2.4　近似逆行列型前処理つき CG 法の並列性能評価

藤野 清次 (九州大学情報基盤センター)

1　はじめに

一般に, 共役勾配 (Conjugate Gradient：以下, CG と略す) 法の前処理行列に望まれる性質として, 元の行列の疎 (スパース) 性の保持とその近似度の高さが上げられる. さらに, 反復計算の高い並列性を実現するには, 前処理行列のタイプも重要になってくる. 代表的な前処理の 1 つが不完全コレスキー (Incomplete Cholesky：以下, IC と略す) 分解である. IC 分解は, 連立 1 次方程式の係数行列を三角行列どうしの積の形に不完全分解する方法である. しかしながら, この前処理はその計算順序が本質的に逐次的であり, そのため CG 法の算法中の前処理部分の並列化が困難になる.

最もシンプルな並列化手法は, ブロック IC 分解と呼ばれる方法である. この方法は 80 年代にすでに提案され, 通称並列ブロックジャコビ法とも呼ばれる. また, ブロック IC 分解は Argonne 研究所で開発された PETSc と呼ばれるライブラリーに実装され, 広く使われている [3][26]. また, Hypre という並列ライブラリも知られている [15]. しかし, ブロック IC 分解はノードに跨るフィルインをまったく考慮しないので, 収束性があまりよくない. そこで, 要素番号を並べ直すことによる改良版ブロック IC 分解が開発された [17]. また, Azte と呼ばれるライブラリーも Sandia 研究所で開発され公開されている [2]. さらに, 最小 2 乗法を利用し近似逆行列を計算するタイプの並列化手法として, Grote らが実装した SPAI[13] や Chow が実装した ParaSails[11] という並列ライブラリーもよく知られている.

一方, 選択的ブロック化によりフィルインを選択する並列化手法も研究されている [16][23][24]. これは, フィルインの発生するブロックや位置に応じて，幾つかの分解処理手順の中から適切なものを選択 (select) するという考え方に基づく方法である. さらに, シュール補元を利用した並列化手法も研究されている [25]. また, Benzi らにより, 近似逆行列に基づく並列化法の要素番号のオーダリングに基づく解析なども行なわれている [5]. また, 色付け順序法による並列化手法として, 岩下らによる ABRB 法 [17] や ABRB 法においてフィルインを考慮する井上らによる並列化手法 [16] が研究されている. この他にも並列版の前処理の有望な方法に MultiGrid 法や AMG (Algebraic MultiGrid) 法などがある [31]. 並列化の最近の動向は文献 [12][14][26] などを参照されたい.

本論文では, IC 分解あるいはロバスト IC 分解に基づき不完全分解された三角行列の逆行列を近似的に計算し前処理行列を構成する方法を提案する [32]. これと同様の考え方は, 須田により回路解析においてすでに提案され, 並列計算機 AP1000 上で実装されている [27]. しかし, 本研究の特長は, 反復法の収束安定性をよりロバストにするために, フィルインの棄却のとき対角要素を補償することにより, 並列性能のより一層の安定化を図ったところにある. また, IC 分解つき CG 法の反復計算中に現れる前進 (後退) 代入計算を行列とベクトルの積の計算に置き換えたところも大きな特長の 1 つである.

2　各種前処理の特徴

実数対称正定値行列 $A(=(a_{ij}))\in R^{n\times n}$ を持つ線形方程式

$$A\boldsymbol{x} \quad = \quad \boldsymbol{b} \tag{2.1}$$

を反復法の 1 つである共役勾配 (Conjugate Gradient：以下, CG と略す) 法で解くことを考える. $\boldsymbol{x}$, $\boldsymbol{b}$ は n 次元の解ベクトルおよび右辺ベクトルとする. 特に, 行列 A が大型で疎行列の場合, 線形方程式 (2.1) は前処理つき CG 法で解かれることが多い.

ここで, 前処理 (preconditioning) とは, $K\approx A^{-1}$ となるような適当な行列 K を定め,

$$KA\boldsymbol{x}=K\boldsymbol{b} \tag{2.2}$$

のように変換することを指す. 具体的な計算においては，CG 法の反復中で，前処理行列 K とベクトルの積を計算することになる. 前処理の結果，反復法の

収束性が高められ，収束までの反復回数が大幅に減少することが多い．代表的な前処理の 1 つである IC 分解では, 前処理行列 K を直接求めず, その代わり次のように分解した上三角行列 $U(=(u_{ij}))$ を求める．

$$A \approx U^T U \quad = \quad K^{-1}. \tag{2.3}$$

ここで, 上付き添え字 “T” は転置を表す．このように行列 A を上三角行列 U とその転置 U^T の積の形で近似する分解は不完全分解型の前処理と呼ばれる．また, CG 法の反復ループ中に現れる前処理行列とベクトルの積の計算は, 前進代入と後退代入という二段階で求められる．

2.1 IC(tol) 分解と RIC(tol) 分解

IC 分解は (完全) コレスキー分解を閾値などを使って不完全に分解するもので, 様々な改良版がある．例えば, IC 分解で分解過程で生じたフィルインを閾値 (=tol) で棄却 (dropping) する方法がある．以下では, この分解を IC(tol) 分解と呼ぶ．この方法は, 前処理行列の疎性の保持と前処理行列の生成時間の短縮を主な目的に開発された．IC(tol) 分解の算法を以下に示す．

- **IC(tol) 分解の算法**

for $i = 1, \ldots, n$

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2} \tag{2.4}$$

for $j = i+1, \ldots, n$

$$u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj})/u_{ii}$$

dropping fillins with tolerance (2.5)

end for

end for

式 (2.5) において, 以下の棄却判定を行う．

```
if (|u_ij| ≤ tol) then  u_ij = 0
end if
```

しかし, 式 (2.4) の右辺の根号の中が零または負の数になり分解が途中で破綻 (breakdown) することがある．そこで, 対称正定値行列に対して, 理論的に破綻が起きないことを保証する前処理として, **RIC(tol) 分解**が提案された [1] [19]．この分解では, $A \approx U^T U + R + R^T - D$ と分解する．ここで, D は対角行列, R は形式的な行列とする．RIC(tol) 分解では, フィルインの棄却を行うと同時に, 前処理行列の対角要素にフィルインの絶対値に対応する量を加えることで破綻を未然に防いでいる．具体的には, 式 (2.5) においてフィルイン u_{ij} を棄却するとき, u_{ij} に対応する 2 つの対角要素 u_{ii}, u_{jj} を次のように修正・補償する．

```
if (|u_ij| ≤ tol) then  u_ij = 0
    u_ii = (1 + |u_ij|/√(u_ii u_jj)) u_ii
    u_jj = (1 + |u_ij|/√(u_ii u_jj)) u_jj
end if
```

図 1 にフィルイン u_{ij} と対応する 2 つの対角要素 u_{ii}, u_{jj} の配置の様子を示す．補償の対象は対角要素 u_{ii} だけでなく, 行列の対称性から, フィルイン u_{ji} に対する対角要素 u_{jj} も補償の対象になる．

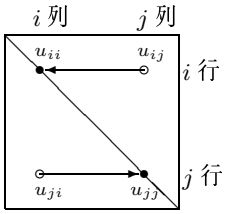

図 1: RIC(tol) 分解におけるフィルインと修正される対角要素の対応

2.2 SAInv 前処理

SAInv 前処理は, 行列 A の逆行列 A^{-1} を

$$A^{-1} \approx Z Z^T \tag{2.6}$$

と近似分解する方法である．この前処理は, Benzi らにより, **SAInv** (Stabilized Approximate Inverse) **前処理**と呼ばれた [6]．SAInv 前処理は, 行列が対称正定値行列のとき, 理論的に分解の破綻を起こさない, 前処理行列の計算が行列ベクトル積の計算で済む, など並列処理に向いた方法である．

3　前処理の並列化

ここでは, IC 分解の並列化についてを考える. IC 分解の前処理行列の作成部分および CG 法の反復ループの中の前進 (後退) 代入計算は本質的に逐次的なアルゴリズムである. そのため, IC 分解の並列化は非常に難しいことが知られている [18]. 次に, 古典的だが有用な IC 分解の並列手法であるブロック IC 分解と改良版について述べる.

3.1　ブロック IC(tol), ブロック RIC(tol) 分解

まず, 通常のブロック IC 分解による前進 (後退) 代入の並列化について記述する [4][28]. 図 2 に, 通常のブロック IC 分解において, 行列の非零要素が無視される領域および下三角行列 U^T の非零要素が含まれるブロック (図 2 中の Bl_0, Bl_1, Bl_2, Bl_3 を指す) さらに, 各ブロックとノードとの対応関係を示す. ノード数は 4 とする. このように, ブロック IC 分解では, 前進 (後退) 代入計算の並列実行を可能にするために, 非零要素は前処理行列を作成するとき, ノード間に跨る非零要素は無視される. したがって, ブロック IC 分解では, ノード数が増えれば増える程非零要素を無視する領域が拡大する. その結果, 前処理行列の近似精度が低下し, 反復回数が急激に増加することが多い.

一方, 本研究で扱う**ブロック IC(tol) 分解**とは, 非零要素が含まれるブロック (図 2 中の BL_0, BL_1, BL_2, BL_3) 内では, あらかじめ定めた閾値より大きいフィルインは前処理行列の要素として認めるという方法である. この処置により, 反復回数の急激な増加を抑えることを図る. さらに, 分解の破綻を防ぐために, 第 2.1 節で述べたフィルインを棄却するときの対角要素の補償処理を行なう. 以下, **ブロック RIC(tol) 分解**と呼ぶ.

3.2　RIC ベースの近似逆行列生成

ここでは, $A \approx U^T U$ と不完全分解する閾値つき IC(tol) 分解 あるいは $A \approx U^T U + R + R^T - D$ と不完全分解する閾値つきロバストな RIC(tol) 分解において, 上三角行列 U の逆行列を求めることを考える [9]. そのために, ガウス・ジョルダン (Gauss-Jordan：以下, GJ と略す) 法と同様の手順を採用する. ただし, GJ 法は線型方程式求解のための数値解法ではなく, あくまで逆行列を求めるための数値解法と位置づける [8] [10][30]. GJ 法自体の並列化については文献 [21][22] を参照されたい. GJ 法で逆行列を求めるとき, 行列 U と逆行列 U^{-1} が各々上三角行列であることから, 作業用の行列 S を用いて, 以下に示すように計算法が得られる.

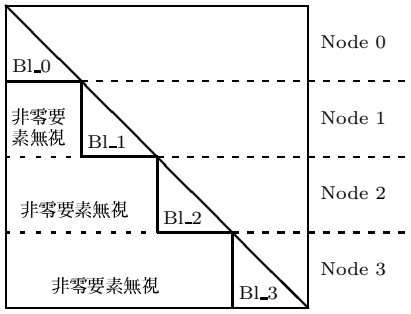

図 2: ブロック IC 分解において, 非零要素が無視される領域および下三角行列 U^T の各ブロックとノードとの対応 (ノード数は 4)

- **上三角行列に対する逆行列計算**

$$S = I \tag{3.7}$$

for $j = 1, \ldots, n$

$$s_{jj} = \frac{s_{jj}}{u_{jj}} \tag{3.8}$$

for $i = j+1, \ldots, n$

$$u_{ji} = \frac{u_{ji}}{u_{jj}} \tag{3.9}$$

end for

for $i = j+1, \ldots, n$

for $k = i+1, \ldots, n$

$$u_{jk} = u_{jk} - \frac{u_{ji}}{u_{ii}} u_{ik} \tag{3.10}$$

end for

$$s_{ji} = -\frac{u_{ji}}{u_{ii}} s_{ii} \tag{3.11}$$

end for

end for

ここで, 式 (3.7) は行列表現とする. 以上から, IC(tol) 分解および RIC(tol) 分解をベースにした近似逆行列型分解が以下のように得られる. この分解法を **IC(tol)AInv** (AInv は Approximate Inverse の略) 分解および **RIC(tol)AInv** 分解と各々呼ぶ.

- **IC(tol)AInv 分解の算法**

for $i = 1, \ldots, n$

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2}$$

for $j = i+1, \ldots, n$

$$u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} u_{ki}u_{kj})/u_{ii} \quad (3.12)$$

dropping fillins with tolerance (3.13)

end for

end for

$***$ IC(tol) decomposition is over. $***$

$$W = U \quad (3.14)$$

$$Z = I \quad (3.15)$$

for $j = 1, \ldots, n$

$$z_{jj} = \frac{z_{jj}}{w_{jj}} \quad (3.16)$$

for $i = j+1, \ldots, n$

$$w_{ji} = \frac{w_{ji}}{w_{jj}} \quad (3.17)$$

end for

for $i = j+1, \ldots, n$

for $k = i+1, \ldots, n$

$$w_{jk} = w_{jk} - \frac{w_{ji}}{w_{ii}} w_{ik} \quad (3.18)$$

end for

$$z_{ji} = -\frac{w_{ji}}{w_{ii}} z_{ii} \quad (3.19)$$

end for

end for

IC(tol) 分解が終了した直後の式 (3.14) では, 前処理行列 U が一時的作業用配列 W に収められ, 次に GJ 法に従って上三角行列 Z の要素が求まる. また, 式 (3.19) において近似逆行列を構成する行列 Z の要素が確定する. さらに, 式 (3.13) は IC(tol) 分解において閾値 (=tol) によるフィルインの棄却の判定処理であり, IC(tol) 分解をベースにした IC(tol)AInv 分解および RIC(tol) 分解をベースにした RIC(tol)AInv 分解における棄却処理は次のように行なわれる.

if ($|u_{ij}| \leq$ tol) then $u_{ij} = 0$
  if IC(tol)AInv is adopted
  then goto (3.14)
  else if RIC(tol)AInv is adopted then
    $u_{ii} = (1 + |u_{ij}|/\sqrt{u_{ii}u_{jj}})u_{ii}$
    $u_{jj} = (1 + |u_{ij}|/\sqrt{u_{ii}u_{jj}})u_{jj}$
  end if
end if

4 数値実験

4.1 計算環境と計算条件

表 1 に計算機環境を示す. 計算はすべて倍精度演算で行なった. CG 法の収束判定条件は反復第 k 回目の残差ベクトル $\boldsymbol{r}_k$ の 2 ノルムが, $||\boldsymbol{r}_k||_2/||\boldsymbol{r}_0||_2 < 10^{-12}$ を満たしたとき収束とした. 初期近似解 $\boldsymbol{x}_0$ はすべて 0 とした. 係数行列は対角スケーリングを行い対角項を 1.0 に揃える正規化処理をした.

表 1: 計算機環境

項目	仕様
計算機	IBM eServer p5 モデル 595
設置場所	九州大学情報基盤センター
CPU	Power5 (1.9GHz)
CPU 数	16
許容メモリ量	48GB
OS	AIX 5L
コンパイラ	XL Fortran 9.1
並列化ライブラリ	OpenMP

表 2: テスト行列の特徴

行列	次元数	非零要素	非零要素/次元数	分野
TUBE1-2	21,498	459,277	21.4	タイヤチューブ [20]
NASASRB	54,870	1,366,097	24.9	シャトル ロケット [29]
TCASE	134,856	4,830,936	35.8	応力解析
ENGINE	143,571	2,424,822	35.8	エンジン 応力解析 [29]
SBEAM	352,496	13,528,771	38.4	応力解析
MODEL3	374,229	11,165,692	29.8	応力解析
GRID_w	634,335	22,385,096	35.3	設計計算
GRID_s	838,395	13,566,835	16.2	設計計算

表 2 にテストに用いた 8 つの行列を示す．そのうち TUBE1-2 は R. Kouhia の疎行列データベースから, NASASRB と ENGINE は Florida 大学の疎行列データベースから選んだ [20][29]．それ以外の 5 つの行列は工学分野の解析で実際に現れた行列である．

4.2 逐次版の安定性能の評価

表 3 と表 4 に 8 つの係数行列に対する逐次版の前処理つき CG 法の CPU 時間を掲載した．調べた前処理は IC(tol) 分解, RIC(tol) 分解, IC(tol)AInv 分解, RIC(tol)AInv 分解の 4 つである．ただし, 表中では, 各々IC 分解, RIC 分解, ICAInv 分解, RICAInv 分解と略記した．各前処理の閾値 (=tol) は 0.01, 0.05, 0.1 と 3 通り変化させ, そのうち最も合計時間が少なかったものを表に載せた．CPU 時間の測定は Fortran90 の組み込み関数 etime を用いて行なった． 表中で,「precond.」は前処理,「tol」は閾値,「fill-in ($\times 10^6$)」は前処理行列で使われたフィルインの個数を表す．単位は 10^6 である．「Itr.」は反復回数,「pre-t」は前処理行列作成に要した時間,「CG-t」は CG 法の反復計算の時間,「tot-t」はそれらの合計時間を各々表す．また, 反復回数の欄で「max」は最大反復回数までで CG 法が収束しなかったことを表す．表 3 と表 4 から以下のことがわかる．

- IC(tol) 分解と IC(tol)AInv 分解は 8 つの行列のうち, 僅か 2 つの行列 (行列 ENGINE, SBEAM) しか収束せず, 収束が非常に不安定である．
- 一方, RIC(tol) 分解と RIC(tol)AInv 分解は 8 つすべての行列で収束し, 収束が安定である．
- メモリ量については, IC(tol) 分解が収束した行列 ENGINE, SBEAM の場合でも, RIC(tol) 分解つきの方が IC(tol) 分解の場合よりも少ない．

以上の逐次処理の結果から，収束が安定である RIC(tol) 分解および RIC(tol)AInv 分解の場合の並列性能について次の 4.3 節で報告する．また, 比較のために SAInv 分解の性能についても報告する．

4.3 並列性能評価

並列ライブラリ OpenMP を使用し, スレッド数を 1, 2, 4, 8, 16 と変化させて並列性能を調べた．スレッド数の上限を 16 としたのは計算機が設置された情

表 3: 係数行列 (TUBE1-2, NASASRB, TCASE and ENGINE) に対する前処理つき CG 法の時間

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	-	-	max	-	-	-
RIC	0.01	0.38	1973	0.19	10.80	10.99
ICAInv	-	-	max	-	-	-
RICAInv	0.01	0.71	2147	1.21	15.59	16.80

(a)matrix TUBE1-2

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	-	-	max	-	-	-
RIC	0.05	0.47	5987	0.2	71.4	71.6
ICAInv	-	-	max	-	-	-
RICAInv	0.05	0.34	7349	0.7	80.6	81.3

(b)matrix NASASRB

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	-	-	max	-	-	-
RIC	0.01	4.49	1743	4.5	200.4	204.9
ICAInv	-	-	max	-	-	-
RICAInv	0.05	1.02	5272	3.7	254.1	257.8

(c)matrix TCASE

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	0.01	3.46	453	2.6	29.8	32.4
RIC	0.01	2.96	874	1.8	50.7	52.5
ICAInv	0.05	1.57	1884	2.8	71.1	73.9
RICAInv	0.05	0.78	2127	1.4	62.2	63.5

(d)matrix ENGINE

表 4: 係数行列 (SBEAM, MODEL3, GRID_w and GRID_s) に対する前処理つき CG 法の時間

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	0.01	10.61	556	11.0	164.4	175.4
RIC	0.01	9.02	1008	8.3	239.3	247.6
ICAInv	0.01	14.30	814	48.0	221.2	269.2
RICAInv	0.01	8.58	1282	32.1	267.1	299.3

(e)matrix SBEAM

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	-	-	max	-	-	-
RIC	0.01	6.57	2303	5.0	403.0	408.0
ICAInv	-	-	max	-	-	-
RICAInv	0.01	7.15	2486	20.2	418.8	439.0

(f)matrix MODEL3

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	-	-	max	-	-	-
RIC	0.05	5.46	4052	3.5	994.4	997.9
ICAInv	-	-	max	-	-	-
RICAInv	0.05	2.84	4435	8.0	926.3	934.3

(g)matrix GRID_w

precond.	tol	fill-in ($\times 10^6$)	Itr.	pre-t	CG-t	tot-t
IC	-	-	max	-	-	-
RIC	0.01	16.44	15749	9.6	5917.1	5926.7
ICAInv	-	-	max	-	-	-
RICAInv	0.1	2.43	40348	4.1	6958.0	6962.1

(h)matrix GRID_s

報基盤センターの運用上の制約である. 経過時間の測定は合計 3 度行いその平均値を表に掲載した. また経過時間の測定は Fortran90 の組み込み関数 SYSTEM_CLOCK を用いて行なった. 本節の並列性能評価では, 収束が安定な前処理であるブロック RIC(tol) 前処理 (以下, 表中では BRIC と略す) つき CG 法 , SAInv 前処理 (以下, 表中では SAInv と略す) つき CG 法, RIC(tol)AInv 前処理 (以下, 表中では RICAInv と略す) つき CG 法の並列性能を評価した. 調べた閾値は 0.1 と 0.05 の場合の 2 ケースで, あまり性能面で差がなかったので, 閾値が 0.1 の場合の結果を以下の表では掲載した. また, 行列は対称であるため一般には下三角部分のみ保持すればよいが, 全非零要素を保持する方が並列計算において経過時間が短かったので, 数値実験では行列の全非零要素を行列ベクトルの積の計算で使用した.

表 5 から表 12 に各行列に対する実験結果を示す. 表中で, 「precond.」は前処理, 「th」はスレッド数, 「fill-in ($\times 10^6$)」は前処理行列で使われたフィルインの個数を表す. 単位は 10^6 である. 「Itr.」は反復回数, 「pre-t」は前処理行列の作成に要した時間, 「CG-t」は CG 法の反復計算の時間, 「tot-t」はそれらの合計時間, 「speedup」はスレッド数が 1 のときの経過時間を 1.0 としたときの各スレッドでの経過時間の比率, すなわち「台数効果」を各々表す. 時間の単位はすべて秒とする. 台数効果は, スレッド数が 1 のケースの合計時間を 1.0 としたときに, 各スレッド数で並列化によって得られた速度向上の倍率を指す. また各表の合計時間の欄で太字の数字はスレッド数が 16 のとき経過時間が最も短かったものを指す. 表 10 の行列 MODEL3 の前処理 SAInv で, 反復回数の欄で「max」は最大反復回数までで CG 法が収束しなかったことを表す.

表 5: 行列 TUBE1-2 に対する並列版 PCG 法の性能

pre-cond.	th	fill-in ($\times 10^6$)	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	0.071	6137	0.04	28.45	28.50	1.00
	2	0.070	6193	0.05	14.67	14.72	1.94
	4	0.069	6275	0.04	7.44	7.48	3.81
	8	0.068	6347	0.04	3.90	3.94	7.23
	16	0.067	6514	0.05	2.80	2.85	10.00
SAInv	1	0.194	3363	0.80	20.20	21.01	1.00
	2	0.194	3555	0.81	10.90	11.71	1.79
	4	0.194	3416	0.81	5.32	6.13	3.43
	8	0.194	3361	0.82	2.80	3.62	5.81
	16	0.194	3426	0.83	2.00	**2.83**	7.42
RIC-AInv	1	0.056	7603	0.08	37.49	37.57	1.00
	2	0.056	7601	0.08	19.28	19.36	1.94
	4	0.056	7601	0.08	9.76	9.84	3.82
	8	0.056	7599	0.08	5.32	5.40	6.96
	16	0.056	7600	0.08	3.41	3.49	10.78

表 6: 行列 NASASRB に対する並列版 PCG 法の性能

pre-cond.	th	fill-in ($\times 10^6$)	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	0.201	10885	0.1	186.2	186.3	1.00
	2	0.201	10883	0.1	76.4	76.5	2.44
	4	0.199	10940	0.1	38.8	39.0	4.78
	8	0.196	10938	0.1	20.4	20.6	9.07
	16	0.189	11066	0.1	14.1	14.3	13.08
SAInv	1	0.576	11401	3.0	276.7	279.7	1.00
	2	0.576	11410	3.1	138.4	141.5	1.98
	4	0.576	11412	3.1	52.6	55.6	5.03
	8	0.576	11398	3.0	25.8	28.8	9.70
	16	0.576	11404	3.0	13.7	16.7	16.75
RIC-AInv	1	0.111	14522	0.2	253.8	254.0	1.00
	2	0.111	14519	0.2	115.9	116.1	2.19
	4	0.111	14525	0.2	51.3	51.6	4.92
	8	0.111	14521	0.2	26.0	26.2	9.68
	16	0.111	14525	0.2	13.9	**14.1**	17.97

表 7: 行列 TCASE に対する並列版 PCG 法の性能

pre-cond.	th	fill-in ($\times 10^6$)	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	0.845	5445	0.6	413.3	413.9	1.00
	2	0.837	5482	0.6	214.0	214.6	1.93
	4	0.815	5446	0.6	88.4	89.0	4.65
	8	0.790	6945	0.6	48.5	49.1	8.43
	16	0.731	7437	0.6	26.2	26.8	15.46
SAInv	1	2.125	6446	18.3	615.5	633.8	1.00
	2	2.125	6449	18.4	323.1	341.4	1.86
	4	2.125	6437	18.4	149.7	168.1	3.77
	8	2.125	6443	18.4	56.8	75.2	8.42
	16	2.125	6438	18.4	26.6	44.9	14.10
RIC-AInv	1	0.328	7697	1.1	549.3	550.4	1.00
	2	0.328	7686	1.1	272.5	273.6	2.01
	4	0.328	7673	1.1	104.6	105.6	5.21
	8	0.328	7676	1.1	50.4	51.5	10.69
	16	0.328	7677	1.1	23.6	**24.7**	22.30

表 8: 行列 ENGINE に対する並列版 PCG 法の性能

pre-cond.	th	fill-in ($\times 10^6$)	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	0.452	2612	0.3	112.2	112.5	1.00
	2	0.408	2915	0.3	50.8	51.0	2.20
	4	0.376	2997	0.3	24.1	24.3	4.63
	8	0.362	3042	0.2	12.8	13.1	8.61
	16	0.346	3074	0.2	7.9	**8.2**	13.78
SAInv	1	0.736	1390	6.6	69.1	75.7	1.00
	2	0.736	1390	6.6	31.9	38.5	1.97
	4	0.736	1390	6.7	15.7	22.4	3.38
	8	0.736	1390	6.8	8.4	15.2	5.00
	16	0.736	1390	6.7	5.0	11.7	6.49
RIC-AInv	1	0.301	2904	0.5	130.0	130.4	1.00
	2	0.301	2903	0.5	57.0	57.5	2.27
	4	0.301	2903	0.5	25.3	25.8	5.06
	8	0.301	2902	0.5	15.0	15.4	8.45
	16	0.301	2902	0.5	9.1	9.6	13.59

表 9: 行列 SBEAM に対する並列版 PCG 法の性能

pre-cond.	th	fill-in $\times 10^6$	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	0.604	2859	1.0	531.7	532.8	1.00
	2	0.604	2859	1.1	276.4	277.4	1.92
	4	0.601	3067	1.0	156.8	157.9	3.37
	8	0.594	3261	1.0	85.8	86.8	6.13
	16	0.581	3430	1.0	43.4	44.5	11.98
SAInv	1	.1304	1748	17.4	362.2	379.6	1.00
	2	1.304	1748	17.5	185.6	203.0	1.87
	4	1.304	1748	17.5	96.2	113.6	3.34
	8	1.304	1746	17.5	50.2	67.6	5.61
	16	1.304	1747	17.6	25.0	**42.7**	8.90
RIC-AInv	1	0.410	4126	1.1	805.1	806.3	1.00
	2	0.410	4121	1.1	412.0	413.2	1.95
	4	0.410	4116	1.1	213.0	214.2	3.76
	8	0.410	4106	1.1	113.3	114.4	7.05
	16	0.410	4107	1.1	52.1	53.2	15.14

表 10: 行列 MODEL3 に対する並列版 PCG 法の性能

pre-cond.	th	fill-in ($\times 10^6$)	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	0.888	7413	1.0	1223.0	1224.0	1.00
	2	0.882	7422	1.0	641.9	642.9	1.90
	4	0.875	7443	1.0	317.1	318.1	3.85
	8	0.860	7494	1.0	149.6	150.6	8.13
	16	0.823	7615	1.0	66.8	**67.8**	18.06
SAInv	1	-	max	-	-	-	-
	2	-	max	-	-	-	-
	4	-	max	-	-	-	-
	8	-	max	-	-	-	-
	16	-	max	-	-	-	-
RIC-AInv	1	0.568	7875	1.2	1337.9	1339.1	1.00
	2	0.568	7873	1.2	685.2	686.5	1.95
	4	0.568	7872	1.2	354.3	355.6	3.77
	8	0.568	7871	1.2	161.8	163.1	8.21
	16	0.568	7870	1.2	75.8	77.1	17.37

表 11: 行列 GRID_w に対する並列版 PCG 法の性能

pre-cond.	th	fill-in ($\times 10^6$)	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	1.625	5457	1.9	1797.6	1799.5	1.00
	2	1.620	5522	1.9	918.1	920.0	1.96
	4	1.615	5553	1.9	468.9	470.9	3.82
	8	1.574	5720	1.9	254.0	255.9	7.03
	16	1.535	5762	1.9	161.2	163.1	11.03
SAInv	1	3.979	2504	52.8	912.7	965.4	1.00
	2	3.979	2501	52.7	462.7	515.5	1.87
	4	3.979	2501	53.7	251.4	305.2	3.16
	8	3.979	2501	53.0	127.4	180.3	5.35
	16	3.979	2501	53.0	69.4	**122.3**	7.89
RIC-AInv	1	0.815	6458	2.3	2116.7	2119.0	1.00
	2	0.815	6455	2.3	1041.1	1043.4	2.03
	4	0.815	6454	2.3	537.0	539.3	3.93
	8	0.815	6454	2.3	283.5	285.7	7.42
	16	0.815	6454	2.3	152.8	155.1	13.67

表 12: 行列 GRID_s に対する並列版 PCG 法の性能

pre-cond.	th	fill-in ($\times 10^6$)	Itr.	pre-t [s]	CG-t [s]	tot-t [s]	speed-up
BRIC	1	3.796	35099	1.9	9411.3	9413.2	1.00
	2	3.788	36218	1.9	4651.3	4653.2	2.02
	4	3.768	38638	1.9	2736.9	2738.8	3.44
	8	3.732	40237	1.9	1345.3	1347.1	6.99
	16	3.705	43245	1.9	755.1	756.9	12.44
SAInv	1	9.712	14947	70.3	5155.8	5226.0	1.00
	2	9.712	14947	70.7	2651.8	2722.4	1.92
	4	9.712	14947	71.6	1392.0	1463.6	3.57
	8	9.712	14947	70.3	692.0	762.4	6.86
	16	9.712	14946	70.5	355.8	**426.3**	12.26
RIC-AInv	1	2.439	40348	4.0	10291.7	10295.7	1.00
	2	2.439	40345	4.0	5166.3	5170.3	1.99
	4	2.439	40344	4.0	2684.5	2688.6	3.83
	8	2.439	40342	4.0	1335.0	1339.1	7.69
	16	2.439	40341	4.0	692.2	696.3	14.79

これらの表から以下の観察ができる.

- 16 スレッドのとき, 合計時間が最も短かったのは RIC(tol)AInv 前処理のときが 2 ケース (行列 NASASRB, TCASE), BRIC(tol) 分解のときが 2 ケース (行列 ENGINE, MODEL3) そして SAInv 前処理のときが 4 ケースであった.
- スレッド数が増加したときの収束までの反復回数について, BRIC(tol) 分解では段々と増加するのに対して, SAInv 前処理と RIC(tol)AInv 前処理ではほぼ一定回数で変動が少ない. このように似た傾向を示すのは, SAInv 前処理と RIC(tol)AInv 前処理が, プロセッサ台数によって前処理行列が変わらないことから，至極当然の結果であると考えられる.
- RIC(tol)AInv 前処理で必要な前処理行列のフィルインの個数は相対的に他の 2 つの前処理のときよりも少なくて済むことが多い.
- SAInv 前処理では, 表 10 のときのように収束しないケースがあり, BRIC(tol) 分解と RIC(tol)AInv 分解に比べてその収束が不安定である.

解析結果をより詳細に検討するために, 表 13 に並列版前処理つき CG 法の 16 スレッドを使用した場合の台数効果の比をまとめた. 表中の太字の数字は各行列で最も台数効果が高かった前処理を示す. 表 13 の結果から, RIC(tol)AInv 前処理が他の 2 つの前処理に比べて台数効果が高いことがわかる. SAInv 前処理と RIC(tol)AInv 前処理は，このように CG 法の反復時間に関していずれも高い台数効果を示すが，前処理行列の作成の負荷は，SAInv 前処理の負荷の方がずっと重い. したがって，前処理行列の作成部分を並列化しない場合には，全体での台数効果は RIC(tol)AInv 前処理の方が高くなる. また, 16 スレッドを越えるような多数スレッドの場合には RIC(tol)AInv 前処理が有用になると思われるが, 前述のように今回の数値実験では運用上の制約から実験出来なかった.

同様に, 16 スレッドを使用した場合において, 表 14 に SAInv 前処理と RIC(tol)AInv 前処理において, 前処理つき CG 法における前処理行列作成の時間と CG 法の反復計算の時間との関係をまとめた. 表中の比 (ratio) は CG 法の反復時間を 1 としたときの前処理時間の比率である. SAInv 前処理のときの前処理行列作成の時間に比べて RIC(tol)AInv 前処理のときの同時間が大幅 (約 10%以下) に短くなったことがわかる. その結果, SAInv 前処理のときのように, 多数のスレッドを使用したとき前処理行列を作成する時間が顕在化するという現象は現れなくなった.

表 13: 並列版前処理つき CG 法の台数効果の評価 (16 スレッドのとき)

Matrix	preconditioning		
	BRIC	SAInv	RICAInv
TUBE1-2	10.00	7.42	**10.78**
NASASRB	13.08	16.75	**17.97**
TCASE	15.46	14.10	**22.30**
ENGINE	13.78	6.49	**13.59**
SBEAM	11.98	8.90	**15.14**
MODEL3	**18.06**	-	17.37
GRID_w	11.03	7.89	**13.67**
GRID_s	12.44	12.26	**14.79**

表 14: 前処理行列作成の時間と CG 法の反復計算の時間との関係 (16 スレッドのとき)

Matrix	SAInv			RICAInv		
	pre-t [s]	CG-t [s]	ratio	pre-t [s]	CG-t [s]	ratio
TUBE1-2	0.8	2.0	0.415	0.08	3.41	0.023
NASASRB	3.0	13.7	0.219	0.2	13.9	0.014
TCASE	18.4	26.6	0.692	1.1	23.6	0.047
ENGINE	6.7	5.0	1.340	0.5	9.1	0.055
SBEAM	17.6	25.0	0.704	1.1	52.1	0.021
MODEL3	-	-	-	1.2	75.8	0.016
GRID_w	53.0	69.4	0.764	2.3	152.8	0.015
GRID_s	70.5	355.8	0.198	4.0	692.2	0.006

5 HPC2500 並列計算機上での性能評価

評価した解法は近似逆行列 (SAINV) 前処理つき CG 法である. テスト行列は構造解析の実数対称正定値行列 (行列 Grid_s と行列 Grid_w) である. 使用した計算機は, 富士通社 HPC2500 である. 最適化オプションは-KOMP -Kfast,largepage,parallel -Cpp -KV9 とした. 表 15 と表 16 に測定結果を示す. ただし, 京都大学での測定は, 通常運転のときの計測であり, 単独実行による計測でないことを付記する. 表中の時間の単位はすべて秒とする.

表 15: 近似逆行列型前処理 (SAINV), 行列 Grid_s

Thread	HPC2500			p5/595
	1.3GHz	1.56GHz	2.08GHz	1.9GHz
	富士通沼津	京都大学	京都大学	九州大学
1	-	-	-	5226
2	-	4480.9	4598.6	2722
4	-	3293.8	2445.9	1463
8	-	1736.4	1748.5	762
16	-	977.0	986.6	426
32	401.7	700.5	520.3	-
64	318.4	450.1	-	-
126	289.2	254.4	-	-

表 16: 近似逆行列型前処理 (SAINV), 行列 Grid_w

Thread	HPC2500			p5/595
	1.3GHz	1.56GHz	2.08GHz	1.9GHz
	富士通沼津	京都大学	京都大学	九州大学
1	-	-	-	965.4
2	-	767.6	755.2	515.5
4	-	555.2	420.1	305.2
8	-	328.9	391.1	180.3
16	-	210.3	210.8	122.3
32	133.0	178.1	129.6	-
64	121.3	126.1	-	-
126	115.8	92.2	-	-

6 おわりに

RIC(tol) 分解ベースに近似的に逆行列を計算するという並列計算向きの CG 法の前処理を新しく提案した. 提案した RIC(tol)AInv 前処理は, (i) 前処理行列の作成に要する時間が従来の SAInv 前処理のときよりも少なくて済み, (ii) 並列計算のときの CG 法の収束性が安定化し, (iii) 並列台数に係わらず, ほぼ一定の反復回数で収束し, 台数効果が得やすい, などのいくつかの優れた特徴を有することがわかった.

今後の課題は, (i)16 スレッドを越えるような多数スレッドでの性能評価, (ii) 最適な閾値の場合の各前処理の性能の比較をすること, そして (iii) 前処理行列の作成部分の並列化などである.

参考文献

[1] Ajiz, M.A., Jennings, A.: A Robust Incomplete Choleski-Conjugate Gradient Algorithm, *Int. J. Numer. Methods Engrg.*, Vol. 20, pp. 949–966 (1984).

[2] Aztec: A massively Parallel Iterative solver library for solving sparse linear systems の HP, http://www.cs.sandia.gov/CRF/aztec1.html

[3] Balay, S., et al.: PETSc Users manual, ANL-95/11 Rev.2.1.5, Argonne National Laboratory, (2004).

[4] Barton,M.L.: Three-dimensional magnetic field computation on a distributed memory parallel processor, *IEEE Trans. Magns.*, Vol. 26-2, pp. 834–836 (1990).

[5] Benzi, M., Marin, J., Tuma, M.: A two-level parallel preconditioner based on sparse approximate inverse, In D. Kincaid ed., Iterative methods in Scientific computation IV, *IMACS series in Comput. and Appl. Math.*, Vol. 5, IMACS, pp. 167–178 (1999).

[6] Benzi, M., Cullum, J.K. and Tuma, M.: Robust Approximate Inverse Preconditioning for the Conjugate Gradient Method, *SIAM J. Sci. Comput.*, Vol. 22, pp. 1318–1332 (2000).

[7] Benzi, M., Tuma, M.: A Robust Incomplete Factorization Preconditioner for Positive Definite Matrices, *Numerical Linear Algebra with Applications*, Vol. 10, pp. 385–400 (2003).

[8] Bogacki, P.: HINGES - An illustration of Gauss-Jordan reduction, MathDL: *J. of Online Math. and its Appli.*, (2006).

[9] Bollhöfer, M., Saad, Y.: On the relations between ILUs and factored approximate inverses, *SIMAX*, Vol. 24, pp. 219–237 (2002).

[10] Chen, K., Evans, D.: An efficient variant of Gauss-Jordan type algorithms for direct and parallel solution of dense linear systems, *Int. J. of Computer Math.*, Vol. 76, pp. 387–410 (2000).

[11] Chow, E.: Parallel implementation and practical use of sparse approximate inverses with a priori sparsity patterns, *Int. J. High Perf. Comput. Appl.*, Vol. 15, pp. 56–74 (2001).

Vol. 10, pp. 385–400 (2003).

[12] Duff, I.S., van der Vorst, H.A.: Developments and trend in the parallel solution of linear systems, *Parallel Computing*, Vol. 25, pp. 1931–1970 (1999).

[13] Grote, M., Huckle, T.: Parallel preconditioning with sparse approximate inverses, *SIAM J. Sci. Comput.*, Vol. 18, pp. 838–853 (1997).

[14] Henon, P., et al.: Blocking issues for an efficient parallel block ILU preconditioner, *Abstracts of International Workshop Preconditioning 2005*, Atlanta U.S., May (2005).

[15] Hypre: Scalable Linear Solvers の HP, http://www.llnl.gov/CASC/linear_solvers/

[16] 井上明彦, 藤野清次：フィルインの選択に基づく改良版 ABRB 順序付け法による ICCG 法の並列化, 情報処理学会論文誌 コンピュータシステム, Vol. 46 No.SIG16 (ACS12), pp. 119–128 (2005).

[17] Iwashita, T., Shimasaki, M.: Parallel precessing of 3-D eddy current analysis with moving conductor using parallelized ICCG solver with renumbering process, *IEEE Trans. Magn.*, Vol. 36, pp. 1504–1509 (2000).

[18] 岩下 武史, 島崎 眞昭, 同期点の少ない並列化 ICCG 法のためのブロック化赤ー黒順序付け, 情報処理学会論文誌：Vol. 43, No.4, pp. 893–904 (2002).

[19] 柿原正伸, 藤野清次: 緩和係数 ω を自動決定する対角緩和準ロバスト ICCG 法の収束性, 情報処理学会論文誌 コンピュータシステム, Vol. 46 No.SIG4(ACS9), pp. 45–55, (2005).

[20] R. Kouhia, Sparse Matrices web page: http://www.hut.fi/~kouhia/sparse.html.

[21] Melab, N., Petiton, S. and Talbi, E.-G.: A new parallel adaptive block-based Gauss-Jordan algorithm, Publication Interne AS-180, LIFL, Université de Lillel, Fév (1998).

[22] Melab, N.: A parallel adaptive Gauss-Jordan algorithm, *The Journal of Supercomputing*, Vol. 17, pp. 167–185 (2000).

[23] Nakajima, K., Okuda, H.: Parallel iterative solvers with selective blocking preconditioning for simulations of fault-zone contact, *Numer. Linear Algebra Appl.*, Vol. 11, pp. 831–852 (2004).

[24] Raghavan, P., Teranishi, K., Ng, E.: A latency tolerant hybrid sparse solver using incomplete Cholesky factorization, *Numer. Linear Algebra Appl.*, Vol. 10, pp. 541–560 (2003).

[25] Saad, Y.: Distributed Schur complement techniques for general sparse linear systems, SIAM J. Sci. Comput., Vol. 21, pp. 1337-1356 (1999).

[26] Saad, Y.: Iterative methods for sparse linear systems 2nd ed., SIAM Philadelphia (2003).

[27] Suda, R.: Large scale circuit analysis by preconditioned relaxation methods, *Proc. of PCG'94*, Keio Univ., pp. 189–197 (1994).

[28] Vollaire, C., Nicolas, L.: Preconditioning techniques for the Conjugate Gradient solver on a parallel distributed memory computer, *IEEE Trans. Magn.*, Vol. 34, pp. 3347–3350 (1998).

[29] University of Florida Sparse Matrix web page: http://www.cise.ufl.edu/research/sparse/matrices.

[30] 山本 哲朗, 数値解析入門 [増訂版], サイエンス社 (2002).

[31] Yavneh, I. (Editor): Special Issue: Multigrid methods, *Numer. Linear Algebra Appl.*, Vol. 11, No.2–3 (2004).

[32] 吉田正浩, 不完全分解の逆行列を用いた前処理つき CG 法の並列化, 九州大学大学院システム情報科学府情報工学専攻, 修士論文, 2006.3.