分散メモリ型並列計算機での対称行列のデータ構造と多重スカイライン法への応用

寒川 光†

分散メモリ型の並列計算機のための密行列のデータ構造として，2 次元ブロックサイクリック分割・分散が定着している．この方法は一般行列にはうまく機能し，自動並列化も可能である．しかし対称行列の場合は，上または下三角部分の要素だけを扱うため，この一般行列の場合の利点の大部分が失われる．分散メモリ型の並列計算機では，対称行列のデータ構造として，正方行列の記憶域を確保し，その半分だけを使用する方法が普及している．本論文では，三角行列の記憶域だけを確保すれば計算可能な圧縮法を提案する．これには CC 型と CB 型の 2 通りの配置方式がある．前半で密行列について，典型的な行列計算である行列ベクトル積と三角分解を例に，データ構造による比較を行い，正方行列の記憶域を確保する方法から圧縮法への変換が比較的簡単に行えることを示す．後半では多重スカイライン法による疎行列の三角分解の最終段階に現れる準密行列に対して，2 章の方法に変更を加えて，ゼロ演算を回避することで性能向上を達成した例について報告する．これらの結果から，対称行列に CB 型圧縮法を用いることの有効性を示す．

A Data Structure for Symmetric Matrices and Its Application for Multiple Skyline Method on Distributed Memory Parallel Computers

Hikaru Samukawa†

Two-dimensional array decomposition/distribution is widely used for dense matrices on distributed memory parallel computers. This approach works well for general-matrices, which leads compilers to automatic parallelization. However in case that only upper or lower triangular elements of symmetric matrix are treated, the most of advantages of this method is lost. For distributed memory parallel computers, a method of allocating region of a square global matrix and using half of its region is commonly used. In this paper we propose a compaction method which enables computing with allocating triangular matrix region only. There are two ways of elements assignment, i.e. CC type and CB type. In the former part of this paper, we show a migration path from square matrix method to compaction method based on comparison among data structures through examples of typical dense-matrix operations, that is, matrix-vector product and triangular-factorization. In the latter part, we describe a factorization of semi-dense matrix which appears in the final part of sparse matrix factorization by multiple skyline method, in which our CB type method described in the former chapter can efficiently be applied with small modifications and achieved performance enhancement. Through these experiences, we show effectiveness of the compaction method with CB type for symmetric matrices.

1. はじめに

分散メモリ型並列計算機で密行列を扱う場合，2 次元ブロックサイクリック分割・分散が基本となる．この方法は，行列を適当なサイズのブロック（小行列）に分割し，これらを 2 次元のグリッド状に配置されたプロセッサ上に，次元ごとにサイクリック分散させる．この方法の利点は，密行列に関する行列ベクトル積から三角分解などにいたる幅広い線形代数計算の並列化で，通信量の削減と計算の高速化を両立できるところにある．この分割・分散方式の特長は，複数の小行列を連結して 1 つの大きな行列として扱うことを可能にする点にある．こうするとコンパイラによる自動並列化にも道が開かれる．このため High-Performance Fortran（HPF）や ScaLAPACK などの数値計算ライブラリでも用いられ，標準的なデータ構造と考えられる[1)]．

しかし対称行列では事情は異なる．対称性を活かして上または下三角行列だけを扱うと，小行列 A_{IJ}

† 日本アイ・ビー・エム株式会社東京基礎研究所
Tokyo Research Laboratory, IBM Research

に A_{JI} の役割も担わせるため，通信が増加する．また，ノードプログラムで小行列の配置の規則性が失われ，計算方法が複雑化する．ScaLAPACK では，記憶域に対しては正方行列全体を確保し，ここに上または下三角行列のみを格納する方法をとっている．この ScaLAPACK のデータ構造に合わせるライブラリも少なくない（たとえば文献 2)）．

本論文では，対称行列の上三角部分だけの記憶域を確保すれば計算可能なデータ構造を提案し，それに対応するアルゴリズムを示す．逐次計算では Fortran プログラムで対称行列を扱う場合，2 次元配列を用いて，その半分だけを使用する方法と，1 次元配列を用いて a_{ij} 要素を $j(j-1)/2+i$ 番目に圧縮して格納する方法とが用いられる．ScaLAPACK の方法は前者に対する並列化，提案方法は後者に対する並列化と考えられる．そこで両者をそれぞれ **ScaLAPACK 法**，**圧縮法**と呼ぶことにする．

本論文ではデータ構造として，行列を小行列に分割する方式，小行列を複数のノードへ分散する方式，各ノードの記憶域上での行列要素を配置する方式の 3 つに着目する．圧縮法の分割・分散方式は 2 次元ブロックサイクリックであるが，配置方式は CC 型と CB 型の 2 通りをとりうる（CC 型 CB 型については後述する）．密な対称行列に対しては両者に大きな差異はないが，CB 型は密だがまとまったゼロ要素が存在する準密行列に対しても対応しやすい柔軟性を備える．

以下 2 章で，行列ベクトル積と三角分解アルゴリズムを対象に，2 次元ブロックサイクリック分割・分散を対称行列に適用した場合と一般行列に適用した場合との違いを述べたうえで，圧縮法でのアルゴリズムを示す．3 章で，多重スカイライン法による対称疎行列に対する三角分解の最終段階で現れる準密行列の分解について述べる．なお対称行列は便宜上，上三角行列を対象として記述する．

2. 圧縮法による対称密行列演算

行列ベクトル積と三角分解を例に，一般行列と対称行列の計算方法の違いを明らかにしたうえで，圧縮法と ScaLAPACK 法の違いがプログラミングのどの部分に現れるかを示す．なおプログラミング言語は Fortran，メッセージ交換は MPI を用いる[3]．

2.1 一般行列の 2 次元ブロックサイクリック分割・分散

対称行列の場合との比較のために，サイズ $n\times n$ の一般行列 A を $b\times b$ の小行列に分割する．

$$A \rightarrow \{A_{IJ}\}_{I=0:(n-1)/b,\ J=0:(n-1)/b}$$

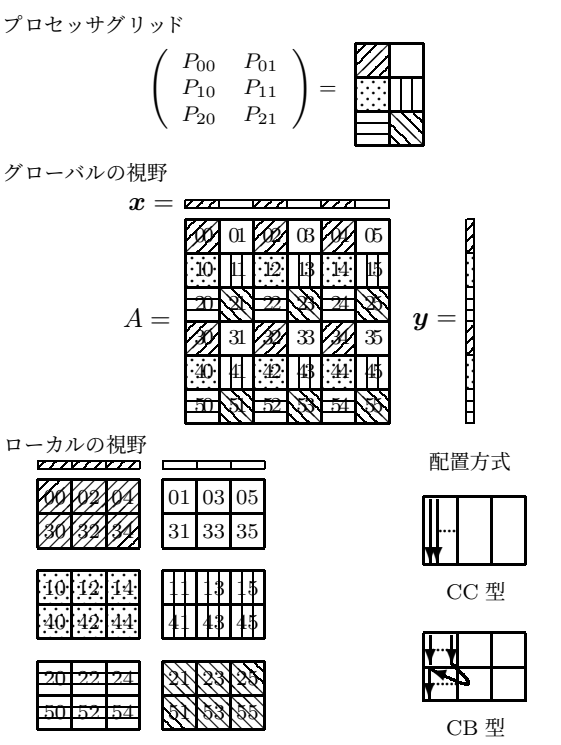


図 1 2 次元配列分割と行列要素の配置
Fig. 1 2-D array decomposition/distribution and elements assignment.

なお，A_{IJ} などの大文字の添字は小行列の添字で，小行列数が N のとき 0 から $N-1$ を使用する．a_{ij} など小文字の添字は行列要素の添字を表し，1 から使用する．

2 次元ブロックサイクリック分散は，プロセッサの配置も $p\times q$ の 2 次元グリッド状とし，各小行列を行方向，列方向ごとにサイクリック分散する．**図 1** に行列を 6×6 に分割して，3×2 のプロセッサグリッド上に分散した状態を示した．ノードプロセッサの番号も 2 次元とする．図の例では 3×2 で使用するので，P_{00} から P_{21} になる．P_{rs} は小行列を（その添字について）行方向は r から p おき，列方向は s から q おきに所有する．グローバルの視野で見ると A_{00} は P_{00}，A_{10} は P_{10}，…，A_{55} は P_{21} の所有となる．ローカルの視野で見ると P_{00} の所有は $A_{00}, A_{30}, \ldots, A_{34}$ となる．図に 2 通りの配置方式を示した．A_{00} の 1 列目に続いて A_{30} の 1 列目を置く方式は，**列が連続的**になるので Column Continuous（CC）型と呼ぶことにする．CC 型は複数の小行列を連結して，1 つの大きな小行列として扱うことを可能にする．これに対し**各小行列ごとに**配置する方式を Column Blocked（CB）型と呼ぶことにする．図では A_{00} の b 列目に続いて

A_{30} の 1 列目が置かれることを強調して，カーブ矢印を加えた．CB 型ではつねに $b \times b$ の小行列として扱う必要がある．

2.2 一般行列とベクトルの積

行列ベクトル積 $\boldsymbol{y}_I = \sum_{J=0}^{N-1} A_{IJ}\boldsymbol{x}_J$ の性能を，“ベクトル $\boldsymbol{x}$ を行グループ 0 のノードに集めた状態”から計測する．“行グループ 0” は 2 次元のグリッド状に配置されたプロセッサ番号の行添字が 0 のものである．図の例では，P_{00} が $\boldsymbol{x}_0, \boldsymbol{x}_2, \boldsymbol{x}_4$ を，P_{01} が $\boldsymbol{x}_1, \boldsymbol{x}_3, \boldsymbol{x}_5$ を所有する．計測の終了は “ベクトル $\boldsymbol{y}$ が列グループ 0 のノードに揃った状態” とする．この場合通信は，同一列グループのノード間で P_{0s} から $\boldsymbol{x}$ を放送 *mpi_bcast* し，自分の持ち分の $\{A_{IJ}\}$ を用いる計算を行う．図 1 の例では P_{00} は次の計算を行う．

$$\begin{pmatrix} \boldsymbol{y}_0 \\ \boldsymbol{y}_3 \end{pmatrix} = \begin{pmatrix} A_{00} & A_{02} & A_{04} \\ A_{30} & A_{32} & A_{34} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_4 \end{pmatrix} \quad (1)$$

ここで得られた $\boldsymbol{y}$ は中間結果なので，同一行グループのノード間で P_{r0} 上に縮約する *mpi_reduce*. 通信時間は $p \times q$ のプロセッサグリッドの場合，$n(\lceil \log_2 p \rceil/q + \lceil \log_2 q \rceil/p)$ に比例する．1 次元分散ではプロセッサ数を pq とした場合 $n\lceil \log_2 pq \rceil$ に比例するので，$p = q$ の場合を考えると通信時間が $1/p$ になる．この通信時間の比は他のアルゴリズム，たとえば三角分解でも成立する．なお行列ベクトル積では，分割・分散方式を 2 次元ブロックとしてもよいが，三角分解のように依存性のある計算では，2 次元ブロックサイクリックにしなければ負荷を均等化できない．これが密行列の分割・分散方式を 2 次元ブロックサイクリックにする理由である．

さらに配置方式を CC 型にすると，式 (1) の計算をノードプログラムで小行列の境界を意識することなく，所有する全小行列を連結した 1 つの行列として，逐次計算用の行列ベクトル積ルーチンを 1 回呼び出すだけで処理できる．これに対し CB 型だと小行列単位に計算しなくてはならないので，2 重のループ構造の中から行列ベクトル積サブルーチンを呼び出す形になる．

また CC 型では p, q, b などから行列要素 a_{ij} がノードプログラム上の配列のどこに置かれているかを導くことができる．文献 4) にはこの**一般行列用の関数**が示されているが，これは 1 つの次元につき除算と *mod* 演算 1 回程度の簡単なものである．この関数を用いると，グローバルの視野で記述されたループを，*do* ループの始端と終端をこの関数で得られた値に変更するだけでローカルの視野にループ変換できるので，コンパイラによる自動並列化にとって重要である．

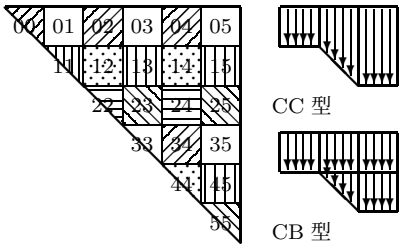


図 2 対称行列のブロックサイクリック分散
Fig. 2 2-D array decomposition/distribution for symmetric matrix.

2.3 対称行列ベクトル積

図 2 に 3×2 のプロセッサグリッドを用いて対称行列の上三角部分に 2 次元ブロックサイクリック分割・分散を行った例を示した．配置方式の違いを P_{01} を例に示した．CB 型がサイズ $b \times b$ の小行列を扱わなくてはならないことは，一般行列の場合と同様であるが，CC 型でも，列によって行要素数が変わるので，所有する全小行列を連結して 1 つの長方形とか台形といった単純形状の小行列にできない．この原因は対称行列を 2 次元分散することにあるので，配置方式を工夫しても解決できない．

対称行列ベクトル積では，小行列ベクトル積 $\boldsymbol{y}_I = \sum_{J=I}^{N-1} A_{IJ}\boldsymbol{x}_J$ と，転置して得られる小行列とベクトルの積 $\boldsymbol{y}_J = \sum_{I=0}^{J} A_{IJ}^t \boldsymbol{x}_I$ を計算する．A_{IJ} を A_{JI} としても使用しなければならないので通信方法も複雑化する．

図 2 の例では P_{01} は次式を計算する．

$$\begin{pmatrix} A_{01} & A_{03} & A_{05} \\ & A_{33} & A_{35} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_3 \\ \boldsymbol{x}_5 \end{pmatrix} \quad (2)$$

$$\begin{pmatrix} A_{01}^t & \\ A_{03}^t & A_{33}^t \\ A_{05}^t & A_{35}^t \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_3 \end{pmatrix} \quad (3)$$

式 (2) は $\boldsymbol{y}_I$ の中間結果，式 (3) は $\boldsymbol{y}_J$ の中間結果を計算する．この計算では，対角位置の小行列 A_{33} には上三角行列に属する要素しか格納されていないので，対角小行列を含むか含まないかで，台形行列用のルーチンと長方形用のルーチンを呼び分ける必要がある．つまり，小行列の境界は隠蔽できないのである．

一方通信は，一般行列の場合のように $\boldsymbol{x}$ を 1 回放送しただけでは計算を開始できない．$\boldsymbol{x}_J$ を同一列グループのノード間で P_{0s} から放送し，さらに同一行グループのノード間で $\boldsymbol{x}_I$ を，対角小行列 A_{II} を所有するノードから放送する．計算結果の縮約は $\boldsymbol{y}_I$ を

同一行グループのノード間で P_{r0} に縮約し，$\boldsymbol{y}_J$ は同一列グループのノード間で A_{JJ} の所有者に縮約したのち，ここから P_{r0} に送る．これらの事柄は，対称行列の上三角部分だけを扱うために生じているので，ScaLAPACK 法でも圧縮法でも同様である．

圧縮法の計算方法を 2 通りの配置方式について比較する．行列は大きな 1 次元配列に CC 型か CB 型で格納し，CC 型は長方形または台形行列とベクトルの積を，CB 型は長方形または三角行列とベクトルの積を計算するサブルーチンを呼び出す．長方形小行列を扱うサブルーチンでは，どちらの型でも 2 次元配列を使用できる．逐次計算では台形行列に対する行列ベクトル積プログラムはあまり使用されないので，逐次計算プログラムを並列化する作業量の観点では CB 型が有利である．性能面では内側ループの反復回数（ベクトル長）を長くとれる CC 型が有利である．

サブルーチンに渡す小行列の**先頭番地**を得るには，図 2 の例では，A_{35} が P_{01} の 5 番目の小行列であることを，I，J，p，q，b および自身のプロセッサ番号 r，s から求める関数を用いる．この関数は，ScaLAPACK 法では一般行列用の関数を使用できるので，I や J が非常に大くなってもその計算時間は無視しうる．しかし圧縮法での先頭番地の計算は，逐次計算での $j(j-1)/2+i$ のような閉じた形の式に表せないので，数えあげ（ループ計算）が必要になる．これを小行列演算のたびに行うと，小行列のサイズ b が小さく，行列サイズ n が非常に大きい場合は計算時間に影響が出る．そこでプロセッサグリッド p と q を定義 *mpi_comm_split* したとき，それらの最小公倍数 lcm を求め，$\{A_{IJ}\}_{I<lcm,\ J<lcm}$ について，各ノードで $\{A_{IJ}\}_{I<lcm,\ J<lcm}$ の先頭番地を I，J から引ける形の 1 次元配列（長さ $\lceil lcm/p \rceil$）にセットしておく．I，J が大きな値になっても，$lcm \times lcm$ の小行列には規則性があるので，I/lcm，J/lcm から得られる位置に $mod(I, lcm)$，$mod(J, lcm)$ で配列参照して得られた値を加えることで，A_{IJ} の置かれた番地をループ計算なしに得る．この手法は CC 型でも CB 型でも大差ない．つまり ScaLAPACK 法を圧縮法に変更するには，この関数を作成し一般行列用の関数を置き換え，小行列ベクトル積ルーチンを改訂すればよい．

次に性能面について述べる．CC 型，CB 型で計算し，演算数を $2n^2$ flops として求めた Mflop/s 性能値を**表 1** に示した☆．行列は $n = 5000$，ブロックサイズは $b = 48$ とし，小行列とベクトルの演算サブルーチンは 4×2 のアンローリングを適用して，計算密度を $\rho = 1.78$ とした☆☆．使用計算機は IBM の RS/6000 SP2 で，プロセッサはクロック周波数が 62.5 MHz で，ピーク性能は 250 Mflop/s である．また相互結合網には High-Performance Switch（HPS）を備える．性能は CC 型が CB 型よりも速いが，これは CB 型がベクトル長が非常に短いループをより多く実行するためである．

☆ 浮動小数点演算数を flops で，演算速度を flop/s で表す．

表 1 対称行列ベクトル積の性能（Mflop/s）
Table 1 Performance of symmetric-matrix vector multiplication (Mflop/s).

	2×2	2×3	3×2	2×4	4×2	3×3
CC	487	695	699	889	850	994
CB	452	608	711	788	746	913

(n=5000, b=48)

2.4 対称行列の三角分解

三角分解アルゴリズムの例として，$N \times N$ 個に分割された小行列の上三角部分に対するコレスキー分解 $A \rightarrow C^tC$ を示す[5]．なお，本論文ではループ構造の変更は構文的な変更の枠内と考える．

コレスキー分解のアルゴリズム

```
do K = 0, N-1
  A(K)_KK → C^t_KK C_KK      ... 分解
  do J = K+1, N-1
    C_KJ = C^{-t}_KK A(K)_KJ  ... 前進代入
    do I = K+1, N-1
      A(K+1)_IJ = A(K)_IJ - C^t_KI C_KJ  ... 更新
```

ただし $N = \lfloor (n-1)/b + 1 \rfloor$，$A^{(0)}_{IJ} = A_{IJ}$ とする．このアルゴリズムによると，対角小行列 A_{II} は I 回更新されて $A^{(I)}_{II}$ となり，分解されて C_{II} になる．非対角小行列 A_{IJ} は I 回更新されて $A^{(I)}_{IJ}$ になり，前進代入によって C_{IJ} となる[5]．小行列単位の演算は，分解，前進代入，更新（$I = J$ の場合は対称階数 k 更新，$I \neq J$ の場合は行列行列積）である．逐次計算で CB 型圧縮法の場合はこのアルゴリズムに忠実にプログラミングできる．つまり，3 重ループ構成の中で I，J，K から各小行列の先頭番地を求め，小行列演算はサブルーチンを呼び出す．ScaLAPACK 法と CC 型圧縮法は逐次計算なら $C_{K,K+1:N}$ を求め，更新を $A^{(K+1)} = A^{(K)} - C^t_{K,K+1:N} C_{K,K+1:N}$ とまとめて計算できるのでループ構造を単純化できる．

2.4.1 通信方法

A_{KK} の所有者がこれを分解したら，同一行グループのノード間で放送し，受信したノードは前進代入を行う．**図 3** は 10×10 個に分割された小行列が，3×4

☆☆ (計算密度：ρ)=(浮動小数点演算回数：flops)÷(記憶域アクセス回数).

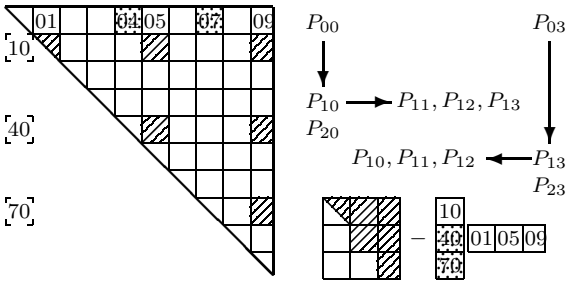


図 3 コレスキー分解
Fig. 3 Cholesky factorization.

のプロセッサグリッド上に分散されたときの，$K = 1$ の計算を示した．P_{00} が A_{00} を分解し，$P_{0,0:3}$ 間で C_{00} を放送し，$C_{0,1:9}$ を求める．$C_{0,1:9}$ は $C^t_{1:9,0}$ でもあるため，これらを同じ列グループのノード間で放送した後，対角小行列 A_{JJ} の所有者から同じ行グループ間で放送する．図では P_{11} が行う更新計算を示しているが，ここで必要になる C_{04} が P_{00} から P_{10} を経由して P_{11} に送られ C_{40} として，また C_{07} が P_{03} から P_{13} を経由して P_{11} に送られ C_{70} として使用されるまでの 2 つの経路を示した．この通信パターンは，対称行列ベクトル積で $\boldsymbol{x}_J$ を通信するのと同じパターンである．なおこの通信方法は，ScaLAPACK 法と圧縮法で同じである．

2.4.2 計算方法

図 3 の右下に P_{11} が行う更新計算を示した．対角位置の小行列があるかないかで演算の種類を変えなくてはならないので，小行列境界を隠蔽できない．CC 型圧縮法，ScaLAPACK 法，CB 型圧縮法の 3 者の比較をする．ここでは C_{KI} に着目する（C_{KJ} は図の例では C_{01}，C_{05}，C_{09} であり，これらは送信バッファに詰めてから送り出されるので，データ構造による違いがない）．複数の C_{KI} は異なる経路で送られるので，CC 型配置でバッファに受信することはできない（CB 型配置では受信可能である）．ScaLAPACK 法と CC 型圧縮法では，受信した C_{KI} を CC 型に並べ直すと，更新計算で扱うベクトル演算のベクトル長を長くできる．しかし並べ直しは，更新計算にキャッシュブロック化を適用する場合は無駄になる．

キャッシュブロック化は行数の多い行列から，適当なサイズの小行列を切り出して連続する作業領域に詰め直すことで，繰り返し参照されるデータのキャッシュミスを防ぐプログラミング技法（ループ変換）である．この例の場合詰直しを C_{KI} に適用すると，更新される行列が連続アクセス（ストライド 1）になるので効果的である[5]．つまりこの場合は，ブロックサイズ b が，キャッシュブロック化にも使用できるサイズであれば，キャッシュブロック化を兼ねて，受信バッファの小行列を直接サブルーチンに渡すのが最も速い．したがって，三角分解では ScaLAPACK 法や CC 型圧縮法であっても，更新計算プログラムは小行列サイズ b を用いた更新計算をするので CB 型との大きな差異はない．

2.4.3 性能測定

行列ベクトル積の場合と同じ環境を用いて CB 型で計測した結果は，$n = 2000$ の行列を $p \times q = 2 \times 3$，$b = 40$ で 800 Mflop/s であった．1 ノードでは約 215 Mflop/s なので，並列化効率は 62%にあたる．$n = 5000$ の行列を $p \times q = 3 \times 3$ で 1450 Mflop/s となり，並列化効率は 75%に達する．なおここで演算数は $n^3/3$ flops とした．

2.5 対称密行列のデータ構造について

対称行列を 2 次元ブロックサイクリック分割・分散すると，ローカルの視野で連結された小行列は複雑な形になり，小行列の境界を意識しなくてはならなくなる．このため小行列の先頭番地を算出しなくてはならなくなる．対称行列を正方行列用の 2 次元配列を前提として格納するデータ構造をとると，一般行列と同じ関数で算出できる．ScaLAPACK はこの方法を用いている．歴史的には，Fortran プログラムによる対称行列の扱いは，2 次元配列の半分を使用する方法と，1 次元配列に圧縮する方法の両方が使い分けられていた．両者は “$a(i,j)$ を，外側の j のループで $j0 = j * (j-1)/2$ と求めたうえで $a(j0+i)$ に変更する” という機械的な修正で変換できるため，扱う行列の規模が大きくなって記憶域が不足した時点で，1 次元配列に移行することも可能である．分散メモリ型の並列計算機では，記憶容量に余裕が出てきたことは事実であるが，大規模な問題では記憶容量の制限で解けなくなる場合も多く，記憶域を節約することは依然として重要である．しかし両者間の移行は逐次計算のように簡単にはできない．

ScaLAPACK が 2 次元配列を前提とした理由を考える．まず，ScaLAPACK が LINPACK+EISPACK（固有値問題のパッケージ）を前身としたこと，そして対称行列のすべての固有値，固有ベクトルを求める場合は，$n \times n$ の配列の半分に係数行列の上三角行列を入力し，n 本の固有ベクトルをここに出力するインタフェースが使用されたため，という解釈である．また自動的な並列化を視野に入れたためとも解釈できる．すなわち “HPF で自動的な並列化により係数行列を生成する際，正方行列全体を作れば並列化されやすい．

行列生成の計算量が $O(n^2)$ の場合，三角分解などの $O(n^3)$ に比較して小さいので，行列生成を行列全体について計算してもその時間は無視しうる．HPF はグローバルの視野で記述するが，ローカルの視野で記述されたサブルーチンを extrinsic 機能を介して呼び出すことができる[1]．この経路で ScaLAPACK を使用することを想定した” という解釈である．この状況では ScaLAPACK 法は適切なデータ構造といえるが，直接 MPI 呼び出しでプログラミングしているユーザにはこの恩恵はない．

本論文の主張の第 1 点は，“圧縮法と ScaLAPACK 法との違いは，小行列の先頭番地を求める関数と，小行列演算サブルーチンの入れ替えで吸収可能なので，圧縮法もあわせて実装すべき” である．圧縮法では配置方式に CC 型と CB 型の 2 通りが考えられるが，これまでの議論の範囲では，どちらが良いかの決定的な差は見出せない．しかし CB 型は，次章で例示するように，少し変形された行列にも柔軟に対応できるという優れた性質を備えている．

3. 多重スカイライン法への応用

本章ではまとまったゼロ要素（穴）が存在する対称密行列を扱う．この行列はデータ構造の比較の目的で作られたものではなく，有限要素法（FEM）構造解析に領域分割を用いた場合に生成される疎行列を，領域分割を行ったときと逆の順序で三角分解すると，その最終段階に現れる．したがって，ここで述べる多重スカイライン法以外でも，領域分割とそれを逆にたどる三角分解のプロセスから類似した形の行列が現れる．そこではじめに，領域分割と疎行列の三角分解の関係を示し，次に CB 型を用いた準密行列の三角分解について述べる．

3.1 多重スカイライン法

多重スカイライン法は解剖法を用いて節点の順序付けを行って得られる対称疎行列を三角分解する．前章の密行列との違いは，小行列のサイズが可変であること，小行列自身も疎のものを含む点にある．

多重スカイライン法のすでに報告済みの範囲では，2 次元領域に適用した場合は十分な性能が達成された[6]．しかし立方体のような 3 次元領域に適用すると，最終段階に現れる準密行列のサイズが大きくなり，この部分を逐次計算により分解していたので，性能が低下し，また解きうる問題の規模も制限された．今回この部分に，前章に述べた CB 型の三角分解を組み込むことで，性能向上と問題規模の制限を緩めることに成功した．

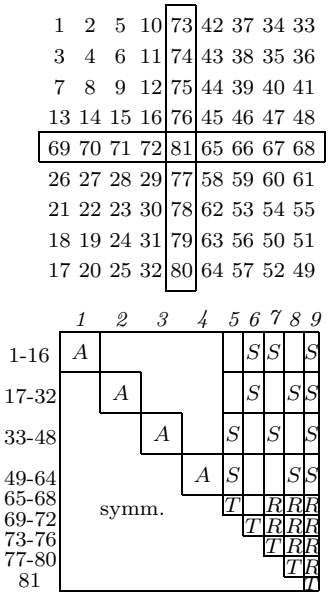
3.1.1 解剖法順序とハイパーテーブル

解剖法は解析領域を 2 つの部分領域と，両者が直接接続しないような境界領域に分割する操作を r ステージ行うことで，2^r 個の部分領域と境界領域とに領域分割する[7]．解剖法によって生成された節点順序を解剖法順序と呼ぶことにする．分割によって作り出される 2 つの部分領域は分離しているので，部分領域に対応する小行列どうしは連成がなく，三角分解 $A \rightarrow C^tC$ によって得られる三角行列 C にも分離によるゼロが小行列単位で保存される（後述）．このため解剖法順序と自然な順序を比較すると，三角分解に要する演算数も大きく異なる．たとえば正方形領域では，係数行列の三角分解に要する計算量を自然な順序による $O(n^2)$ から $O(n^{1.5})$ に削減する．

この効果を “FEM 行列生成と三角分解のプロセス” に反映するための最低限の入力情報は，解剖法順序そのものである．与えられた節点順序から，シンボリック分解と呼ばれる分解後の行列の非ゼロ要素を特定する処理を行えば，三角分解に無駄な演算が混入することを排除できる（後述）．また節点順序から “消去の木”separator tree も生成でき，これにより分解計算を並列化することもできる[8]．

シンボリック分解の計算量は三角分解よりはオーダーが低いが，無視できるほどではなく，また並列化もできない．そこで多重スカイライン法では，シンボリック分解を節点を単位としてではなく，解剖法の分割によって生成される節点のグループを単位として行うことで，シンボリック分解の処理の軽減を図った．そのために入力情報として，節点順序（解剖法順序以外も可），節点の分類（グループ化），節点グループ相互の接続関係を収めたハイパーテーブルを使用する．ハイパーテーブルに消去の木の役割を担わせることで，多重スカイライン法は規則的なループ変換という構文的な範囲で並列化できる．

図 4 は解析領域を 8×8 の 64 個の四辺形要素で構成し，これを 4 つの部分領域（$r = 2$）に分割した例である．たとえば左上の四辺形要素は節点 1，2，4，3 に接続するが，この要素の輪郭線は省略した．左上が節点順序（境界領域に属する節点はグループごとに囲んだ），左下が対応する係数行列，右にハイパーテーブルを示した．g がグループ番号，lg がそのグループの最後の節点番号，lst が接続グループ番号のリストである．グループ分けは，*1* から *4* までが部分領域で，接続する部分領域の組合せによって *5* から *9* が決まる（グループ番号をイタリック体で示した）．A_{11} が A_{22} と連成しない（A_{12} がゼロな）ので，小行列 A_{17} は



g	lg	lst
1	16	6,7,9
2	32	6,8,9
3	48	5,7,9
4	64	5,8,9
5	68	7,8,9
6	72	7,8,9
7	76	8,9
8	80	9
9	81	

図 4 節点順序，係数行列，ハイパーテーブル
Fig. 4 Grid-points ordering, Coefficient matrix, Hypertable.

その下にフィルインを作らない（A_{27} はゼロのまま）．これは “節点 73 から 76” が，やはり境界領域によって “節点 17 から 32” と分離されているからである☆．通常のスカイライン法では[9]，特定の列に着目すると，その列の最も若い非ゼロ要素よりも下に位置する要素はすべて計算の対象になるが，多重スカイライン法では，係数行列の構造（グループ間の接続関係）がハイパーテーブルに与えられるので，このゼロ小行列を認識できる．

3.1.2 データ構造

小行列のデータ形式として，部分領域に含まれる節点に対応する小行列（図 4 の A）はスカイライン形式で，部分領域と境界領域の接続部分（S）は変形したスカイライン形式で扱う．なおスカイライン形式は，行列を列ごとに，最も行番号の若い非ゼロ行要素から対角項までを 1 次元配列に圧縮して格納し，対角項のポインタを 1 次元配列（スカイライン インデックス）に格納する．

境界領域は分解後の三角行列に基づき，データ形式は対角位置の小行列は三角形（T），非対角位置の小行列は長方形行列（R）の密行列である．出力は通常のスカイライン法と同様，入力行列に上書きされるので，記憶域の無駄はない．すなわちユーザは，解剖法によって節点順序の生成とそのグループ分けを行い，シンボリック分解によってフィルインを特定してハイパーテーブルを作成し，小行列の集合として係数行列 A を生成すれば，多重スカイライン法ルーチンによって三角分解できる．

3.1.3 逐次計算アルゴリズム

逐次計算アルゴリズムは前章に示した密行列のコレスキー分解アルゴリズムの I，J の添字を，ハイパーテーブルを介して参照するように変更することで得られる．

疎行列のコレスキー分解のアルゴリズム

$do\ K = 1, ng$
  $A_{KK}^{(K)} \rightarrow C_{KK}^{t} C_{KK}$
  $do\ jl = 1, len(lst(K))$
    $J = lst(jl, K)$
    $C_{KJ} = C_{KK}^{-t} A_{KJ}^{(K)}$
    $do\ il = 1, len(lst(K))$
      $I = lst(il, K)$
      $A_{IJ}^{(K+1)} = A_{IJ}^{(K)} - C_{KI}^{t} C_{KJ}$

ただし ng はグループ数，$len(lst(K))$ はハイパーテーブルの k 番目のグループのリスト長である．また小行列演算は，スカイライン形式用のもの，密行列用のもの，両者を用いるもののため，種類は多くなるが，これらはキャッシュブロック化を適用したスカイライン法プログラムから作ることができる[5]．

正方形の解析領域を 128×128 の 16,384 個の四辺形要素で構成し，これを 64 の部分領域（$r = 6$）に分割した面内応力解析（1 節点は 2 自由度）の問題では，解剖法は演算数を，自然な順序の 20%にまで削減する．この分割に対して多重スカイライン法は，通常のスカイライン法を自然な順序に対して用いた場合の 30%の計算時間で三角分解する[6]．

3.2 多重スカイライン法の並列化

並列化は，逐次計算アルゴリズムに構文的な変更を加えるアプローチを基本とした．以下，疎行列のコレスキー分解アルゴリズムに対する配列分割・分散とループ変換を中心に述べる．

3.2.1 配列の分割・分散

小行列は部分領域や境界領域のグループごとに分散する．グループに属する節点数は異なるので，対称行列の上三角部分で見ると，$\{A_{IJ}\}$ の I による可変ブロック長の 1 次元分散と考えられる．行列本体は分散によってローカルの視野となるが，ハイパーテーブルはグローバルの視野とする．1 次元分散なので更新計

☆ 疎行列の三角分解では 1 つの節点（グループ）を消去すると，その節点（グループ）の接続関係は，その節点（グループ）と直接接続を持つ節点（グループ）に波及する．図の例では，7 と 8 は接続がないので $A_{7,8}$ は初期にはゼロ行列であるが，5 が消去された段階で $-C_{57}^{t} C_{58}$ によって非ゼロ要素（フィルイン）が現れる．

算 $A_{IJ}^{(K+1)} = A_{IJ}^{(K)} - C_{KI}^t C_{KJ}$ の積計算 $-C_{KI}^t C_{KJ}$ までは通信なしでできる，これを $A_{IJ}^{(K)}$ に足し込むところで，通信や同期の考慮が必要になる．

ハイパーテーブルを検索することで，並列に消去できる小行列の組合せを調べ，この結果を**順位**として 1 次元配列 $level$ に格納する．部分領域は順位ゼロ，他の境界領域のグループからの更新を受けないグループを順位 1，順位 1 以下のグループからしか更新を受けないグループを順位 2，... とする．

計算の依存性

```
do K = N1 + 1, ng
  do jl = 1, len(lst(K))
    J = lst(jl, K)
    level(J) = max(level(J), level(K) + 1)
```

分散はノード数を np とすると，部分領域グループはブロック分散（N_1/np 個ずつ分散）し，境界領域グループはハイパーテーブルを検索して通信量が少なくなるように分散する．分散の結果は 1 次元配列 $iown$ に格納する．

$level$ は計算の依存性を表すが，ノード内の計算順序を小行列の添字の昇順に限定すれば，順位を少なくできる．そこで分散が確定した段階でこれを次のように，通信の依存性を考慮するものに更新する．

通信による依存性

```
do K = N1 + 1, ng
  do jl = 1, len(lst(K))
    J = lst(jl, K)
      if(iown(J).ne.iown(K))then
        level(J) = max(level(J), level(K) + 1)
```

配列 $level$ と $iown$ もグローバルの視野に置く．順位はグループの属性なので，対角小行列の分解や前進代入では，小行列の行添字として直接依存性の判定に使用できる（同じ順位の K なら並列計算可能）．この順位が消去の木の代用として機能する．

3.2.2 計算方法と通信方法

解剖法順序による疎行列の三角分解は，解剖法の分割のステージを逆にたどる．最初は依存性のない（順位がゼロの）グループを扱うので並列性が高いが，ステージが 1 つ若い分割で生成されたグループに進むたびに並列性は半分になり，フィルインのために分解すべき小行列も密になってくる．この密になってゆく過程は行列や分割方法によって異なるが，正方形領域と立方体領域に解剖法を適用した場合が典型的で分かりやすいので，以下これらを例とする．

分解計算はこの疎/密の程度によって計算方法/通信方法を切り替える．**図 5** に切り替える位置を示した．“1” から “4” は，計算の進行を更新計算 $A_{IJ}^{(K)} - C_{KI}^t C_{KJ}$ の C_{KI} の位置で表した（C_{KI} が “1” の位置にあれば，以下の “Part 1” の方法で計算される）．図の縦方向の単位はグループ番号とする．N_1 として部分領域数を用いる．つまり “Part 1” は部分領域を消去する．“Part 2”，“Part 3”，“Part 4” を切り替える N_2 は，今回の改良以前は，並列性のなくなるグループ番号を使用していたが，今回の改良で，消去計算の並列性が計算機の並列性よりも少なくなるグループ番号を選択するようにした．したがって，ノード数の多い並列計算機で実行すると，若い N_2 が選択され，Part 2 の演算数が減り Part 4 の演算数が増える．

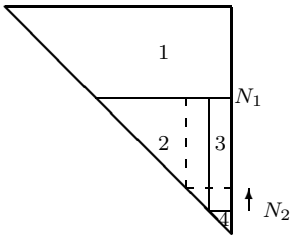


図 5 計算の分割
Fig. 5 Division of computation.

Part 1 の計算は，逐次計算アルゴリズムの “$do\ K = 1, ng$” を次のように変更する．

部分領域のループ変換

```
ns = N1/np
  do ks = 1, ns
    do K = ks, N1, ns
      if(iown(K).eq.myid) then
```

ただし $myid$ はプログラムを実行するプロセッサ番号である．また更新計算は，被更新行列が他のノードの所有の場合は送信し，対応するノードが受信して足し込む．Part 1 の処理の特徴は，このループで処理するグループ（部分領域）どうしは連成がない（足し込みはいつ行ってもよい）こと，計算に比較して通信量があまり多くないことである．そこで通信と計算をオーバラップさせる目的で，2 つの通信バッファを用い，非同期受信 mpi_irecv，更新計算と非同期送信 mpi_isend，最後に mpi_wait で前回のループ反復の通信を待ち合わせてから足し込む，という処理をしている．

Part 2 は，1 つの被更新小行列に足し込まれる積の数はまだ少ないが，複数の積が足し込まれる可能性があるものを扱う．そこで “$A_{IJ} - C_{K_1I}^t C_{K_1J} - C_{K_2I}^t C_{K_2J} - \ldots$” とループ構造を変更する（$K$ を内側にする）．ここで I，J の順位の組合せが同じものをまとめて行うと，複数の通信をオーバラップさせられる．具体的には，逐次計算アルゴリズムの 3 重のルー

プ構造の外側に，順位に関する 2 重のループをかぶせることで，依存性の低い小行列計算から順位順に選択的に行う順位駆動型のループ構成をとっている．次の計算方法の表現で “$\{A_{IJ}\}\ I,J=li,lj$” は，行位置 I の順位が li，列位置 J の順位が lj を満たす小行列とする．

順位駆動型ループ（外側）

```
do lj = 1, level(N2) + 1
  do li = 1, min(lj, level(N2) + 1)
   {F_IJ = -C^t_KI C_KJ} I,J,K = li, lj, 1 : li-1
   mpi_alltoallv F_IJ
   {A_IJ ← A_IJ + F_IJ} I = li, J = lj
```

ノードプログラムは自分の所有する順位が li よりも小さいグループについて，I の順位が li の C_{KI} と，J の順位が lj の C_{KJ} の両方が存在すれば，F_{IJ} を計算する．この処理は次のように，if 文でマスクしたループ構成によって記述することができる．

順位駆動型ループ（内側）

```
do K = N1 + 1, ng
  if (iown(K).eq.myid.and.level(K).lt.li) then
    do il = 1, len(lst(K))
      I = lst(il, K)
      if (level(I).eq.li) then
        do jl = il, len(lst(K))
          J = lst(jl, K)
          if (level(J).eq.lj) then
```

$iown(K)$ と $iown(I)$ が異なる場合には，$\{F_{IJ}\}$ を送り先のノードの昇順にソートし，また I と J についても昇順に詰めて送信バッファに置く．受信側は，自分が所有する被更新小行列 $\{A_{IJ}\}$ について，送信ノードごとに $\{F_{IJ}\}$ の存在を調べられるので，複数の送信と，対応する複数の受信は転置通信 $mpi\_alltoallv$ 1 回で行うことができる．この通信方法は性能の向上に効果的で，後述する立方体データに対して $isend/irecv$ によるよりも，8 ウェイで 20%の性能改善につながった．

Part 3 は，1 つの被更新小行列に足し込まれる更新小行列の数が多いものを扱う．そこで通信量の削減のために縮約通信を用いる．以前は，最終段階の小行列は 1 ノードで所有していたので，更新小行列を各ノードのバッファに Part 4 の形で足し込んだのち，全ノード間で縮約通信 mpi_reduce していた．今回の改良では最終段階の小行列も並列性が残っており，これらは複数のノードが所有しているので，受信側のノード数の回数，縮約通信を行う．この縮約通信は，疎行列を縮約する専用のルーチンによった．2 分木状に $send/recv$ を $\log_2 np$ 段繰り返すことで，1 つのターゲットノードへ縮約できる．

Part 4 は，今回の改良以前は，最終段階の複数のグループはすべて 1 ノードの所有としていた．今回の改良で，並列性を残した状態でも最終段階に入るようになったので，2 次元ブロックサイクリックに再分散して，並列性を復活させ，全ノードで分解する．ここで前章で述べた密行列用のルーチンが利用できる．

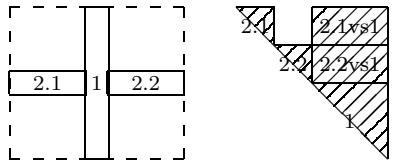


図 6 正方領域の最初の 2 ステージの分割と最終段階の行列
Fig. 6 Matrix of the final triangle (two partitioning stages for square).

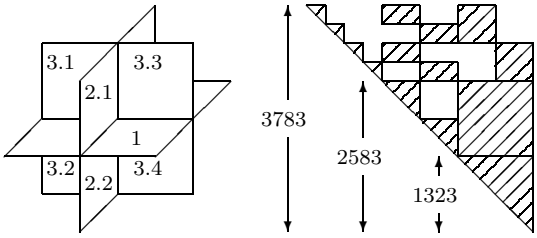
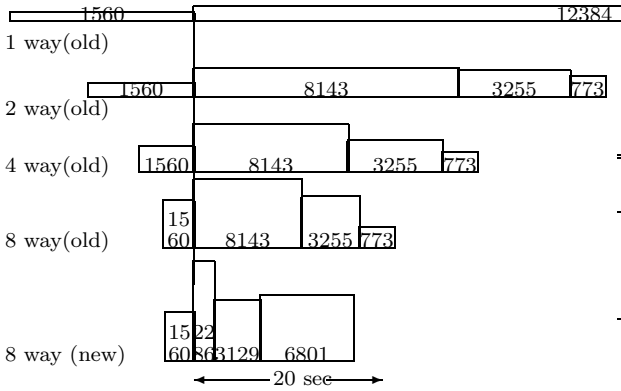


図 7 立方体領域の最初の 3 ステージの分割と最終段階の行列
Fig. 7 Matrix of the final triangle (three partitioning stages for cube).

3.2.3 準密行列の三角分解

解析領域が正方形の場合で，N_2 を解剖法による分割の第 2 ステージまでの境界領域とした場合，Part 4 で扱う小行列は**図 6** のように 1 つの穴が空いている．この理由は，1 が最初の分割でできた境界領域，2.1 と 2.2 が 2 回目の分割でできた境界領域のとき（2.1 と 2.2 は 1 によって分離されているので，その消去計算には 2 ウェイの並列性がある），“2.1 vs 1” と “2.2 vs 1” の部分は，第 r から第 3 ステージで生成したグループを消去すると，フィルインで密になっているからである．全要素に対するゼロ要素数は約 1/8 であるが，これをオペランドとする乗算を回避できると，（回避できないで全体を密行列として計算した場合の）72%の演算数で済む．

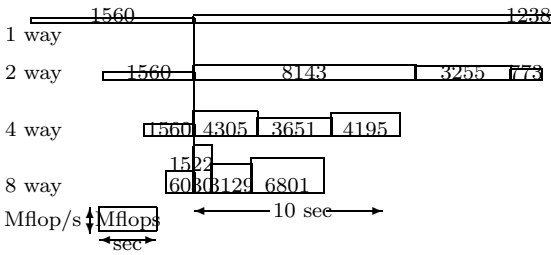
最終段階に 4 ウェイの並列性が残る（Part 2 に 8 ウェイの並列性がある）ように，N_2 を解剖法による分割の第 3 ステージまでの境界領域とした場合，Part 4 で扱う小行列はさらに多くの穴が空いている．立方体領域の場合を**図 7** に示したが，この場合，この穴をスキップできると，全体を密行列として計算した場合の 38%の演算数で済む（後述）．なお，この行列の形は領域分割から生じたもので多重スカイライン法に特有の形ではない．



改良の効果，SP2 での経過時間（秒）と並列化効率
Performance improvement, elapsed time (sec) on SP2s

np	改良前（old）		改良後（new）			
	62.5 M（HPS）		62.5 M（HPS）		77 M（SPS）	
1	102.0	1.0	102.0	1.0	89.8	1.0
2	54.8	.93	54.8	.93	46.4	.97
4	35.8	.71	31.3	.81	25.9	.87
8	24.4	.52	19.9	.64	15.2	.74

SPS は HPS の約 2 倍の通信性能



SP での経過時間（秒）と並列化効率
Elapsed time (sec) and parallel efficiency on SP

np	$t(t1+t2+t3+t4)$ 秒	並列化効率
1	44.6 (8.60+36.0)	1.0
2	23.3 (4.78+11.7+5.07+1.61)	.96
4	13.5 (2.61+3.34+3.93+3.61)	.83
8	8.23 (1.44+.910+2.13+3.81)	.67

図 8 並列版多重スカイライン法の性能（機種別，および改良の効果）
Fig. 8 Performance of parallel multi-skyline method (comparison of before/after modification on different computers).

可変ブロック長の 1 次元分散から固定ブロック長の 2 次元分散への**再分散**は，まず固定ブロック長の 1 次元分散に変換して，次にこれを 2 次元分散する 2 段階方式とした．この再分散の処理を逆に行うと多重スカイライン形式に戻すことができる[10]．

2 次元分散するとき，$b \times b$ の小行列 A_{IJ} の要素がすべてゼロの場合は，記憶領域を詰める．また，各小行列の配列内での位置は番地テーブルに格納する．

穴あき行列の三角分解は，前章で述べた対称密行列の三角分解の，小行列の置かれた先頭番地を引く関数を，再分散ルーチンの出力した番地テーブルを参照するように変更することで行える．また零小行列は計算をスキップする．このように多重スカイライン法の最終段階の計算は，密行列用のアルゴリズムに比較的簡単な修正を加えるだけで実現できる．三角分解が簡単に行えるのは CB 型によるところが大きく，またそれに先立つ再分散においても，CB 型は密行列の再分散ルーチンに簡単な修正を加えるだけで作成できる[10]．

3.3 性能の測定

立方体領域を 20×20×20 にメッシュ切りしたモデル（1 節点 3 自由度としているので $n = (20+1)^3 \times 3 = 27,783$）に対し，解剖法の分割を 6 ステージ反復した 64 部分領域に分割したデータを計測し，経過時間を測定した（**図 8** の右下の表）．分解された三角行列の行列要素数は 15.4×10^6 になり，演算数は約 14 Gflops（部分領域と境界領域はそれぞれ 1.6，12.6 Gflops）である．行列は 342 個のグループに分割され，そのうち 64 個が部分領域である．また固定ブロック長は 36 を用いた．N_2 は 2/4/8 ウェイの計算で，それぞれ第 1/2/3 ステージの解剖法分割の境界領域に対応させた．図 7 に最終段階の行列を示した．これらの行列の次数は，n =1323/2583/3783 である．これを穴を考慮して分解すると，その演算数は 773/4195/6801 Mflops になる．8 ウェイの場合，三角分解全体の半分近くの演算数が Part 4 に集まっている．

もし最終段階の小行列を，CC 型用のルーチンを使用して分解するとなると，穴をつぶしてゼロ演算を行うことになる．この場合の演算数は $n^3/3$ なので，8 ウェイの場合は穴を考慮することで 38%の演算数になる．4 ウェイではこの値は 73%である．

使用した計算機は IBM の SP で，各ノードは RS/6000 の 160 MHz のプロセッサを使用している．相互結合網には SP スイッチ（SPS）を搭載している．

結果を図 8 の右下の表に示したが，括弧内の数字は，Part ごとの処理時間である（逐次処理では Part 2，3，4 は分離されない）．なお Part 4 での 2 次元分散は，プロセッサグリッドを 2×1，2×2，4×2 とした．

今回の改良の効果や，プロセッサと相互結合網の性能バランスで並列化効率がどのように変化するかを示すために，異なる機種でも計測した．計算機は前章で密行列プログラムの計測に使用した，ノードに 62.5 MHz のプロセッサ，相互結合網に HPS を備えた SP2 と，77 MHz のプロセッサに SPS を組み合わせた SP2 である．SPS は HPS の約 2 倍のデータ転送能力を持つ．図 8 の右上の表にこれらの環境で改良前・後のプログラムを計測した結果を示した．改良の効果は 62.5 MHz と HPS を備える SP2 では，8 ウェイで，経過時間を約 80%に短縮し，並列化効率を 52%から 64%に向上させている．並列化効率は 77 MHz と SPS を組み合わせた SP2 では 74%に達している．なお Part 4 で行う再分散の時間は，Part 4 全体の時間の数分の 1 である．

図 8 に，62.5 MHz の SP2 と 160 MHz の SP における各 Part の経過時間，計算速度，演算数を，それぞれボックスの幅，高さ，面積で示した．横軸が時間で，部分領域消去の完了に揃えた．上が SP2 での改良前プログラムによる逐次，2，4，8 ウェイ，改良後プログラムによる 8 ウェイである．下が SP での逐次，2，4，8 ウェイのものであるが，横軸のスケールは 1:2 の違いがある．

改良前のプログラムには，Part 4 はノード数を増やしても速くならないだけでなく，Part 2 の後半で並列性が不足して性能が伸び悩む現象がみられた．今回の改良で N_2 に小さいグループ番号を選べるようにしたので Part 2 の速度も向上している．8 ウェイの改良前・後の経過時間は 20%の改善にあたるが，改良後の時間は改良前の 16 ウェイの時間に等しいので，計算機資源の観点では大きな改善である．ここでは CB 型準密行列処理が重要な役割を果たしている．並列化効率が 8 ウェイで 67%なので，対称密行列の三角分解と比較して，十分な性能に達したと考えている．

3.4 対称行列のデータ構造について

多重スカイライン法を例として，疎行列を出発点とする準密行列の並列化三角分解について述べた．並列化を構文的な変換の範囲内で実現しようとすると，疎行列を 1 次元分散し，関連するテーブル類をグローバルの視野に置く方法が現実的である．しかし 3 次元領域を対象とする領域分割で得られた行列を扱うと，この方法の限界が現れ，再分散と準密行列の分解が求められる．ここでは密行列用 CB 型圧縮法ルーチンに簡単な修正を加えただけで対応し，十分な性能を得ることができた．本論文では 8 ウェイまでの性能測定を行ったが，この規模の問題に対して計算機の並列度を高めると，それに見合う “消去計算の並列性” を持つ小行列のサイズが小さくなり，ゼロ演算の混入のために演算数が増加する．これを解決するためには，$b \times b$ の小行列内部に連続的にゼロ列やゼロ行が存在した場合に，演算を回避できるような手法を実装する必要がある．つまり固定ブロック長による簡単なデータ構造は，$b \times b$ に比べて十分に大きな穴には対応できるが，小さな穴には対応しきれない．

4. おわりに

密行列を 2 次元ブロックサイクリック分割・分散するときの行列要素の配置方式は，標準的には ScaLAPACK 法が使用されている．この方法は一般行列には優れた性能を提供し，またコンパイラの自動並列化までも可能とする合理的な方法である．しかし対称行列に対しては，一般行列に対するときのような合理性は少ない．本論文では記憶域の制限を緩和する目的で，1 次元配列に対称行列の上三角要素だけを圧縮して格納する圧縮法を提案し，ScaLAPACK 法からの変換方法も示した．また対称疎行列に対しても，三角分解を行うと三角行列の一部はかなり密で大きな小行列が現れる．可変ブロック長の 1 次元配列分割を用いる多重スカイライン法の最終段階に，2 次元ブロックサイクリック分割・分散へ再分散を行って，圧縮法を用いる分解計算を実装し，圧縮法の有効性を確かめた．

これらの例に見られるように，種々の行列に対するデータ構造を，できるだけ少ない種類のプログラムでカバーできることが望ましい．この観点から，対称行列に CB 型圧縮法を用いることを提案する．

参 考 文 献

1) Koelbel, C.H., Loveman, D.B., Schreiber, R.S., Steele Jr., G.L. and Zosel, M.E.: *The High Performance Fortran Handbook*, The MIT Press, Cambridge, Massachusetts (1994).
2) Parallel Engineering and Scientific Subroutine Library for AIX Guide and Reference, Order No.SA22-7272, IBM (1998).
3) Gropp, W., Lusk, E. and Skjellum, A.: *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, Cambridge, Massachusetts (1994).
4) Hiranandani, S., Kennedy, K., Mellor-Crummey, J. and Sethi, A.: Compilation Techniques for Block-Cyclic Distribution, *Proc. ACM International Conference on Supercomputing '94*, pp.392–403 (1994).
5) 寒川　光：RISC 超高速化プログラミング技法，

共立出版 (1995).
6) 寒川 光：解剖法順序を活かす多重スカイライン法，情報処理学会論文誌，Vol.38，No.10，pp.1879–1885 (1997).
7) George, J.A.: Block Eliminations on Finite Element Systems of Equations, Rose, D.J. and Willoughby, R.A. (Eds.) *Sparse Matrices and Their Applications*, pp.101–114, Plenum Press (1972).
8) Gupta, A., Karypis, G., and Kumar, V.: Highly Scalable Parallel Algorithms for Sparse Matrix Factorization, *IEEE Trans. Parallel and Distributed Systems*, Vol.8, No.5, pp.502–520 (1997).
9) Bathe, K.J. and Wilson, E.L., 菊池文雄（訳）：有限要素法の数値計算，科学技術出版社 (1979).
10) 寒川 光：分散メモリ型並列計算機での対称行列のデータ構造と多重スカイライン法への適用，並列処理シンポジウム JSPP'99 論文集，pp.191–198 (1999).

(平成 11 年 8 月 27 日受付)
(平成 12 年 3 月 2 日採録)

寒川 光（正会員）

1972 年早稲田大学理工学部機械工学科卒業．同年日本ユニバック（株）入社．1984 年日本アイ・ビー・エム（株）入社．現在東京基礎研究所勤務，金沢工業大学連携大学院客員教授兼任．数値解析，数値計算法，計算機アーキテクチャ，専用計算機に関する仕事に従事．工学博士．計算工学会，日本シミュレーション学会会員．