研究集会「超大規模行列の数理的諸問題とその高速解法」
2007年3月7日（水） 東京大学工学部

行列計算ライブラリインタフェース SILC を用いた非対称線形方程式の数値解析

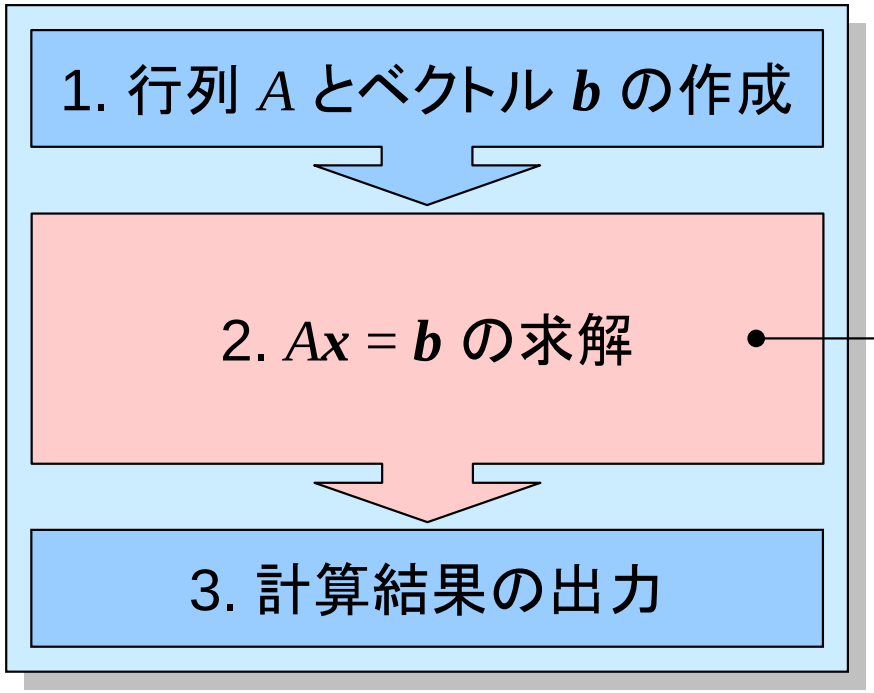
梶山民人（JST） 額田彰（JST） 須田礼仁（東大）
長谷川秀彦（筑波大） 西田晃（中央大）

線形方程式の数値解析

- 解くのは同じ線形方程式 $Ax = b$ だが
  - いろいろな解法で解く
    - 直接法（密, 帯, 疎）, 反復法（多種多様）
    - 異なるインタフェース
  - 倍精度で解いたり多倍精度で解いたりする
  - 丸めモードを変えて解く
- １本の応用プログラムを作るのとは異なる

典型的な数値解析プログラム

- データ入出力部分はほぼ一定で求解部分のバリエーションが多い
- さまざまな求解方法を手軽に試したい

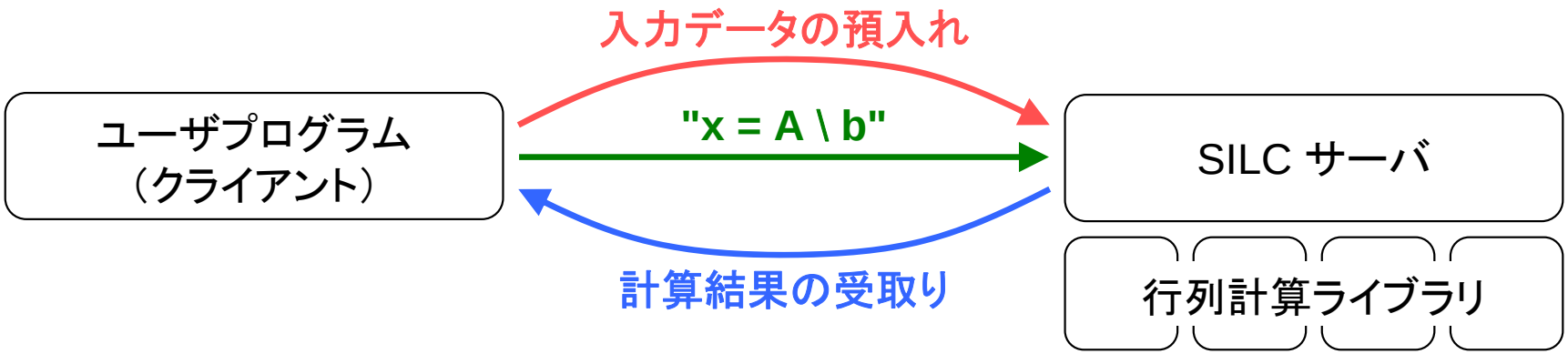


- 解法
- 行列の格納形式
- 演算精度
- 丸めモード
- 環境（SMP マシン，PC クラスタ，ベクトル機など）

に応じて書き換えが必要

行列計算ライブラリインタフェースSILC

- **S**imple **I**nterface for **L**ibrary **C**ollections
  - ライブラリ、計算環境、プログラミング言語に依存しないインタフェース
- 次の3ステップで行列計算ライブラリを利用
  1. 入力データの預入れ
  2. 文字列(数式)による計算の指示
  3. 計算結果の受取り



線形方程式 $Ax = b$ の求解

- 従来法のプログラム（ScaLAPACK を利用する場合）

```
double *A, *B;
int desc_A[9], desc_B[9], *ipiv, info;
/* 行列 A とベクトル b の作成 */
pdgesv(N, NRHS, A, IA, JA, desc_A, ipiv, B, IB,
  JB, desc_B, &info);
```

- SILC のユーザプログラム（ライブラリに依らず一定）

```
silc_envelope_t A, b, x;
/* 行列 A とベクトル b の作成 */
SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("x = A\\b"); /* pdgesv の呼出し */
SILC_GET(&x, "x");
```

SILC の特徴

- 使用するライブラリに依存しない
  - 異なるライブラリの解法、前処理、行列の格納形式、演算精度を同じインタフェースで利用できる
- 計算環境に依存しない
  - 逐次、共有メモリ並列、分散並列の計算環境をサポート
  - ユーザプログラムの修正なしに他の環境を利用できる
- プログラミング言語に依存しない
  - どの言語でも同じ数式でライブラリを呼出せる
  - C, Fortran, Java, Python, Octave から利用可

SILC の2種の実装

- OpenMP 版：逐次環境、共有メモリ並列環境
  - バージョン 1.2 を公開中
- MPI 版：分散並列環境
  - 開発中
- 対応する計算環境の組合せ

ユーザプログラム（クライアント）	SILC サーバ	対応バージョン
逐次	逐次	OpenMP 版
逐次	共有メモリ並列	OpenMP 版
逐次	分散並列	MPI 版
分散並列	分散並列	MPI 版

主な機能

- データ構造
  - データ型：スカラー、ベクトル、行列、3次元配列
  - 演算精度：整数、実数、複素数（それぞれ単精度または倍精度）
  - 行列の格納形式：密, 帯（LAPACK 形式）, 疎（CRS, JDS）
- 数式の構成要素
  - 四則演算（+, −, *, /, %）, 線形方程式の求解（A \ b）
  - 要素ごと乗算, 要素ごと除算
  - 共役転置（A'）, 複素共役（A~）
  - 関数, 手続き
    - 例：n = sqrt(b' * b)　ベクトルの2ノルム
  - 添字
    - 例：A[1:5,1:5]　5×5 の部分行列

SILC の使用例:
非対称線形方程式の数値解析 *

1. $A, \boldsymbol{b}$ を与えて BiCG 法などで解く
2. 演算精度を変える
3. $A^T A\boldsymbol{x} = A^T\boldsymbol{b}$ を CG 法で解く(CGNR 法)
4. CGNR 法を SILC の数式で書く
5. 対称化した拡大方程式に対する CG 法を SILC の数式で書く
6. 2グリッド法を SILC の数式で書く

* H. Hasegawa, T. Sogabe, T. Ogita, T. Kajiyama: Comparison of conjugate gradient method for nonsymmetric matrices. Joint GAMM−SIAM Conference on Applied Linear Algebra, July 24−27, 2006, Düsseldorf, Germany.

1. A, $\boldsymbol{b}$ を与えて BiCG 法などで解く

```
silc_envelope_t A, b, x, h;
/* 行列 A とベクトル b の作成 */
SILC_INIT();
SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("x = A \\ b"); /* 解法の呼出し */
SILC_GET(&x, "x");
SILC_EXEC("h = get_residual_history()");
SILC_GET(&h, "h");
SILC_FINALIZE();
```

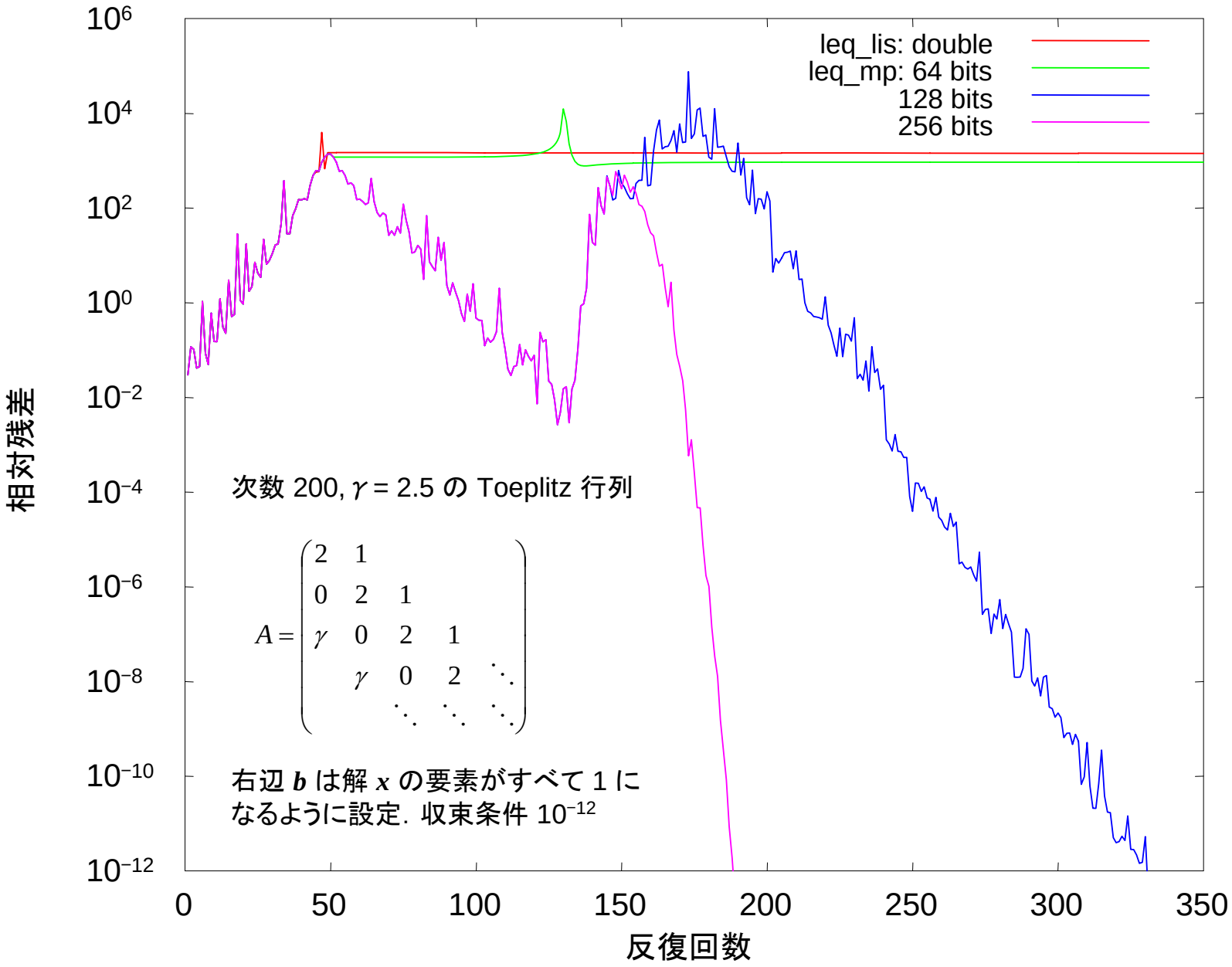
2. 演算精度を変える

- 1つのライブラリにつき1つのモジュール
- 使用するモジュールを prefer 文で選択

```
/* Lis モジュール(倍精度)で求解 */
SILC_EXEC("prefer leq_lis");
SILC_EXEC("x1 = A \\ b");
SILC_EXEC("h1 = get_residual_history()");

/* mp モジュール(GNU MP による多倍精度)で求解 */
SILC_EXEC("prefer leq_mp");
SILC_EXEC("x2 = A \\ b");
SILC_EXEC("h2 = get_residual_history()");
```

BiCG 法の収束の演算精度依存性



$$A = \begin{pmatrix} 2 & 1 & & & \\ 0 & 2 & 1 & & \\ \gamma & 0 & 2 & 1 & \\ & \gamma & 0 & 2 & \ddots \\ & & \ddots & \ddots & \ddots \end{pmatrix}$$

BiCG 法の演算精度と実行時間

- ユーザプログラムと SILC サーバを同じ PC で実行
  - Intel Pentium4 3.40 GHz, 主記憶 1 GB, Windows XP
- 反復回数 over ＝ 1,000 回で収束せず
- 1反復あたりの時間は増えるが反復回数は減る
  - トータルで速くなる可能性がある（試す価値あり）

モジュール	演算精度	反復回数	求解時間（秒）	実行時間（秒）	1反復あたりの時間（秒）
leq_lis	double	over	0.0094	–	0.0000094
leq_mp	64 bits	over	0.5958	–	0.0005958
	128 bits	331	0.2407	0.3353	0.0007272
	256 bits	189	0.1772	0.2735	0.0009378

100倍

3. $A^T Ax = A^T b$ を CG 法で解く（CGNR 法）

```
silc_envelope_t A, b, x, h;
/* 行列 A とベクトル b の作成 */
SILC_PUT("A", &A);
SILC_PUT("b", &b);
SILC_EXEC("AT = A';
           x = (AT * A) \\ (AT * b)");
SILC_GET(&x, "x");
SILC_EXEC("h = get_residual_history()");
SILC_GET(&h, "h");
```

CGNR 法の収束性

- Matrix Market の非対称実行列 80 個
  - 右辺 $\boldsymbol{b}$ は解 $\boldsymbol{x}$ の要素がすべて 1 になるように設定
  - 収束条件 : 10^{-12}, 反復回数の上限 : 10,000
  - 解の精度がよい : 真の解との相対誤差が 10^{-8} 未満
- 約3分の1が CGNR 法でのみ解けた

求解の可否	行列の数	CGNR 法の方が 反復回数が少なく 解の精度がよい行列の数
両方の解法で収束	21	8
BiCG 法 ($A\boldsymbol{x} = \boldsymbol{b}$) のみ収束	7	—
CGNR 法 ($A^TA\boldsymbol{x} = A^T\boldsymbol{b}$) のみ収束	26	5
いずれも収束せず	26	—
合計	80	13

4. CGNR 法を SILC の数式で書く

- ループと収束判定はユーザプログラムの記述言語で実現
- 行列 A の格納形式や演算精度に依らない

```
/* A: 係数行列, b: 右辺ベクトル */
SILC_EXEC("AT = A'");
SILC_EXEC("rho_old = 1.0");
SILC_EXEC("n = length(b)");
SILC_EXEC("p = zeros(n, 1)");
SILC_EXEC("x = zeros(n, 1)");
SILC_EXEC("r = AT * b");
for (i = 1; i <= maxiter; i++) {
  SILC_EXEC("rho = r' * r");
  SILC_EXEC("beta = rho / rho_old");
  SILC_EXEC("p = r + beta * p");
  SILC_EXEC("q = AT * (A * p)");
  SILC_EXEC("alpha = rho / (p' * q)");
  SILC_EXEC("r = r - alpha * q");
  SILC_EXEC("x = x + alpha * p");
  /* 収束判定 */
}
/* x: 解ベクトル */
```

5. 対称化した方程式に対するCG法

- 条件数不変
- c の採り方に選択肢

$$\begin{pmatrix} & A^T \\ A & \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c \\ b \end{pmatrix}$$

```
SILC_EXEC("AT = A'; r = b - A * x; rr = c - AT * y");
SILC_EXEC("p = r; pp = rr");
for (i = 1; i <= maxiter; i++) {
  SILC_EXEC("rho = (r' * r + rr' * rr)");
  SILC_EXEC("tmp = (p' * (A * pp) + pp' * (AT * p))");
  SILC_EXEC("alpha = rho / tmp");
  SILC_EXEC("x = x + alpha * pp");
  SILC_EXEC("r  = r  - alpha * (A * pp)");
  SILC_EXEC("rr = rr - alpha * (AT * p)");
  /* 収束判定 */
  SILC_EXEC("beta = (r' * r + rr' * rr) / rho");
  SILC_EXEC("p = r + beta * p; pp = rr + beta * pp");
}
```

5. 対称化した方程式に対するCG法

- Matrix Market の非対称実行列 80 個
  - 右辺 $\boldsymbol{b}$ は解の要素がすべて 1 になるように設定
  - 収束条件：10^{-12}，反復回数の上限：10,000
  - 6通りの $\boldsymbol{c}$ の選択肢を試した（後述）
  - 解の精度がよい：（$\boldsymbol{c}$ に依らず）真の解との相対誤差が 10^{-8} 未満
- 約1割が対称化した方程式に対する CG 法でのみ解けた

求解の可否	行列の数	拡大方程式に対する CG 法の方が反復回数が少なく解の精度がよい行列の数
両方の解法で収束	13	4
BiCG 法のみ収束	15	—
拡大方程式に対する CG 法のみ収束	9	5
いずれも収束せず	43	—
合計	80	9

5. 対称化した方程式に対するCG法

- 数字は反復回数（over = 10,000 回で収束せず）
- 最良（赤），最悪（青）の $\boldsymbol{c}$ の選択肢は行列依存

行列	$\boldsymbol{c}$ の選択肢					
	$\boldsymbol{c} = \boldsymbol{b}$	$\boldsymbol{c} = P\boldsymbol{b}$	$\boldsymbol{c} = (1,0,...,0)^T$	$\boldsymbol{c} = (1,1,...,1)^T$	$\boldsymbol{c}$ = 乱数 × $\|\boldsymbol{b}\|$	$\boldsymbol{c}$ = 乱数
gre__512	**487**	492	814	488	**1,166**	834
impcol_d	2,842	2,748	**1,804**	3,066	**3,075**	2,573
jpwh_991	1,038	1,023	937	**over**	950	**922**
orani678	7,978	8,126	**7,876**	8,361	**8,655**	8,134
shl__200	4,125	**3,273**	**over**	4,758	8,113	4,147
shl__400	6,035	3,537	**9,117**	over	7,388	**3,066**
shl____0	3,575	2,568	5,580	**2,562**	**9,902**	3,503
str__400	2,499	**2,005**	2,208	2,453	**2,594**	2,259
str__600	7,146	8,687	**5,021**	**over**	7,617	**over**

SILC による非対称線形方程式の解析

- ✓ ライブラリ呼出し（A \ b）
  1. $A, \boldsymbol{b}$ を与えて BiCG 法などで解く
  2. 演算精度を変える
  3. $A^T A\boldsymbol{x} = A^T\boldsymbol{b}$ を CG 法で解く（CGNR 法）
- ✓ 反復解法を SILC の数式で記述
  4. CGNR 法
  5. 対称化した拡大方程式に対する CG 法
- ✓ 合わせ技
  6. 2グリッド法を SILC の数式で書く

2グリッド法を SILC で書く

- 解法全体を SILC の数式で記述
- スムージング, 粗い格子での反復解法を \ 演算子で

```
/* A: 係数行列, b: 右辺ベクトル, R: 制約演算子, P: 補完演算子 */
SILC_EXEC("split(A, L, D, U)");
SILC_EXEC("LD = L + D");
SILC_EXEC("DU = D + U");
SILC_EXEC("Ac = R * A * R'");          /* 粗い格子の係数行列 */
for (i = 1; i <= maxiter; i++) {
  SILC_EXEC("x = LD \\ (b - U * x)");  /* プレスムージング */
  SILC_EXEC("rc = R * (b - A * x)");   /* 残差を粗い格子に移す */
  SILC_EXEC("uc = Ac \\ rc");          /* 粗い格子での反復解法 */
  SILC_EXEC("x += P * uc");            /* 細かい格子で解を修正 */
  SILC_EXEC("x = DU \\ (b - L * x)");  /* ポストスムージング */
  /* 収束判定 */
}
/* x: 解ベクトル */
```

まとめ

- SILC を用いた非対称線形方程式の数値解析
  - ライブラリ呼出し（A \ b）
  - SILC の数式による反復解法の記述
  - 上記2つの併用
- 特定のライブラリや計算環境に依存しない方法でユーザプログラムを作成可能
  - 種々のライブラリをどの言語でも同じ方法で呼び出せる
  - サーバを並列環境に移すだけで計算が並列化される
- 数値解析のアイデアをより容易に試せる

SILC の入手先

http://ssi.is.s.u-tokyo.ac.jp/silc/

- OpenMP 版 SILC v1.2 を公開中
  - ソースコード（Unix/Linux, Mac OS X, Windows）
  - Win32 用バイナリ（ダウンロードしてすぐ実行可）
  - サンプルプログラム、ドキュメント