スカイライン法の一般化による疎行列コレスキー分解の並列処理

宮川佳夫[†] 松田章[†] 加藤隆[†]

有限要素法等によって導出される大規模で疎な連立方程式を，コレスキー分解を用いて高速に求解する一手法について検討し，RISC プロセッサをベースとする共有メモリ型並列計算機に実装する．従来の研究では十分に検討されていない，RISC プロセッサに適したループ構成を選択するとともに，疎行列の格納形式としては，消去木の構造に依存しないスカイライン法を一般化した汎用的な手法を用いる．その結果，RISC プロセッサによる効率的処理のために欠かせない CPU キャッシュを有効に利用するためのブロック化が，密行列と同様な手法で一般疎行列に対して可能となる．また並列処理においては，自動並列化等によって組み込まれるバリア同期よりも各プロセッサの待ち時間が生じにくいイベントカウント同期を導入し，マルチスレッド・プログラミングによって明示的に並列処理を記述する．計算機として Sun Enterprise 450 を使用し，高い処理性能が得られることを示す．

Parallel Processing of Sparse Cholesky Factorization by Generalized Skyline Method

Yoshio Miyakawa,[†] Akira Matsuda[†] and Takashi Kato[†]

This paper presents an efficient method of sparse Cholesky factorization on the shared memory parallel computer based on RISC processors. A looping structure, which is suitable for RISC and hasn't been examined fully yet by other researches, is chosen in the Cholesky factorization. Moreover, in order to handle sparse matrix, a generalized skyline method is developed. The method is independent of the elimination tree and makes the blocking scheme, which uses processor's cache-memory effectively, possible for general sparse matrix in the same way as dense matrix. For parallel processing, instead of the barrier synchronization inserted by automatic parallelization, the eventcount synchronization which reduces the waiting time on each processor is introduced. The parallelized code is described concretely with multithread programming. Experimental results on the Sun Enterprise 450 demonstrate the effectiveness of the new algorithm.

1. はじめに

有限要素法に基づく数値シミュレーションでは，連立方程式の求解プロセスが重要な要素を占める．特に，よりリアルな 3 次元解析モデルから導かれる連立方程式は非常に大規模なものとなり，その求解には多大な計算コストを要する．したがって，その処理過程の高速化は大規模数値解析を実現するための最重要課題であると考えられる．

連立方程式の求解法は直接法と反復法に大別されるが，通常，大規模問題に対しては一般には反復法が多用される．しかし解析対象によっては収束性が問題になるため，事前に計算量が把握可能な直接法が有用な場合も多いと思われる．本論文は，有限要素法から導出される大規模で疎な連立方程式を，直接法に基づいて高速に解くことを目的とする．まず，係数行列が正定値対称となる場合の代表的な求解法であるコレスキー分解について，非零成分が行列中に散在する一般の疎行列に対応させるための簡潔かつ効率的な手法を示す．さらに，RISC プロセッサをベースとする共有メモリ型並列計算機上で効率的に並列処理するための手法について検討し，実機上でこの手法の性能評価を行う．

† 岡山県立大学情報工学部
Faculty of Computer Science and System Engineering, Okayama Prefectural University

2. 疎行列のコレスキー分解

2.1 コレスキー分解による連立方程式の求解

連立 1 次方程式は次のように表すことができる．

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

ここで A は $n \times n$ の正定値対称行列であり，$\boldsymbol{b}$ は既

```
for(j=0; j<n; j++) {
    Factor(j,j);
    for(i=j+1; i<n; i++) {
        Factor(i,j);
    }
}
```

図 1 jik 型コレスキー分解の基本構成
Fig. 1 Basic style of jik-Cholesky factorization.

知ベクトル，$\boldsymbol{x}$ は解となる未知ベクトルである．コレスキー分解とは次式を満たす下三角行列 L を求めることである．

$$A = LL^T \tag{2}$$

L が求められれば，以下の 2 式によって連立方程式の解 $\boldsymbol{x}$ を求めることができる．

$$L\boldsymbol{y} = \boldsymbol{b} \tag{3}$$

$$L^T\boldsymbol{x} = \boldsymbol{y} \tag{4}$$

L の各要素は次の 2 式で計算される．

$$\ell_{jj} = \sqrt{a_{jj} - \sum_{k=0}^{j-1} \ell_{jk}^2} \tag{5}$$

$$\ell_{ij} = \frac{1}{\ell_{jj}} \left(a_{ij} - \sum_{k=0}^{j-1} \ell_{ik}\ell_{jk} \right), \ i > j \tag{6}$$

本論文では，行列やベクトルの各要素の添え字は 0 から始まるものとして表記している．

2.2 ループ構成

式 (5)，(6) を C 言語等のプログラミング言語でコーディングすると，コレスキー分解は式中の添え字 i，j，k をループ変数とする 3 重のループ構成となる．ループ変数の出現順によって 6 通りのループ構成が考えられる．たとえば外側から j，i，k の順にループを構成すると，コレスキー分解の基本的な処理手順は**図 1** のように表すことができる．ここでは擬似的な C 言語コードを用いている．この場合のループ構成を jik 型と呼ぶ．L の対角要素を求める式 (5) の計算は `Factor(j,j)` で行われ，それ以外の要素を求める式 (6) は `Factor(i,j)` で処理される．最も内側の k のループは，ベクトルの内積計算としてそれぞれの `Factor` の中に含まれる．図 1 の処理手順では，行列中の下三角成分の先頭列から順に，対角要素を計算した後に i のループによって同じ j 列にある非対角要素を計算する．

2.3 疎な連立方程式の求解過程

有限要素法に基づく連立方程式のように，行列 A が疎行列となるような工学的問題は非常に多い．そのような連立方程式を直接法を用いて解く場合，途中の処理過程を工夫することによって，全体の計算量や所要記憶容量を大幅に削減することができる．一連の処理過程は次のように表される[1]．

(1) Ordering: 行列 A の行と列を並べ替える．
(2) Symbolic factorization: 行列 L の構造を求める．
(3) Numeric factorization: 行列 L の値を求める．
(4) Triangular solution: 行列 L から解 $\boldsymbol{x}$ を求める．

本論文では Ordering や Symbolic factorization の結果はすでに得られているものとして，コレスキー分解の数値計算過程である Numeric factorization の効率的手法について検討する．

2.4 並列コレスキー分解に関する従来の研究

この節ではコレスキー分解の並列処理に関する主要な研究を概観する．Dongarra ら[2] はコレスキー分解を含む密行列用の種々の行列演算ルーチンの並列化を行っている．George ら[3] は，コレスキー分解のループ構成に着目し，column-Cholesky と命名した jki 型のループ構成を有望な手法として選択し，さらに疎行列への対応[4] を行っている．Zhang ら[5] はこの column-Cholesky アルゴリズムと，これに負荷の均等化を図るための改良を取り入れた手法[6]，さらに fan-in アルゴリズム[7]，およびその改良版である compute-ahead fan-in アルゴリズム[8] の 4 つの手法について性能比較を行っている．Ng ら[9] は supernode の概念を取り入れることで性能改善を図っている．Rothberg ら[10] は疎行列を密な部分行列の集合として扱うブロック fan-out 手法を提唱している．Duff らによるマルチフロンタル法[11] は，消去木の構造に基づいて，フロンタル行列と呼ばれる密三角行列の構築と更新計算を繰り返すことで疎行列のコレスキー分解を遂行する．寒川[12] は疎行列をスカイライン形式で表現可能な小行列の集合として扱う多重スカイライン法を提案している．

2.5 ループ構成と演算形態

前述の各手法のうち，column-Cholesky と fan-in アルゴリズムは jki 型，fan-out アルゴリズムとマルチフロンタル法は kji 型のループ構成をとる．これらの内側ループにおける演算形態はともに，スカラ・ベクトル積のベクトルへの加算，すなわち BLAS ルーチンにおける daxpy 形式となる．一方，図 1 に示した jik 型ではベクトルの内積，すなわち BLAS ルーチンにおける ddot 形式となる．

daxpy 形式の演算はベクトルプロセッサに適したコードとなる．しかしメモリ演算と浮動小数点演算の比率の関係から RISC プロセッサでは必ずしも効率的とはいえず，一般には ddot 形式の方が効率的に処理で

きる[13]．ループ・アンローリングによってその演算比率は変動するが ddot 形式の優位性は変わらない．したがって，RISC プロセッサ上で効率的な処理を行うには，主要な演算が ddot 形式となる jik 型ループ構成を選択する必要があるといえる．しかし，このループ構成を採用した手法は初期の文献 2) 以降，ほとんど議論されていない．そこで本論文では，RISC プロセッサに有利な jik 型ループとなるスカイライン法を基礎として，これを一般化することによって疎行列に対応させる方法を検討する．

3. スカイライン法とその一般化

3.1 スカイライン法

式 (6) 中の主要な計算は i 行と j 行の 2 つの行ベクトルの内積であり，先頭から $(j-1)$ 列までの要素が対象となる．内積のための個々の積和計算では，2 つの行ベクトル中の要素 ℓ_{ik}，ℓ_{jk} の少なくとも一方が 0 である場合，その積和計算は省略できる．そこで，対称行列 A の i 行に関して非零要素が始まる列位置を t_i で表すと，コレスキー分解を表す式 (5)，(6) はそれぞれ次のようになる．

$$\ell_{jj} = \sqrt{a_{jj} - \sum_{k=t_j}^{j-1} \ell_{jk}^2} \qquad (7)$$

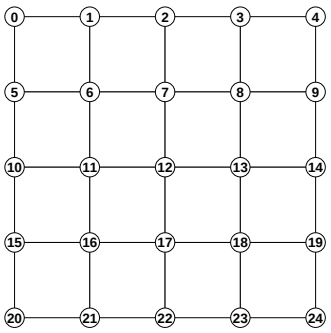
$$\ell_{ij} = \frac{1}{\ell_{jj}} \left(a_{ij} - \sum_{k=\max(t_i,\, t_j)}^{j-1} \ell_{ik}\ell_{jk} \right) \qquad (8)$$

これはスカイライン法としてよく知られている．この手法によって，下三角行列の先頭の列から連続して存在する零要素を計算対象から除外できる．このように行列中の計算範囲を狭めることが可能な理由は，行 i において fill-in が t_i 列の左側（若い列番号）では発生しないことに起因している．

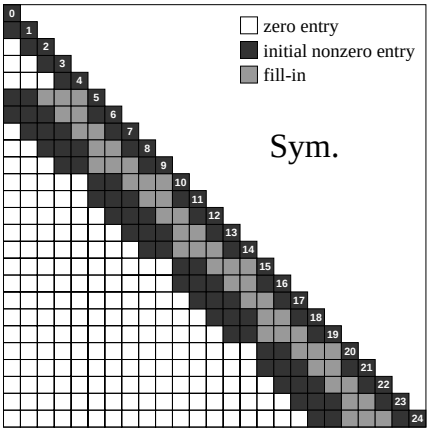
3.2 有限要素法と行列の形態

スカイライン法は，各行の先頭非零要素から対角要素までの範囲の大部分が fill-in を含めた非零要素で占められる場合に都合がよい．実際の有限要素問題において自然な節点番号付けを行った場合，行列は帯状となりスカイライン法が有効である．しかし節点番号付けの手法によっては非零成分が行列中に散在するようになるため，スカイライン法をそのまま適用することは不合理である．このことについて，以下に具体的な例をあげて解説する．

有限要素メッシュの節点番号付けと行列の形態（非零要素の分布）には密接な関係がある．ここでは説明を簡潔にするために，**図 2** (a) に示す 2 次元矩形領域



(a) Finite element grid by natural ordering.

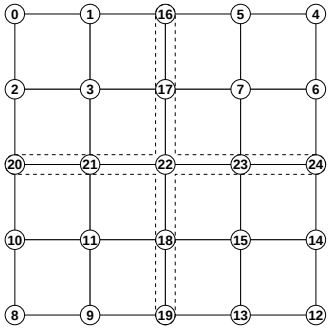


(b) Profile of the matix by natural ordering.

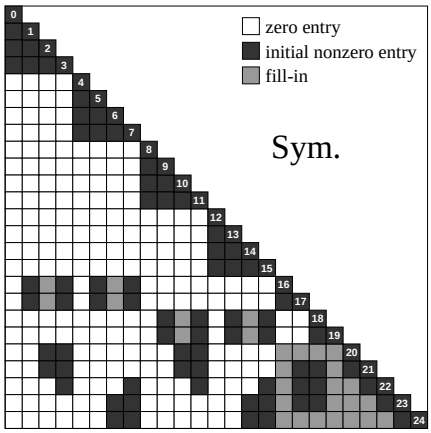
図 2 自然な番号付けによる有限要素メッシュとその行列の形態
Fig. 2 Finite element grid and its matrix by natural ordering.

を 4×4 に要素分割した単純な解析モデルを想定する．要素の種類としては 4 節点 4 辺形アイソパラメトリック要素とする．ここでは節点番号を節点の並びに沿って自然に番号付けしている．このときの剛性マトリックスの形態は図 2 (b) に示す帯行列となる．外側の大きな正方形は行列全体を意味し，対称性から下三角成分のみを表示している．白色の小四角形（以降，マスと表記）は行列 A や L の零要素を示す．黒色マスはコレスキー分解前の行列 A の非零要素を示し，灰色マスはコレスキー分解後に非零となる fill-in の位置を示している．このような帯行列の場合，各行の先頭非零要素から対角要素までの領域は，fill-in の発生によってすべて非零となることが分かる．

次に同様の有限要素モデルについて nested dissection[14] を応用した番号付けの例を**図 3** (a) に示す．ここでは解析対象を十字形の境界領域と残り 4 つの部分領域に分け，部分領域から先に数え上げた後に境界領域を番号付けしている．このときの行列の形態は



(a) Finite element grid ordered by nested dissection.



(b) Profile of the matix by nested dissection.

図 3 Nested dissection を適用した有限要素メッシュとその行列の形態

Fig. 3 Finite element grid and its matrix by nested dissection.

図 3 (b) のように非零要素が散在する状態となる．図 2 の自然な番号付けの場合と異なり，各行の先頭非零要素から対角要素の間に fill-in を生じない部分が存在する．この領域を計算対象から除外することによって，自然な番号付けの場合よりも所要記憶容量や計算量を削減することができる．

3.3 一般化スカイライン法

スカイライン法は，先頭の非零要素からその行の対角要素まではすべて非零と見なすため，図 3 (b) のように，行列中に非零成分が分散する場合には適さない．そのため本論文では，スカイライン法での疎行列の表現方法を拡張した一般化スカイライン法を提案する．

疎行列を表現するために，行列を構成する各行ベクトルについて，連続する非零要素の集まり（本論文では chunk と呼ぶ）の数と，それぞれの chunk の開始点と終了点を保持する方法を用いる．**図 4** (a) はある疎なベクトルを表している．図中のマスはベクトルの

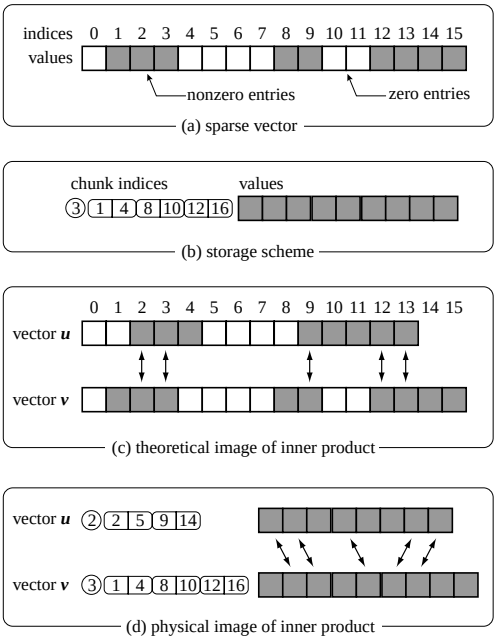


図 4 疎なベクトルの表現形式と内積

Fig. 4 Storage scheme of sparse vector and sparse inner product.

各要素を示している．白色マスは零要素，灰色マスは非零要素（fill-in を含む）を表している．それぞれのマスの上に記した整数はベクトル中の各要素の位置を表す添え字である．このベクトルが行列中の行ベクトルを表すとすれば，添え字は列番号に相当する．非零要素は，この図のように隣接し合って分布することが多いため，その性質を利用することで効率的な表現が可能になる．図 4 (b) は，(a) のベクトルの格納方法を示している．この格納形式は，非零要素の分布を表す整数データ（chunk index）と個々の非零要素の値を示す実数データ（value）から構成される．整数データとしては，ベクトル中の chunk 数とそれぞれの chunk の開始位置と終端位置（零要素の開始位置）を順に格納する．実数データとしては非零要素の値を順に格納する．

図 4 (c) は 2 つの疎なベクトルの内積計算の理論的イメージを示している．同じ添え字を持つ 2 つの要素について，どちらか一方もしくは両方が零要素の場合は，それらの積和計算は省略する．図 4 (d) は先述の格納方法を用いたときの内積計算の物理的イメージを示している．非零要素の位置情報をもとに積和計算が必要な実数データの位置を求めてから，積和計算を行って内積を得る．なお，chunk にはあらかじめ fill-in の領域を含めておくため，コレスキー分解の進行に従っ

```
    for(j=0; j<n; j++) {
        i0 = MyFirst(tid,j);
        if (i0 == j) {
            Factor(j,j);
            i0 += nth;
        }
        Barrier;
        for(i=i0; i<n; i+=nth) {
            Factor(i,j);
        }
    }
```

図 5 バリアを用いた並列コレスキー分解
Fig. 5 Parallel Cholesky factorization with barrier.

て chunk の位置情報を更新する必要はない．このデータ構造に対応した内積計算コードを図 1 中の Factor に組み込むことによって，与えられた疎行列に対して必要最小限の計算量でコレスキー分解の逐次処理が可能となる．

4. コレスキー分解の並列処理手法

ここでは共有メモリ型並列計算機の使用を想定し，マルチスレッドによるコレスキー分解の並列処理を検討する．まずスレッド間の同期方法が異なる 2 つの手法を示し，さらにブロック化の方法について解説する．

4.1 バリアによる同期

図 1 で示したコレスキー分解の逐次処理を，バリア同期を用いて並列化した例を**図 5** に示す．この図に示したコード部分は複数のスレッドで並列実行される．それぞれのスレッドにおける処理内容はスレッド ID（図中の tid）によって差別化される．計算対象となる正方行列の下三角成分に対し，先頭の列から順番に処理する．各スレッドへの処理領域の割当ては行単位で静的かつサイクリックに行っている．

MyFirst(tid,j) はスレッド tid に関する j 行以降の最初の担当行番号を返す．その行番号が j と等しければ，そのスレッドは Factor(j,j) で対角要素を計算する．その次に配置した Barrier によって全スレッドが足並みを揃える．その後 i のループによって Factor(i,j) で非対角要素の計算を行う．i の増分はスレッド数 nth となっている．これは各スレッドに担当行をサイクリックに割り当てているため，その間隔がスレッド数と等しくなることに起因する．

非対角要素の計算では，その列の対角要素（分解計算後の値）を参照する．ここで用いたバリアは，その列の対角要素の分解計算を担当しないスレッドに対して，非対角要素の計算時に参照する対角要素の計算が完了していることを保証するものである．Factor(i,j) の計算では，j 行の先頭から j 列目（対角要素）までの行ベクトルと i 行の先頭から j-1 列目までの行ベクトルを参照する．このとき j 行の対角要素の計算が完了しているならば，同じ行中の他の要素も計算が完了していることは自明である．また，i 行に関しては自己スレッドの担当範囲なので，Factor(i,j) を実行する時点で，先頭から j-1 列目までの計算が完了していることは自明である．したがってこの計算手順では，非対角要素の計算開始時には，その列の対角要素の計算完了のみを保証すればよいことが分かる．

このようにバリアを用いる手法はループ処理の並列化として標準的なものである．自動並列化コンパイラを用いれば，同様の実行コードを逐次処理ソースコードから生成可能であると考えられる．しかし，このようなバリアによる同期は必ずしも効率が良いとはいえない．なぜなら，対角要素を計算し終えたスレッドが他のスレッドに先行してバリアに到達した場合，数値計算上はその列に関するバリアが不要になるにもかかわらず，そのスレッドは残りのスレッドがすべてバリアに到達するまでそのまま待ち続けることになるためである．この問題点を解消する手法を次節で示す．

4.2 イベント カウントによる同期

バリアによる同期の欠点を解消するため，イベントカウント[15] による同期を用いたコレスキー分解の並列処理手法[16] を導入する．**図 6** はイベントカウントによる並列コレスキー分解の構成例である．ここに示した変数 E はイベントカウントであり，操作関数 Advance や Await によって，すべてのスレッドから更新や参照が可能である．E は -1 に初期化され，対角要素の分解計算が完了した時点で Advance(E) によってインクリメントされる．換言すれば，E の値はつねに対角要素の分解計算が完了した列番号を示している．Await(E,j) は E の値が j 以上になるまで待機する．コードを順に見ていくと，まず ID 番号 0 のスレッドが最初の対角要素を計算し Advance する．各スレッドは，j に関するループ内で Await(E,j) によって j 列の対角要素が計算されるのを待つ．次に，(j+1) 行を担当するスレッドは，条件 (i0 == j+1) が成立するため，Factor(j+1,j) によって j 列の最初の非対角要素を計算した後，続けて Factor(j+1,j+1) によって右に隣接する対角要素をただちに計算し，Advance する．その他のスレッドおよび上記の処理を終えたスレッドは，i のループ内にある Factor(i,j) によって j 列の非対角要素を計算する．

```
if (tid == 0) {
    Factor(0,0);
    Advance(E);
}
for(j=0; j<n-1; j++) {
    Await(E,j);
    i0 = MyFirst(tid,j+1);
    if (i0 == j+1) {
        Factor(j+1,j);
        Factor(j+1,j+1);
        Advance(E);
        i0 += nth;
    }
    for(i=i0; i<n; i+=nth) {
        Factor(i,j);
    }
}
```

図 6 イベントカウントを用いた並列コレスキー分解
Fig. 6 Parallel Cholesky factorization with eventcount.

この手法の特長は，対角要素の計算が可能になった時点で，残りの非対角要素の計算を行う前にその対角要素の計算を先行させることにある．これによって，他のスレッドは待ち状態から早期に開放される．

なお，自動並列化コンパイラによって逐次処理用のコレスキー分解コードからこのような並列化コードを自動生成することは現状では困難である．そのため本論文では，マルチスレッド・プログラミングにより図示の並列処理手続きを明示的に記述している．

4.3 ブロック化

近年の RISC 型計算機は，プロセッサ内に豊富な浮動小数点レジスタを持つとともに，主記憶との間に高速なキャッシュメモリを有する．これらを有効利用する手法はブロック化として知られている．レジスタの有効利用には外側ループ・アンローリング，キャッシュの有効利用には外側ループ・ストリップマイニングが用いられる[17]．コレスキー分解は 3 重のループから構成されるので，外側および中間のループに対してこれらを適用する．

有限要素解析では，扱う問題によって各節点あたりの自由度が異なるが，3 次元問題で 1 節点あたりの自由度が 3 となる場合には，行列全体の非零成分は 3×3 の小行列単位で分布する．したがって，この小行列を処理単位とすることによって疎行列の構造を表すデータ量を削減できると同時に，このサイズでの外側ループ・アンローリングを行うことで浮動小数点レジスタの有効利用が図られる．この場合，図 4 に示した疎なベクトルは，chunk の位置情報を節点番号で表現することができる．さらに 図 4 (d) に示した内積にともなう積和計算は，3×3 の小行列を単位とする行列積和計算に置き換えられる．

キャッシュメモリを有効利用するための外側ループ・ストリップマイニングは，さらに大きな部分正方行列を単位として行う．本論文では，その部分行列の大きさ（行数）をブロックサイズと呼び，コレスキー分解の計算中はつねに一定のブロックサイズを用いる．最適なブロックサイズは，使用する計算機や扱う行列の規模あるいは疎行列中の非零要素の分布形態によって変動するため，計算実行時にこれを変数として最適な値を与えるものとする．

このブロック化手法は密行列の数値計算で通常行われるブロック化と同等であり，非常に簡潔である．疎行列の場合には，行列を構成する各ベクトルの非零要素の含有率によって，実質的にアクセスされるメモリ領域の範囲が変動する．そのため，コレスキー分解中のキャッシュ占有率とブロックサイズの関係が一定ではない．しかし次章に示す実験結果によって，効率的な処理のためには疎行列の場合でもブロックサイズの適切な選択が有効であることが分かる．

なお，先述した各同期手法による並列処理アルゴリズムでは，`Factor` 内で行われる分解計算は行列中の 1 要素が単位となっている．これに改良を加え，3×3 の小行列を単位とする外側ループ・アンローリングと，ブロックサイズを単位とする外側ループ・ストリップマイニングを導入することによって，コレスキー分解全体をブロック化することができる．この場合，各スレッドへの行ベクトルの割当ては，ブロックサイズに等しい行数ごとにサイクリックに行われることになる．

5. 性能評価実験

本報に示した並列処理手法の性能評価では，計算機として Sun Enterprise 450 を使用した．本機は 400 MHz の UltraSPARC II プロセッサを 4 基搭載した共有メモリ型の対称型マルチプロセッサ・システムである．浮動小数点演算の理論ピーク性能は 1 プロセッサで 800 Mflop/s，4 プロセッサで 3,200 Mflop/s となっている．また，数値計算の性能評価指標として定評のある LINPACK ベンチマークの TPP 値[18] は，1 プロセッサで 552 Mflop/s，4 プロセッサで 1,841 Mflop/s と報告されている．

並列処理性能の評価に用いた連立方程式は，3 次元有限要素法で導出される剛性方程式を用いる．有限要

表 1 評価に用いた連立方程式データ
Table 1 Statistics of data sets for evaluation.

データ名	各辺分割数 N	次数 n	密度 %	計算量 Gflop
D30L0	30	89,373	6.46	759.35
D30L1	30	89,373	3.28	242.37
D30L2	30	89,373	2.20	159.36
D30L3	30	89,373	1.93	151.33
D30L4	30	89,373	1.92	151.33

素モデルとしては，最も基本的な 3 次元形状である立方体を想定し，各辺とも均等な要素分割とする．

要素タイプとして 8 節点 6 面体アイソパラメトリック要素を採用する．各節点の自由度は 3 である．節点番号付けの手法としては nested dissection を適用する．評価に用いたデータの一覧は**表 1** のとおりである．データ名に含まれる L0〜L4 の指標は，nested dissection における nesting のレベルを表し，指標の数字が大きいほど nesting が深い．ただし L0 は nested dissection を適用せずに自然な番号付けを行ったものである．このとき連立方程式の係数行列は帯行列となる．L1 は立方体領域を 8 つの部分領域（立方体）とそれらの境界領域に分割して番号付けしたものである．L2 はそれらの部分立方体に対してさらに分割したものであり，L3 や L4 は同様の手続きを再帰的に施したものである．また，この表中の密度とは，行列中の全要素に対する fill-in を含めた非零要素の割合である．計算量とは，コレスキー分解の際に必要な浮動小数点演算の総数である．なお本論文では，計算量を flop，計算速度を flop/s で表す．

5.1 帯行列に対する処理性能

RISC プロセッサを使用する数値計算では，プログラミングの技法や種々のコンパイルオプションの有無等によって実行速度が大きく変化する．そのため数値計算ソフトウェアを作成して並列処理性能を論じる場合，逐次処理時について，使用する計算機本来の性能を発揮しているかどうかをあらかじめ確認しておく必要がある．この節では，まず作成した一般化スカイライン法の実行コードの演算性能を確認するため，行列が単純な帯状となるデータ D30L0 に対して演算性能を計測した．結果を**図 7** に示す．1 プロセッサ使用時（逐次処理）においてはブロックサイズ 174 のときに最大性能 511 Mflop/s が得られた．この値は TPP 値と比較して遜色なく，逐次処理コードとして計算機本来の性能を十分に発揮していると考えられる．

並列処理においては，バリアを使用する場合と比較して，イベントカウントを使用する場合の方がブロックサイズの変化に対して広い範囲で安定した高い性能を示している．4 プロセッサ使用時には最大 1,955 Mflop/s を得ており，これは TPP 値（1,841 Mflop/s）を上回るものである．

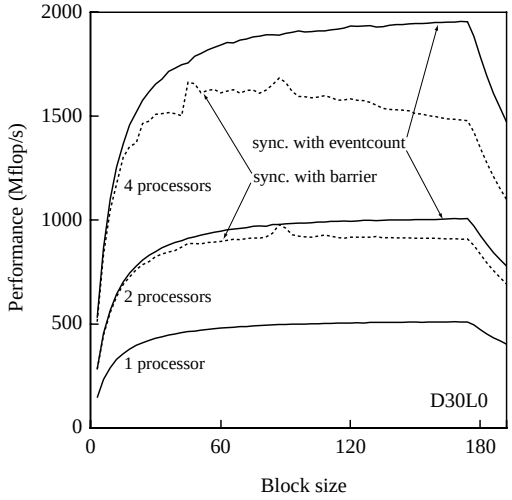


図 7 帯行列に対するコレスキー分解時の浮動小数点演算性能
Fig. 7 Floating-point performance on Cholesky factorization of band matrix.

5.2 Nesting の深さと処理速度

nested dissection によって計算量の削減を図ることができるが，nesting が深くなるにつれて疎行列中の非零要素が細かく散在するようになる．そのため配列データの位置決めにともなうオーバヘッドが増大し，処理効率の低下が予想される．そこで，異なる深さの nesting から得られる疎行列データに関して，4 並列でコレスキー分解を実行した結果を**図 8** に示す．ここでは，自然な節点番号付けによるデータ D30L0 について最適なブロックサイズを選択した際の処理速度を基準値 1 としている．本図から，nested dissection の適用によって演算速度（Mflop/s 値）は低下するが，計算量の削減効果によって実質的な処理速度は数倍に向上していることが分かる．

5.3 他の手法との性能比較

ここでは，本論文で提案する一般化スカイライン法と既存の他の手法との性能比較を行う．一般化スカイライン法では，並列処理の同期機構としてバリアを用いたものとイベントカウントを用いたものとを評価する．また，比較対象として大規模疎行列の直接解法の中で，特に近年有望視されているマルチフロンタル法について評価する．同手法を実装したソフトウェアとして MUMPS パッケージ[19] を使用する．行列データ D30L3 に対してコレスキー分解を実行する速度について，一般化スカイライン法の逐次処理速度を基準に

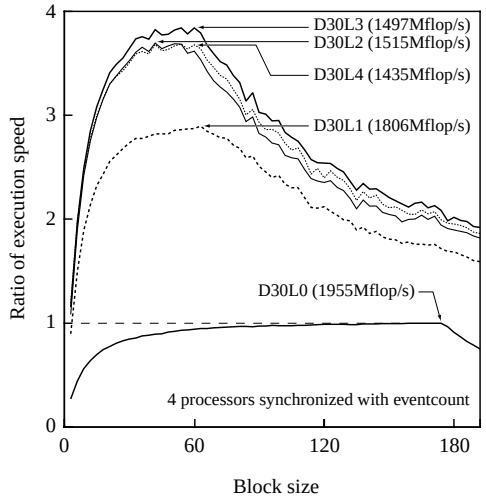


図 8 疎行列の形態と並列コレスキー分解の処理性能
Fig. 8 Performance of parallel Cholesky factorization to various sparse matrices.

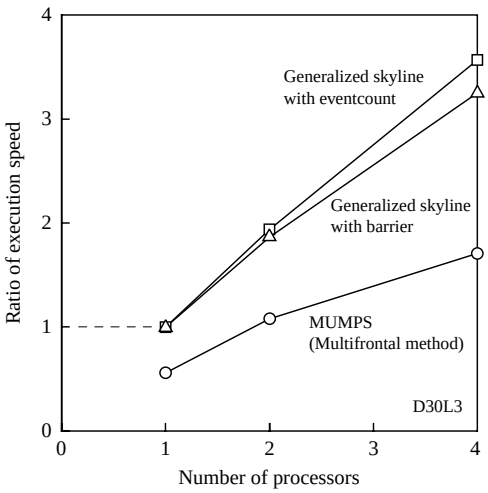


図 9 疎行列コレスキー分解の各手法の性能比較
Fig. 9 Performance comparison of each technique for sparse Cholesky factorization.

して相対比で表した結果を**図 9** に示す．ここでは，ブロックサイズはそれぞれ最適な値を選択している．イベントカウントを使用した一般化スカイライン法は，バリアを使用したものと比較して並列数が大きいときにより高い性能を示している．逐次処理時の速度を基準にすると，バリアを用いた場合に 4 並列で 3.3 倍となったのに対し，イベントカウントを用いた場合には 3.6 倍となっている．

一方，マルチフロンタル法は逐次処理，並列処理ともに一般化スカイライン法よりも低い性能を示している．マルチフロンタル法ではフロンタル行列と呼ばれる密な三角行列を構成し，更新計算を行った後に新たなフロンタル行列を再構成することによってコレスキー分解を遂行する．そのためメモリ中のデータ移動が頻発し，性能の低下を招いていると考えられる．一般化スカイライン法では疎行列の全体構造をメモリ中に静的に確保するため，このようなデータ移動は不要となり，効率が良いといえる．

6. おわりに

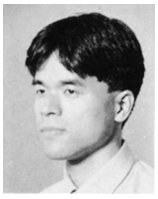
本論文では，大規模で疎な連立方程式を解くためにスカイライン法を一般化したコレスキー分解手法を提案し，評価を行った．通常のスカイライン法では，疎行列の構造を表すために各行における先頭の非零要素の位置情報のみを保持するが，一般化スカイライン法では行列中に散在する非零要素の分布を完全に表現する．そのため nested dissection 等の順序付け手法によって計算量や記憶容量の削減を図ることができる．また有力な手法の 1 つであるマルチフロンタル法と異なり，疎行列を主記憶中に静的に確保しているため，数値データの移動にともなう効率低下を避けることができる．さらに並列処理においては，プロセッサ間の同期手法としてイベントカウントを採用したことによって，ループ計算の単純な並列化に用いられるバリアよりも効率的な処理を実現した．Sun Enterprise 450 を使用した性能評価では，マルチフロンタル法と比較して逐次処理・並列処理ともに，より高速に実行されることを確認した．今後はその他の有力な疎行列ソルバとの性能比較を行うとともに，より並列数の大きい分散メモリ型並列計算機への実装を検討していきたい．

参考文献

1) Heath, M.T., Ng, E. and Peyton, B.W.: Parallel Algorithms for Sparse Linear Systems, *SIAM Review*, Vol.33, No.3, pp.420–460 (1991).
2) Dongarra, J.J. and Hiromoto, R.E.: A Collection of Parallel Linear Equation Routines for the Denelcor HEP, *Parallel Computing*, Vol.1, pp.133–142 (1984).
3) George, A., Heath, M.T. and Liu, J.: Parallel Cholesky Factorization on a Shared-Memory Multiprocessor, *Linear Algebra and its Applications*, Vol.77, pp.165–187 (1986).
4) George, A., Heath, M.T., Liu, J. and Ng, E.: Solution of Sparse Positive Definite Systems on a Shared-Memory Multiprocessor, *International Journal of Parallel Programming*, Vol.15, No.4, pp.309–325 (1986).
5) Zhang, G. and Elman, H.C.: Parallel Sparse

Cholesky Factorization on a Shared Memory Multiprocessor, *Parallel Computing*, Vol.18, No.9, pp.1009–1022 (1992).
6) Geist, G.A. and Ng, E.: Task Scheduling for Parallel Sparse Cholesky Factorization, *International Journal of Parallel Programming*, Vol.18, No.4, pp.291–314 (1989).
7) Ashcraft, C., Eisenstat, S.C. and Liu, J.W.H.: A Fan-In Algorithm for Distributed Sparse Numerical Factorization, *SIAM Journal on Scientific and Statistical Computing*, Vol.11, No.3, pp.593–599 (1990).
8) Ashcraft, C., Eisenstat, S.C., Liu, J.W.H., Peyton, B. and Sherman, A.H.: A Compute-Ahead Implementation of the Fan-In Sparse Distributed Factorization Scheme, Tech. Report ORNL-TM-11496, Oak Ridge National Laboratory (1990).
9) Ng, E. and Payton, B.W.: A Supernodal Cholesky Factorization Algorithm for Shared-Memory Multiprocessors, *SIAM Journal of Scientific Computing*, Vol.14, pp.761–769 (1993).
10) Rothberg, E. and Gupta, A.: An Efficient Block-Oriented Approach to Parallel Sparse Cholesky Factorization, *SIAM Journal on Scientific Computing*, Vol.15, No.6, pp.1413–1439 (1994).
11) Duff, I.S. and Reid, J.K.: The Multifrontal Solution of Indefinite Sparse Symmetric Linear Equations, *ACM Trans. Math. Software*, Vol.9, No.3, pp.302–325 (1983).
12) 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).
13) Dowd, K.: *High Performance Computing*, O'Reilly & Assicuates, Inc. (1993).
14) George, A.: Nested Dissection of a Regular Finite Element Mesh, *SIAM Journal on Numerical Analysis*, Vol.10, No.2, pp.345–363 (1973).
15) Reed, D.P. and Kanodia, R.K.: Synchronisation with Eventcounts and Sequencers, *Comm. ACM*, Vol.22, No.2, pp.115–123 (1979).
16) 宮川佳夫，松田　章，加藤　隆：対称型マルチプロセッサによるコレスキー分解の並列処理，情報処理学会論文誌，Vol.38, No.1, pp.1–8 (1997).
17) 寒川　光：RISC 超高速化プログラミング技法，共立出版 (1995).
18) Dongarra, J.J.: Performance of Various Computers Using Standard Linear Equations Software, Technical Report CS-89-85, University of Tennessee (2000).
19) Amestoy, P., Duff, I. and L'Excellent, J.-Y.: Multifrontal Parallel Distributed Symmetric and Unsymmetric Solvers, *Comput. Methods in Appl. Mech. Eng.*, Vol.184, pp.501–520 (2000).

(平成 12 年 8 月 30 日受付)
(平成 13 年 3 月 9 日採録)



宮川　佳夫（正会員）

1990 年広島大学大学院工学研究科（設計工学専攻）博士課程前期修了．同年広島大学工学部助手．1993 年より岡山県立大学情報工学部助手，現在に至る．日本機械学会，日本塑性加工学会，日本シミュレーション学会各会員．

松田　　章（正会員）

1980 年山梨大学大学院（計算機科学専攻）修士課程修了．同年（株）日立製作所入社．岐阜大学，通産省工業技術院名古屋工業技術研究所を経て，1994 年岡山県立大学情報工学部講師，現在に至る．博士（工学）．人工知能学会，日本塑性加工学会，品質管理学会各会員．

加藤　　隆（正会員）

1975 年名古屋大学大学院工学研究科博士課程修了．同年名古屋大学工学部助手．1987 年広島大学工学部助教授．1990 年通産省工業技術院名古屋工業技術研究所主任研究官．1993 年岡山県立大学情報工学部教授，現在に至る．工学博士．日本機械学会，日本塑性加工学会各会員．