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

襲田 勉[†] 丸山訓英[†] 鷲尾巧[†]
土肥俊[†] 山田進[††]

共有メモリベクトル並列計算機の演算性能を最大限に引き出すような，ランダムスパース行列のための Block（ブロック）ILU 前処理付き反復法のベクトル・並列化手法を提案し，その手法を並列ベクトル型スーパコンピュータ SX-4（SRAM 版，1 CPU のピーク性能 2 Gflops）上で性能評価した結果を示す．ここでブロックとはある格子点上に定義された複数の未知数からなる集合とする．ベクトル・並列処理をすることが難しいとされる BILU 前処理の前進・後退代入演算のベクトル・並列化のために IDS-MJAD（InDependent Set Multiple Jagged Diagonal）形式を導入した．IDS-MJAD 形式の導入により CPU 間の同期回数を低減した代入演算の実装が可能になる．3 次元構造解析問題（GeoFEM Tiger V1.0）を用いた約 100 万自由度の評価例題を使った数値実験において，1 CPU で 1.0 Gflops，8 CPU で 6.8 Gflops を達成した．

Vectorization and Parallelization Technique of Block ILU Preconditioning for Unstructural Problems

Tsutomu Osoda,[†] Kunihide Maruyama,[†] Takumi Washio,[†]
Shun Doi[†] and Susumu Yamada[††]

In this paper, we propose techniques to extract vector and shared memory parallel performance of a parallel vector machine, and we evaluate the proposed techniques on an NEC super computer SX-4. As the linear solver, we have implemented the block ILU preconditioned iterative method, which is frequently used for many large sparse problems. Here, a block corresponds to unknowns on one node in a mesh or in a grid. As for the vectorization and parallelization technique, we propose IDS-MJAD (Independent set multiple jagged diagonal) format. With this technique, we can decrease the number of synchronizations. The numerical experimental results show that we achieved about 1.0 Gflops on 1 CPU and about 6.8 Gflops on 8 CPUs for some FEM problems.

1. はじめに

偏微分方程式を離散化したときに生成される連立一次方程式の解法として ILU（Incomplete LU）前処理付き反復法[2)]が知られている．ILU 前処理とは係数行列 A の LU 分解において，すべてまたは部分的に fill-in を発生させずに近似的な LU 分解を行い，これを前処理行列 M とするものである．また，本論文で使った BILU（Block ILU）前処理におけるブロックとは，物理空間内の節点に複数の変数が定義されている問題に対し，節点上に定義された複数の未知数をまとめて扱う単位のことで，そのときにまとめて計算される小ベクトルもしくは小ベクトルに対する係数行列の小行列を指している．特にここでは，非ゼロ小行列を密行列として取り扱い，ブロックサイズを一定とする．このような場合コーディングの際に，ブロックごと演算をループ内で陽に書き下すことにより，メモリアクセスの負荷を低減でき，高いベクトル演算性能を実現できる．

BILU 分解行列 M は以下のように表される．

$$M = (L + D)D^{-1}(D + U) \quad (1)$$

ここで，L，D，U はそれぞれ，ブロック単位の下三角，対角，上三角行列で次のようにブロック単位で表現されるものとする．n はブロックの総数を示す．

† NEC 情報通信メディア研究本部
Computer & Communication Media Research Laboratories, NEC Corporation
†† 日本原子力研究所計算科学推進センター
Center for Promotion of Computational Science and Engineering, Japan Atomic Energy Research Institute

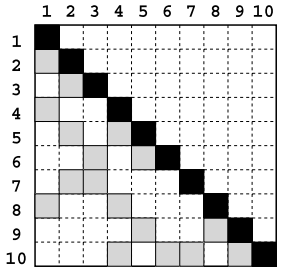

図 1 独立集合構成前の非ゼロブロックパターン
Fig. 1 A pattern of nonzero elements of original matrix.

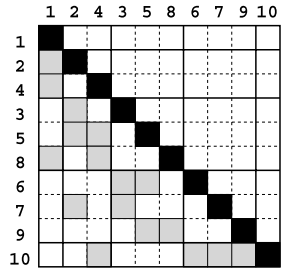

図 2 独立集合構成後の非ゼロブロックパターン
Fig. 2 The pattern of nonzero elements after the construction of the independent set.

$$L = (L_{i,j})_{1 \le i,j \le n}$$
$$U = (U_{i,j})_{1 \le i,j \le n}$$
$$D = \mathrm{diag}(D_{i,i})_{1 \le i \le n}$$

BILU 分解した後，反復計算においては $Mx = y$ という方程式を解く必要があるがそれは以下のようにして計算する．

$$p = D^{-1}(y - Lp)$$
$$x = p - D^{-1}Ux$$

前式の演算は前進代入と呼ばれ，後式の演算は後退代入と呼ばれる．これらの式をブロック単位で記述すると以下のようになる．

$$p_i = D_{i,i}^{-1}\left(y_i - \sum_{k=1}^{i-1} L_{i,k}p_k\right) \quad (i = 1, \cdots, n) \tag{2}$$

$$x_i = p_i - D_{i,i}^{-1}\sum_{k=i+1}^{n} U_{i,k}x_k \quad (i = n, \cdots, 1) \tag{3}$$

この式に示すように ILU 前処理の前進・後退代入演算はブロック間の参照関係が存在するための逐次性があり，このままではベクトル・並列化ができない．そこで，前処理演算のベクトル化を行うために，互いに参照関係にないブロックにより構成される独立集合[2]を構成する．独立集合内では前進代入計算は行列ベクトル積と同様な演算となるためベクトル化が可能となり[8]，行列ベクトル積をベクトル化するときの手法を適用することで BILU 前処理のベクトル化を実現している．

以下に独立集合の構成方法を説明する．**図 1** は非ゼロブロックを独立集合順に並べかえる前の係数行列を表している．係数行列に対し以下のアルゴリズムを適用することによって独立集合を構成する．

(1) 最初は，他のブロックを参照しなくても計算できるブロックを探し，1 番目の独立集合とする．図 1，**図 2** では，1 番目のブロックだけなので，1 番目のブロックを第 1 の独立集合とする．

(2) すでに選ばれたブロックのみを参照するブロックを選び，次の独立集合を構成する．1 番目のブロックのみを参照しているブロックは 2 と 4 なので 2 番目の独立集合として 2 と 4 を選ぶ．

(3) すべてのブロックが選ばれるまで，2 の操作を繰り返す．

(4) 最後に，同じ独立集合内のブロックが連続に並ぶように行と列を並べる．

このアルゴリズムは以下のように再帰的に定式化できる．$IDS(k)$ は k 番目の独立集合を示し，PB は行列 L，U の非ゼロブロック番号どうしを結んだ辺の集合を示している．

$$IDS(1) := \{I | I \le J \text{ for }, \forall (I,J) \text{ or } (J,I) \in PB\} \tag{4}$$

$$IDS(k) := \{I | I \le J \text{ for }, \forall (I,J) \text{ or } (J,I) \in PB, J \notin IDS(1) \cup \ldots \cup IDS(k-1)\} \tag{5}$$

図 2 は，図 1 の係数行列を独立集合の順に並べ替えを行ったときの非ゼロパターンを示している．太線が独立集合の区切り目に対応する．これから分かるように独立集合内の演算に依存関係はなく，式 (2)，(3) の演算はすでに演算が完了した独立集合のみを参照しているため，独立集合内における前進・後退代入演算は行列ベクトル積演算となり，独立集合内の演算にベクトル・並列化のための手法を適用することで，前処理全体をベクトル・並列化することができる．

次に行列ベクトル積をベクトル・並列化するためのデータ構造について説明をする．有限要素法によって離散化された係数行列を格納するデータ構造として CRS（Compressed Row Sparse）形式が知られている．しかしこのデータ構造を使うと行列ベクトル積演算においてループ長が短くなりベクトルマシン上では十分な演算性能を引き出すことができない．そこでそれを改良したデータ構造として JAD（Jagged Diago-

nal）形式[2]が提案されている．JAD 形式は非ゼロブロックの個数の多い順に行の並べ替えを行い，行列を列方向に格納し直すことにより，行列ベクトル積を以下のように実装する方法である．ここで

- MZ：1 行あたりの非ゼロブロックの個数の最大
- $PNTR$：JAD 形式の列方向セグメントの始まりを示すポインタ
- COL_INDX：JAD 形式の非ゼロブロックの列インデックス
- A：JAD 形式の係数行列
- x，y：ベクトル

とすると $y = Ax$ を計算する行列ベクトル積演算は以下のようになる．ただしここでは各ブロックのサイズが一定でその値が 3 であるとしている．

```
do i=1,MZ
   do j=PNTR(i),PNTR(i+1) − 1
      row=j − PNTR(i)+1
      col= COL_INDX(j)
      y(row,1)=y(row,1)+∑_{k=1}^{3} A(j,1,k) ∗ x(col,k)
      y(row,2)=y(row,2)+∑_{k=1}^{3} A(j,2,k) ∗ x(col,k)
      y(row,3)=y(row,3)+∑_{k=1}^{3} A(j,3,k) ∗ x(col,k)
   enddo
enddo
```

CRS 形式では 1 行あたりの非ゼロ成分の個数 MZ の長さしかループ長が確保されないのに対し，JAD 形式ではブロックの個数に相当する長さのループ長 $PNTR(i+1)-PNTR(i)$ が確保される．さらに，共有メモリベクトル並列化に関しては，以下のよう実装することも可能である．ただし PARDO という指示行によって，直後のループの処理を分割し各プロセッサに割り当てている．

```
do i=1,MZ
!PDIR PARDO FOR
   do j=PNTR(i),PNTR(i+1) − 1
      row=j − PNTR(i)+1
      col= COL_INDX(j)
      y(row,1)=y(row,1)+∑_{k=1}^{3} A(j,1,k) ∗ x(col,k)
      y(row,2)=y(row,2)+∑_{k=1}^{3} A(j,2,k) ∗ x(col,k)
      y(row,3)=y(row,3)+∑_{k=1}^{3} A(j,3,k) ∗ x(col,k)
   enddo
enddo
```

しかしこの実装方法では j の 1 つの値に対し 1 回の同期を必要とするので（**図 3**），CPU 間の同期のオーバヘッドが大きくなってしまう．行列ベクトル積全体では，(1 行あたりの非ゼロ成分数)（$= MZ$）回の同期が必要になる．

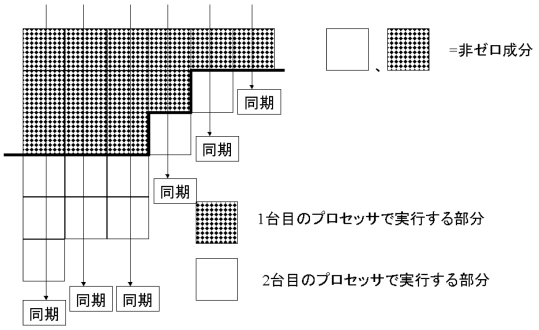

図 3 指示行を使うことによって JAD 形式の行列ベクトル積を並列化したとたとき必要となる同期のタイミングを示した図．横方向は行列の行方向を示し，縦方向は行列の列の方向を示している．各列の演算を終了した後でプロセッサ間の同期が必要となる．

Fig. 3 This figure shows the timinig of synchronization necessary for the execution of the matrix-vector multiplication with the JAD format. The horizontal direction coressponds to the row direction and the vertical direction coressponds to the column direction. Synchronization takes place by each column operation.

そこで，本論文においては行列ベクトル積演算における同期のオーバヘッドを低減させるために MJAD（Multiple JAD）形式を提案する．さらに BILU 前処理のベクトル・並列化のために，各独立集合ごとに MJAD 形式を適用した IDS-MJAD（InDependent Set Multiple JAD）形式を提案する．つまり，共有メモリベクトル並列計算機に行列ベクトル積と BILU 前処理演算を実装するためのデータ構造として，以下の 2 つのデータ構造を提案する．

- **行列ベクトル積演算** MJAD 形式（2.1 節）
- **前進・後退代入演算** IDS-MJAD 形式（2.2 節）

次に BILU 前処理演算において並列性を向上させるためのオーダリングついて説明をする．BILU 前処理付き反復法の収束性と並列性にはトレードオフの関係があり，規則格子の場合には種々のオーダリングにおいてトレードオフが確認されている[4)~7)]．収束性の悪化を極力おさえながら，十分なベクトル性能を引き出す多色マルチカラー法が提案されており[5),6)]，ここではその不規則格子への拡張を行う．一般の不規則格子に対する代表的なオーダリング方法として以下のものがある．

- 収束性：Reverse Cuthill-Mckee（RCM）オーダリング[1)]

  RCM オーダリングは以下のようなものである．

  (1) 任意に選んだ出発ノードから Cuthill-

Mckee オーダリングを構成し，その最後のノードを出発ノード v_0 とする．

(2) ノード v_0 から再び Cuthill-Mckee オーダリングを適用する．

RCM オーダリングでは v_0 はグラフの疑似的な端点と見なすことができ，端点から出発し距離の順に番号付けすることにより，隣接するノードの番号が近い番号になり，オーダリング後の行列のバンド幅を小さくすることができる．

- 並列性：マルチカラー（MC）オーダリング[1)]

MC オーダリングは以下のようなものである．

(1) 色数を最小にするという条件のもとで隣接ノードの色が異なるようにノードを塗り分ける．

(2) グラフを塗り分けた後，同色のノードに対応するブロックは連続な番号になるように並べかえる．

MC オーダリングでは前処理の前進後退代入において，同色のブロックにおける演算の間に依存関係がなくなり，最小の色数にすることによって各色に含まれるブロックの個数が増えループ長が長くなるため，ベクトル演算効率の良い前処理が実現できる．

MC オーダリングを使うと高いベクトル演算性能が得られ，マシンのレジスタ長と比較するときわめて長いループ長が確保される．そこで，適度なループ長を保ちながら色数を多くし，収束性を向上させることで計算時間を短縮させることが期待できる．本論文で使用する多色マルチカラー（MC2）オーダリングは，MC オーダリングのときの演算性能を十分に保ったまま，反復回数を低減することにより計算時間を短縮する方法である．

2. BILU 前処理付き反復法のベクトル・並列化方法

本章では，共有メモリベクトル並列マシン上でのベクトル化・並列化手法について説明する．ILU 前処理付き反復法をベクルト・並列化するとき，同期のオーバヘッドを低減させるデータ構造・収束性の低下をおさえながらベクトル演算性能を発揮させるオーダリングを使うことでマシンの演算性能を十分に引き出すことができる．本論文では以下のデータ構造・オーダリングを提案する．

- データ構造
  - 行列ベクトル積：MJAD 形式
  - 前処理：IDS-MJAD 形式

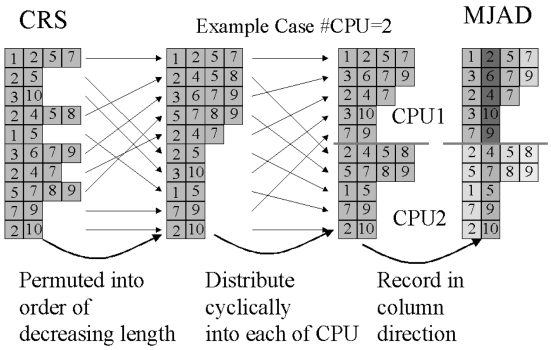

図 4 CRS 形式の行列から MJAD 形式の行列に変換するためのアルゴリズム．各ステップにおいて，横方向は行列の行方向を示し，縦方向は行列の列の方向を示している．非ゼロ成分数の大きなものから各プロセッサにサイクリックに割り当てていくことにより，行列ベクトル積の演算量が均等化される．

Fig. 4 The algorithm for the transformation from the CRS format to the MJAD fotmat. The horizontal direction corresponds to the row direction and the vertical direction corresponds to the column direction. The computational load is well-balanced among CPUs because rows are cyclically allocated to each CPU with descending ordering of the number of nonzero elements.

- オーダリング：MC2 オーダリング

2.1 MJAD（Multiple JAD）形式による並列化方法

Multiple JAD（MJAD）形式を使う並列化の方法は，単純な JAD 形式の並列化方法に比べ同期の回数を減らすことで計算時間を短縮する方法である．MJAD 形式への変換方法および，MJAD 形式を使った行列ベクトル積の演算方法について説明をする．**図 4** にプロセッサ台数が 2 台の場合の，CRS 形式の行列から MJAD 形式の行列に変換するためのアルゴリズムを模式図で示す．その方法は以下のようなアルゴリズムである．まず CRS 形式の行列が入力として与えられたとする．

(1) 各行の非ゼロブロックの個数が多い行から降順に行を並べ替える．

(2) 並べ替えた行を 1 番目の行から順に各プロセッサにサイクリックに割り当てながら行を並べ替える．

(3) 各プロセッサに割り当てられた行列に JAD 形式を適用する．

以上のようなアルゴリズムにより，CRS 形式から MJAD 形式への変換をすることができる．第 1，2 ステップによって計算負荷は各 CPU に均等になるように割り当てられる．

MJAD 形式を使った行列ベクトル積の演算は以下

のようなアルゴリズムによって実行される．ここで P は CPU 台数であるとする．

```
!PDIR PARDO FOR
do k=1,P
   do i=MZ(k),MZ(k+1)-1
      do j=PNTR(i),PNTR(i+1) − 1
         row=PNTR2(k) + j− PNTR(i)
         col= COL_INDX(j)
         y(row,1)
            =y(row,1)+∑_{k=1}^{3} A(j,1,k) ∗ x(col,k)
         y(row,2)
            =y(row,2)+∑_{k=1}^{3} A(j,2,k) ∗ x(col,k)
         y(row,3)
            =y(row,3)+∑_{k=1}^{3} A(j,3,k) ∗ x(col,k)
      enddo
   enddo
enddo
```

ここで

- PNTR2：各 CPU の先頭の行を記憶するための配列
- MZ(k+1)-MZ(k)：k 番目の CPU に割り当てられた，1 行の非ゼロブロックの個数の最大

である．このアルゴリズムを JAD 形式を単純に並列化した場合のアルゴリズムと比較すると，最外側のループで並列化がなされており，粒度が増大していることが分かる．この実装方法では最外側のループが並列化されているので（**図 5**），1 回の行列ベクトル積演算に対し最後に 1 回の同期しか必要としない．

2.2 IDS-MJAD（Independent Set Multiple JAD）形式による前処理演算の並列化方法

IDS-MJAD 形式は 1 章に示した独立集合 $IDS(1), \cdots, IDS(k)$ それぞれに 2.1 節の MJAD 形式を適用することによって構成するデータ構造である．また CRS 形式から IDS-MJAD 形式への変換方法は以下のようになる．

- 独立集合の順番になるように行列の行と列を入れ替える．
- 行列を行方向に独立集合ごとに区切り，各独立集合上の部分行列に MJAD 形式を適用する．

2.2.1 多色マルチカラー（MC2）オーダリング

RCM オーダリングの持つ高収束性を保ちつつ，ベクトル・並列演算性能を高めるため，MC2 オーダリングは以下のように構成する．使用する色数を n_{color} とする．

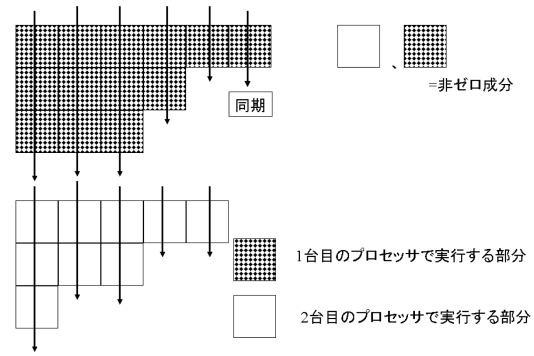

図 5 MJAD 形式を構成することによって並列化を行った行列ベクトル積に必要な同期のタイミングを示した図．横方向は行列の行方向を示し，縦方向は行列の列の方向を示している．すべての列を計算し終えた後でプロセッサ間の同期を 1 度だけとる必要がある．

Fig. 5 This figure shows the timinig of synchronization necessary for the execution of the matrix-vector multiplication with the MJAD format. The horizontal direction coressponds to row direction and the vertical direction coressponds to the column direction of matrix. Synchronization takes place once after all column operations.

(1) RCM オーダリングを作成する．

(2) RCM オーダリングのもとで独立集合を作成する．作成された独立集合を $IDS(1), IDS(2), \cdots$ とする．

(3) $IDS(1), IDS(1 + n_{color}), IDS(1 + 2n_{color}), \cdots$，に 1 番目の色を塗り（カラー集合 $C(1)$ の作成），$IDS(2), IDS(2 + n_{color}), IDS(2 + 2n_{color}), \cdots$，に 2 番目の色を塗り（カラー集合 $C(2)$ の作成），というように独立集合を n_{color} 色の色で塗り分ける．

この構成方法から分かるように，色数を増大させるほど MC2 オーダリングは収束性が良いとされている RCM オーダリングに近付くことが分かる．

2.2.2 多色マルチカラー（MC2）オーダリングにおける色数と並列性能の関係

共有メモリ並列ベクトル計算機において多色マルチカラー（MC2）オーダリングを用いて最短の計算時間で計算をするときには，最適な色数を選択する必要がある．そのときにベクトル・並列性能を発揮するのに必要な色数の見積りをこの節で行う．以下のようにパラメータを定義をする．

- N：未知数の個数,
- n_{color}：色数
- P：プロセッサ台数
- X（Mflops）：ループ長が十分に長いときの演算の実効性能

- S（μsec）：同期に必要な時間
- M：1 行あたりの平均非ゼロ成分の個数
- L：1 CPU あたりのループ長
- V：マシンのベクトルレジスタ長

共有メモリ並列ベクトル計算機の演算性能を十分に発揮させるためには十分にベクトル演算性能を発揮させ，そのうえで演算時間に比べ同期時間を十分に小さくさせる必要がある．

ベクトル演算性能を十分に発揮させるためには，ループ長をマシンのベクトルレジスタ長よりも長くなるようにする必要がある．仮に独立集合に含まれる未知数の個数は等しいと仮定し，1 行あたりの非ゼロ成分の個数も等しいと仮定すると，ループ長は式 (6) のようになる．

$$L = \frac{N}{n_{color} \times P} \tag{6}$$

ここでベクトル演算性能を十分に発揮させるためは，ループ長がベクトルレジスタ長の k 倍必要であるとすると，色数は式 (7) の条件を満たすように決める必要がある．

$$n_{color} < \frac{N}{kVP} \tag{7}$$

また演算時間に比べ同期時間を十分に小さくするために，演算時間は同期時間の 10 倍以上必要であると仮定する．非ゼロ成分 1 つあたりの演算個数は 2 回のため，同期のオーバヘッドを小さくするためには，式 (8) を満たさなければならない．

$$\frac{2LM}{X \times 10^6} > 10S \times 10^{-6} \tag{8}$$

ゆえに

$$n_{color} < \frac{MN}{5SXP}$$

である．つまり色数を上記式よりも少ない色数にしないと同期オーバヘッドによる性能低下が顕著になる可能性がある．

まとめると

$$n_{color} < \min\left(\frac{N}{kVP}, \frac{MN}{5SXP}\right)$$

を満たすような最大の色数 n_{color} を決めることで，共有メモリ並列ベクトル計算機の十分に発揮させたまま計算時間を短縮にすることができる．

3. 数値実験

この章ではベクトル・並列化の効果を数値実験により調べた結果を示す．以下のような環境のもとで数値実験を行った．

- 評価例題：3 次元構造解析（GeoFEM/Tiger（RIST）[3]，規則構造メッシュ，1 行あたり約 27 個の非ゼロブロック，1 節点上に 3 つの未知数）
- 使用マシン：SX-4（Peak：2 Gflops，6 GByte，ベクトルレジスタ長 256），測定には共用時間を利用．反復法のみの計算時間を測定．
- 基本反復法：共役勾配法
- 収束判定条件：$\frac{||r||_2}{||b||_2} < 10^{-8}$
- MC2 オーダリングの色数：2.2.2 項の式で $k = 2$ として計算された色数

3.1 IDS-MJAD 形式による並列演算性能の向上

本論文で提案した MJAD，IDS-MJAD 形式を採用したアルゴリズムが従来の単純に JAD 形式を並列化した場合に比べて高い演算性能を持っているということを示す．この節では以下の条件のもとで行った．

- オーダリング：MC オーダリング
- 未知数の個数：375000
- ブロックサイズ：1（1 行あたりの非ゼロ成分数 = 80）

その数値実験の結果を**図 6** に示す．図 6 から以下のことを読み取ることができる．

- JAD 形式を指示行によって並列化した場合には CPU 台数の増加とともに演算時間が増大している．8 CPU のとき並列効率は 4%である．
- IDS-MJAD 形式を使って並列化した場合には CPU 台数の増加とともに計算時間が短縮されている．8 CPU のとき並列効率は 63%である．

この原因として，反復法を実行するのに必要な同期のオーバヘッドの大小にある．1 行あたり約 80 個の非ゼロ成分数が存在するため，JAD 形式では前処

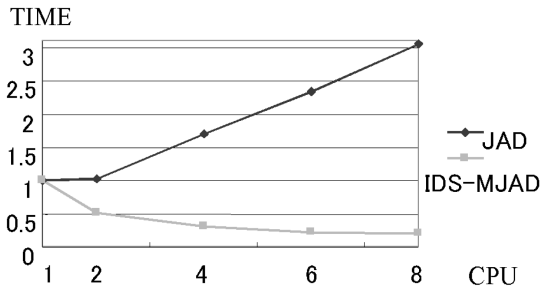

図 6 計算に使用する CPU 台数を変えたときの演算時間を JAD 形式と IDS-MJAD 形式で比較した図．横軸は CPU 台数，縦軸は 1 CPU での計算時間を 1 としたときの計算時間を示している．

Fig. 6 This figure shows the total computational time of the iterative method with upto 8 CPUs case. The horizontal line represents the number of CPUs. The vertical line represents the computational time over the computaional time of 1 CPU case.

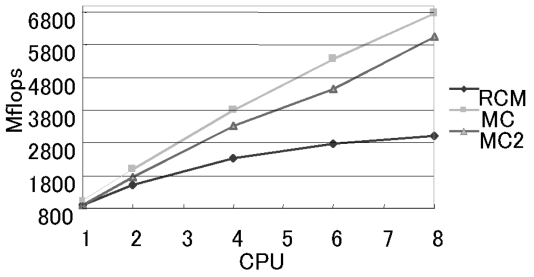

図 7 使用 CPU 数を変えたときの，SX4 における BILU 前処理付き反復法の演算性能を表したグラフ．横軸は CPU 台数，縦軸は演算性能を表している．3 本の線は上から順に MC オーダリング，MC2 オーダリング，RCM オーダリングである．

Fig. 7 This figure shows the MFLOPS of the BILU-BICGSTAB method with upto 8 CPUs case. The horizontal line represents the number of CPUs. The vertical line represents the MFLOPS. Three lines represents the MFLOPS with the MC ordering, the MC2 ordering, and the RCM ordering from the top to the bottom.

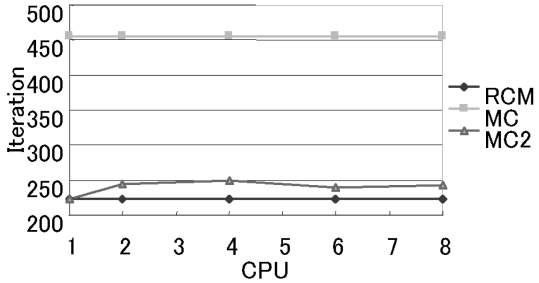

図 8 使用 CPU 数を変えたときの，BILU 前処理付き反復法の反復回数を表したグラフ．横軸は CPU 台数，縦軸は反復回数を表している．3 本の線は上から順に MC オーダリング，MC2 オーダリング，RCM オーダリングである．

Fig. 8 This figure shows iteration counts of the BILU-BICGSTAB method with upto 8 CPUs case. The horizontal line represents the number of CPU. The vertical line represents iteration counts. Three lines represents the iteration counts with the MC ordering, the MC2 ordering, and the RCM ordering from the top to the bottom.

理演算の同期のオーバヘッドが大きくなるのに対し，IDS-MJAD 形式ではその分の同期のオーバヘッドがなくなる．SX-4 においては同期に数 μsec 要し，JAD 形式を並列化した場合には全計算時間に占める同期時間の割合が大きくなり，演算性能が低下してしまう．CPU 台数を多くするほど演算時間は短くなるため同期時間の割合は大きくなり，CPU 台数を多くするほど同期のオーバヘッドが顕在化してくる．

3.2 オーダリングと並列演算性能

BILU 前処理付き反復法をベクトル並列計算機に効率良く実装するために工夫したオーダリングが，並列演算性能・演算時間に与える影響を調べる．数値実験は以下の条件のもとで行った．

- データ構造：IDS-MJAD 形式
- 未知数の個数：約 102 万
- ブロックサイズ：3

まず並列演算性能に関する数値実験の結果を**図 7** に示す．図 7 から以下のことを読み取ることができる．

- RCM：CPU 台数の増大とともに Mflops 値も増大するが，演算性能の向上率は徐々に低下している．8 CPU で約 3 Gflops である．
- MC：CPU 台数の増大とともに Mflops 値も増大し，8 CPU で約 6.8 Gflops である．
- MC2：CPU 台数の増大とともに Mflops 値も増大し，8 CPU で約 6.0 Gflops である．

この原因として各オーダリングの前処理における平均ループ長が大きく関係している．問題サイズを固定して CPU 台数を増やした場合，1 CPU あたりの平均ループ長は CPU 台数に反比例して短くなる．そのため，比較的平均ループ長の短い RCM オーダリングでは，1 CPU の場合にたかだかマシンのレジスタ長の数倍のループ長しか確保されていないため，CPU 台数を増やしたときの平均ループ長短縮による演算性能の低下が顕著になり，使用する CPU 台数を増やすにつれて性能向上が得られなくなる．逆に MC オーダリングでは，1 CPU の場合にマシンのレジスタ長の数万倍のループ長が確保されており，CPU 台数を増やしてもベクトル演算性能の低下が起こらないため，顕著な性能向上の低下は見られない．MC2 ではループ長がレジスタ長の 2 倍の長さになるように色数を決めているため，MC と同様顕著な性能向上の低下は見られない．

一方，オーダリングが BILU 前処理付き反復法の反復回数に与える影響を調べた数値実験の結果を**図 8** に示す．図 8 から以下のことを読み取ることができる．

- MC2：CPU 台数の増大とともに反復回数も増大する傾向にある．

MC2 オーダリングのとき色数は，レジスタ長の 2 倍程度の長さのループ長になるように決めている．そのため，CPU 台数が多いとき色数を少なくする必要があり，逆に少ないときには色数を多くする必要がある．問題サイズを固定した場合，MC2 オーダリングのもとでは，収束までに必要な反復回数は，使用する CPU 台数を少なくするほど RCM での反復回数に近くなり，使用する CPU 台数を多くするほど多くなる．

オーダリングが BILU 前処理付き反復法の計算時間に与える影響を**図 9** に示す．図 9 から以下のことを読み取ることができる．

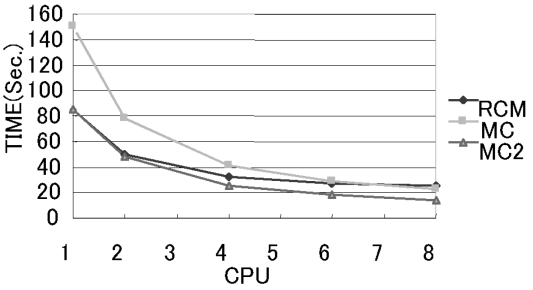

図 9 使用 CPU 数を変えたときの，SX4 における BILU 前処理付き反復法の計算時間を表したグラフ．横軸は CPU 台数，縦軸は計算時間を表している．グラフの右端において 3 本の線は上から順に RCM オーダリング，MC オーダリング，MC2 オーダリングである．

Fig. 9 This figure shows the computational time of the BILU-BICGSTAB method with upto 8 CPUs case. The horizontal line represents the number of CPU. The vertical line represents the total computational time. Three lines represents the computational time with the RCM ordering, the MC ordering, and the MC2 ordering from the top to the bottom.

- RCM オーダリングと MC オーダリングの優劣は問題サイズによって変わってくる．比較的少ない CPU 台数においては RCM オーダリングの方が計算時間が短く，多い CPU 台数においては MC オーダリングの方が計算時間が短くなる．
- MC2 オーダリングは RCM，MC オーダリングよりもつねに計算時間が短くなっている．

先の 2 つの数値実験から，以下のことが分かっている．

- MC：つねに十分な演算性能は発揮されるものの反復回数が多い．
- RCM：反復回数は少ないものの，CPU 台数を増やしたときには十分なベクトル演算性能が発揮されない．

つまり，演算性能と反復回数のトレードオフによって最適なオーダリングは決まるものの，計算前にはどちらのオーダリングが最適なのかを決められない．ところが MC2 オーダリングはつねに RCM，MC オーダリングよりも短い計算時間となり，最適なオーダリングであると結論付けできる．

4. ま と め

本論文では，共有メモリベクトル並列計算機において大規模ランダムスパース行列を係数行列に持つ連立一次方程式を効率良く解くための BILU 前処理付き反復法のベクトル・並列化手法を報告した．ベクトル・並列化するときに課題となるのは ILU 前処理における前進・後退代入演算で，我々は同期のオーバヘッドを低減させるために

- M-JAD，IDS-MJAD という新しいデータ構造の導入
- 多色マルチカラーオーダリングの採用

を行った．その有効性を数値実験によって検証し．約 100 万自由度の例題を使って，SX-4 の 1 CPU で約 1 Gflops，8 CPU で約 6.8 Gflops を達成した．

参 考 文 献

1) Duff, I. and Meurant, G.: The Effect of Ordering on Preconditioned Conjugate Gradients, *BIT 29*, pp.635–657 (1989).
2) Saad, Y.: *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company (1996).
3)（財）高度情報科学技術研究機構, http://www.rist.or.jp/home2.htm.
4) Doi, S.: On Parallelism and Convergence of Incomplete LU Factorization, *Appl. Numer. Math.*, 7, pp.417–436 (1991).
5) Doi, S. and Hoshi, A.: Large-numbered Multicolor MILU Preconditioning on SX-3/14, *Int. J. Comput. Math.*, 44, pp.143–152 (1992).
6) Fujino, S. and Doi, S.: Optimizing Multicolor ICCG Methods on Some Vector Computers, *Proc. IMACS International Symposium on Iterative Methods in Linear Algebra*, Beauwens, R. (Ed.) (1991).
7) Doi, S. and Washio, T.: Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations, *Parallel Computing*, 25, pp.1995–2014 (1999).
8) 丸山訓英，鷲尾 巧，土肥 俊：非構造メッシュ用 Block ILU 前処理付き反復法のベクトル化手法，HPC 研究会論文集，pp.73–78 (Dec. 1999).

(平成 12 年 4 月 26 日受付)
(平成 12 年 9 月 5 日採録)

襲田 勉

平成 7 年東京大学大学院理学研究科情報科学専攻修士課程修了．同年日本電気株式会社入社．大型計算機上でのライブラリの研究開発等に従事．計算工学会会員．

丸山　訓英

1998 年北海道大学大学院理学研究科数学専攻修士課程修了．同年日本電気（株）入社．

鷲尾　　巧（正会員）

昭和 39 年生．平成元年大阪大学大学院理学研究科数学専攻修士課程修了．同年日本電気株式会社入社．コンピュータ技術本部にてコンピュータ HW の開発に従事．平成 3 年より C&C 研究所にて，大規模疎行列連立一次方程式の高速並列解法の研究に従事．平成 6 年より NEC ヨーロッパ C&C 研究所にて，マルチグリッド法の研究に従事．平成 11 年より C&C メディア研究所にて，地球シミュレータ用ライブラリの開発に従事．日本応用数学会会員．

土肥　　俊（正会員）

昭和 59 年北海道大学大学院工学研究科精密工学専攻博士課程修了．工学博士．同年日本電気（株）入社．現在 NEC 情報通信メディア研究本部勤務．日本計算工学会，可視化情報学会，日本応用数理学会各会員．

山田　　進（正会員）

昭和 45 年生．平成 10 年東北大学大学院情報科学研究科博士課程修了．同年東北大学大学院工学研究科寄付講座教員．平成 11 年より日本原子力研究所博士研究員．平成 12 年より日本原子力研究所研究員．常微分方程式の数値解法および並列計算に関する研究に従事．博士（情報科学）．日本応用数理学会，計算工学会各会員．