第 5 章

周期境界問題に対する並列計算機向きの前処理法

本章では，周期境界条件を課した偏微分方程式を離散化して得られた線形方程式求解に対して，非定常反復法を用いて求解する際に併用する，並列計算向きの新しい前処理法について説明する．x 方向に周期境界条件を課した 2 次元移流拡散方程式に対して，5 点中心差分したときに得られる線形方程式の係数行列は，対角ブロックに周期境界要素を持つ．ブロック前処理をほどこすとき，この対角ブロックの構造に着目して Sherman-Morrison 公式を適用したブロック前処理は，従来のブロック不完全分解と比べて計算コストを増大させずに対角ブロックに対する完全な分解を可能にする．数値実験をとおして，不完全分解に基づく従来のブロック前処理よりも収束性が良く，計算時間も短縮されていることが確認された．また，並列計算により，プロセッサ台数に対する効果も示した．

5.1 はじめに

周期境界条件を課した偏微分方程式に対して，離散化し近似的に解を求めるとき，大規模でスパースな線形方程式

$$A\boldsymbol{x} \;=\; \boldsymbol{b} \tag{5.1}$$

を解くことに帰着されることが多い．このとき，係数行列 A は周期境界要素（偏微分方程式の離散化で周期境界条件から生ずる行列の要素）を持つ．

このような問題に対する解法として，係数行列が対称の場合の前処理付き CG 法 [23] をはじめ，非対称な場合の前処理付き BiCGSTAB 法 [65] など，前処理を併用した反復法が用いられている．反復法の収束は係数行列の性質 (スペクトル特性など) に依存する [5][35]．前処理とは，このような観点で線形方程式を収束性が良くなるよう変換することであり，その変換を行うのが前処理行列である．前処理行列が何らかの方法により係数行列 A を近似していると，変換された線形方程式では，収束までの所要反復回数が減少することが期待できる [5].

一方で，大規模な問題に対しては，分散メモリ型の並列計算機を用いた数値計算が行われる．このときの前処理では，各プロセッサ毎に独立して演算することが重要である．これまでに，係数行列をブロック分割してブロック化前処理を施したときの収束性に関する効果が報告されている [2][9]．このアイデアに基づいて，ベクトル計算や並列計算向けの前処理としても応用されている [3][36]．

本論文では，係数行列に周期境界要素を持つ線形方程式に対して並列計算するときの，新しい前処理法について説明する．従来の代表的な方法では，ブロック化された各対角ブロックに対して IC(不完全コレスキー) 分解または ILU(不完全 LU) 分解を用いて不完全に分解するが [34][35]，本研究で提案する方法では，演算量を大幅に増大させずに対角ブロックに対して厳密な分解を行う．

本章の第 5.2 節では，対象とする問題について述べる．第 5.3 節では，文献 [9] による係数行列のブロック分割の方法について説明する．また，このブロック分割した際の従来の前処理法について説明する．第 5.4 節において，本研究で提案する新しい前処理のアイデアとその特長について述べ，具体的な計算手法について説明する．第 5.5 節の数値実験では，本前処理法が従来の方法よりも収束性が良く，計算時間も短縮することを示す．さらに，並列計算における台数効果も示す．

5.2　対象とする問題

対象とする問題は，2 次元正方領域 $\Omega = [0,1] \times [0,1]$ における移流拡散方程式

$$
\begin{aligned}
-\Delta u + v_1 \frac{\partial u}{\partial x} + v_2 \frac{\partial u}{\partial y} &= f, \qquad (x,y) \in [0,1] \times (0,1), && (5.2)\\
u(0,y) &= u(1,y), \qquad \text{(periodic boundary conditions)}, && (5.3)\\
u(x,0) &= g_0, \qquad \text{(Dirichlet conditions)},\\
u(x,1) &= g_1, \qquad \text{(Dirichlet conditions)}
\end{aligned}
$$

である．ここで，v_1, v_2 は各々x, y 方向の移流速度を表す．係数行列 A は，式 (5.2) を 5 点中心差分で離散化して作成し，x, y の各方向とも $(n+1)$ 等分すると，周期境界要素を持つ $n(n+1) \times n(n+1)$ の非対称でスパースな 5 重対角行列となる．

係数行列 A を，次のようにブロック化する．A は (n ブロック)×(n ブロック) 行列であり，各ブロックは $(n+1) \times (n+1)$ である．

$$
A = \begin{bmatrix}
P_1 & B_1 & & & 0 \\
A_2 & P_2 & B_2 & & \\
 & \ddots & \ddots & \ddots & \\
 & & A_{n-1} & P_{n-1} & B_{n-1} \\
0 & & & A_n & P_n
\end{bmatrix} \qquad (5.4)
$$

式 (5.2) の問題の場合，ブロック行列 P_l(l はブロックの番号である) は以下のとおりであり，周期境界要素を持つ 3 重対角行列である．ここで，$p_1^{(l)}, q_m^{(l)}$ が第 l ブロック内の周

期境界要素である ($m = n + 1$ である).

$$P_l = \begin{bmatrix} d_1^{(l)} & b_1^{(l)} & & 0 & p_1^{(l)} \\ a_2^{(l)} & d_2^{(l)} & b_2^{(l)} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{m-1}^{(l)} & d_{m-1}^{(l)} & b_{m-1}^{(l)} \\ q_m^{(l)} & 0 & & a_m^{(l)} & d_m^{(l)} \end{bmatrix}, \tag{5.5}$$

このようなシステムに対して，多くの場合，前処理付き反復法を用いて求解が行われる．係数行列が対称な場合，通常では前処理付き共役勾配法 (**アルゴリズム** 2.6) が用いられる．

係数行列が非対称な場合，効果的な解法の一つとして，前処理付き BiCGSTAB 法 (**アルゴリズム** 5.1) が用いられる．

アルゴリズム: 5.1 (前処理付き BiCGSTAB 法)

$\boldsymbol{x}_0$: is initial guess,
$\boldsymbol{r}_0^* = K^{-1}(\boldsymbol{b} - A\boldsymbol{x}_0)$,
$\boldsymbol{p}_0 = \boldsymbol{t}_0 = \boldsymbol{r}_0 = \boldsymbol{r}_0^*$,
for $k = 0, 1, \cdots$, until $\|\boldsymbol{r}_k\| \leq \varepsilon \| \boldsymbol{b} \|$ do:
begin

$$\begin{aligned} \alpha_k &= \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_k)}{(\boldsymbol{r}_0^*, K^{-1}A\boldsymbol{p}_k)}, \\ \boldsymbol{t}_k &= \boldsymbol{r}_k - \alpha_k K^{-1} A \boldsymbol{p}_k, \\ \zeta_k &= \frac{(K^{-1}A\boldsymbol{t}_k, \boldsymbol{t}_k)}{(K^{-1}A\boldsymbol{t}_k, K^{-1}A\boldsymbol{t}_k)}, \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k + \zeta_k \boldsymbol{t}_k, \\ \boldsymbol{r}_{k+1} &= \boldsymbol{t}_k - \zeta_k K^{-1} A \boldsymbol{t}_k, \\ \beta_k &= \frac{\alpha_k (\boldsymbol{r}_0^*, \boldsymbol{r}_{k+1})}{\zeta_k (\boldsymbol{r}_0^*, \boldsymbol{r}_k)}, \\ \boldsymbol{p}_{k+1} &= \boldsymbol{r}_{k+1} + \beta_k (\boldsymbol{p}_k - \zeta_k K^{-1} A \boldsymbol{p}_k), \end{aligned}$$

end

これらのアルゴリズムにおいて，行列 K が前処理行列である．一般に，前処理を施すと，各々の反復において $\boldsymbol{s'} = K^{-1}\boldsymbol{s}$ の方程式を解くことに帰着される．

5.3　ブロック前処理

本研究では，ブロック前処理について考える．ブロック前処理は，並列計算において非常に有効である．ブロック前処理 K は以下のとおりである．

$$K = \begin{bmatrix} K_1 & & & & 0 \\ & K_2 & & & \\ & & \ddots & & \\ & & & K_{n-1} & \\ 0 & & & & K_n \end{bmatrix}, \tag{5.6}$$

ここで，K_l は $(n+1)\times(n+1)$ サイズのブロック行列である．

従来の代表的な方法では，係数行列の各対角ブロックに対して IC(不完全コレスキー) 分解または ILU(不完全 LU) 分解を用いて不完全に分解することで，周期境界要素に起因する *fill–in* を生じないように前処理行列を生成する [5][62]．すなわち，

$$(K_l^{\mathrm{IF}})^{-1} \approx P_l^{-1} \tag{5.7}$$

である．このときの分解は，

$$K_l^{\mathrm{IF}} = \begin{bmatrix} \delta_1^{(l)} & & & & \\ a_2^{(l)} & \delta_2^{(l)} & & 0 & \\ & \ddots & \ddots & & \\ & 0 & a_{m-1}^{(l)} & \delta_{m-1}^{(l)} & \\ q_m^{(l)} & & & a_m^{(l)} & \delta_m^{(l)} \end{bmatrix} \begin{bmatrix} \delta_1^{(l)} & b_1^{(l)} & & & p_1^{(l)} \\ & \delta_2^{(l)} & b_2^{(l)} & 0 & \\ & & \ddots & \ddots & \\ & 0 & & \delta_{m-1}^{(l)} & b_{m-1}^{(l)} \\ & & & & \delta_m^{(l)} \end{bmatrix} \tag{5.8}$$

である．これにより，前処理演算にかかる計算コストを軽減することができる．

一方，収束性の観点からは，対角ブロックに対する完全な分解

$$(K_l^{\mathrm{CF}})^{-1} = P_l^{-1} \tag{5.9}$$

は，不完全分解 (5.7) よりも良い収束が期待できる．このときの分解は，

$$K_l^{\mathrm{CF}} = \begin{bmatrix} \delta_1^{(l)} & & & & \\ a_2^{(l)} & \delta_2^{(l)} & & 0 & \\ & \ddots & \ddots & & \\ 0 & & a_{m-1}^{(l)} & \delta_{m-1}^{(l)} & \\ q_m^{(l)} & \star & \star & \varrho_m^{(l)} & \delta_m^{(l)} \end{bmatrix} \begin{bmatrix} \delta_1^{(l)} & b_1^{(l)} & & 0 & p_1^{(l)} \\ & \delta_2^{(l)} & b_2^{(l)} & & \star \\ & & \ddots & \ddots & \star \\ & 0 & & \delta_{m-1}^{(l)} & \rho_{m-1}^{(l)} \\ & & & & \delta_m^{(l)} \end{bmatrix} \tag{5.10}$$

である．ここで，“$\star$” は，*fill–in* により生ずる非ゼロ要素を表す．しかし，この分解では多くの演算量を要する．

5.4　Splitting Correction 前処理

本研究では，前処理付き反復法を並列計算するために，対角ブロックに対して

$$(K_l^{\mathrm{SC}})^{-1} \;=\; P_l^{-1} \tag{5.11}$$

と表される，式 (5.9) と同値な前処理を生成し，かつ，不完全分解 (5.7) と比較して，演算量を大幅に増大させない方法を提案する．

5.4.1　周期境界要素の分離

まず，次式(5.12)のとおり，係数行列の対角ブロックP_lから周期境界要素を分離(splitting)する．

$$\begin{aligned}
K_l^{\mathrm{SC}} &= \begin{bmatrix} d_1^{(l)}-p_1^{(l)} & b_1^{(l)} & & & 0 \\ a_2^{(l)} & d_2^{(l)} & b_2^{(l)} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{m-1}^{(l)} & d_{m-1}^{(l)} & b_{m-1}^{(l)} \\ 0 & & & a_m^{(l)} & d_m^{(l)}-q_m^{(l)} \end{bmatrix} + \begin{bmatrix} p_1^{(l)} & 0\cdots 0 & p_1^{(l)} \\ 0 & 0\cdots 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0\cdots 0 & 0 \\ q_m^{(l)} & 0\cdots 0 & q_m^{(l)} \end{bmatrix} \\
&= T_l + \begin{bmatrix} p_1^{(l)} \\ 0 \\ \vdots \\ 0 \\ q_m^{(l)} \end{bmatrix} \begin{bmatrix} 1\,0\cdots 0\,1 \end{bmatrix} \\
&= T_l + \boldsymbol{u}_l \boldsymbol{v}_l^T . \qquad (5.12)
\end{aligned}$$

このとき，K_l^{SC} は 3 重対角行列 T_l と周期境界要素から構成される階数 (rank)1 の行列に分離でき，後者を二つのベクトルの積による修正 (correction) の項として表すことができる．したがって，前処理された残差ベクトル $\boldsymbol{r'}_l$ は，元の $\boldsymbol{r}_l$ に対して，

$$\boldsymbol{r'}_l \;=\; (K_l^{\mathrm{SC}})^{-1}\boldsymbol{r}_l = (T_l + \boldsymbol{u}_l\boldsymbol{v}_l^T)^{-1}\boldsymbol{r}_l \tag{5.13}$$

と表される．

5.4.2　Sherman-Morrison 公式の適用

これに対して，Sherman-Morrison 公式 [19][20][22]

$$(T+\boldsymbol{u}\boldsymbol{v}^T)^{-1} \;=\; T^{-1} - T^{-1}\boldsymbol{u}(1+\boldsymbol{v}^T T^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T T^{-1}, \tag{5.14}$$

$$T：m\times m \text{ 行列}, \qquad \boldsymbol{u},\boldsymbol{v}：m \text{ 次ベクトル}$$

を適用すると，前処理演算 (5.13) は，以下のとおりである．

$$
\begin{aligned}
\boldsymbol{r'}_l &= T_l^{-1}\boldsymbol{r}_l - T_l^{-1}\boldsymbol{u}_l(1+\boldsymbol{v}_l^T T_l^{-1}\boldsymbol{u}_l)^{-1}\boldsymbol{v}_l^T T_l^{-1}\boldsymbol{r}_l \\
&= \boldsymbol{y}_l - \boldsymbol{z}_l\left(1+\boldsymbol{v}_l^T\boldsymbol{z}_l\right)^{-1}\boldsymbol{v}_l^T\boldsymbol{y}_l \\
&= \boldsymbol{y}_l - \boldsymbol{z'}_l\begin{bmatrix} 1 & 0 & \cdots & 0 & 1 \end{bmatrix}\begin{bmatrix} y_1^{(l)} \\ y_2^{(l)} \\ \vdots \\ \vdots \\ y_m^{(l)} \end{bmatrix} \\
&= \boldsymbol{y}_l - (y_1^{(l)} + y_m^{(l)})\boldsymbol{z'}_l. \qquad (5.15)
\end{aligned}
$$

ここで，

$$
\begin{aligned}
\boldsymbol{y}_l &= T_l^{-1}\boldsymbol{r}_l \equiv \left[y_1^{(l)} y_2^{(l)} \cdots y_m^{(l)}\right]^T, \qquad (5.16)\\
\boldsymbol{z}_l &= T_l^{-1}\boldsymbol{u}_l, \qquad (5.17)\\
\boldsymbol{z'}_l &= \boldsymbol{z}_l\left(1+\boldsymbol{v}_l^T\boldsymbol{z}_l\right)^{-1}. \qquad (5.18)
\end{aligned}
$$

以上から，前処理演算 (5.13) は，式 (5.15) と変形される．これは，式 (5.16)∼(5.18) を解くことに帰着される．ここで，周期境界要素から生成されたベクトル $\boldsymbol{u}_l$ と $\boldsymbol{v}_l$ は反復過程において不変のベクトルなので，式 (5.17)(5.18) は，反復過程の初回の計算結果を毎回利用できる．結局，反復過程における前処理演算では，式 (5.16) のみを計算すれば良い．また，T_l は周期境界要素を伴わない 3 重対角行列なので，演算量は対角ブロックに対する完全分解よりも少なく抑えられる．

以上の前処理行列の生成過程から，本研究では，この前処理を “SC(Splitting Correction, 分離修正)” 前処理と呼ぶ．

以上のブロック前処理をまとめると，次の表 5.1 のとおりである．

表 5.1: ブロック前処理のまとめ.

	ブロック前処理行列	収束性	演算量 (比率)
block IF	$K_l^{\mathrm{IF}} = \widetilde{L}_l\widetilde{U}_l = P_l - R_l$	劣	$11mn$ (1.00)
block CF	$K_l^{\mathrm{CF}} = L_lU_l = P_l$	良	$17mn$ (1.55)
SC	$K_l^{\mathrm{SC}} = T_l + \boldsymbol{u}_l\boldsymbol{v}_l^T = P_l$	良	$13mn$ (1.18)

演算量の比率は，ブロック不完全分解を `1.00` とした．これより，SC 前処理は，不完全分解に基づく前処理よりも収束性が良く，求解に要する計算時間も増大しないと期待できる.

5.5　数値実験

数値実験は，式 (5.2) で表される方程式に基づき，対称な系であるポアソン方程式と非対称な系である移流拡散方程式について行った．解析解は，どの実験においても $u(x, y) = \sin(2\pi(x + y))$ である.

このときの係数行列の対角ブロックに対して，従来の前処理である不完全分解を施した反復法と本研究で提案する SC を施した反復法とでの収束性と CPU 時間とを比較した．また，並列計算におけるプロセッサ台数に対する計算効率の確認のための実験も実施した.

CPU 時間を測定する各実験において，計算機は IBM RS/6000 SP RISC プロセッサ (Power2, 66MHz) を使用した．**第** 5.5.1**，**5.5.2 **節の対称・非対称問題に対する前処理の効果**を確認する実験では，収束性を評価するためにプロセッサ 1 台のみを使用し，コンパイラは Fortran77 ベースの `xlf` を用いた．**第** 5.5.3 **節の並列計算効率**の確認では，プロセッサ台数を 1, 2, 4, 8, 16, 32 とし，コンパイラは Fortran77 +MPI ベースの `mpxlf` を用いた．計算は倍精度実数で行い，収束判定基準は，相対残差ノルム (2-norm) が 10^{-12} となったところで停止した．時間測定は，前処理行列の分解開始から，反復法が収束して完了するところまでである.

5.5.1　対称問題に対する前処理の効果

式 (5.2) において，$v_1 = v_2 = 0$ とした方程式 (ポアソン方程式) を扱った．このときの係数行列 A は対称行列なので，反復法には CG 法を用い，比較する前処理はブロック IC(不完全コレスキー) 分解である．すなわち，ブロック ICCG 法と SCCG 法とで比較した．問題のサイズは，格子点数 $(m \times n)$ において，65×64, 97×96, 129×128 である．

収束の振舞いを，各モデルについてグラフに示した．65×64 のグラフが図 5.1 であり，97×96 のグラフが図 5.2 であり，129×128 のグラフが図 5.3 である．これらのグラフにおいて，横軸は反復回数を表し，縦軸は log スケールの相対残差ノルム (2-norm) の値を表す．

これらの結果から，SCCG 法の方がブロック ICCG 法よりも速く収束していることが確認できる．また，これらの収束の振舞いの傾向は，ほぼ同じである．

ここで，ブロック ICCG と SCCG の収束の振舞いに注目すると，次のような特徴がある．これらは途中までほぼ同じように収束するが，途中から分岐する．ブロック ICCG の方はある程度横這いとなり，その後再び収束する．SCCG の方は，滑らかに（横這いのような特徴無く）収束する．結局，SC 前処理を施した方の収束の速さは，この収束の振舞いの違いに因ると考えられる．これに関する考察は，文献 [28] として掲載され，本論文の**「付録 B 前処理の効果に対するスペクトル特性からの評価」**にも示した．

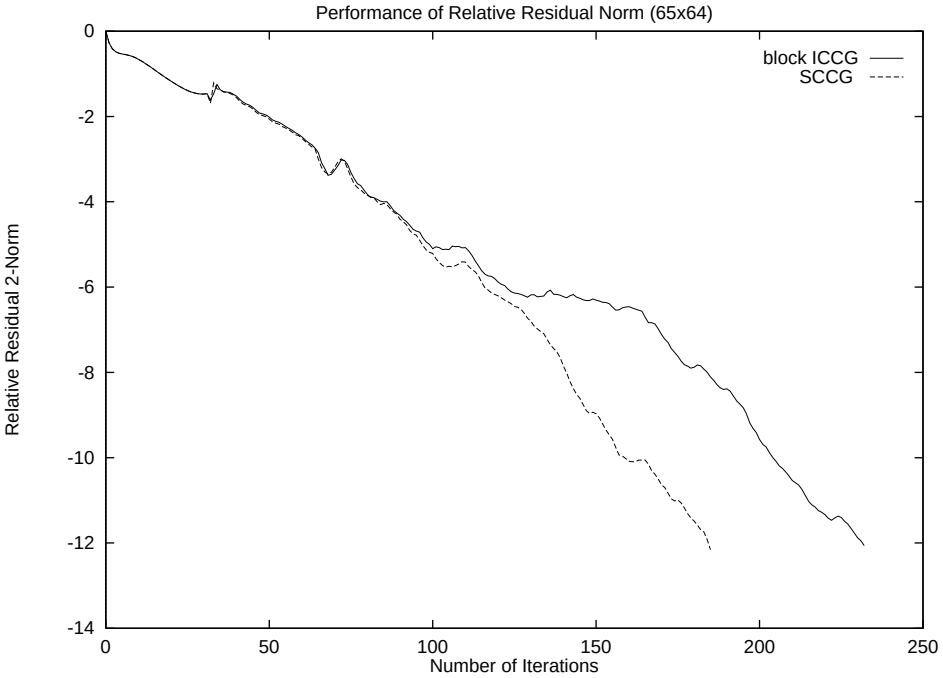

図 5.1: 残差ノルムの振舞い (65×64).

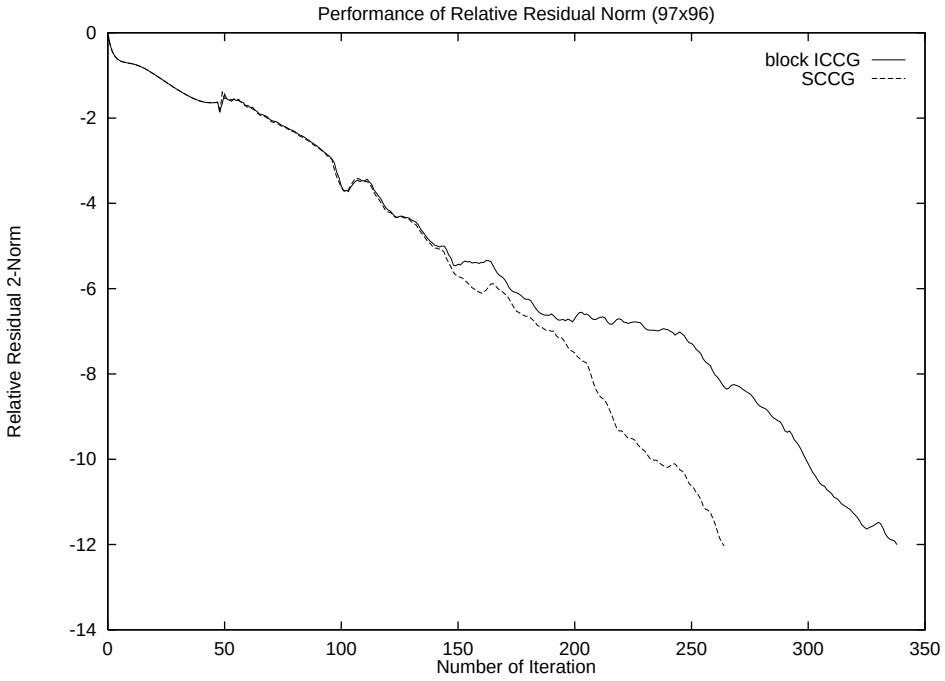

図 5.2: 残差ノルムの振舞い (97×96).

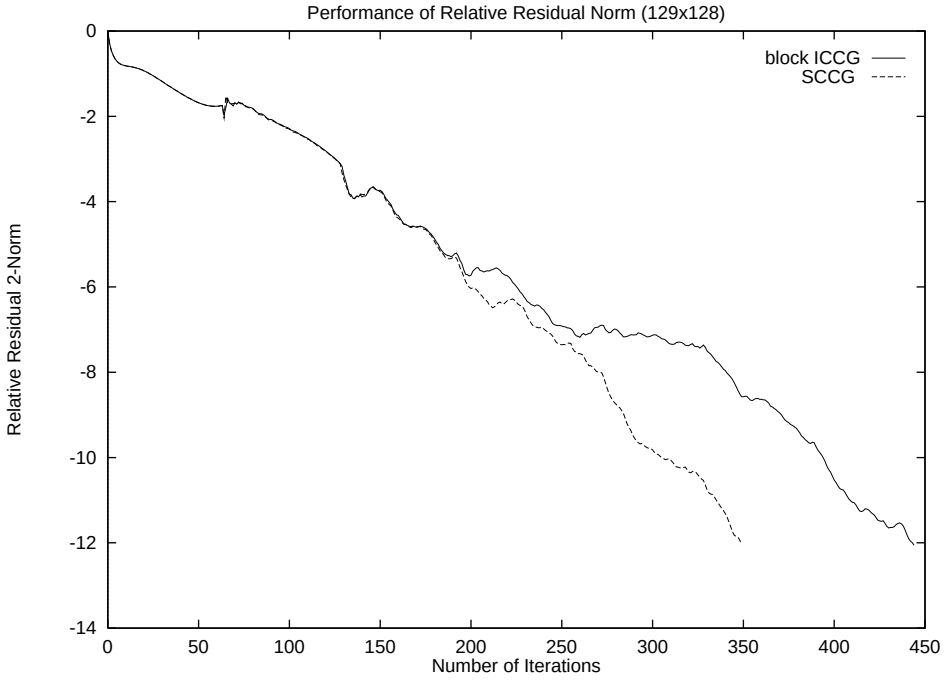

図 5.3: 残差ノルムの振舞い (129×128).

これらの問題に対する実験結果を表 5.2 にまとめた.

表 5.2: 対称問題に対する収束までの反復回数 (CPU time[sec]).

grid size	65×64	97×96	129×128
block ICCG	232(1.43)	338(4.53)	444(10.23)
SCCG	185(1.15)	264(3.56)	349(8.32)
SC/IC [%]	79.7(80.4)	78.1(78.6)	78.6(81.3)

これらの結果から，対称問題に対して SC の方が収束性が改善され，計算時間も短縮していることが確認された.

5.5.2　非対称問題に対する前処理の効果

数値実験は，移流拡散方程式(5.2)において，移流速度 $v_1 = 0.0 \sim 10.0,\ v_2 = 0.0 \sim 10.0$ とした方程式を扱った．このときの係数行列Aは非対称行列なので，反復法にはBiCGSTAB法を用い，比較する前処理は ILU 分解である．問題のサイズは，65×64 である．本章では，移流速度 $(v_1, v_2) = (1.0, 1.0), (0.0, 1.0), (1.0, 0.0)$ の結果を示し，それ以外のグラフは**「 付録 D 非対称問題に対する SC 前処理の効果」** にまとめて掲載した.

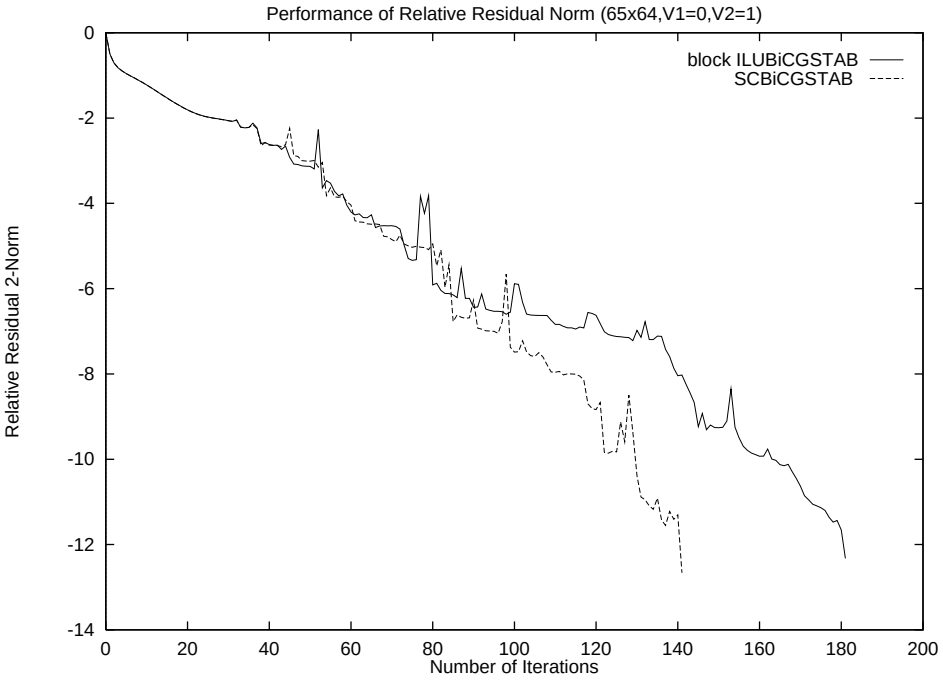

図 5.4: 残差ノルムの振舞い $(v_1 = 0.0, v_2 = 1.0)$.

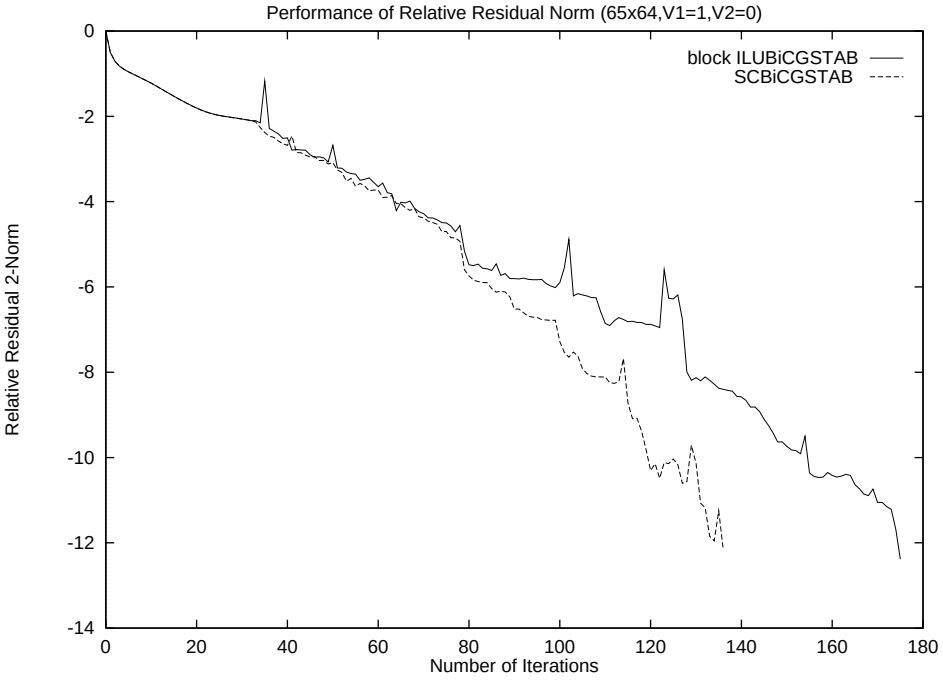

図 5.5: 残差ノルムの振舞い $(v_1 = 1.0, v_2 = 0.0)$.

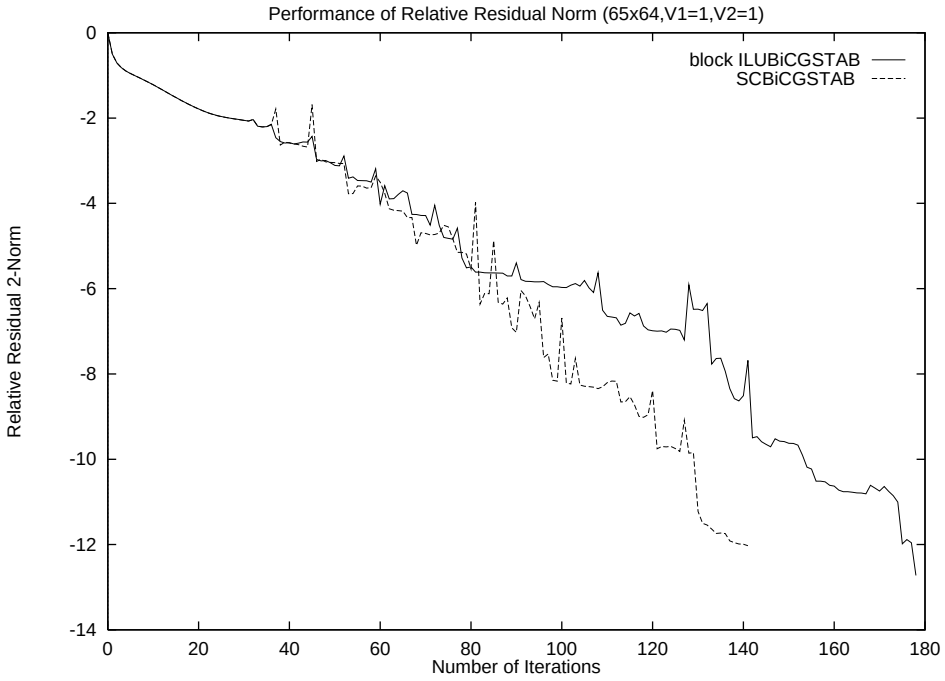

図 5.6: 残差ノルムの振舞い $(v_1 = 1.0, v_2 = 1.0)$.

収束の振舞いを，各モデルについてグラフに示した．$(v_1, v_2) = (0.0, 1.0)$ のグラフが図 5.4 であり，$(v_1, v_2) = (1.0, 0.0)$ のグラフが図 5.5 であり，$(v_1, v_2) = (1.0, 1.0)$ のグラフが図 5.6 である．これらのグラフにおいて，横軸は反復回数を表し，縦軸は対数スケールの相対残差ノルム (2-norm) の値を表す．

これらの結果から，SC 前処理を施した方がブロック ILU を施した方よりも速く収束していることが確認できる．また，非対称問題についても，これら収束の振舞いの傾向は，ほぼ同じである．

これらの問題に対する実験結果を表 5.3 にまとめた．

表 5.3: 非対称問題に対する収束までの反復回数 (CPU time[sec]).

grid size	65×64	65×64	65×64
(v_1, v_2)	$(0.0, 1.0)$	$(1.0, 0.0)$	$(1.0, 1.0)$
block ILUBiCGSTAB	181(4.37)	175(4.21)	178(4.32)
SCBiCGSTAB	141(3.43)	136(3.37)	141(3.49)
SC/ILU[%]	77.9(78.5)	77.7(80.0)	79.2(80.8)

これらの結果から，非対称問題に対しても SC の方が収束性が改善され，計算時間も短

縮していることが確認された.

5.5.3　並列計算効率

ここでは，**第**5.5.1**節**の問題に対するSCCG法と**第**5.5.2**節**の問題に対するSCBiCGSTAB法に対して，並列実行したときのプロセッサ数ごとの台数効果を示す．問題のサイズは，129×128, 193×192, 257×256 である．

対称問題のときのプロセッサ台数に対する全サイズのスピードアップのグラフを，図 5.7 に示した．実線で示した “IDEAL” は，完全な台数効果が得られるときの理想値である．スピードアップを示す値は，

(1CPU における計算時間)/(各 CPU 台数の計算時間)

で算出した.

非対称問題のときのプロセッサ台数に対する全サイズのスピードアップのグラフは，図 5.8 に示した．ここでは，2CPU を基準にスピードアップの効果を示した.

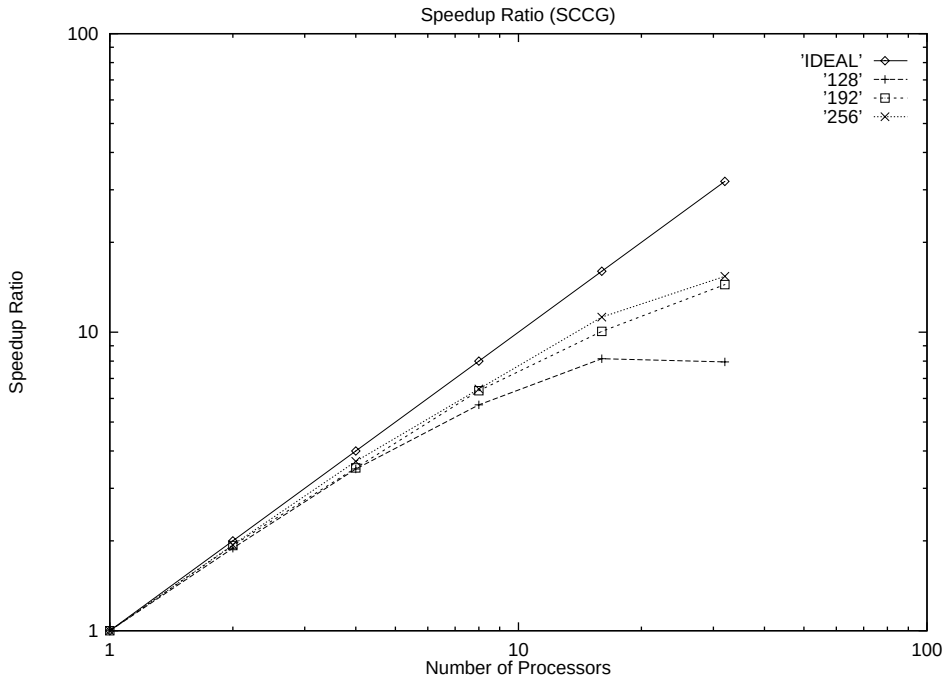

図 5.7: 対称問題に対するスピードアップ.

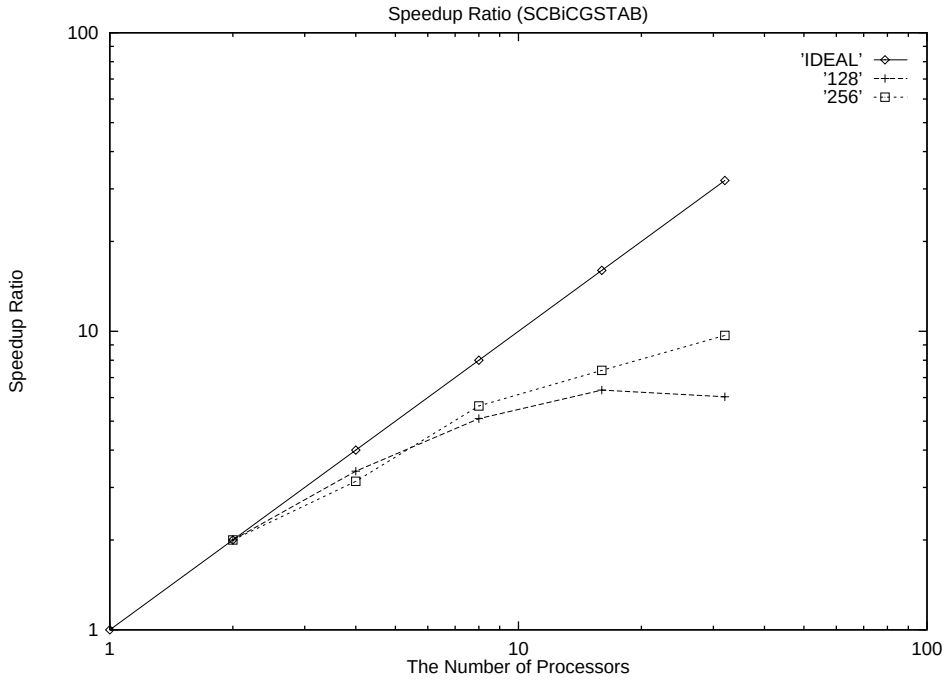

図 5.8: 非対称問題に対するスピードアップ.

以上では，SC前処理における式(5.16)(5.17)が，3重対角行列を係数行列とする線形方程式であるため，コレスキー分解ないしLU分解に基づく計算を行った．さらに，これらの計算部分に対しては，ツイスト分解 [63][64]，リカーシブダブリング (recursive doubling)[57] や巡回縮約法 (cyclic reduction)[21] などの 3 重対角系に特化した手法の適用が容易である．これにより，並列性の向上や効率の良いベクトル化などが期待できる．

5.6　まとめ

本研究では，周期境界条件を課した偏微分方程式を離散化して得られる線形方程式の求解に対する並列計算向きの前処理法 Splitting Correction 前処理を提案した．本前処理法は，ブロック前処理行列を生成して対角ブロックを厳密に分解し，かつ，演算量を増大させない前処理法である．数値実験では，係数行列が対称・非対称の問題を取り上げ，各々に対しても，従来の前処理よりも SC を用いた方が収束性が改善され，収束までの所要反復回数，計算時間共に 20% 程短縮していることが確認された．

この成果は，文献 [27] としてまとめられた．また，この特徴ある収束の振舞いに関する考察は文献 [28] として掲載された．

さらに，並列計算におけるプロセッサ台数に対する計算効率確認のための実験も行い，並列計算における台数効果が確認された．この成果は，文献 [29] としてまとめられた．