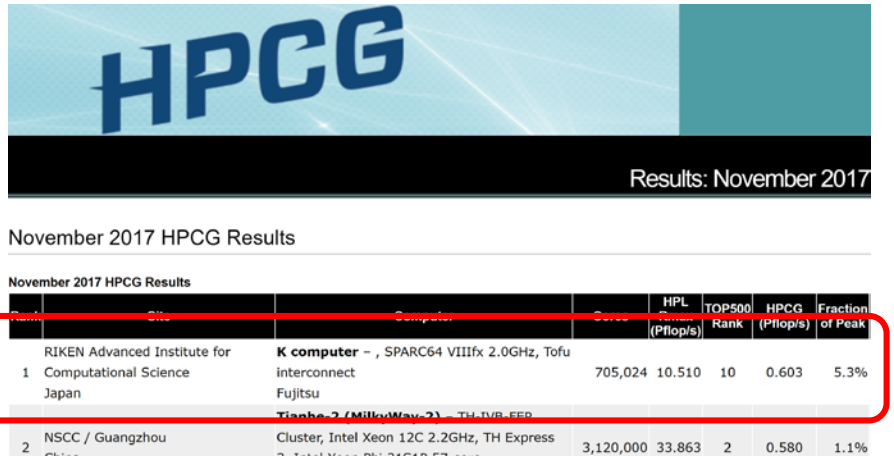
- 2017年11月のTop500
- 4, 9 , 10位に日本スパコン

反復法で日本は世界一に！＝HPCG

疎行列の例



DRIVCAV/cavity04: 317 x 317, 非零成分数 7327 (全要素数の約7.3%)



理研の京がNo1

CG法のアルゴリズム

$\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$, $\mathbf{p}_0 := \mathbf{r}_0$ とする。

$k = 0, 1, 2, \cdots$ に対して以下を計算する。

(a) $\alpha_k := \dfrac{(\mathbf{r}_k, \mathbf{p}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)}$

(b) $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$

(c) $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k A\mathbf{p}_k$ (又は $:= \mathbf{b} - A\mathbf{x}_{k+1}$)

(d) $\beta_k := \dfrac{\|\mathbf{r}_{k+1}\|_2^2}{\|\mathbf{r}_k\|_2^2}$

(e) 収束判定

(f) $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$

- CG法・・・BLAS Level 1, 2のみで構成
- 疎行列×ベクトルの演算(SpMV)の強化が必要
- IMKL, cuSPARSEの疎行列演算機能を使おう！
- ベンチマークは大変

4. LAPACKの機能と例題

- 連立一次方程式の直接解法
- 行列の固有値問題

連立一次方程式のベンチマーク

表 2.2: LAPACK: DGESV 関数

$\mathbf{b} := A^{-1}\mathbf{b}$

#include "lapache.h"	引数の意味
int LAPACKE_dgesv(	
int matrix_order,	行列 A の格納方式 LAPACK_ROW_MAJOR(行優先) または LAPACK_COL_MAJOR (列優先)
int n, int nrhs,	行列サイズ
double *a,	行列 A(n × n) の格納先ポインタ
int lda,	解ベクトルの本数
int *ipiv,	ピボット列へのポインタ
double *b,	ベクトル $\mathbf{b}$(n × nrhs) 格納先
int ldb	ベクトル $\mathbf{b}$ の本数
);	

返り値

$$\text{int info} = \begin{cases} 0 & \text{正常終了} \\ -i & i \text{ 番目の引数が異常値} \\ i & i \text{ 番目のピボットがゼロ} \end{cases}$$

- 連立一次方程式を直接法で解く
  - 自作プログラム
  - LAPACK(DGESV関数)
  - IMKL(DGESV関数)
- 単-倍混合精度反復改良法(DSGESV)
  - IMKL
  - MAGMA (on CUDA)

linear_eq.c : DGESVルーチン使用

```
// vec_b := 1.0 * mat_a * vec_x + 0.0 * vec_b
alpha = 1.0; beta = 0.0;
cblas_dgemv(CblasRowMajor, CblasNoTrans, dim, dim, alpha, mat_a, dim, vec_x,
inc_vec_x, beta, vec_b, inc_vec_b);


// ピボット初期化
pivot = (lapack_int *)calloc(dim, sizeof(lapack_int));


// solve A * X = C -> C := X
info = LAPACKE_dgesv(LAPACK_ROW_MAJOR, dim, 1, mat_a, dim, pivot, vec_b, 1);


// print
printf("calculated x = ¥n");
for(i = 0; i < dim; i++){
        printf("%3d -> %3d: ", i, pivot[i]);
        printf("%25.17e ", vec_b[i]);
        printf("¥n");
}
```

直接法とは？＝LU分解と前進・後退代入

$$\begin{cases} 3x_1 - x_2 = -1 & \cdots ① \\ -6x_1 + 3x_2 = 2 & \cdots ② \end{cases}$$

①×2＋②

$$\begin{array}{rl} & 6x_1 - 2x_2 = -2 \\ +) & -6x_1 + 3x_2 = 2 \\ \hline & x_2 = 0 \end{array}$$

$x_2 = 0$を①に代入

$$3x_1 = -1$$

$$x_1 = -\frac{1}{3}$$

(答) $\begin{cases} x_1 = -\frac{1}{3} \\ x_2 = 0 \end{cases}$

- 左辺の未知数が１次式で記述できる複数式の方程式を連立「一次」方程式と呼ぶ
- 実用上重要な方程式
  - 微分方程式に基づくシミュレーション
  - 建築物の構造解析

  等々・・・
- 未知数は大量にあるケースが多い

→短い計算時間で求めるにはどうすればよいか？

LU分解と前進・後退代入(2×2の場合)

係数行列 A $\begin{bmatrix} 3 & -1 \\ -6 & 3 \end{bmatrix}$ $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ $=$ $\begin{bmatrix} -1 \\ 2 \end{bmatrix}$ 定数ベクトル$\mathbf{b}$

解$\mathbf{x}$

1. LU分解

$$\begin{bmatrix} 3 & -1 \\ -6/3 & 3-(-6/3)\times(-1) \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 3 & -1 \\ -2 & 1 \end{bmatrix}$$

$$\rightarrow L = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}, U = \begin{bmatrix} 3 & -1 \\ 0 & 1 \end{bmatrix}$$

$$A = LU \Leftrightarrow L(U\mathbf{x}) = \mathbf{b}$$

2. 前進代入

$$L\mathbf{y} = \mathbf{b} \Leftrightarrow \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

$$\Leftrightarrow \begin{cases} y_1 = -1 \\ -2y_1 + y_2 = 2 \end{cases}$$

$$\rightarrow \begin{cases} y_1 = -1 \\ y_2 = 2 + 2y_1 = 0 \end{cases}$$

3. 後退代入

$$U\mathbf{x} = \mathbf{y}$$

$$\Leftrightarrow \begin{bmatrix} 3 & -1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

$$\Leftrightarrow \begin{cases} 3x_1 - x_2 = -1 \\ x_2 = 0 \end{cases}$$

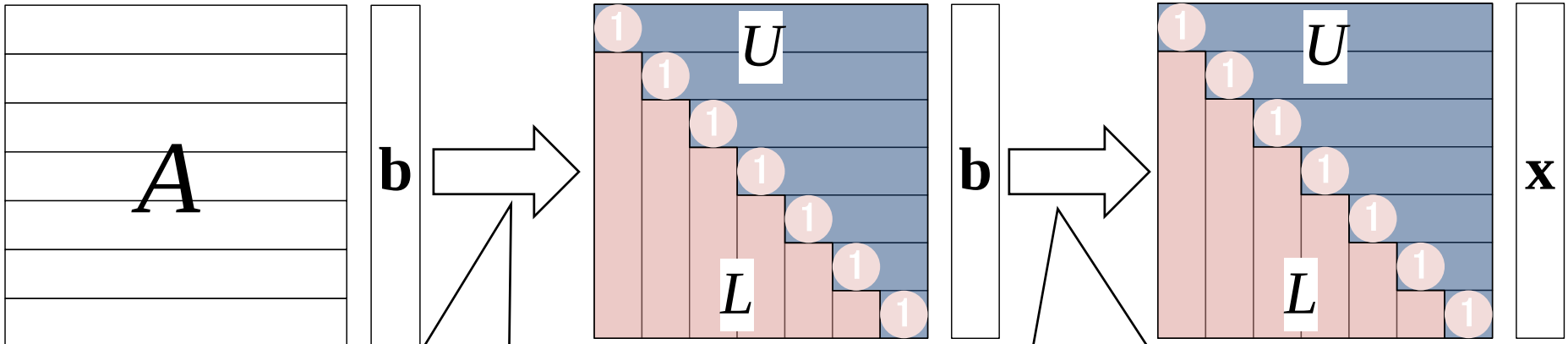
$$\rightarrow \begin{cases} x_1 = \dfrac{-1 + x_2}{3} = -\dfrac{1}{3} \\ x_2 = 0 \end{cases}$$

(答) $\begin{cases} x_1 = -\dfrac{1}{3} \\ x_2 = 0 \end{cases}$

規模の大きなLU分解と前進・後退代入

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

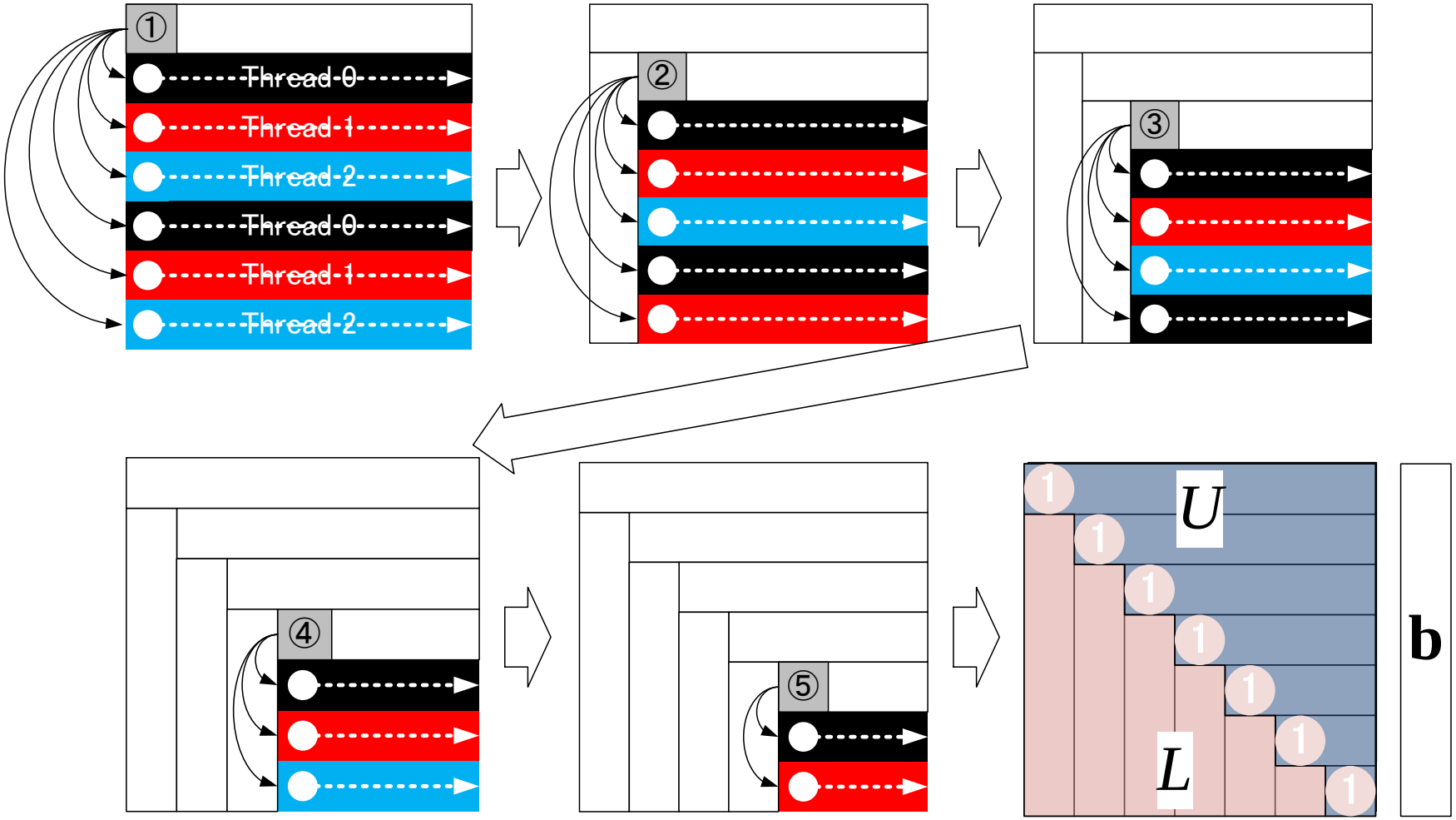
係数行列 A　　解$\mathbf{x}$　　定数ベクトル$\mathbf{b}$



LU分解
n^3に比例した計算量

前進・後退代入
n^2に比例した計算量

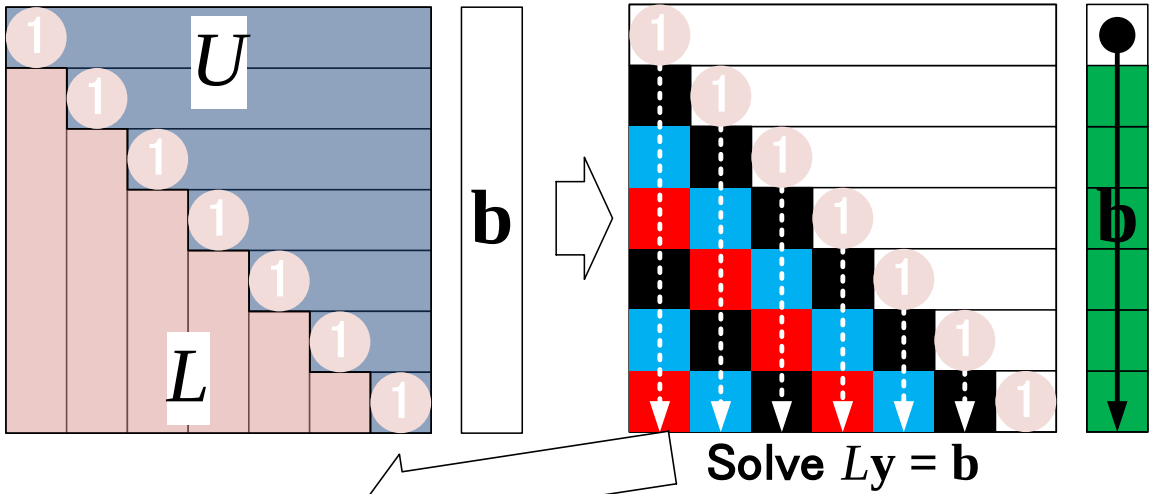
LU分解の並列化



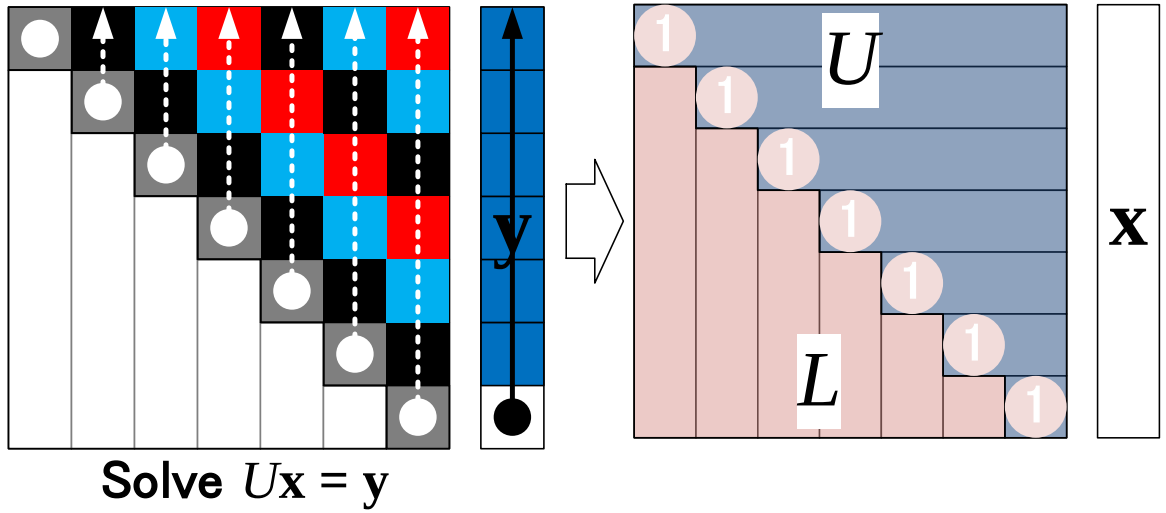
- 行（横方向）単位でスレッドに計算を割り当てる
- 計算が進むにつれて，並列化できる余地が減る

前進・後退代入の並列化

前進代入

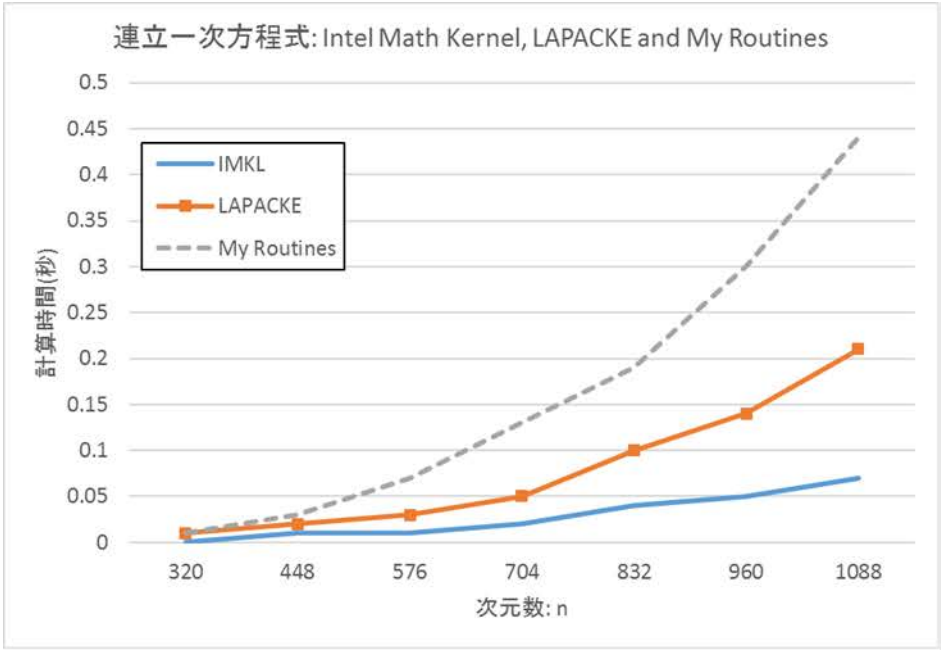


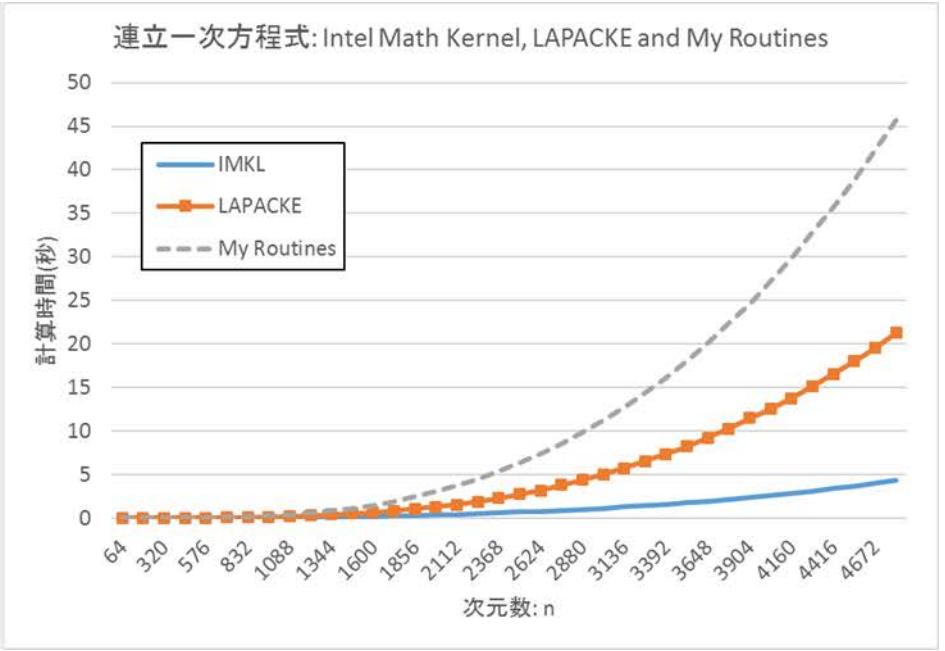
後退代入



- スレッド割り当ては市松模様に
- LU分解より計算量が少ないので並列化の効果も少ない

DGESV関数ベンチマーク





- ①IMKL，②LAPACKE（ソースをそのままコンパイルして利用），③自作ルーチンの順に高速
  - ブロック化アルゴリズム
  - SIMD命令利用
- 大規模問題になるとさらにその差が増す
- 「LAPACK/BLAS相当の線型計算を自作するな！」（ｂｙ　福井大・細田）

混合精度反復改良法（DSGESV）

表 4.5: LAPACK: DSGESV 関数

単精度-倍精度混合精度反復改良法で $\mathbf{b} := A^{-1}\mathbf{b}$ を求める

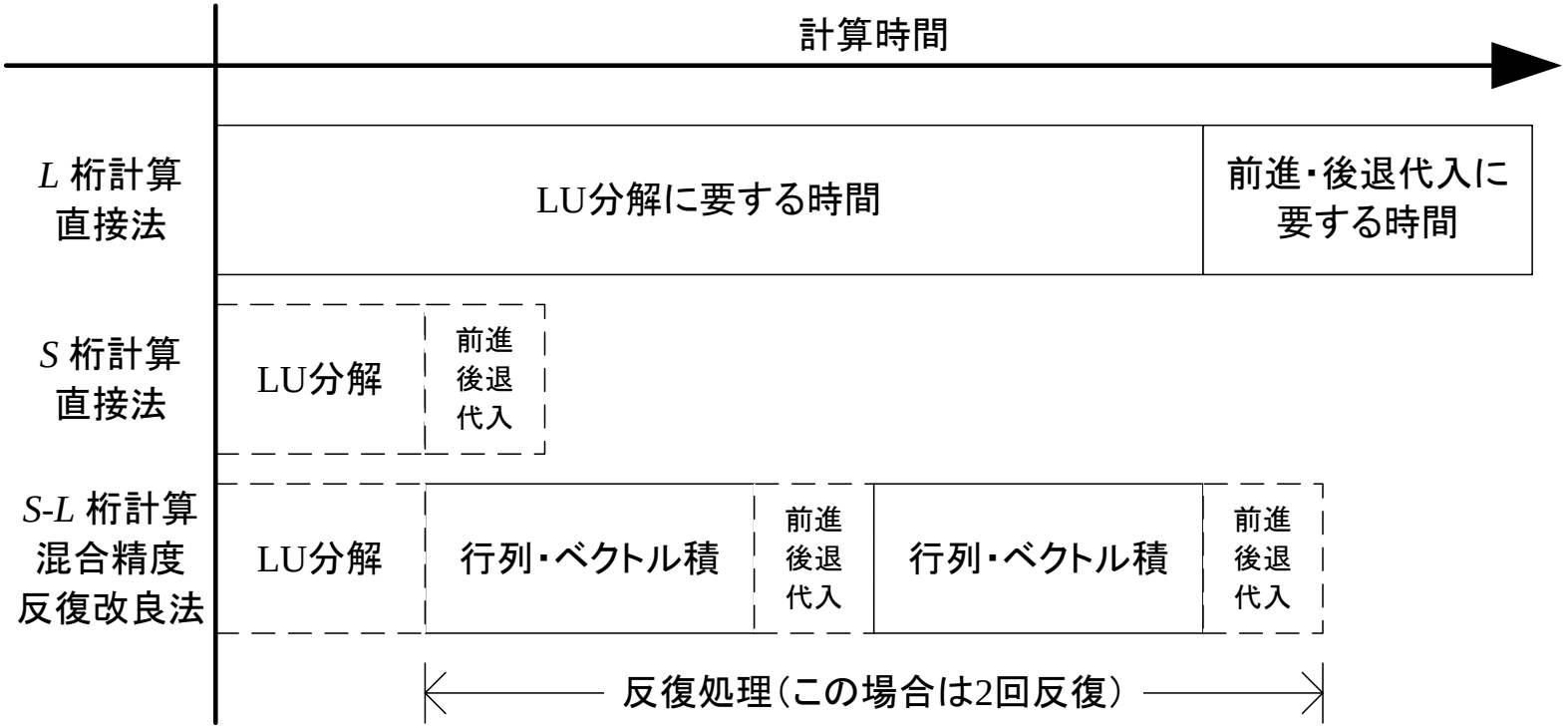
#include "lapache.h"	引数の意味
int LAPACKE_dsgesv(	
int matrix_order,	行列格納方式 LAPACK_ROW_MAJOR(行優先) または LAPACK_COL_MAJOR (列優先)
int n, int nrhs,	行列サイズ
double *a,	行列 A 格納先ポインタ (n × n)
int lda,	解ベクトルの本数
int *ipiv,	ピボット列へのポインタ
double *b,	ベクトル $\mathbf{b}$ 格納先 (n × nrhs)
int ldb	ベクトル $\mathbf{b}$ の本数
double *x,	ベクトル $\mathbf{x}$ 格納先
int ldx,	ベクトル $\mathbf{x}$ の本数
int *iter	反復回数
);	

返り値

$$\texttt{int info} = \begin{cases} 0 & \text{正常終了} \\ -i & i \text{ 番目の引数が異常値} \\ i & i \text{ 番目のピボットがゼロ} \end{cases}$$

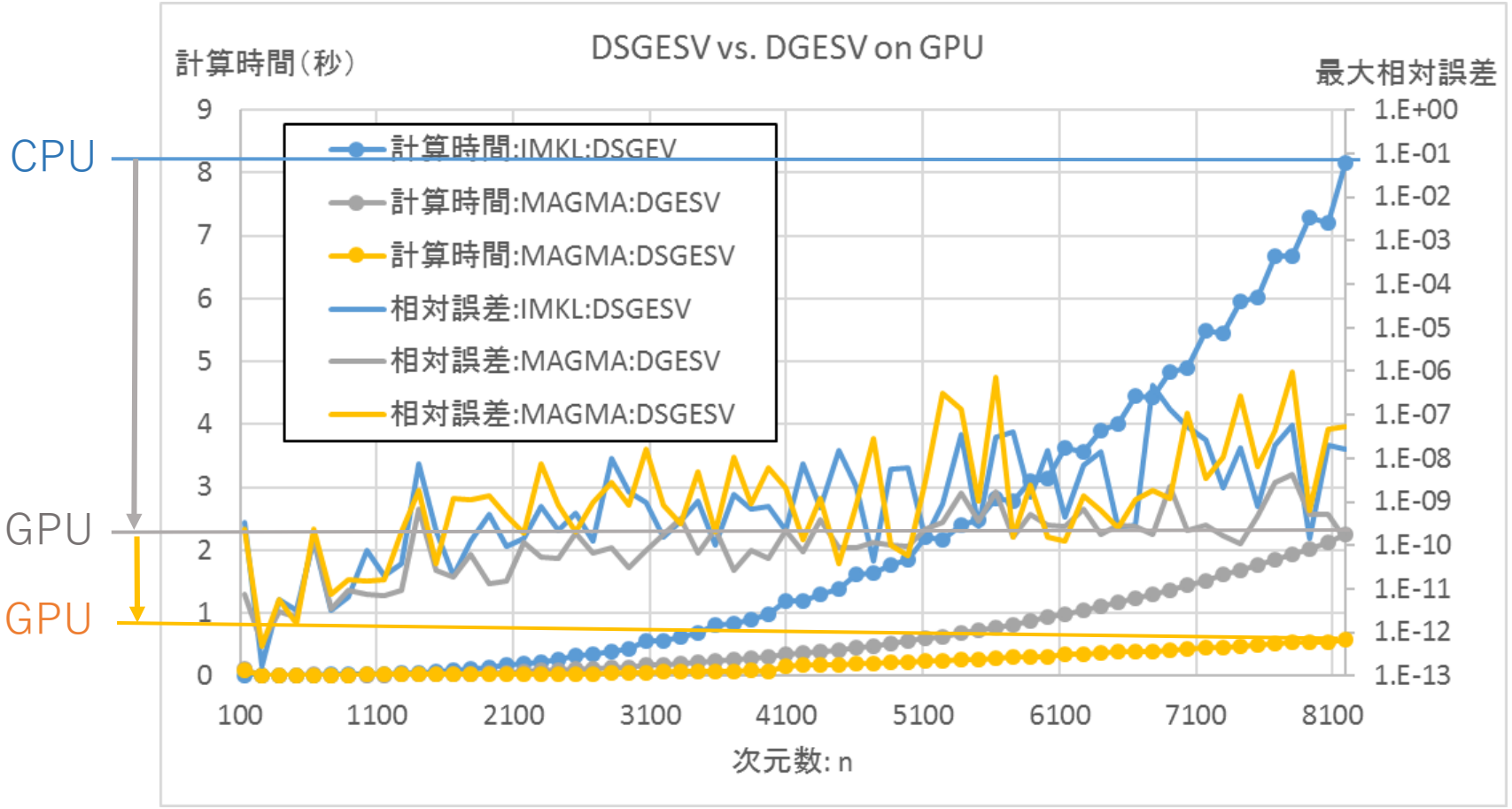
- LAPACK Version 3.2 (2008年)から追加
- 単精度と倍精度の混合精度バージョン

混合精度反復改良法（DSGESV）



- 単精度計算（*S* 桁）が高速なGPUでは効果的
- 倍精度計算（*L* 桁）が遅い環境＆良条件問題に対しては実用性が高い

CPUとGPUの処理速度の比較



- GPUはCPUより４倍近い高速化を達成
- DSGESVを使用することで更に倍以上の高速化を達成

(標準)固有値問題

- 右固有ベクトル $\mathbf{x}$: $A\mathbf{x} = \lambda\mathbf{x}$
- 左固有ベクトル $\mathbf{y}$: $\mathbf{y}^H A = \lambda\mathbf{y}^H$

  ここで $\mathbf{y}^H = \bar{\mathbf{y}}^T$

[質問1] 固有値問題と連立一次方程式の解を求める問題との本質的な違いは？

[質問2] 対角化不可能な正方行列と対角化可能な正方行列の違いは？

DSYEV(実対称), DGEEV(一般)

- 複素数が必要となる計算　→
- 実数で閉じている計算↓

表 4.6: LAPACK: DSYEV 関数

実対称行列 A の固有値・固有ベクトルを求める	
`#include "lapache.h"`	引数の意味
`int LAPACKE_dsyev(`	
`int matrix_order,`	行列 A の格納方式 `LAPACK_ROW_MAJOR`: 行優先 `LAPACK_COL_MAJOR`: 列優先
`char jobz,`	`'N'`: 固有値のみ求める, `'V'`: 固有ベクトルも求める (A に上書き格納)
`char uplo,`	`'U'`: A の上三角要素のみ格納, `'L'`: A の下三角要素のみ格納
`int n,`	行列 A のサイズ
`double *a,`	行列 A(n × n) の格納先ポインタ
`int lda,`	行列 A の実質サイズ
`double *w`	行列 A の固有値へのポインタ
`);`	

返り値

$$\texttt{int info} = \begin{cases} 0 & \text{正常終了} \\ -i & i \text{ 番目の引数が異常値} \\ i & i \text{ 個分のリダクションした非対角要素がゼロに収束しない} \end{cases}$$

表 4.8: LAPACK: DGEEV 関数

実行列 A の固有値・固有ベクトルを求める	
`#include "lapache.h"`	引数の意味
`int LAPACKE_dgeev(`	
`int matrix_order,`	行列 A の格納方式 `LAPACK_ROW_MAJOR`: 行優先 `LAPACK_COL_MAJOR`: 列優先
`char jobvl,`	`'N'`: 固有値のみ求める, `'V'`: 左固有ベクトルも求める (vl に格納)
`char jobvr,`	`'N'`: 固有値のみ求める, `'V'`: 右固有ベクトルも求める (vr に格納)
`int n,`	行列 A のサイズ
`double *a,`	行列 A(n × n) の格納先ポインタ
`int lda,`	行列 A の実質サイズ
`double *wr,`	A の固有値の実数部へのポインタ
`double *wi,`	A の固有値の虚数部へのポインタ
`double *vl,`	左固有ベクトルへのポインタ
`int ldvl,`	左固有ベクトルの本数
`double *vr,`	右固有ベクトルへのポインタ
`int ldvr`	右固有ベクトルの本数
`);`	

返り値

$$\texttt{int info} = \begin{cases} 0 & \text{正常終了} \\ -i & i \text{ 番目の引数が異常値} \\ i & i \text{ 個分の固有値を得て収束に失敗} \end{cases}$$

5. [応用] 積分方程式

積分方程式をLAPACKで解く！(1/5)

ここでは閉区間 $s,t \in [a,b] \subset \mathbb{R}$ において定義される解 $x(s)$ に対して定義された，次のハマースタイン (Hemmerstein) 型積分方程式を考えることにします。

$$x(s) = f(s) + \int_a^b K(s,t)g(t,x(t))dt \tag{8.1}$$

解が関数になりますので，その解析式が導出できれば良いのですが，一般的には積分自身が簡単に有限の式の形で表現できるものではありませんので，数値計算としては離散的な解の値，即ち

$$\mathbf{x} = [x(t_1)\ x(t_2)\ ...\ x(t_N)]^T = [x_1\ x_2\ ...\ x_N]^T$$

が得られれば良しとします。ここで $t_i \in [a,b]$, $x_i = x(t_i)$ $(i = 1,2,...,N)$ です。

このような前提があれば，この積分方程式 (8.1) を解くには，積分の部分を離散的な解の値を使った近似公式

$$\int_a^b K(s,t)g(t,x(t))dt \approx \sum_{j=1}^{N} w_j K(s,t_j)g(t_j,x_j)\ (w_j\text{は近似公式によって決まる定数}) \tag{8.2}$$

を使い，(8.1) 式に代入して，次のように設定すれば良いことになります。

$$x_i = f_i + \sum_{j=1}^{N} w_j K(s_i,t_j)g(t_j,x_j) \tag{8.3}$$

$$(i = 1,2,...,N)$$

積分方程式をLAPACKで解く！(2/5)

8.2　ゴラブ-ウェルシュの方法によるガウス型積分公式の分点計算

ガウス (Gauss) 型積分公式の分点 $\alpha_1, \alpha_2, ..., \alpha_N$ ($i < j$ の時，$\alpha_i > \alpha_j$ とする) は，三項漸化式

$$p_j(x) = (a_j x + b_j)p_{j-1}(x) - c_j p_{j-2}(x) \ (j = 1, 2, ..., N) \tag{8.4}$$

に基づいて定義される N 次直交多項式 $p_N(x)$ の零点

$$p_N(\alpha_i) = 0 \ \ (i = 1, 2, ..., N)$$

として表現できます。ここで，$a_j, b_j, c_j \in \mathbb{R}$ は直交多項式ごとに定まる定数であり，今回計算したルジャンドル (Legendre)，ラゲール (Laguerre)，エルミート (Hermite) 多項式の場合は表 8.1 のようになります。

表 8.1: 3 項漸化式の係数，重み関数，積分区間

直交多項式名	$p_{-1}(x)$	$p_0(x)$	a_j	b_j	c_j	$w(x)$	a	b
ルジャンドル	0	1	$(2j-1)/j$	0	$(j-1)/j$	1	-1	1
ラゲール	0	1	$-1/j$	$(2j-1)/j$	$(j-1)/j$	$\exp(-x)$	0	$+\infty$
エルミート	0	1	2	0	$2j-2$	$\exp(-x^2)$	$-\infty$	$+\infty$

積分方程式をLAPACKで解く！(3/5)

$$J = DTD^{-1}$$

$$= \begin{bmatrix} -\frac{b_1}{a_1} & \sqrt{\frac{c_2}{a_1 a_2}} & & & \\ \sqrt{\frac{c_2}{a_1 a_2}} & -\frac{b_2}{a_2} & \sqrt{\frac{c_3}{a_2 a_3}} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{\frac{c_{N-1}}{a_{N-2} a_{N-1}}} & -\frac{b_{N-1}}{a_{N-1}} & \sqrt{\frac{c_N}{a_{N-1} a_N}} \\ & & & \sqrt{\frac{c_N}{a_{N-1} a_N}} & -\frac{b_N}{a_N} \end{bmatrix}$$

1. Gauss-Legendre積分公式をDSTEQRで求める
2. 離散化した非線型方程式をDerivative-free解法で求める。

表 8.2: LAPACK: DSTEQR 関数

実対称三重行列 $T := S^T AS$ の固有値・固有ベクトルを求める

```
#include "lapache.h"
int LAPACKE_dsteqr(
  int matrix_order,
  char compz,
  int n,
  double *d,
  double *e,
  double *z,
  int ldz
);
```

引数の意味

- `int matrix_order`: 行列 T の格納方式
  - `LAPACK_ROW_MAJOR`: 行優先
  - `LAPACK_COL_MAJOR`: 列優先
- `char compz`:
  - `'N'`: 固有値のみ求める，
  - `'V'`: T に変換する前の実対称行列の固有ベクトルも求める (z に相似変換行列 S をあらかじめ格納しておく)
  - `'I'`: T の固有ベクトルも求める (z に格納)
- `int n`: 行列 A のサイズ
- `double *d`: 行列 T の対角要素格納先ポインタ (計算後に固有値が格納される)
- `double *e`: 行列 T の副対角要素格納先ポインタ
- `double *z`: 固有ベクトル格納先ポインタ
- `int ldz`: 固有ベクトルの本数

返り値

$$\texttt{int info} = \begin{cases} 0 & \text{正常終了} \\ -i & i \text{ 番目の引数が異常値} \\ > 0 & 30n \text{ 回反復後，収束せず失敗} \end{cases}$$

積分方程式をLAPACKで解く！(4/5)

8.3 デリバティブフリーな非線型方程式の解法

n 次元非線型写像 $\mathbf{F}:\mathbb{R}^N \to \mathbb{R}^N$ を用いて定義される非線型方程式 (8.11)

$$\mathbf{F}(\mathbf{x}) = 0 \tag{8.11}$$

に対しては様々な解法が提案されており，今現在も盛んに研究されています。ここではそのごく一部のみを LAPACK/BLAS を使って実装していきます。

通常，$\mathbf{F}$ がある領域 $\Omega \subset \mathbb{R}^N$ において滑らかな関数であれば，ニュートン (Newton) 法が適用できます。

$$\mathbf{x}_{k+1} := \mathbf{x}_k - \left[\frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x}_k)\right]^{-1} \mathbf{F}(\mathbf{x}_k) \tag{8.12}$$

ここで $[\partial\mathbf{F}/\partial\mathbf{x}]$ は $\mathbf{F}$ の偏導関数 $\partial F_i/\partial x_j$ を要素とするヤコビ (Jacobi) 行列です。ヤコビ行列の逆行列を乗じている部分は連立一次法方程式を解くことで得るようにします。これは LAPACK の xGESV 関数 (表 2.2) を使うことで計算できます。

$\mathbf{F}$ が微分不可能な場合にも適用するために，このヤコビ行列を 1 階差分商行列 $[\mathbf{x}_{k-1}, \mathbf{x}_k; \mathbf{F}]$ で置き換えた割線法というものがあります。

$$\begin{aligned} \mathbf{y}_k &:= \mathbf{x}_k - [\mathbf{x}_{k-1}, \mathbf{x}_k; \mathbf{F}]^{-1}\mathbf{F}(\mathbf{x}_k) \\ \mathbf{x}_{k+1} &:= \mathbf{y}_k - [\mathbf{x}_{k-1}, \mathbf{x}_k; \mathbf{F}]^{-1}\mathbf{F}(\mathbf{y}_k) \end{aligned} \tag{8.13}$$

ここで 1 階差分商行列

$$[\mathbf{u}, \mathbf{v}; \mathbf{F}]_{ij} = \frac{1}{u_j - v_j}\left(F_i(u_1, \ldots, u_j, v_{j+1}, \ldots, v_n) - F_i(u_1, \ldots, u_{j-1}, v_j, \ldots, v_n)\right)$$

として計算できます。但し，差分商を構成する二つのベクトル $\mathbf{u}$, $\mathbf{v}$ が近接していると桁落ちが起きますし，ニュートン法に比べると収束性に劣る，即ち，収束のスピードが遅くなりがちです。

積分方程式をLAPACKで解く！(5/5)

$$x(s) = 1 + \frac{1}{2}\int_0^1 K(s,t)(|x(t)| + (x(t))^2)dt$$

$$(s \in [0,1])$$

$$\text{ここで} K(s,t) = \begin{cases} (1-s)t & t \le s \\ s(1-t) & t \ge s \end{cases}$$

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\|_2 \le 10^{-10}\|\mathbf{x}_k\|_2 + 10^{-50}$$

- 反復回数1回分，Derivative-free解法が高速(n>128)
- GPUで高速になるかどうかは研究課題

表 8.4: 割線法とデリバティブフリー法の計算時間と並列化効率

割線法				
N	反復回数	LAPACK(a)	IMKL(4 スレッド)(b)	速度向上率=(a)/(b)
128	6	0.032	0.035	0.92
256	6	0.18	0.31	0.59
512	6	1.2	0.52	2.35
1024	6	9.6	3.3	2.94
2048	6	83.3	23.5	3.55
デリバティブフリー法				
N	反復回数	LAPACK(a)	IMKL(4 スレッド)(b)	速度向上率=(a)/(b)
128	5	0.034	0.23	0.15
256	5	0.15	0.11	1.36
512	5	1.1	0.45	2.33
1024	5	8.3	2.8	2.94
2048	5	71.2	19.1	3.72

まとめ

- 線型計算のグローバルスタンダードである LAPACK/BLASの概要と機能（のごく一部）
- 派生型の高性能ライブラリの性能評価
- 非線型問題への応用事例

[サポートページ] https://na-inet.jp/lapack/

→ GitHubにてサンプルプログラムは公開中