P1-4　複数の前処理と格納形式を持つ反復解法ライブラリ

小武守 恒（JST）　藤井 昭宏（工学院大学）　中島 研吾（東京大学）
長谷川 秀彦（筑波大学）　西田 晃（東京大学）

1. はじめに

線型方程式 $Ax = b$ の解法として多数の反復解法と前処理が提案されている。解くべき問題に依存して、反復解法と前処理の選択によっては、収束するまでに多くの反復回数が必要となったり、収束しない場合がある。また、これらのアルゴリズムの計算時間は、計算環境とデータの格納形式にも強く依存して変化する。この問題を解決するためには、反復解法、前処理の組み合わせ、データ格納形式を様々な計算環境上で同時に評価できる反復解法ライブラリが必要である。

2. 解法・前処理・格納形式・並列化

対象とする反復解法は、大規模実行列用で定常（SOR 等）、非定常（GPBiCG 等）など 10 種類程度を用いる。前処理としては、よく知られているスケーリング、不完全 LU 分解などに加えて、最近提案されている $I + S$ 型[1)]、SA-AMG[2)]、SOR などの反復法を用いる Hybrid[3)] などである。データの格納形式は CRS（Compressed Row Storage）など 10 種類程度である。並列化には、共有メモリに対しては OpenMP を、分散メモリに対しては MPI を用いる。領域間の「通信テーブル」は、GeoFEM[4)] を参考にする。

3. 実行例

分散共有メモリ型計算機 SGI Altix3700（CPU：Intel Itanium2 1.3GHz、主記憶：32GB、OS：SGI Advanced Linux 2.4）を使用した。Intel C/C++ Compiler ver. 8.1 で、最適化オプション-O3 を使用した。係数行列 A は、直方体領域において、境界条件を $\phi = 0, Z = Z_{\min}$ としたポアソン方程式 $-\Delta\phi = 1$ を離散化したものである。行列の格納形式は CRS とした。収束判定条件は 10^{-8} とした。

次数 N を 27,000 とした場合の計算時間と反復回数を**表 1** に示す。CG 法に対して前処理を施した場合の計算時間と反復回数を**表 2** に示す。上記の 2 つの例の残差履歴を**図 1** に示す。$N = 10^6$ の場合の OpenMP による並列化を行った例を**表 3** に示す。

4. おわりに

ポスターでは、開発中のライブラリの詳細な機能や並列化について示す。また、係数行列が非対称の場合についても示す予定である。

結果として、解法・前処理・格納形式の組み合わせが、複数の問題、多様な環境（逐次・共有並列・分散並列）で利用できるようにすることで、ユーザに、問題と環境に応じたよりふさわしいプログラムを提供する。

謝辞　本研究は、科学技術振興機構（JST）戦略的創造研究推進事業（CREST）「大規模シミュレーション向け基盤ソフトウェアの開発」プロジェクトの一部として実施した。

表 1　各解法の計算時間と反復回数（$N = 27,000$）

反復解法	計算時間（sec.）	反復回数
CG	0.40	94
BiCG	0.93	94
CGS	0.64	70
BiCGSTAB	0.65	67
BiCGSTAB(2)	0.70	70
GPBiCG	0.58	57
Orthomin(6)	0.57	93
GMRES(60)	1.00	143
QMR	1.08	93

表 2　前処理つき CG 法の計算時間と反復回数（$N = 27,000$）

前処理	計算時間（sec.）	反復回数
なし	0.40	94
対角スケーリング	0.14	29
ILU(0)	0.84	42
SSOR	0.87	43

表 3　OpenMP による並列化（CG、前処理なし、CRS、$N = 10^6$）

Threads	計算時間（sec.）	速度向上
1	52.14	1.00
2	30.23	1.73
4	14.91	3.50
8	7.12	7.32
16	3.57	14.60
32	2.05	25.43

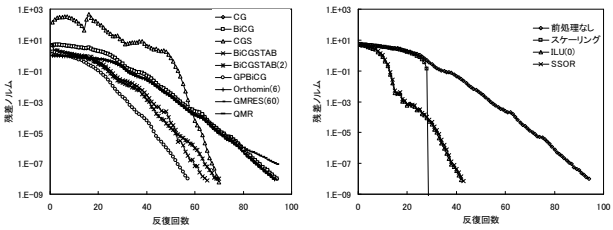


図 1　各解法の収束履歴と前処理つき CG 法の収束履歴

参考文献

1) Toshiyuki Kohno, Hisashi Kotakemori and Hiroshi Niki: Improving the Modified Gauss-Seidel Method for Z-matrices, *Linear Algebra and its Applications*, Vol.267, pp. 113–123 (1997).
2) 藤井昭宏, 西田晃, 小柳義夫: 領域分割による並列 AMG アルゴリズム, 情報処理学会論文誌, Vol.44, No.SIG6(ACS1), pp. 1–8 (2003).
3) 阿部邦美, 張紹良, 長谷川秀彦, 姫野龍太郎: SOR 法を用いた可変的前処理付き一般化共役残差法, 日本応用数理学会論文誌, Vol. 11, No. 4, pp. 157–170 (2001).
4) 奥田洋司, 中島研吾: 並列有限要素解析 [I] クラスタコンピューティング, 培風館 (2004).

P1-4 複数の前処理と格納形式を持つ反復解法ライブラリ

小武守 恒(JST) 藤井 昭宏(工学院大学) 中島 研吾(東京大学)
長谷川 秀彦(筑波大学) 西田 晃(東京大学)

目標

- 様々な格納形式の入力を受け付ける
- 様々な解法と前処理を組み合わせて使える

今後の課題

- 分散並列(MPI)

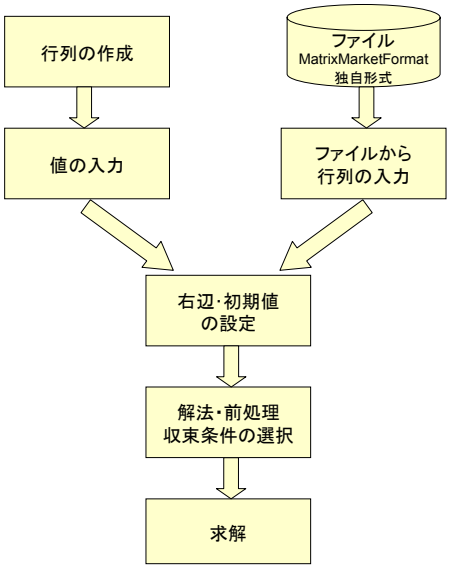
ライブラリの構成

主に以下の3つのルーチンからなる

- 行列処理ルーチン
  - 作成、ファイル入力、変換
- 前処理ルーチン
- 反復解法ルーチン

求解までの流れ

- 指定の格納形式の行列の作成
- 行列に値を入力
- 右辺・初期値の設定
- 解法・前処理・収束条件の選択
- 求解



主な関数

```
int ssi_matrix_create(SSI_MATRIX_PARAMS
    params, SSI_MATRIX **Amat)
```

機　能:　指定された格納形式の行列を作成
入　力:　params(行列の格納形式、次数等)
出　力:　Amat(指定された格納形式の行列)

```
int ssi_matrix_input(SSI_MATRIX **A,
    SSI_SCALAR *b[], SSI_MATRIX_INOUT minout,
    int matrix_type)
```

機　能:　ファイルから行列データ等を読み込み指定の格納形式の行列に代入
入　力:　minout(ファイル名、ファイルフォーマット等の指定)
　　matrix_type(格納形式の指定)
出　力:　A(指定された格納形式の行列)
　　b(右辺)

```
int ssi_matrix_convert(SSI_MATRIX *Ain,
    SSI_MATRIX **Aout, int matrix_type)
```

機　能:　指定の格納形式へ行列を変換
入　力:　Ain(変換前の行列)
　　matrix_type(格納形式の指定)
出　力:　Aout(変換後の行列)

```
int ssi_solve(SSI_MATRIX *A, SSI_SCALAR
    b[], SSI_SCALAR x[], SSI_SCALAR
    params[], int options[],
    SSI_COMMTABLE *comm)
```

機　能:　求解
入　力:　A(係数行列)
　　b(右辺)
　　x(初期値)
　　params[](収束判定条件等)
　　options[](解法・前処理の指定等)
　　comm(通信テーブル)
出　力:　x(解)

格納形式

Point	Dense	(DNS)
	Coordinate	(COO)
	Compressed Row Storage	(CRS)
	Compressed Column Storage	(CCS)
	Modified Compressed Sparse Row	(MSR)
	Diagonal	(DIA)
	Ellpack-Itpack generalized diagonal	(ELL)
	Jagged Diagonal	(JDS)
Block	Block Sparse Row	(BSR)
	Block Sparse Column	(SBC)
	Variable Block Row	(VBR)

反復解法

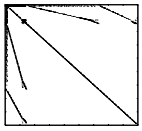
対称	CG
非対称	BiCG
	CGS
	BiCGSTAB
	BiCGSTAB(l)
	GPBiCG
	Orthomin(m)
	GMRES(m)
	QMR
	Jacobi
	Gauss-Seidel
	SOR

実行例

実行環境：分散共有メモリ型計算機 SGI Altix3700(Itanium2 1.3GHz, 32GB, SGI Advanced Linux2.4)

格納形式：CRS

収束条件：10^{-8}



MatrixMarket

ポアソン方程式 $-\Delta\phi = 1$

境界条件 $\phi = 0, Z = Z_{\min}$

MEMPLUS: Computer component design memory circuit

27,000 x 27,000 681,472 entries

17,758 x 17,758 126,150 entries

解法	ポアソン方程式		MatrixMarket MEMPLUS	
	計算時間(sec.)	反復回数	計算時間(sec.)	反復回数
CG	0.40	94	1.72	1011
BiCG	0.93	94	2.82	798
CGS	0.64	70	1.84	473
BiCGSTAB	0.65	67	4.21	998
BiCGSTAB(2)	0.70	70	9.32	2104
GPBiCG	0.58	57	3.50	651
Orthomin(6)	0.57	93	3.05	852
GMRES(60)	1.00	143	6.99	1700
QMR	1.08	93	3.54	721

前処理

- Jacobi
- SSOR
- Block Jacobi
- ILU
- Hybrid

  前処理行列をMとする。反復過程で解かなければならないMz=vの代わりにAz=vをSOR法などで近似的に解く
- SA-AMG

  問題行列AからAと似た性質を持つサイズの小さい行列A'を生成する(構築部)。その後、サイズの小さい問題A'x'=b'とAx=bを交互に緩和法を適用しながら効率よく解く(解法部)。
- I+S型

  単位行列と係数行列A=(a_{ij})のi+1の部分の符号を逆にしたもの

$$S = \begin{cases} -a_{ij} & j = i+1 \\ 0 & else \end{cases}$$

バリエーション

- I+S(m) :対角側より非対角要素をm個使用
- I+Smax :各行絶対値最大の要素を1個使用

実行例

前処理つきCG法の計算時間と反復回数

ポアソン方程式

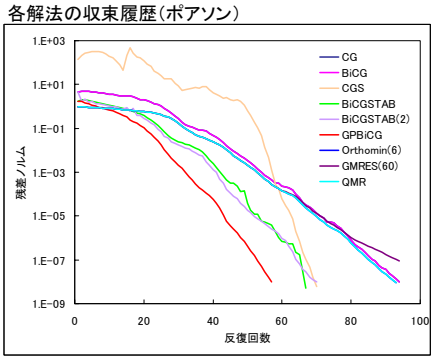
前処理	計算時間(sec.)	反復回数
なし	0.40	94
対角スケーリング	0.14	29
ILU(0)	0.84	42
SSOR	0.87	43
Hybrid(SOR)	2.70	30
SA-AMG	2.07	19

前処理つきCGS法の計算時間と反復回数

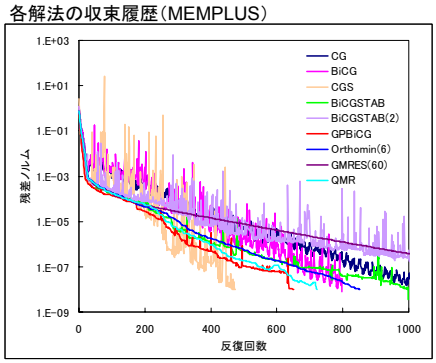
MatrixMarket MEMPLUS

前処理	計算時間(sec.)	反復回数
なし	1.84	473
対角スケーリング	0.64	140
ILU(0)	0.92	133
SSOR	0.52	63
Hybrid(SOR)	1.11	18
I+S(5)	0.78	137

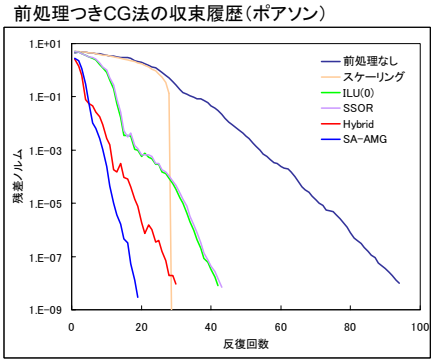
各解法の収束履歴(ポアソン)



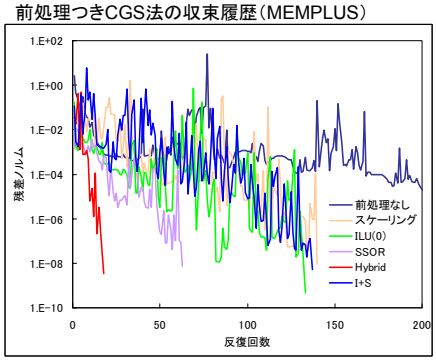
各解法の収束履歴(MEMPLUS)



前処理つきCG法の収束履歴(ポアソン)



前処理つきCGS法の収束履歴(MEMPLUS)



OpenMPによる並列化

- (分散)共有メモリ型を対象
- 右のルーチンのループを並列化
- Altixでは行列の初期化の並列化が重要

(最初にアクセスしたプロセスのローカルメモリに配置されるため)

Threads	初期化の並列化あり		初期化の並列化なし	
	計算時間(sec.)	速度向上	計算時間(sec.)	速度向上
1	52.14	1.00	52.00	1.00
2	30.23	1.73	30.00	1.73
4	14.91	3.50	22.54	2.31
8	7.12	7.32	35.55	1.46
16	3.57	14.60	52.41	0.99
32	2.05	25.43	61.94	0.84

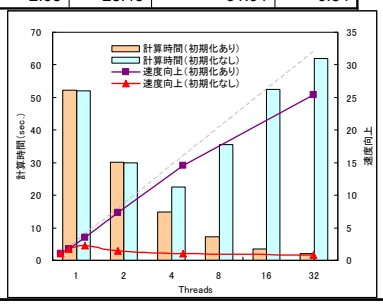
ポアソン方程式

$N=10^6$

解法:CG法

前処理:なし

格納形式:CRS



ベクトルの加減(DAXPY)

```
#pragma omp parallel for private(i)
for(i=0; i<n; i++)
{
        vy[i] += alpha * vx[i];
}
```

内積(DDOT)

```
dot = 0;
#pragma omp parallel for reduction(+:dot)
for(i=0; i<n; i++)
{
        dot += vx[i]*vy[i];
}
```

行列ベクトル積(MATVEC)

```
#pragma omp parallel for private(i,j,js,je,t,jj)
for(i=0; i<n; i++)
{
        js = A->row[i];
        je = A->row[i+1];
        t = beta * y[i];
        for(j=js;j<je;j++)
        {
                jj  = A->index[j];
                t += alpha * A->value[j] * x[jj];
        }
        y[i] = t;
}
```