同期点の少ない並列化ICCG法のための ブロック化赤－黒順序付け

岩下武史[†] 島崎眞昭[††]

不完全コレスキー分解前処理付き共役勾配法（ICCG 法）の代表的な並列化手法の 1 つとして，並列オーダリングの利用がある．これまでにも様々な並列オーダリングが提案されており，たとえば高い収束性と並列度を同時に実現する優れたオーダリングとして，大きな色数による多色順序付けがある．本論文では，収束性，並列度に加えて，並列化されたアルゴリズム中の同期点について考慮し，各代入計算の並列実行における同期点の数を 1 つにし，かつ前述の手法と同程度の高い収束性が得られる並列オーダリングとして，ブロック化赤－黒順序付けを新たに提案する．本手法は，節点のいくつかをブロック化し，そのブロックに対して赤－黒順序付けを適用するものである．各プロセッサがいくつかのブロックを担当することにより，代入計算が並列実行される．本手法の収束性，並列度，速度向上に関して，オーダリンググラフによる解析的検討，スカラ並列計算機上での数値解析的検討を行う．解析的検討では，ブロックサイズを増加させることにより収束性が改善されることを示し，実行計算機の並列度に合わせた最適なブロックサイズを導く．数値解析的検討では，提案手法がスカラ並列計算機上での実装において高い収束性とキャッシュデータの再利用性を持ち，約 100 万自由度を持つ差分解析の計算例において逐次型 ICCG 法と比べて 16CPU で 14.3 倍の速度向上を得ることを示す．

Block Red-Black Ordering for Parallelized ICCG Solver with Fewer Synchronization Points

Takeshi Iwashita[†] and Masaaki Shimasaki[††]

A use of parallel ordering is one of typical strategies for parallel processing of the Incomplete Choleskey Conjugate Gradient (ICCG) method. While various parallel ordering techniques have been proposed, “the large-numbered multi-color ordering method,” which is one of the most effective techniques, achieves both fast convergence and high parallelism. In addition to convergence and parallelism, the present paper considers synchronization points in parallelized ICCG method. Consequently, a new parallel ordering, “block red-black ordering,” is proposed for a parallelized ICCG solver with fewer synchronization points and a high convergence rate. The new method, which has only one synchronization point in each substitution, attains approximately the same convergence as the large-numbered multi-color ordering method. In the method, nodes in an analyzed grid are divided into several blocks, and red-black ordering is applied to the blocks. Several blocks are assigned to each processor and the substitution is carried out in parallel. For evaluation of the new method, we perform analytical investigation by means of the ordering graph theory and computational tests on a scalar parallel computer. The analytical investigation shows that a convergence rate can be improved by an increase in the block size. The analytical study also presents the optimal block size that depends on the number of processors. Computational tests show that the proposed method attains both fast convergence and effective utilization of data cache on scalar parallel computers and that it achieves a 14.3-fold speed-up by 16 processors on an finite difference analysis with approximately 10^6 degrees of freedom.

1. はじめに

偏微分方程式の境界値問題を有限要素法や差分法で解く際に生ずる正値対称な係数行列を持つ連立一次方程式の一般的な解法として ICCG（Incomplete Cholesky Conjugate Gradient）法[1] がある．ICCG 法は，共役勾配ソルバと不完全コレスキー分解に基づいた前処理部により構成される．このうち，不完全コレスキー分解と前処理として行われる代入計算は，逐次型計算であるためにその並列化が困難である．なか

† 京都大学大型計算機センター
Data Processing Center, Kyoto University
†† 京都大学大学院工学研究科
Graduate School of Engineering, Kyoto University

でも，代入計算は反復のたびに実行されるので，全体の計算コストに対する影響が大きく，その並列化は課題である．これまでにも，代入計算を含むICCG法の並列化に関して様々な研究がなされている．Van der Vorstらは文献2)において，不完全分解前処理の代表的な並列化手法として 1. 打ち切りノイマン系列によるもの 2. リオーダリングによるもの 3. 領域分割法によるもの，をあげている．本論文ではこれらのなかで，リオーダリング（並列オーダリング）による手法に注目し，構造型の差分格子を対象とした場合について考える．

差分格子を対象とした並列オーダリングとしてよく知られているものには，赤－黒順序付け，Dissection orderingなどがある．Duffらは文献3)において様々なオーダリングの優劣に関して論じているが，一般に並列オーダリングには，並列化の代償として反復数が増加するというトレードオフが生ずる．このトレードオフを解決するために，Doiらはオーダリンググラフ理論によるオーダリングの定量的評価から，大きな色数を使用した多色順序付けを提案した[4),5)]．従来，多色順序付けでは2色（この場合は赤－黒順序付けとなる）ないし4色といった少数の色数を用いていたが，同手法では30〜100色の色数を用いる．他の並列オーダリングと比較した場合，同手法は少ない反復数の増加で高い並列度を実現している．そこで，著者らも同手法に注目し，工学分野における応用を目指し検討を行ってきた[6)]．しかし，それらの研究過程において，実行計算環境や対象とする問題のサイズによっては，同手法により得られる高い並列度が十分に活用されない場合があることが分かった．その原因の1つに同期（通信）コストがある．m 色を用いた多色順序付け法では，各代入計算のアルゴリズム中に $m-1$ 個の同期点があり，それにともなう同期ないし通信が必要となる．収束性の改善のために大きな色数を用いた場合，アルゴリズム中の同期点の個数が増え，同期（通信）コストのために十分な台数効果が得られないことがある．そこで，本論文ではアルゴリズム中に同期点が少なく，かつ十分な並列度を持つ並列オーダリングについて検討を行う．

本論文では，まず多色順序付け法において収束性を維持しながら同期コストを少なくする手法として，ブロック化による方法を提案する．節点のいくつかをブロック化することにより色数を削減し，代入計算における同期点を少なくする．このとき収束性が維持されることをオーダリンググラフにより示す．

次に，このブロック化の考えに基づいて，各代入計算中の同期点を1つにしたブロック化赤－黒順序付け法を新たに提案する．本手法では，収束性を高めるための戦略は，多色順序付けにおいて色数を増やす場合と異なり，ブロック内の節点数を増加させることによる．本論文では，同手法における代入計算の並列実行手順，並列度，オーダリンググラフに基づいた収束性について詳細に述べる．

本論文ではプログラミングの容易さから，ブロック化赤－黒順序付け法をスカラ並列計算機上に適用する場合の実装方法，数値解析結果を示す．その結果，同手法において，ブロック内の節点数（ブロックサイズ）を増加させることにより収束性が改善されること，また，逐次実行の場合と比べて高い台数効果が得られることを示す．

2. 対象とする問題とICCG法

本論文では，偏微分方程式を差分公式により離散化したときに生ずる正値対称な連立一次方程式を対象とする．提案手法の説明，従来手法との比較を容易にするために問題を2次元とする．ただし，本論文で議論の対象とする多色順序付け法[5)]や，本論文で提案するブロック化赤－黒順序付け法は，そのアルゴリズムにおいて2次元問題に特化した部分を持たないため，3次元問題への応用が可能である．ブロック化赤－黒順序付け法の3次元問題への応用手法については4.4節で簡単に述べる．

ここでは，対象とする偏微分方程式を5点差分公式により離散化し，得られる連立一次方程式を解くものとする．未知数の置かれる差分格子として，$nx \times ny$（それぞれx方向，y方向の格子点数）の格子を用い，各格子点を

$$(i,j) \in G = \{(i,j) | 1 \leq i \leq nx, 1 \leq j \leq ny\} \quad (1)$$

と表すと，5点差分公式は

$$a_{ij,1}u_{ij-1} + a_{ij,2}u_{i-1j} + a_{ij,3}u_{ij} + a_{ij,4}u_{i+1j} + a_{ij,5}u_{ij+1} = f_{ij} \quad (2)$$

で与えられる．ここで，u_{ij} は格子点 (i,j) における求めるべき未知数である．本論文で扱う問題は次式のような対称性が成り立つ場合である．

$$a_{ij,1} = a_{ij-1,5},\ a_{ij,2} = a_{i-1j,4} \quad (3)$$

このとき，式(2)は

$$c_{ij-1}u_{ij-1} + b_{i-1j}u_{i-1j} + a_{ij}u_{ij} + b_{ij}u_{i+1j} + c_{ij}u_{ij+1} = f_{ij} \quad (4)$$

と書き換えることができる．ここで，$a_{ij} = a_{ij,3}$，$b_{ij} = a_{ij,4}$，$c_{ij} = a_{ij,5}$ である．格子点の総数を

$n = nx \times ny$ とし，格子点全体 G に対して式 (4) を連立させると，解くべき方程式として

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f} \tag{5}$$

のような n 元連立一次方程式が得られる．ここで，G 上の未知数 $\{u_{ij}\}$, 右辺値 $\{f_{ij}\}$ のベクトル $\boldsymbol{u}$，$\boldsymbol{f}$ へのマッピングが節点のオーダリング（番号付け，順序付け）であり，集合 $V = \{1, \cdots, n\}$ を用いると，G, V 間の 1 対 1 写像で与えられる．節点のオーダリング（以下，オーダリングとも表記）は，係数行列 $\boldsymbol{A}$ の非零要素パターンを決める要因となるので，解法によっては，求解に要する計算時間や計算の並列実行可能性に大きな影響を与える．

本論文で対象とする連立一次方程式は，係数行列 $\boldsymbol{A}$ が対称正定値な場合である．このような連立一次方程式の解法として最も一般的なものとして ICCG 法がある．ICCG 法は前処理付き共役勾配法の一種で，前処理行列として係数行列 $\boldsymbol{A}$ の不完全コレスキー分解行列を用いる．ICCG 法の反復部は，前処理部である前進・後退代入計算と共役勾配ソルバ部である内積計算，行列 · ベクトル積計算，ベクトル更新により構成される．これらの計算部分において，共役勾配ソルバ部については，係数行列，各ベクトルを行方向に分割することにより容易に並列化できる．一方，前処理行列を求めるための不完全コレスキー分解と反復中の代入計算は，逐次計算であるためにその並列化が困難である．したがって，これまでの並列化 ICCG 法に関する研究では，主としてこれらの計算部分の並列化について検討されてきた．そこで，本論文でも不完全コレスキー分解と代入計算の並列化を中心に述べることにする．

ICCG 法では，係数行列を

$$\boldsymbol{A} \simeq \boldsymbol{L}\boldsymbol{D}^{-1}\boldsymbol{L}^T \tag{6}$$

のように不完全コレスキー分解して得られる行列 $\boldsymbol{L}\boldsymbol{D}^{-1}\boldsymbol{L}^T$ を前処理行列として用いる．ここで，$\boldsymbol{L}$ は下三角行列，$\boldsymbol{D}$ は対角行列である．5 点差分公式を用いた差分解析では，係数行列 $\boldsymbol{A}$ の狭義下三角部分 $\boldsymbol{L}_s$ を用いて，

$$\boldsymbol{L}\boldsymbol{D}^{-1}\boldsymbol{L}^T = (\boldsymbol{L}_s + \boldsymbol{D})\boldsymbol{D}^{-1}(\boldsymbol{L}_s + \boldsymbol{D})^T \tag{7}$$

と書け，行列 $\boldsymbol{D}$ は次式を前進代入計算することにより求められる．

$$\boldsymbol{D} = \mathrm{diag}(\boldsymbol{A}) - \mathrm{diag}(\boldsymbol{L}_s\boldsymbol{D}^{-1}\boldsymbol{L}_s^T) \tag{8}$$

ここで，$\mathrm{diag}(\boldsymbol{X})$ は行列 $\boldsymbol{X}$ の対角部分を表す．

一方，反復中の前処理部はある残差ベクトル $\boldsymbol{r}$ に対して，

$$\{(\boldsymbol{L}_s + \boldsymbol{D})\boldsymbol{D}^{-1}(\boldsymbol{L}_s^T + \boldsymbol{D})\}\boldsymbol{z} = \boldsymbol{r} \tag{9}$$

を解くことで与えられ，前進代入計算

$$\boldsymbol{y} = \boldsymbol{D}^{-1}(\boldsymbol{r} - \boldsymbol{L}_s\boldsymbol{y}) \tag{10}$$

および後退代入計算

$$\boldsymbol{z} = \boldsymbol{y} - \boldsymbol{D}^{-1}\boldsymbol{L}_s^T\boldsymbol{z} \tag{11}$$

により求められる．ここで，式 (8)，(10)，(11) の代入計算における並列実行性は $\boldsymbol{L}_s$ の非零要素パターン，すなわち節点のオーダリングにより決まる．また，式 (8) により求められる前処理行列はオーダリングにより異なり，その結果，オーダリングにより前処理効果，すなわち ICCG 法の収束性が通常異なることになる．

3. 多色順序付けによる並列化 ICCG 法

3.1 並列オーダリングとオーダリンググラフ理論

前章で述べたように，式 (8)，(10)，(11) の代入計算が並列実行可能であるかどうかは，節点のオーダリングにより決まる．たとえば，一般に最もよく用いられる自然な順序付け（Natural ordering，辞書式順序付けとも呼ばれる）では，これらの代入計算を並列化することはできない．そこで，代入計算を並列実行することができるようなオーダリング（並列オーダリング：Parallel ordering と呼ばれる）として，様々なオーダリングが提案されている．たとえば，代表的なものとして，赤－黒順序付け，Multi-wavefront 順序付けなどがあげられる．しかし，これまでの研究において，多くの並列オーダリングが逐次型のオーダリングと比較して反復法の収束性を低下させる，すなわち不完全分解前処理の効果を低下させることが分かってきた[3]．そこで，Doi らはオーダリングを定量的に扱う手法として，**図 1** に示されるようなオーダリンググラフによる解析を提案している[4]．オーダリンググラフ中において，矢印の終点にあたる節点は始点にあたる節点よりも後に番号付けされている．つまり，オーダリンググラフは節点間のデータ依存関係を表す．同じオーダリンググラフを持つ順序付けは等価な前処理行列を持ち，その前処理効果，すなわち（反復法の）収束性は等しい．

Doi らは，このオーダリンググラフの中で，incompatible node の割合が収束性に影響することを数値実験で示している．ここで，incompatible node とは，図 1 に示されるような節点であり，節点 (i, j) の節点番号を p_{ij} として，以下のように与えられる．すなわち，全節点のうち，

$$\{(i,j) \in G |\ p_{ij} < p_{i+1j},\ p_{ij} < p_{i-1j}\} \tag{12}$$

または，

$$\{(i,j) \in G |\ p_{ij} < p_{ij+1},\ p_{ij} < p_{ij-1}\} \tag{13}$$

を満たす節点が incompatible node であり，それぞれ，x-incompatible node，y-incompatible node と呼ばれ

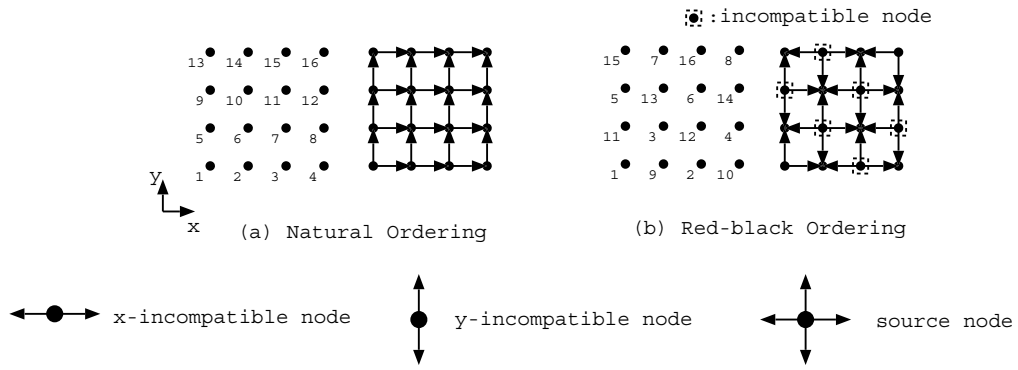

図 1 オーダリンググラフの例
Fig. 1 Examples of ordering graph.

る．また，incompatible node のうち，特に式 (12), (13) をともに満たすような節点は，source node または fully incompatible node と呼ばれる．Doi らの数値実験は，このような incompatible node が全節点に対して占める割合が高ければ高いほど収束性が悪化することを示している．しかしその一方で，incompatible node の一種である source node は，代入計算を並列実行するうえでの開始点となりうる．したがって，ある程度 source node が存在しないと十分な並列度は得られない．ここに，並列オーダリングにおける並列度と収束性のトレードオフが生ずることになる．

3.2 多色順序付けによる並列代入計算

Doi らは上記のオーダリンググラフ理論に基づいて，incompatible node の割合を適度に保ちながら高い並列度を実現するオーダリングとして，多色順序付けにおいて多くの色数を用いることを提案した[4),5)]．多色順序付けでは，同色に塗られた節点はお互いにデータ依存関係がなく，並列に扱うことができる．以下に多色順序付けの手法について述べる．まず，ある色数 $m(\geq 2)$ を決め，節点全体 G を

$$C(l) = \{(i,j) \in G | \mathrm{mod}(i+j-2, m)+1 = l\} \quad (14)$$

のように m 個の部分集合 $(C(l), l = 1, \cdots, m)$ に分割する．ここで，$C(l)$ は色 l が塗られた節点の集合と考えればよい．多色順序付けでは，節点を $C(1)$ から $C(m)$ まで色の順に順序付けする．このとき，同色に塗られた節点は互いにデータ依存関係を持たず，式 (8), (10), (11) の代入計算は各色ごとに並列化できる[8)]．

図 2 に例として 4 色による多色順序付けを示す．図中の C1〜C4 は色を表し，黒丸は incompatible node を表す（図 3, 4 とも）．図中において，色数を増やすと，全節点に対して incompatible node の割合が下がり，前節で述べた理論から収束性が増すことが分かる．

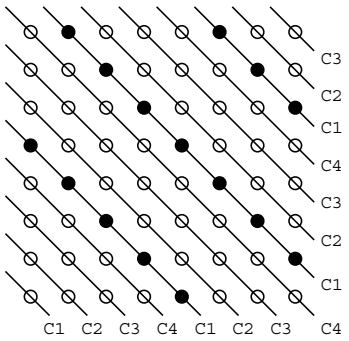

図 2 4 色順序付け
Fig. 2 4-color ordering.

ここで，incomaptible node の割合を定量的に扱うために，Doi らにより定義された incompatibility ratio R_{ic}[5)]，

$$R_{ic} = N_{ic}/N_{icp} \quad (15)$$

を用いる．式 (15) において，N_{ic} はオーダリンググラフ中の incompatible node の数を表す．一方，N_{icp} は全節点に対して最大可能な incompatible node の数を表し，赤−黒順序付けにおける incompatible node 数に等しく，

$$N_{icp} = \frac{nx \cdot ny}{2} = \frac{n}{2} \quad (16)$$

で与えられる．m 色順序付けでは，図 2 に示されるように，C(1) に属する節点が incomptible node（source node）となる．したがって，m 色順序付けにおいて各色に割り当てられる節点数が等しいとしたとき，incompatible node 数 N_{ic} は，

$$N_{ic} = n/m \quad (17)$$

で与えられ，これらはすべて source node である．式 (15), (16), (17) より，incompatibility ratio R_{ic} は，

$$R_{ic} = \frac{(n/m)}{(n/2)} = \frac{2}{m} \tag{18}$$

となる．一方，並列度 P は 1 色に割り当てられた節点数となるので，

$$P = n/m \tag{19}$$

で与えられる．これまでの研究にでは，適当な並列度と収束性を得る値として，$m = 30 \sim 100$，$R_{ic} = 0.02 \sim 0.13$ 程度が用いられている[5]．

4. ブロック化赤－黒順序付け

4.1 ブロック化多色順序付け

前節で述べた大きな色数による多色順序付け法は，高い並列度と収束性を持つ優れた並列順序付け法であるが，代入計算中の同期点の個数が多いという問題点がある．

そこで本論文では，大きな色数による多色順序付け法により得られる収束性を維持しながら，色数を少なくして同期コストを削減する手法として，いくつかの節点をブロック化する手法（ブロック化多色順序付け法）を提案する．ただし，ブロック内のオーダリングは，元の多色順序付けとオーダリンググラフが同一となるようにする．**図 3** に前節で述べた 4 色順序付けのブロック化の例を示す．図 3 においては，図 2 の 4 色順序付けにおける 2 色を 1 色にまとめることにより色数を削減している．この場合，オーダリンググラフは 4 色順序付けとまったく同じであり，収束性も等しい．ただし，ブロック内の節点はまとめて扱う必要があり，並列度は 1 色に割り当てられたブロック数となる．

図 3 のようなブロック化の場合，ブロック化された節点数にばらつきがでるため，プロセッサ間の負荷バランスがとりにくい．また，メモリへの間接参照も避けられない．そこで，**図 4** のようなブロック化を行う．ここで，ブロック内のオーダリングは辞書式順序付けとする．図 4 のブロック化では，すべてのブロック内の節点数が等しいので，プロセッサ間の負荷バランスを均等に保つことができる．また，図 4 のブロック化によっても，ブロック内節点数（ブロックサイズ）を調整することにより，R_{ic} をある値，たとえば 0.13 以下に抑えながら多色順序付けの色数を削減できる．ここで，このようなブロック化による色数の削減を進めていくと，同期コストの観点から見た場合，最も少ない色数である 2 色（赤－黒）の場合が最も有効となる．すなわち，本手法では色数の増加により収束性を高める（incompatible node の割合を減らす）のではなく，ブロック化により収束性を高めることにより，

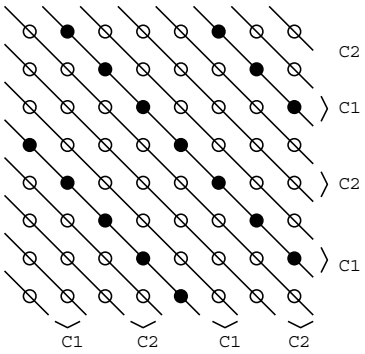

図 3 4 色から 2 色へのブロック化
Fig. 3 Reduction of number of colors by blocking.

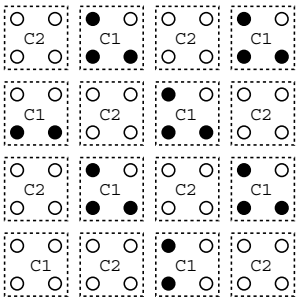

図 4 ブロック化多色順序付け（2 色）
Fig. 4 Block multi-color ordering (2 colors).

代入計算中の同期点を最小に保ちながら，大きな色数での多色順序付け法と同等の収束性が期待される．ここでは，本手法をブロック化赤－黒順序付け法と呼び，その詳細を次節で述べる．

4.2 ブロック化赤－黒順序付け

本節では，本論文で新たに提案するブロック化赤－黒順序付け法に関して，その手法の詳細を述べる．同手法では，まずブロックサイズ $nxb \times nyb$ を決定する．ここで，nxb，nyb は 1 ブロック内の x 方向，y 方向の節点数である．さらに，このブロック単位に基づいて，解析対象の差分格子を**図 5** のようにブロック分割し，各ブロックを赤－黒順序付けによって 2 色に塗り分け，番号を付ける．ここで，簡単のために，

$$\mathrm{mod}(nx, nxb) = 0, \quad \mathrm{mod}(ny, nyb) = 0 \tag{20}$$

が成り立つものとする．式 (20) が成り立たない場合に関する取扱いについては，計算機への実装方法により異なるため 4.4 節で述べる．ブロックの番号に従い赤ブロックの節点から順に番号を付け，その後に黒ブロックの節点に番号を付けると，同色のブロック間は互いにデータ依存関係がなく，代入計算はブロックごとに並列化できる．ここで，ブロック内の節点はそのオーダリンググラフが辞書式順序付けと一致するよう

に付ける．すなわち，通常はブロック内の節点に辞書式順序付けを適用すればよい．プロセッサは各色ごとに 1 つないし複数のブロックを担当し，代入計算はプロセッサ間で並列実行される．本手法におけるアルゴリズム中の同期点は各代入計算中に 1 回である．以下にブロック化赤－黒順序付けを用いた場合に行われる 1 つのブロックに対する前進・後退代入計算を示す．以下の計算は逐次的な計算であり，この計算が複数のプロセッサで並列に同時実行される．手法の説明のための表記として，赤ブロックに属する節点の集合を R, 黒ブロックに属する節点の集合を B とし，ブロック内の節点に次式で与えられるブロック内座標を定義する．

$$(i_b, j_b) = (1 \le i_b \le nxb,\ 1 \le j_b \le nyb) \quad (21)$$

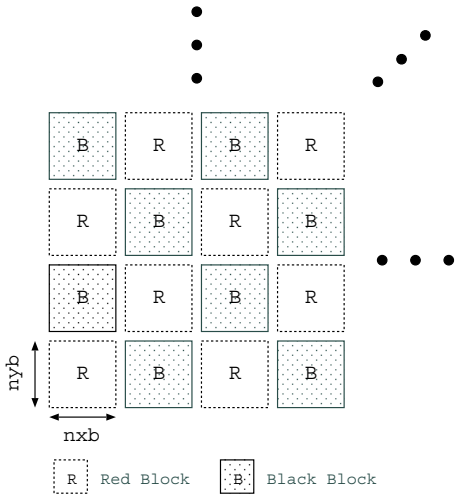

図 5 ブロック化赤－黒順序付けにおけるブロック分割と色分け
Fig. 5 Colored blocks in block red-black ordering.

まず，前進代入計算について考える．赤ブロック内の節点に対する前進代入計算は以下のように行われる．まず，赤ブロックに属し，ブロック内座標が $(i_b, j_b) = (1,1)$ の節点，すなわち $\{(i,j) \in R | i_b = 1,\ j_b = 1\}$ の節点（**図 6** 中の I）に対して，

$$y_{ij} = r_{ij}/d_{ij} \quad (22)$$

が行われる．ここで，y_{ij}，r_{ij}，d_{ij} はそれぞれ，節点 (i,j)（i，j は全体格子座標）に対応するベクトル $\boldsymbol{y}$, $\boldsymbol{r}$ の要素，および，行列 $\boldsymbol{D}$ の対角要素である．次に，$\{(i,j) \in R | 2 \le i_b \le nxb,\ j_b = 1\}$ の節点（図 6 中の II 部分）に対して，

$$y_{ij} = (r_{ij} - b_{i-1j} y_{i-1j})/d_{ij} \quad (23)$$

が行われ，次に，$\{(i,j) \in R | i_b = 1,\ 2 \le j_b \le nyb\}$ の節点（図 6 中の III 部分）に対して，

$$y_{ij} = (r_{ij} - c_{ij-1} y_{ij-1})/d_{ij} \quad (24)$$

が行われる．最後に $\{(i,j) \in R | 2 \le i_b \le nxb,\ 2 \le j_b \le nyb\}$ の節点（図 6 中の IV 部分）に対して，

$$y_{ij} = (r_{ij} - b_{i-1j} y_{i-1j} - c_{ij-1} y_{ij-1})/d_{ij} \quad (25)$$

が実行される．各プロセッサでいくつかのブロックに対して，上記の前進代入計算を行った後，プロセッサ間の同期をとり黒ブロックに対する前進代入計算が以下のように行われる．

まず，$\{(i,j) \in B | 1 \le i_b \le nxb - 1,\ 1 \le j_b \le nyb - 1\}$ の節点（図 6 中の V）に対して，

$$y_{ij} = (r_{ij} - b_{i-1j} y_{i-1j} - c_{ij-1} y_{ij-1})/d_{ij} \quad (26)$$

が行われ，次に，$\{(i,j) \in R | i_b = nxb,\ 1 \le j_b \le nyb - 1\}$ の節点（図 6 中の VI 部分）に対して，

$$y_{ij} = (r_{ij} - b_{i-1j} y_{i-1j} - c_{ij-1} y_{ij-1} - b_{ij} y_{i+1j})/d_{ij} \quad (27)$$

が行われ，続いて，$\{(i,j) \in R | 1 \le i_b \le nxb - 1,\ j_b =$

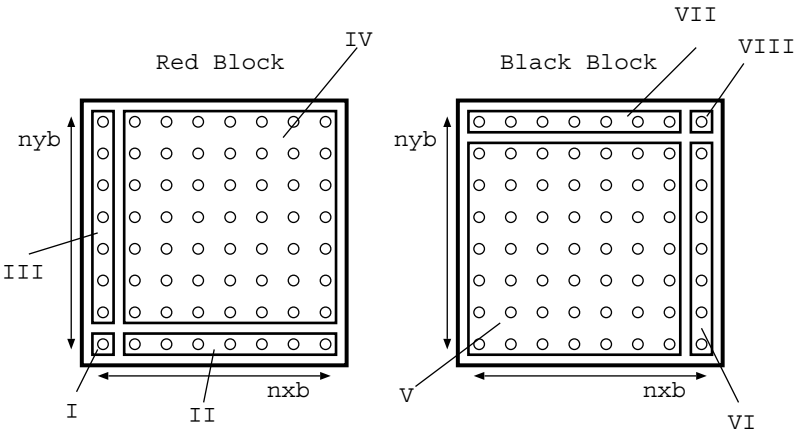

図 6 各ブロックにおける前進代入計算の計算順序（ブロック内の節点の区分け）
Fig. 6 Sequence of forward substitution in each block (Partition of nodes in each block).

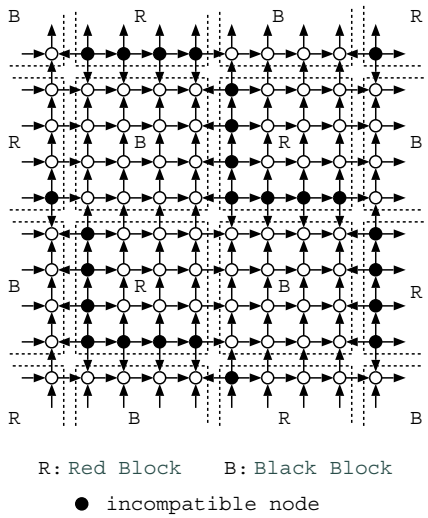

図 7 ブロック化赤−黒順序付けにおけるオーダリンググラフ
Fig. 7 Ordering graph of block red-black ordering.

$nyb-1\}$ の節点（図 6 中の VII 部分）に対して，

$$y_{ij}=(r_{ij}-b_{i-1j}y_{i-1j}-c_{ij-1}y_{ij-1}-c_{ij}y_{ij+1})/d_{ij} \quad (28)$$

が行われる．最後に $\{(i,j)\in R|i_b=nxb,\ j_b=nyb\}$ の節点（図 6 中の VIII 部分）に対して，

$$y_{ij}=(r_{ij}-b_{i-1j}y_{i-1j}-c_{ij-1}y_{ij-1}-b_{ij}y_{i+1j}-c_{ij}y_{ij+1})/d_{ij} \quad (29)$$

が実行され，前進代入計算が終了する．

一方，後退代入計算は，まず黒ブロックに対して，図 6 の VIII → VII → VI → V の順に計算され，すべての黒ブロックに関する計算が終了した後，赤ブロックに対して，図 6 の IV → III → II → I の順に行われる．不完全コレスキー分解は前進代入計算と同様の手順で行われる．

次にブロック化赤−黒順序付け法における並列度と収束性について考える．ブロック化赤−黒順序付け法では，同色のブロックが並列に扱われる．したがって，並列度 P は 1 色に割り当てられたブロック数となる．ここで，簡単のために，$nx=ny$，$nb=nxb=nyb$ とすると，総ブロックの個数 ntb は，

$$ntb=(nx\cdot ny)/(nxb\cdot nyb)=n/nb^2 \quad (30)$$

となる．1 色あたりのブロック数は ntb の半数となるので，並列度 P は，

$$P=ntb/2=n/2nb^2 \quad (31)$$

で与えられる．次に incompatible node について考える．**図 7** にブロック化赤−黒順序付けにおけるオーダリンググラフを示す．図 7 から分かるように，ブロック化赤−黒順序付けにおいて incompatible node となるのは，赤ブロック中でブロック内座標が，図 6 I，II，III に属している部分であり，このうち領域 I に属する節点のみが source node となる．1 つの赤ブロックにおいて，領域 I に属する節点数は 1 であり，領域 II，III に属する節点数はいずれも $nb-1$ で与えられる．したがって，1 つの赤ブロックにおける incompatible node の数 N_{icrb} は

$$N_{icrb}=1+(nb-1)+(nb-1)=2nb-1 \quad (32)$$

となる．全 incompatible node の総数 N_{ic} は 1 赤ブロック内の incompatible node 数 N_{icrb} と赤ブロックの数 $ntb/2$ の積で与えられ，

$$N_{ic}=N_{icrb}\cdot\frac{ntb}{2}=\frac{(2nb-1)n}{2nb^2}=\frac{n}{nb}-\frac{n}{2nb^2} \quad (33)$$

となる．このうち source node は各赤ブロックに 1 つであるので，その総数は $ntb/2$ で与えられる．式 (15)，(16)，(33) より，incompatibility ratio R_{ic} は，

$$R_{ic}=N_{ic}/(n/2)=2/nb-1/nb^2 \quad (34)$$

となる．

表 1 ブロック化赤−黒順序付け法と m 色順序付け法の比較
Table 1 Comparison between block red-black ordering method and m-color ordering method.

	ブロック化赤−黒順序付け法	m 色順序付け法
各代入計算中の同期点数	1	$m-1$
並列度	$n/2nb^2$	n/m
R_{ic}	$2/nb-1/nb^2$	$2/m$

4.3 ブロック化赤−黒順序付け法と多色順序付け法との比較

表 1 にブロック化赤−黒順序付け法（ブロックサイズ：$nb\times nb$）と m 色順序付け法の比較を示す．まず，代入計算中の同期点数に関しては，ブロック化赤−黒順序付け法の方が有利である．次に，収束性について考える．式 (18)，(34) より，ブロック化赤−黒順序付け法において $nb=m$ とすれば，式 (34) の右辺第 2 項のために m 色順序付け法と比べて小さい R_{ic} 値が得られ，より高い収束性が得られることが分かる．ただし，nb ないし m が大きくなるにつれ，この右辺第 2 項の影響は小さくなり，収束性は同程度となる．一方，並列度に関しては，多色順序付け法の方が有利である．しかし，著者らの検討では，実用的な問題を対象とした場合，ブロック化赤−黒順序付け法において十分な並列度が確保できると考えている．たとえば，32 色順序付け法と同程度の収束性が得られるブロック赤−黒順序付け法（ブロックサイズ：32×32）について考えると，1024×1024 の二次元問題において得られる並列度は 512 である．現在の計算機の性能と本

問題サイズを勘案すると，この 512 の並列度は十分に大きな値であるといえる．また，今後のプロセッサの性能向上を考えると，ある問題サイズに対して適当な並列度（プロセッサ数）はより小さくなると想定され，ブロック化赤－黒順序付け法において十分な並列度が確保できると考えられる．

また，ブロック化赤－黒順序付け法では，実行計算機に応じた並列度を設定できるという利点がある．多色順序付け法の場合，実行計算機の並列度（プロセッサ数など）が小さい場合において，収束性を高めるために色数を非常に大きくとることは，同期（通信）コストの点から有利ではない．すなわち，最適な色数の決定は，収束性，実行並列度，同期（通信）コストのすべてを勘案する必要があり，容易ではない．一方，ブロック化赤－黒順序付け法では，同期点の数はブロックサイズに無関係であり，実行環境の並列度に一致するまでブロックサイズを増加させれば，その並列度に対して最良の収束度が得られる．すなわち，最適なブロックサイズ $nb_{op} \times nb_{op}$ は，並列度 P（1 色あたりのブロック数）が実行計算機の並列度 N_p と一致する

$$P = N_p \tag{35}$$

の場合に得られ，式 (31) より

$$\frac{n}{2 \cdot nb_{op}^2} = N_p \tag{36}$$

である．式 (36) より，

$$nb_{op} = \sqrt{\frac{n}{2N_p}} \tag{37}$$

となる．

4.4 ブロック化赤－黒順序付け法のスカラ並列計算機上での実装と 3 次元問題への応用

本章で述べたブロック化赤－黒順序付け法では，計算機に応じた複数の実装方法が考えられる．本論文ではその中で，スカラ並列計算機向きでプログラミングが容易な方法について述べる．

ブロック化赤－黒順序付け法では，各プロセッサは各色において 1 つないし複数のブロックを担当する．担当するブロックに属する節点の番号が分かれば，代入計算を行うことができる．そこで，係数行列などの各種ベクトルを辞書式順序付けに基づいて作成し，担当の各ブロックの左下点，右上点の x，y 格子座標を保持するとする．たとえば，それらを LLx，LLy，RUx，RUy とする．このとき，各ブロック内の節点は，

$$\{(i,j)|LLx \leq i \leq RUx,\ LLy \leq j \leq RUy\} \tag{38}$$

で与えられ，辞書式順序付けに基づく節点番号 i_{dic} は，

$$i_{dic} = i + nx \cdot j \tag{39}$$

となる．節点 (i,j) に関連する節点 $(i,j-1)$，$(i-1,j)$，$(i+1,j)$，$(i,j+1)$ の節点番号は，それぞれ $i_{dic}-nx$，$i_{dic}-1$，$i_{dic}+1$，$i_{dic}+nx$ で与えられ，代入計算は間接参照を必要とすることなく実行される．付録に本実装手法に基づいた前進代入計算のプログラムコード例を示す．

本実装方法において，プログラム中の代入計算部分は，ブロックの左下点，右上点格子座標に基づいて書かれるため，各ブロックのサイズは異なっていてもよい．すなわち，同色のブロック間にデータ依存関係が生じないようにブロックが 2 色に塗り分けられていれば，ブロックサイズは異なっていてもよい．したがって，式 (20) が成り立たない場合においても，対象格子の右端，上端部のブロックにおいて x，y 方向の節点数を調整することで対応することができる．

次に本節では，ブロック化赤－黒順序付け法の 3 次元問題への応用について簡単に述べる．ここでは，対象格子の z 方向の格子点数を nz とし，z 方向のブロックサイズを nzb とする．3 次元問題では，まず格子全体（$nx \times ny \times nz$）をブロックサイズ $nxb \times nyb \times nzb$ の直方体により分割し，生じたブロックに対して赤－黒順序付けを適用する．ブロック内のオーダリングには辞書式順序付けを適用する．このとき，2 次元問題と同様に同色のブロックは依存関係を持たず，並列に処理することができる．スカラ並列計算機への実装においては，2 次元の場合と同様に，ブロックの左下手前角点 (LLx, LLy, LLz)，右上奥角点 (RUx, RUy, RUz) の格子座標に基づいてプログラムを書けばよい．

5. 解 析 結 果

5.1 解析対象と実行計算環境

本論文では，解析対象として次式で与えられるポアソン方程式の境界値問題を用いる．

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u(x,y)) &= f \ in\ \Omega(0,1) \times (0,1) \quad (40)\\ u(x,y) &= 0 \ on\ \delta\Omega \\ if\ (\frac{1}{4} \leq x \leq \frac{3}{4}\ \&\ \frac{1}{4} \leq y \leq \frac{3}{4})\ then & \\ \kappa &= 100 \\ else \quad \kappa &= 1 \end{aligned}$$

ここで f は，格子点を辞書式順序付けで並べた場合の格子番号を k として，$0.5 \sin(k+1)$ である．本論文では，未知数の個数を 1025×1025 とする．式 (40) を 5 点差分公式 (2) により離散化し，ある節点のオーダリングに基づいて得られる連立一次方程式 (5) を ICCG 法により解く．ここで，解析プログラム上において，係数行列と前処理行列を表す a_{ij}，b_{ij}，c_{ij}，d_{ij} は辞書式順序付けに基づいて一次元配列 $\boldsymbol{a}$，$\boldsymbol{b}$，$\boldsymbol{c}$，$\boldsymbol{d}$ にそれ

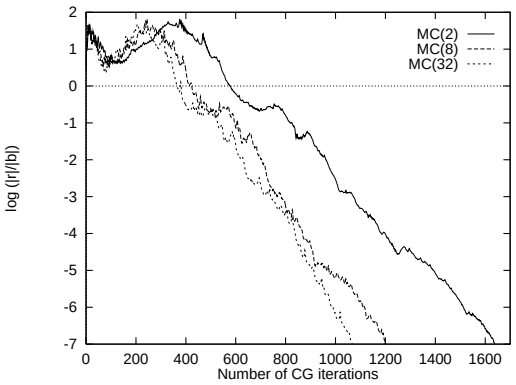

図 8 多色順序付け法の収束性

Fig. 8 Convergence behavior in multi-color ordering method.

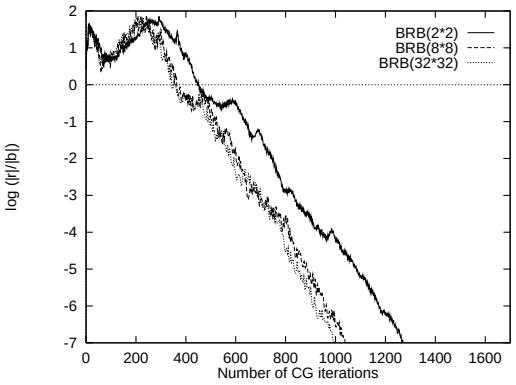

図 9 ブロック化赤－黒順序付け法の収束性

Fig. 9 Convergence behavior in block red-black ordering method.

ぞれ格納される．ICCG 法の収束判定基準として，右辺ベクトルと残差ベクトルの比が 10^{-7} 以下となる条件を用いる．解析は京都大学大型計算機センターの富士通製 GP-7000F model 900 上での倍精度実数計算により行った．プログラミング言語として Fortran を用い，Open MP を用いたマルチスレッドによる並列化を行った．コンパイラには富士通製 Fortran Compiler Version 5.1 を使用し，コンパイルオプションとして-KOMP -O2 を指定した．

5.2 解析結果

図 8，**9** に多色順序付け法とブロック化赤－黒順序付け法の収束性の比較を示す．ここで，MC(m) および BRB($nbx \times nby$) は，それぞれ m 色順序付け法，ブロックサイズ ($nbx \times nby$) のブロック化赤－黒順序付け法を表すものとする．ここで，m 色順序付け法の実装方法については，Washio，Doi らが文献 5)，8) に示している間接参照を必要としない手法を用いた．同手法では，各種ベクトルは辞書式順序付けに基づいて作成され，代入計算は m ストライドのループで構成される．ただし，色数に関する制限として

$$\mathrm{mod}(nx - 1, m) = 0 \tag{41}$$

が課せられる．

まず，図 9 より，ブロック化赤－黒順序付け法において，ブロック内節点数を増加させることで収束性の改善が図れることが分かる．すなわち，代入中の同期点を最小に抑えながら反復法の収束性を高めることができる．次に，図 8，9 において，MC(m)，BRB($nb \times nb$) で $m = nb$ とした場合の比較を行うと，ブロック化赤－黒順序付け法の方が収束性が良い．両手法における収束性の差異は，m または nb が大きくなるに従い小さくなっているので，主として式 (34) の右辺第 2 項の影響によるものと考えられる．

表 2 に両手法における，計算時間，反復回数，速

表 2 計算時間，反復回数，速度向上

Table 2 Computation time, iteration, speed-up.

(a) 逐次型 ICCG 法（1CPU）の実行結果

計算時間 (sec)	反復回数
1135.2	944

(b) m 色順序付け法による実行結果

MC (m)	プロセッサ数	計算時間 (sec)	反復回数	速度向上
2	2	928.2	1638	1.22
2	8	230.1	1638	4.93
2	16	153.3	1638	7.40
8	2	1903.3	1200	0.60
8	8	463.9	1200	2.44
8	16	226.0	1205	5.02
32	2	1411.3	1072	0.80
32	8	328.0	1067	3.46
32	16	163.3	1070	6.95

(c) ブロック化赤－黒順序付け（$nb \times nb$）法による実行結果

BRB ($nb \times nb$)	プロセッサ数	計算時間 (sec)	反復回数	速度向上
8×8	2	890.8	1039	1.27
8×8	8	226.6	1039	5.01
8×8	16	115.8	1039	9.80
32×32	2	760.5	997	1.49
32×32	8	167.4	994	6.78
32×32	16	85.4	997	13.29
64×64	2	662.0	985	1.71
64×64	8	158.3	990	7.17
64×64	16	79.5	992	14.3

(d) Localizd ICCG 法による実行結果

プロセッサ数	計算時間 (sec)	反復回数	速度向上
2	684.4	1219	1.66
8	176.6	1267	6.43
16	88.15	1273	12.9

表 3 1 回の ICCG 反復の計算時間（ミリ秒）
Table 3 Computation time of one ICCG iteration (msec).

逐次型 ICCG	MC (4)	MC (6)	MC (8)	MC (16)
1180	1807	2493	3213	3179

BRB (8×8)	BRB (32×32)	BRB (128×128)
1659	1349	1252

度向上を示す．速度向上については，辞書式順序付けに基づく逐次型 ICCG 法の計算時間に対する比率で表す．表 2 において全体的にブロック化赤－黒順序付け法の方が高い速度向上を得ていることが分かる．多色順序付け法では，色数を大きくした場合，反復回数が減少しているものの速度向上に大きな改善が見られない．この原因としては，同期コストとともにキャッシュデータの再利用性がストライドアクセスにより低下したためと考えられる．そこでキャッシュの影響を調べるために，**表 3** に 1 回の ICCG 反復の計算時間を示す．ここで，表 3 における各計算時間は，すべて 1CPU での逐次実行において計測されたものである．表 3 より，多色順序付け法において色数を増加させると 1 反復あたりの計算時間が増加することが分かる．また，実行計算機のキャッシュラインサイズが 64B（倍精度実数 8 要素分）であるために，8 色以上の色数を用いた場合（ストライドが 8 以上となる）キャッシュデータを効率良く利用することができず，逐次型 ICCG 法に比べて 1 反復あたりの計算時間が約 2.7 倍に増加している．その結果，表 2 (b) に見られるように，色数を増加させることにより収束性が改善されても，十分な速度向上が得られない．

一方，表 2 (c) において，ブロック化赤－黒順序付け法では，ブロックサイズを増加させるに従い反復回数が減少し，その効果による速度向上の改善が見られる．また，同手法では，表 3 に見られるように，ブロックサイズの増加により 1 反復あたりの計算時間が減少している．これは，ブロックサイズを大きくすることにより，一次元配列上で連続的にアクセスする部分が増え，キャッシュデータの再利用性が向上したためと考えられる．したがって，ブロック化赤－黒順序付け法では，ブロックサイズを大きくすることにより，収束性の改善とキャッシュデータの再利用性の向上の両面が期待できる．

次に，ブロック化赤－黒順序付け法における並列処理による速度向上について検討する．ここでは，比較の対象としてスカラ並列計算機向きとされる Localized ICCG 法[9),10)]（Block ICCG 法とも呼ばれる）を加える．これは，多色順序付け法は元来ベクトル計算機向けの手法であり，上記で述べたようにスカラ計算機上での実装に向かない点があるためである．Localized ICCG 法による計算結果を表 2 (d) に示し，同手法と比較したブロック化赤－黒順序付け法の速度向上を**図 10** に示す．図 10 において，ブロックサイズを 32×32，64×64 とした場合には，ブロック化赤－黒順序付け法の方が Localized ICCG 法と比べて高い速度向上を得ている．また，プロセッサ数を増加した場合，Localized ICCG 法では，収束性は維持されず一般に悪化するのに対し，ブロック化赤－黒順序付け法では，決まったブロックサイズに対して収束性は維持される．たとえば，本数値実験では，式 (31) より，ブロックサイズ 64×64 の場合に得られる並列度は 128 で，128 CPU までは収束性が維持される．したがって，プロセッサ数が 16 以上と，さらに増加していった場合，ブロック化赤－黒順序付け法の Localized ICCG 法に対する優位性はさらに高まると考えられる．次に，図 10 中で，2CPU，8CPU 時において式 (37)（$n = 1024 \times 1024$ として計算）によって得られる最適なブロックサイズを用いた場合の結果を △ 印により示す．図 10 において，最適なブロックサイズを用いることにより，与えられたプロセッサ数に対して最も高い速度向上が得られていることが分かる．

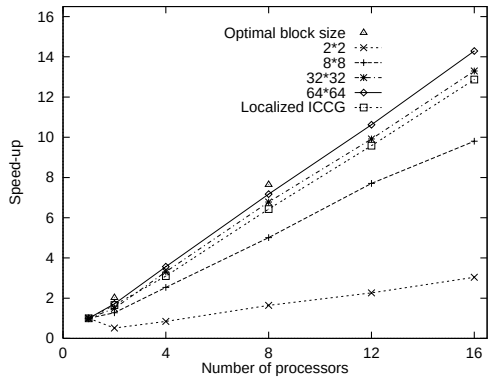

図 10 ブロック化赤－黒順序付け法による速度向上
Fig. 10 Speed-up in block red-black ordering method.

6. おわりに

本論文では，ICCG 法の並列化手法の 1 つである並列オーダリングの利用に関して，代入計算中に同期点の少ないブロック化赤－黒順序付け法を提案した．同手法の利点として，以下の点があげられる．

- 各代入計算中の同期点が 1 個だけである．

- ブロック内の節点数を増加させることより収束性を改善することが可能であり，大きな色数による多色順序付け法と同程度の収束性を得ることができる．
- 最適なブロックサイズの選択が容易である．すなわち，実行環境の並列度に合わせてなるべく大きなブロックを用いれば，その並列度に対して最良の収束性が期待できる．

これらの利点について，オーダリンググラフに基づいた解析的検討，数値計算による検証を行い，同手法の有効性を確かめた．また，スカラ並列計算機上での数値計算において，ブロックサイズを大きくすることにより，収束性の改善に加えて，キャッシュデータの再利用性の向上が得られることを示した．

参 考 文 献

1) Meijerink, J. and van der Vorst, H.A.: An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix Is a Symmetric M-matrix, *Mathematics of Computation*, Vol.31, pp.148–162 (1977).
2) van der Vorst, H.A. and Chan, T.F.: Parallel Preconditioning for Sparse Linear Equations, *ZAMM. Z. angew. Math. Mech.*, Vol.76, pp.167–170 (1996).
3) Duff, I.S. and Meurant, G.A.: The Effect of Ordering on Preconditioned Conjugate Gradients, *BIT*, Vol.29, pp.635–657 (1989).
4) Doi, S. and Lichnewsky, A.: A Graph-Theory Approach for Analyzing the Effects of Ordering on ILU Preconditioning, INRIA Report 1452 (1991).
5) Doi, S. and Washio, T.: Ordering Strategies and Related Techniques to Overcome the Trade-off Between Parallelism and Convergence in Incomplete Factorization, *Parallel Computing*, Vol.25, pp.1995–2014 (1999).
6) Iwashita, T. and Shimasaki, M.: Construction and Ordering of Edge Elements for Parallel Computation, *IEEE Trans. Magnetics*, Vol.37, pp.3498–3502 (2001).
7) Duff, I.S. and van der Vorst, H.A.: Developments and Trend in the Parallel Solution of Linear Systems, *Parallel Computing*, Vol.25, pp.1931–1970 (1999).
8) Washio, T. and Hayami, K.: Overlapped Multicolor MILU Preconditioning, *SIAM Journal of Scientific Computing*, Vol.16, pp.636–650 (1995).
9) Dongarra, J.J., Duff, I.S., Sorensen, D.C. and van der Vorst, H.A.: *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM (1991).
10) Iwashita, T. and Shimasaki, M.: Parallel Processing of 3-D Eddy Current Analysis with Moving Conductor Using Parallelized ICCG Solver with Renumbering Process, *IEEE Trans. Magnetics*, Vol.36, pp.1504–1509 (2000).
11) Barrett, R., et al.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, *SIAM* (1994).
12) 襲田 勉，丸山訓英，鷲尾 巧，土肥 俊，山田 進：非構造メッシュ用 BILU 前処理付き反復法のベクトル・並列化手法，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.41, No.SIG 8(HPS 2), pp.92–99 (2000).
13) 小国 力（編著），村田健郎，三好俊郎，ドンガラ J.J.，長谷川秀彦著：行列計算ソフトウェア WS，スーパーコン，並列計算機，丸善 (1991).
14) 岩下武史，島崎眞昭：多色順序付けを用いた並列化 ICCG ソルバに関する検討—ブロック化による性能向上と工学的応用，第 8 回「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ，pp.55–60 (2001).

付 録

A.1 前進代入計算のプログラム例

ブロック化赤－黒順序付け法における前進代入計算 $\boldsymbol{y} = \boldsymbol{D}^{-1}(\boldsymbol{r} - \boldsymbol{L}_s\boldsymbol{y})$ のプログラムを以下に示す．ただし，説明を容易にするために実際のプログラムコードに若干の変更を加えている．

```
myid: プロセッサのランク
ntr(myid): 各プロセッサの持つ赤ブロック数
ntb(myid): 各プロセッサの持つ黒ブロック数
rbs(myid, inrb, 1 or 2): 赤ブロックの左下点の
            x, y 格子座標 (0 ～ nx-1, 0 ～ ny-1)
rbe(myid, inrb, 1 or 2): 赤ブロックの右上点の x, y
                                    格子座標
bbs(myid, inrb, 1 or 2): 黒ブロックの左下点の x, y
                                    格子座標
bbe(myid, inrb, 1 or 2): 黒ブロックの右上点の x, y
                                    格子座標

------------------------------------------------
! 前進代入計算
! 赤ブロック

do inrb=0,ntr(myid)-1
  llx=rbs(myid,inrb,1)
  lly=rbs(myid,inrb,2)
  rux=rbe(myid,inrb,1)
  ruy=rbe(myid,inrb,2)

! 領域 I
    y(lly*nx+llx)=r(lly*nx+llx)/diag(lly*nx+llx)
```

```
! 領域 II
  do i=lly*nx+llx+1,lly*nx+rux
    y(i)=(r(i)-b(i-1)*y(i-1))/diag(i)
  enddo

! 領域 III
  do i=(lly+1)*nx+llx,ruy*nx+llx,nx
    y(i)=(r(i)-c(i-nx)*y(i-nx))/diag(i)
  enddo

! 領域 IV
  do iy=lly+1,ruy
    do ix=llx+1,rux
    i=iy*nx+ix
    y(i)=(r(i)-b(i-1)*y(i-1)-c(i-nx)*y(i-nx)) &
                                      /diag(i)
  enddo
 enddo

enddo

!$OMP BARRIER
! 黒ブロック

do inbb=0,ntb(myid)-1
  llx=bbs(myid,inbb,1)
  lly=bbs(myid,inbb,2)
  rux=bbe(myid,inbb,1)
  ruy=bbe(myid,inbb,2)

! 領域 V
  do iy=lly,ruy-1
    do ix=llx,rux-1
      i=iy*nx+ix
      y(i)=(r(i)-b(i-1)*y(i-1)-c(i-nx)*y(i-nx)) &
                                        /diag(i)
    enddo
  enddo

! 領域 VI
  do i=ruy*nx+llx,ruy*nx+rux-1
    y(i)=(r(i)-b(i-1)*y(i-1)-c(i-nx)*y(i-nx) &
                      -c(i)*y(i+nx))/diag(i)
  enddo

! 領域 VII
  do i=lly*nx+rux,ruy*nx+rux-nx,nx
    y(i)=(r(i)-b(i-1)*y(i-1)-c(i-nx)*y(i-nx) &
                      -b(i)*y(i+1))/diag(i)
  enddo

! 領域 VIII
  ii=ruy*nx+rux
  y(ii)=(r(ii)-b(ii-1)*y(ii-1)           &
        -c(ii-nx)*y(ii-nx)-b(ii)*y(ii+1) &
        -c(ii)*y(ii+nx))/diag(ii)

enddo
```

(平成 13 年 8 月 13 日受付)
(平成 14 年 2 月 13 日採録)

岩下　武史（正会員）

昭和 46 年生．平成 10 年京都大学大学院工学研究科電気工学専攻博士課程修了．平成 10 年より京都大学工学部電気工学科リサーチアソシエイト．平成 12 年より京都大学大型計算機センター助手，現在に至る．ハイパフォーマンスコンピューティング，特に並列化線形ソルバ，AMG 法，高速電磁界解析に関する研究に従事．京都大学博士（工学）．IEEE，電気学会，日本 AEM 学会各会員．

島崎　眞昭（正会員）

昭和 18 年生．昭和 46 年京都大学大学院工学研究科博士課程単位修得退学．昭和 46 年同大学工学部助手（情報工学科），同大学助教授を経て，平成元年九州大学教授（大型計算機センター），平成 9 年京都大学教授（工学研究科電気工学専攻），現在に至る．工学博士．スーパーコンピューティング，計算科学，計算機ソフトウェアの研究に従事．著書「スーパーコンピューティングとプログラミング」．電子情報通信学会，電気学会，日本応用数理学会，日本ソフトウェア科学会，ACM，IEEE，SIAM 各会員．