One-way dissection オーダリングによる連立一次方程式の直接解法の並列化

中野 智輝[1,a)]　横川 三津夫[1]　深谷 猛[2]　山本 有作[3]

概要：近年，導電性高分子がウェアラブルデバイスとして注目されている．安定した導電性を持つ素材を設計するためには，導電性高分子の電子状態を求めることが重要である．この支配方程式は，時間依存シュレディンガー方程式であるが，ある離散化により連立一次方程式を解く問題に帰着される．
本稿では，この方程式のモデル問題として２次元ポアソン方程式を取り上げ，one-way dissection オーダリングによる正定値対称疎行列を係数行列にもつ連立一次方程式の並列直接解法に対して，いくつかの疎行列格納方式を用いた場合の性能評価結果について述べる．また，新しいスカイライン格納方式を提案し，その有効性を確認した．

キーワード：並列解法，ポアソン方程式，コレスキー分解，疎行列格納形式

Parallelization of the direct solver for system of linear equations by one-way dissection ordering

Tomoki Nakano[1,a)]　Mitsuo Yokokawa[1]　Takeshi Fukaya[2]　Yusaku Yamamoto[3]

Abstract: In recent years, conductive polymers have attracted a lot of attention as materals of wearable devices. It is required to make clear electronic states of the conductive polymers in order to design materials which have electrically stable conductivities. The electronic states are represented by the time-dependent Schrödinger equation and a linear system of equations is derived from its discretization.
In this paper, we considered the two-dimensional Poisson's equation as a model problem. We applied two sparse matrix storage formats, or CCS format and a new skyline-type format, to hold the coefficient matrix of a linear system of equations which is obtained by discretization of the Poisson's equation with one-way dissection ordering, where the coefficient matrix is sparse, symmetric, and positive definite. The performance of the formats was evaluated and the new formant was found to be efficeint.

Keywords: Parallel computation, Poisson equation, Cholesky decomposition, sparse matrix storage format

1. はじめに

数値シミュレーションにおいては，偏微分方程式を離散化して大規模な連立一次方程式を解く問題に帰着させることが多い．また，連立一次方程式を解く時間は全体のシミュレーションの大部分を占めることが多い．よって，連立一次方程式を高速に解くことは非常に重要である．一般的に，シミュレーションの方法によって，行列のサイズや性質が大きく異なる．また，同じ係数行列を繰り返し解く問題や一度しか解かない問題など問題設定も様々である．一方，近年の計算機のアーキテクチャは多様化・複雑化していて，同じ問題でも計算機によって最速となる解法が異なることがある．よって，対象とする問題と使用する計算機の特性の両方を考慮したある種の専用解法を考えることはシミュレーションを効率化するための一つの有効な手段である．

現在，我々は，導電性高分子材料のシミュレーションで

[1] 神戸大学
[2] 北海道大学
[3] 電気通信大学
[a)] 1445065t@stu.kobe-u.ac.jp

ⓒ 2017 Information Processing Society of Japan

現れる連立一次方程式を効率的に解くことを研究している．具体的には，問題の物理的な特徴に由来する疎行列の疎構造，シミュレーションの手法に由来する問題設定，そして，近年のマルチコア CPU の特徴，の三点を踏まえて，one-way dissection オーダリングを用いた疎行列向けの並列直接解法を検討している．本稿では，上述の手法を実際の問題二適用するための前段階として，類似形状を持つシンプルな疎行列（細長い領域上のポアソン方程式に由来する行列）を用いて行った手法の高速化に関する実験結果を報告する．特に，疎行列の格納形式について，CCS(Compressed Column Storage) 形式とスカイライン格納方式に着目し，密行列形式を含めた三種類の実装を行い，それぞれの性能を評価する．

2. 導電性高分子材料のシミュレーションの概要

1976 年に導電性高分子が発明された [1]．通常は絶縁体，もしくは半導体だが，ある種の分子をドーピングすると，電気を通すことができる物質である．導電性高分子は柔らかい性質を持つため，ウェアラブルデバイスに用いるなど，様々な工学的応用が期待されている．

導電性高分子の導電性は，導電性高分子を構成する π 電子の移動によるものである．一般的に，導電性高分子の構造は乱雑で電子が移動し難いため，この乱雑さを抑えることが求められている [2]．そのためには，高分子の電子状態を知ることが必要であり，計算機シミュレーションで時間発展のシュレディンガー方程式

$$\frac{\partial \psi}{\partial t} = -i\mathcal{H}\psi \qquad (1)$$

を解く必要がある．ここで，$\mathcal{H}$ はハミルトニアンと呼ばれるエルミート作用素である．これを基底関数によって離散化すると，次のような連立常微分方程式が得られる．

$$\frac{\partial \mathbf{x}}{\partial t} = -iH\mathbf{x} \qquad (2)$$

このとき，H は正定値エルミート疎行列である．H は一般にブロック三重対角行列であり，例えば，PPE（ポリフェニレンエチニレン）の場合，行列 H の先頭 180×180 の非零要素形状は**図 1** のようになる[*1]．式 (2) を以下のように Crank-Nicolson 法によって離散化する．

$$\frac{\mathbf{x}(t+\Delta t)-\mathbf{x}(t)}{\Delta t} = \frac{1}{2}\{iH\mathbf{x}(t) + iH\mathbf{x}(t+\Delta t)\} \qquad (3)$$

これを変形すると，

$$\left(I - \frac{1}{2}iH\Delta t\right)\mathbf{x}(t+\Delta t) = \left(I + \frac{1}{2}iH\Delta t\right)\mathbf{x}(t) \qquad (4)$$

となり，$\left(I - \frac{1}{2}iH\Delta t\right)$ を係数行列，$\left(I + \frac{1}{2}iH\Delta t\right)\mathbf{x}(t)$ を右辺ベクトルとする連立一次方程式をタイムステップごとに解くことによって，電子状態を計算することができる．未知ベクトル $\mathbf{x}(t)$ の次元は原子の数にほぼ比例するため，高分子では多大な計算時間を要する．

この方程式は既に反復法によって解かれているが [4]，最近のマルチコア・メニーコア CPU に対しては，この疎行列の性質を利用した並列の直接解法によって効率的に解くことが期待できる．また，シュレディンガー方程式を原子位置を固定して時間発展させた場合，係数行列が一定で右辺ベクトルのみが変わるので係数行列の構造を利用すれば，複素対称コレスキー分解により，高速に解けることが期待される．以上の背景により，我々は，導電性高分子のシミュレーションで現れる連立一次方程式を，並列直接解法で効率的に解くことを研究している．

*1 この行列データは ELSES[3] によって作成されたものである．

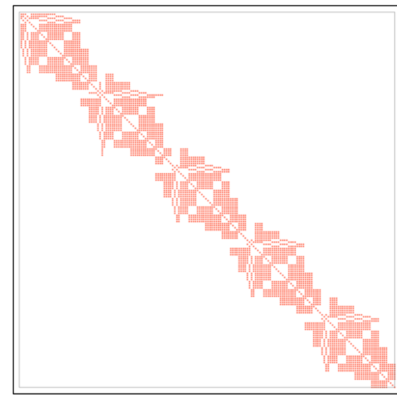


図 1: Pattern of nonzero elements in the top-left 180 × 180 part of H

3. 疎行列直接解法

以下では，疎行列直接解法について，2 次元ポアソン方程式を 5 点差分近似で離散化した場合に現れる連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を行列分解を用いて解くことを例に説明する．係数行列 A は正定値対称なので，コレスキー分解を用いる．手順としては，まず $A = LL^{\mathrm{T}}$ と分解を行う．そして，$L\boldsymbol{y} = \boldsymbol{b}$, $L^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{y}$ を順に解くことで解 $\boldsymbol{x}$ を求める（前進後退代入）．したがって，数学的には密行列に対する直接解法の場合と何ら変わりはない．

しかし，数値計算を行うプログラムとしては，疎行列の疎性を考慮する．つまり，フィルインも考慮した非零要素のみを記憶し演算する必要がある．そのため，分解後の非零要素の位置を先立って計算する必要がある．これをシンボリック分解という．また，実際の数値をコレスキー分解することを数値分解という．コレスキー分解前に行列の行と列を置換する適切なオーダリングにより，フィルインを減らし，並列計算を可能にすることができる．まとめると，疎行列直接解法は一般的に以下の 4 つの処理を順に行うことにより解を求めることができる．

(1) オーダリング
(2) シンボリック分解

ⓒ 2017 Information Processing Society of Japan

Vol.2017-HPC-162 No.19
2017/12/19

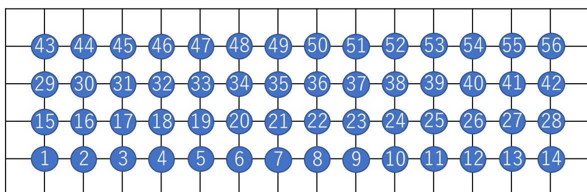


図 2: Natural ordering

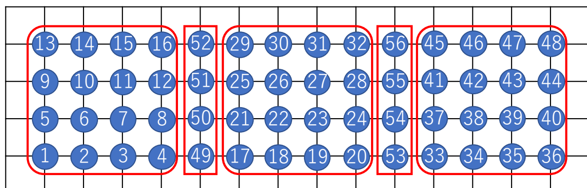
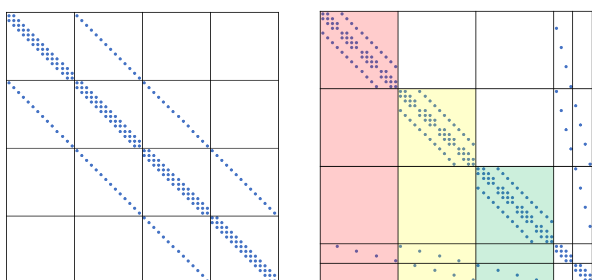


図 3: One-way dissection ordering



(a) Natural ordering (b) One-way dissection ordering

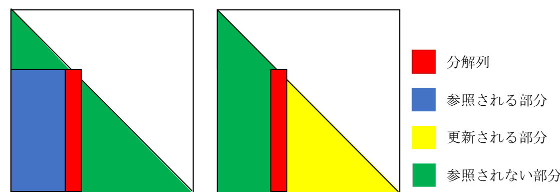
図 4: Pattern of nonzer elements in the coefficent matrix

(3) 数値分解
(4) 前進後退代入

原子位置を固定して時間発展計算をする場合のように係数行列が不変で右辺ベクトルのみが変わる場合は，(1)，(2)，(3) を一度行い，各右辺ベクトルに対して (4) を行うだけでよい．以下の各節では，上記の 4 つの処理について，より詳しく述べる．

3.1 オーダリング

2 次元の計算領域を離散化し，**図 2** のように格子に番号付けした場合，係数行列の非零要素形状は**図 4**(a) のようになる．この係数行列のコレスキー分解には，変数の依存関係があるため，並列計算ができない．並列計算を可能にするオーダリング方法として，one-way dessection オーダリング [5] が知られている．このオーダリングは，まず全領域をいくつかのサブ領域に分け，分かれた領域ごとに節点番号をつけていき，最後に境界領域に節点番号をつけていくという方法である．**図 3** は，3 つのサブ領域に分けた場合である．このオーダリングを行った場合の非零要素形状は図 4 (b) のようになり，サブ領域はそれぞれ依存関係がないため，図中の色のついた部分はそれぞれ並列に計算できる．実際のコレスキー分解の手順は 3.3 節で説明する．



(a) Left-looking algorithm (b) Right-looking algorithm

図 5: Algorithms for computing Cholesky decomposition

3.2 シンボリック分解

シンボリック分解ではフィルインによる要素も含めた非零パターンのみに着目してコレスキー分解で現れる非零要素の位置を求める．定理 1 を用いることでシンボリック分解の計算コストを減らすことができる [6], [7].

定理 1 分解の過程での行列 A のすべてのフィルインを計算するためには，シンボリック分解の第 j ステップで第 w 番目の列のフィルインされる要素番号を決定すれば良い．ここで，w は j 列の非零要素の行番号の集合 $\mathcal{R}_j$ の最小要素番号である．

3.3 コレスキー分解（数値分解）

コレスキー分解の left-looking アルゴリズムと right-looking アルゴリズムについて述べる [8], [9]．Left-looking アルゴリズムでは，分解列より前の計算済みである列を参照し，計算する (**図 5**(a))．逆に，right-looking アルゴリズムでは，分解列より後の列を常に更新し，計算済みである分解列より前の列は参照されない (図 5(b))．本研究では疎行列に向いている right-looking アルゴリズムを用いる．

ここで，one-way dissection オーダリングを行った場合のブロックレベルでのコレスキー分解について説明する．**図 6** は空間を格子数 29×4 に離散化し，one-way dissection オーダリングにより格子点を 6 つのサブ領域に分割した場合のポアソン方程式に現れる係数行列の非零要素形状である．容量を節約するために，色のついた部分のブロック行列のみを保持する．ここで，行列 C_i は分解前では零行列であるが，コレスキー分解のフィルインにより非零成分が生成される．このときのコレスキー分解のアルゴリズムを **Algorithm 1** に示す．n_1 はサブ領域の数，n_2 は境界領域の数 $(= n_1 - 1)$ である．1 行目〜11 行目までの 3 つのループでは，各ループの内部では（i に関する）依存関係がないため，処理を並列に行うことが可能である．

3.4 前進後退代入

One-way dissection オーダリングを行った場合のブロックレベルでの前進後退代入のアルゴリズムについて説明する．前進代入のアルゴリズムを **Algorithm 2**，後退代入のアルゴリズムを **Algorithm 3** に示す．ただし，$\boldsymbol{b}$, $\boldsymbol{x}$ は

ⓒ 2017 Information Processing Society of Japan

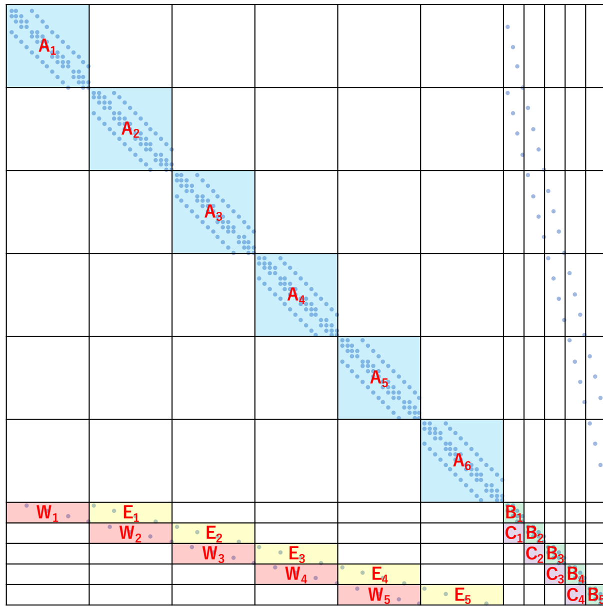


図 6: Pattern of nonzero elements in the coefficient matrix obtained by one-way dissection ordering with 6 subregions (The number of grid points: 29 x 4)

Algorithm 1: Cholesky decomposition for a matrix obtained by one-way dissection ordering

```
1  for i = 1, n1 do
2      decompose A_i
3  end
4  for i = 1, n2 do
5      decompose W_i using deomposed A_i
6      decompose E_i using deomposed A_{i+1}
7      B_i = B_i − W_i W_i^T − E_i E_i^T
8  end
9  for i = 1, n2 − 1 do
10     C_i = −W_{i+1} E_i^T
11 end
12 for i = 1, n2 − 1 do
13     decompose B_i
14     decompose C_i using decomposed B_i
15     B_{i+1} = B_{i+1} − C_i C_i^T
16 end
17 decompose B_{n2}
```

A の分割に合わせ，

$$\boldsymbol{b} = [\boldsymbol{b}_{1,1}^{\mathrm{T}},\ \boldsymbol{b}_{1,2}^{\mathrm{T}},\ \ldots,\ \boldsymbol{b}_{1,n_1}^{\mathrm{T}},\ \boldsymbol{b}_{2,1}^{\mathrm{T}},\ \boldsymbol{b}_{2,2}^{\mathrm{T}},\ \ldots,\ \boldsymbol{b}_{2,n_2}^{\mathrm{T}}]^{\mathrm{T}}$$
$$\boldsymbol{x} = [\boldsymbol{x}_{1,1}^{\mathrm{T}},\ \boldsymbol{x}_{1,2}^{\mathrm{T}},\ \ldots,\ \boldsymbol{x}_{1,n_1}^{\mathrm{T}},\ \boldsymbol{x}_{2,1}^{\mathrm{T}},\ \boldsymbol{x}_{2,2}^{\mathrm{T}},\ \ldots,\ \boldsymbol{x}_{2,n_2}^{\mathrm{T}}]^{\mathrm{T}}$$

とする．その他の文字の定義は **Algorithm 1** と同じである．**Algorithm 2** では 1 行目〜9 行目の 3 つのループ，**Algorithm 3** では 6 行目〜14 行目の 3 つのループがそれぞれ，ループ内部の処理を並列化可能となっている．

Algorithm 2: Forward substitution for a matrix obtained by **Algorithm 1**

```
1  for i = 1, n1 do
2      solve A_i x_{1,i} = b_{1,i}
3  end
4  for i = 1, n2 do
5      b_{2,i} = b_{2,i} − W_i x_{1,i}
6  end
7  for i = 1, n2 do
8      b_{2,i} = b_{2,i} − E_i x_{1,i+1}
9  end
10 for i = 1, n2 − 1 do
11     solve B_i x_{2,i} = b_{2,i}
12     b_{2,i+1} = b_{2,i+1} − C_i x_{2,i}
13 end
14 solve B_{n2} x_{2,s} = b_{2,s−1}
```

Algorithm 3: Backward substitution for a matrix obtained by **Algorithm 1**

```
1  solve B_{n2}^T x_{2,n2} = b_{s−1}
2  for i = n2 − 1, 1, −1 do
3      b_{2,i} = b_{2,i} − C_i^T b_{2,i+1}
4      solve B_i^T x_{2,i} = b_{2,i}
5  end
6  for i = 1, n2 do
7      b_{1,i} = b_{1,i} − W_i^T b_{2,i}
8  end
9  for i = 1, n2 do
10     b_{1,i+1} = b_{1,i+1} − E_i^T b_{2,i}
11 end
12 for i = 1, n1 do
13     solve A_i^T x_{1,i} = b_{1,i}
14 end
```

4. 疎行列格納形式

本稿では，前節で説明した疎行列直接解法を適用する場合の疎行列の格納方法に主眼を置く．そこで，本節では，本研究で検討の対象とする，CCS 形式，スカイライン格納方式について述べる．なお，本稿で考えている方程式の係数行列には，図 6 から分かるように，対称部分行列 A_i, B_i と非対称部分行列 W_i, E_i, C_i が含まれる．よって，以下では，次の対称行列 M，一般行列 N を例として説明する．

$$M = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 4 & 5 \\ 2 & 4 & 6 & 0 \\ 0 & 5 & 0 & 7 \end{bmatrix},\ N = \begin{bmatrix} 1 & 3 & 0 & 8 \\ 2 & 4 & 6 & 9 \\ 0 & 0 & 7 & 0 \\ 0 & 5 & 0 & 0 \end{bmatrix} \qquad (5)$$

4.1 CCS 形式

CCS(Compuressed Column Storage) 形式とは，疎行列を列方向に走査し，零要素を省いて，非零要素のみを格納する格納形式である．行列 M, N を CCS 形式で格納した場合の例を示す．

- 行列 M
  - $val : [1, 2, 3, 4, 5, 6, 7]$
  - $row_ind : [1, 3, 2, 3, 4, 3, 4]$
  - $col_ptr : [1, 3, 6, 7, 8]$
- 行列 N
  - $val : [1, 2, 3, 4, 5, 6, 7, 8, 9]$
  - $row_ind : [1, 2, 1, 2, 4, 2, 3, 1, 2]$
  - $col_ptr : [1, 3, 6, 8, 10]$

ここで，M は対称行列なので，左下半分のみを格納している．配列 val は非零要素の値，配列 row_ind は非零要素の行番号を格納している．配列 col_ptr は val と row_ind における各列の先頭配列要素番号を格納していき，最後の要素に行列全体の非零要素数に 1 を足したものを格納している．

4.2 スカイライン格納方式

スカイライン格納方式とは，三角行列や対称行列に使われるもので，ある列の対角要素からその列の最後の非零要素までをすべて格納する格納形式である．本稿では，行列 M のような対称行列に対しては，下三角部分に対して，上述のようなスカイライン格納形式を適用し，非対称行列には，ある列の先頭要素からその列の最後の非零要素までをスカイライン格納形式と同様の形式ですべて格納する，スカイラインライク格納方式を用いる．なおスカイライン格納形式で非対称行列を格納する場合には，上記のスカイラインライク格納方式ではなく，非対称行列を上三角行列とした三角行列に分離し，それぞれに対してスカイライン格納方式を適用するのが一般的である．

行列 M をスカイライン格納方式で，行列 N をスカイラインライク格納方式で格納した場合の例を示す．配列 val, col_ptr の扱いはともに CCS 形式と同じである．

- 行列 M
  - $val : [1, 0, 2, 3, 4, 5, 6, 7]$
  - $col_ptr : [1, 4, 7, 8, 9]$
- 行列 N
  - $val : [1, 2, 3, 4, 0, 5, 0, 6, 7, 8, 9]$
  - $col_ptr : [1, 3, 7, 10, 12]$

5. 疎行列格納方式を用いた直接解法の検討

本研究では，疎行列直接解法を実装する際に採用する格納形式として，CCS 形式とスカイライン格納方式の 2 種類を検討する．本節では，それぞれの格納形式を採用した場合のアルゴリズムや実装の特徴について簡単に説明する．

Algorithm 4: Symbolic decomposition using CCS format

```
1  for j = 1, n do
2  |   R_j = {i | a_ij ≠ 0, i ≥ j}
3  end
4  col_ptr(1) = 1
5  for j = 1, n do
6  |   if num(R_j) ≥ 2 then
7  |   |   w = min(R_j \ {j})
8  |   |   R_w = R_w ∪ (R_j \ {j})
9  |   end
10 |   col_ptr(j + 1) = col_ptr(j) + num(R_j)
11 end
12 allocate array row_ind of size col_ptr(n + 1)
13 p = 0
14 for j = 1, n do
15 |   for i = 1, num(R_j) do
16 |   |   p = p + 1
17 |   |   row_ind(p) = min(R_j)
18 |   |   R_j = R_j \ {min{R_j}}
19 |   end
20 end
```

なお，どちらの格納形式を採用した場合にも，行列全体を一つの形式として格納するのではなく，各ブロック行列を単位として格納する．

5.1 CCS 形式を用いた直接解法

シンボリック分解については，先に述べた定理 1 を用いることで高速に行うことができる．このアルゴリズムを **Algorithm 4** に示す．**Algorithm 4** において，n は行列サイズ，$\mathcal{R}_j$ の初期値は j 列にある対角成分とそれより下にある非零要素の行番号の集合である．また，$\mathbf{num}(\mathcal{X})$ は集合 $\mathcal{X}$ の要素数を返す関数，$\mathbf{min}(\mathcal{X})$ は集合 $\mathcal{X}$ の要素の最小値を返す関数である．

プログラムで集合を扱うには，大きめの配列 $(n \times n)$ を用意しなくてはいけない．ただし，今回の行列では最も大きなブロック行列 1 つ分だけ用意すればよい．他のブロック行列にも対して使うことができるからである．また，最大バンド幅 w がわかっていれば，1 つの列は非零要素を高々 w しか含まないので，配列は $w \times n$ で済む．基本的にシンボリック分解は逐次で行うが，5 行目〜11 行目までのループをそれぞれのブロック行列で並列計算し，領域確保を逐次で行い，14 行目〜20 行目までを並列計算する方法も可能である．ただし，集合用の配列が並列数だけ必要となる．

数値分解のアルゴリズムを **Algorithm 5** に示す．2 行目で対角成分の平方根を求め，3 行目〜5 行目でそれより下の成分の値を求めている．そして 6 行目〜15 行目で分解列より右の列を更新している．p は val の更新場所のポイ

Algorithm 5: Numeric Cholesky decomposition using CCS format

```
1  for j = 1, n do
2      val(col_ptr(j)) = sqrt(val(col_ptr(j)))
3      for i = col_ptr(j) + 1, col_ptr(j + 1) − 1 do
4          val(i) = val(i)/val(col_ptr(j))
5      end
6      for i = col_ptr(j) + 1, col_ptr(j + 1) − 1 do
7          p = col_ptr(row_ind(i))
8          for h = i, col_ptr(j + 1) − 1 do
9              while row_ind(h) ≠ row_ind(p) do
10                 p = p + 1
11             end
12             val(p) = val(p) − val(i) ∗ val(h)
13             p = p + 1
14         end
15     end
16 end
```

Algorithm 6: Symbolic decomposition using skyline storage format

```
1  for j = 1, n do
2      row_max(j) = max({i|a_ij ≠ 0, i ≥ j})
3  end
4  col_ptr(1) = 1
5  for j = 1, n − 1 do
6      row_max(j + 1) = max(row_max(j), row_max(j + 1))
7      col_ptr(j + 1) = col_ptr(j) + row_max(j) − j + 1
8  end
9  col_ptr(n + 1) = col_ptr(n) + row_max(n) − n + 1
```

ンタである．更新する列は，分解列の非零成分の行番号と一致する．それぞれの更新列に対して，更新する場所は分解列に存在する非零要素と同じ行である．よって，val の更新には，更新列の対角成分から順に走査していき，分解列の非零要素と行番号が一致するインデックスを見つけなければならない．そのため，条件分岐が必要となる．

5.2 スカイライン格納方式を用いた直接解法

スカイライン格納方式では対角要素から最後の非零要素までをすべて非零要素とみなす．そのため，定理 1 における w は常に $j+1$ となる．よってシンボリック分解は分解列の 1 つ右の列のフィルインを計算していくことになる．アルゴリズムを **Algorithm 6** に示す．$row_max(j)$ の初期値は j 列の最後の非零要素の行番号である．

数値分解のアルゴリズムを **Algorithm 7** に示す．$p1$ は更新する列を，$p2$ は更新する val のインデックスを保持している．零要素も陽的に含んでいるので，演算回数の点では CCS 形式を用いた場合よりも不利となる．要素数によっては CCS の方が速くなると考えられる．ただし，すべての列が密ベクトルとなっているので，条件分岐は不要であり，計算時間の意味では CCS 形式よりも高速となる可能性を秘めている．

Algorithm 7: Numeric Cholesky decomposition using skyline storage format

```
1  for j = 1, n do
2      val(col_ptr(j)) = sqrt(val(col_ptr(j)))
3      for i = col_ptr(j) + 1, col_ptr(j + 1) − 1 do
4          val(i) = val(i)/val(col_ptr(j))
5      end
6      p1 = j
7      for i = col_ptr(j) + 1, col_ptr(j + 1) − 1 do
8          p1 = p1 + 1
9          p2 = col_ptr(p1)
10         for h = i, col_ptr(j + 1) − 1 do
11             val(p2) = val(p2) − val(i) ∗ val(h)
12             p2 = p2 + 1
13         end
14     end
15 end
```

6. 数値実験

6.1 実験に用いた行列

以下の 2 次元ポアソン方程式を 5 点差分近似を用いて解くことについて考える．

$$\begin{aligned} &\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -10 \quad \text{in } \Omega = [0, L] \times [0, 1] \\ &u = 0 \quad \text{on } \partial\Omega \end{aligned} \tag{6}$$

今回の実験では，$L = 32$ とし，x 方向に長い領域を考えた．One-way dissection オーダリングにおけるサブ領域の数の変化に対しての計算時間を調べるために，x 方向，y 方向の分割数をそれぞれ 2048，64 とし，行列サイズを固定して評価した．評価条件を以下に示す．

- 計算格子数：2047×63
- サブ領域の数：16, 64, 256, 1024
- thread 数：1, 2, 4, 8, 16

6.2 実行環境とプログラム

プログラムは Fortran で実装し，OpenMP を用いてスレッド並列化を行った上で，神戸大学の π-computer（FUJITSU Supercomputer PRIMIEHPC FX10）で性能を評価した．π-computer の主な仕様と使用したコンパイルオプションを**表 1** に示す．プログラムは，二次元配列を使って各ブロック行列全体を持つ NORMAL, CCS 形式を用いて実装した CCS, スカイライン格納方式とスカイラインライク格納方式を用いて実装した SKYLINE の 3 つを作成

表 1: Specification of π-computer(FX10)

CPU	SPARC64^{TM} IXfx
Cores	16
Clock	1.65GHz
Memory	32GB
Compiler	富士通製クロスコンパイラ
Compile option	-Kopenmp -Kfast

表 2: Amount of required memory

分割数	NORMAL (MB)	CCS (MB)	SKYLINE (MB)	非零要素率 (%)
16	7931	272	184	0.235
64	1984	137	95	0.119
256	503	105	78	0.093
1024	155	103	96	0.102

し，それぞれについて実行時間を計測した．

6.3 使用メモリ量の比較

まず，それぞれのプログラムにおいて，行列を保持するのに必要なメモリ量を比較する．また，サブ領域への分割数によって，行列の帯幅が変化し，コレスキー分解で生じるフィルインの数が異なるので，これも併せて比較する．**表 2** にその結果を示す．なお，整数型を 4 バイト，倍精度実数型を 8 バイトとして計算した．一番右の列は，コレスキー分解後の非零要素率[*2] を示している．非零要素率が小さければそれだけフィルインを抑えていることになる．表より，256 分割で最もフィルインが少ないことがわかる．

必要なメモリを比較すると，すべての分割数において，NORMAL, CCS, SKYLINE の順で小さくなっている．SKYLINE では零要素を少し含んでしまうが，配列 row_ind が不要なため，CCS より小さくなったと考えられる．NORMAL では分割数を増やすとメモリ量が大幅に減少している．非零要素率は 16 分割から 64 分割ではかなり減少しているが，それ以外では少ししか変化していない．つまり，分割数が少ないときはメモリの無駄が多く，分割数が多いと改善されていくということである．これに対し，CCS と SKYLINE のメモリ容量では，少し誤差があるものの非零要素率と比例の関係にあるといえる．メモリを効率的に使えていることがわかる．

6.4 疎行列直接解法全体の比較

表 3 にプログラムを逐次で実行したときの計算時間を示す．ここで，

- 区間 1 : メモリ領域確保，係数行列の生成，シンボリック分解（CCS と SKYLINE）の場合の実行時間
- 区間 2 : 数値分解（コレスキー分解）の実行時間

[*2] 非零要素率 (%) $= \frac{\text{非零要素数}}{(2047\times63)^2} \times 100$

表 3: Computational time in the case of using 1 thread

Method	Number of sub-regions	区間 1 (sec.)	区間 2 (sec.)	区間 3 (sec.)	total (sec.)
NORMAL	16	3.5541	3449.3139	1.4344	3454.3024
	64	0.8949	229.0601	0.4094	230.3644
	256	0.2322	18.3945	0.1878	18.8145
	1024	0.0747	1.8024	0.0912	1.9683
CCS	16	0.6258	23.0795	0.1895	23.8948
	64	0.2365	12.1667	0.0667	12.4700
	256	0.1875	11.5551	0.0574	11.8001
	1024	0.1910	14.4351	0.0736	14.6997
SKYLINE	16	0.1722	6.8538	0.1334	7.1595
	64	0.0891	1.5440	0.0555	1.6886
	256	0.0724	0.9700	0.0471	1.0894
	1024	0.0780	1.0716	0.0530	1.2026

- 区間 3 : 前進後退代入の実行時間

としている．なお，計算開始時の係数行列は COO（Coordinate Format）形式[*3] で与えられるものとしている．また，one-way dissection オーダリングを適用済み（オーダリング適用後の列と行番号が与えられる）としている．

区間 1 では，NORMAL は，COO 形式で与えられた各ブロック行列の要素の値を密行列（二次元配列）にセットする．CCS では, この入力データを集合用の配列に格納し，シンボリック分解して行列を生成する．SKYLINE では，それぞれの列の最大の行番号を row_max にセットし，シンボリック分解を行って行列を生成する．

表より，すべての手法，分割数で計算時間の多くを占めているのはコレスキー分解であるのがわかる．そこで，以下の評価ではコレスキー分解の計算時間について詳しく評価する．また，時間発展の計算を踏まえて，前進後退代入の計算時間についても細かく評価する．

6.5 コレスキー分解の評価

それぞれのプログラムのコレスキー分解の計算時間をスレッド数を変えて計測した．この結果を**表 4** に示す．16 分割した場合，すべてのスレッド数で NORMAL, CCS, SKYLINE の順に速くなった．だが，1024 分割では，CCS, NORMAL, SKYLINE の順に速くなっている．CCS が SKYLINE より遅いのは，数値分解において，CCS では条件分岐が必要だが，SKYLINE は不要であることが主な原因であると推測される．ただし，SKYLINE では零要素を含むこともあるため，その割合によっては CCS の方が速くなることもある．16 スレッド実行時に最速となったのは 64 分割，SKYLINE で 0.1369sec. となった．これは NORMAL の最速である 1024 分割の計算時間の約 5.08 倍の速さである．

CCS，SKYLINE のコレスキー分解にかかる時間を逐次部分，並列部部分にわけて計測した．その結果をそれぞれ**図 7**，**図 8** に示す．グラフの横軸は分割数とスレッド数，

[*3] 非零要素の（列番号，行番号，値）を要素数分持たせる形式

表 4: Computational time of the Cholesky decomposition

Method	Number of sub-regions	Number of threads(sec.) 1	2	4	8	16
NORMAL	16	3449.3139	1749.0685	894.3046	478.4436	414.1043
	64	229.0601	125.6869	68.1501	38.4945	28.1965
	256	18.3945	9.2547	4.6992	2.4411	1.4861
	1024	1.8024	1.1920	0.8962	0.7525	0.6950
CCS	16	23.0795	12.0532	6.1236	3.1377	1.6362
	64	12.1667	6.4386	3.5061	2.0293	1.2918
	256	11.5551	6.9265	4.6000	3.4330	2.8504
	1024	14.4351	11.7812	10.4548	9.7867	9.4545
SKYLINE	16	6.8538	3.5109	1.7647	0.8915	0.4530
	64	1.5440	0.8005	0.4230	0.2320	0.1369
	256	0.9700	0.5683	0.3671	0.2658	0.2154
	1024	1.0716	0.8648	0.7628	0.7110	0.6860

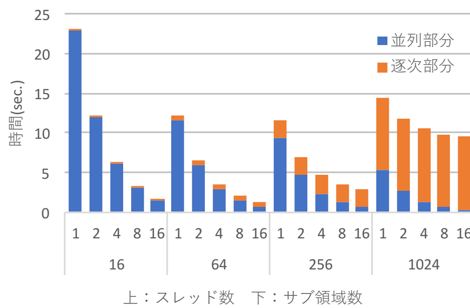


図 7: Breakdown of the computational time of the Cholesky decomposition (using CCS format)

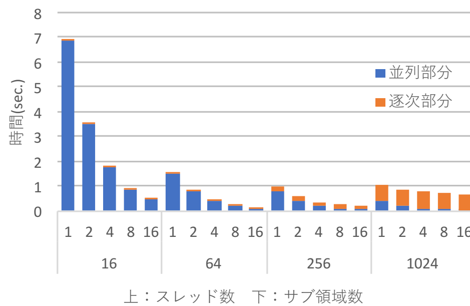


図 8: Breakdown of the computational time of the Cholesky decomposition (using skyline storage format)

縦軸は計算時間である．両方のプログラムで，分割数を増やすにつれて，逐次部分が増加していることがわかる．これにより，スレッドの増加による計算時間の減少率が落ちている．また，ポアソン方程式の場合，逐次部分における部分行列（図 6 の行列 B_i, C_i）はコレスキー分解する際には密行列となる．よって，CCS における密行列のコレスキー分解は非常に遅いことから，1024 分割では，NORMAL が CCS より速くなったと考えられる．

6.6 SKYLINE のより詳細な評価

SKYLINE の有効性がわかったので，その特性についてより詳しく考察する．ひとつのサブ領域の格子数を a，ひとつの境界領域の格子数を b とすると，行列 A_i のサイズは $a \times a$，行列 W_i，E_i のサイズは $b \times a$ となる．ここで，

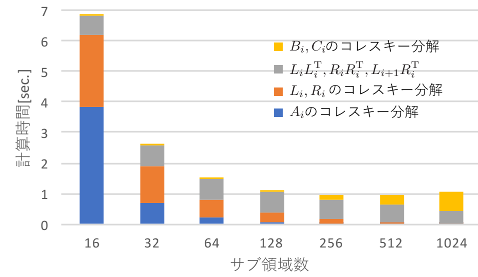


図 9: Relationship between the number of sub-regions and computational time in each step of the Cholesky decomposition (using skyline storage format)

A_i の半帯幅を w とすると，A_i のコレスキー分解の計算量は $\mathcal{O}(aw^2)$ となる．行列 W_i，E_i のコレスキー分解の計算量は，それぞれの列の非零要素数の平均を h とすると，$\mathcal{O}(ahw)$ となる．また，行列 W_i，E_i の行列積の計算量は，$\mathcal{O}(ah^2)$ となる．サブ領域数を 2 倍にすると，a，b，w は半分になるが，b と h は一定である．A_i，W_i，E_i の部分行列の数が 2 倍になることを考慮すると，A_i のコレスキー分解の計算時間の合計は 1/4 倍，W_i，E_i のコレスキー分解の計算時間の合計は 1/2 倍，行列積の計算時間の合計は一定である．また，B_i，C_i の行列のサイズは一定で数が 2 倍になるので，B_i,C_i のコレスキー分解の計算時間の合計も 2 倍になる．SKYLINE のコレスキー分解をより細かい区間に分け，サブ領域数を変化させて逐次実行した場合の計算時間の結果を**図 9** に示す．サブ領域数は今までの 16，64，256，1024 に加えて 32，128，512 も追加実験した．先に述べたことと測定結果が一致しているのがわかる．

6.7 前進後退代入の評価

コレスキー分解の利点は，連立方程式の係数行列が変わらない（右辺ベクトルのみが変わる）場合に，一度分解をしてしまえば，あとは前進後退代入のみで高速に計算できる点である．したがって，繰り返し解く回数によっては，前進後退代入の時間が重要となる．それぞれのプログラムの前進後退代入の計算時間を**表 5** に示す．すべてのスレッド数，分割数で SKYLINE が最速となった．16 スレッド実行時に，NORMAL では 256 分割，CCS と SKYLINE では 64 分割のときが最速となり，この中で最も最速なのは，SKYLINE で 0.0059sec. である．これは NORMAL の最速である 256 分割の計算時間の約 3.85 倍の速さである．

CCS，SKYLINE の前進後退代入にかかる時間を逐次部分，並列部部分にわけて計測した．その結果を，**図 10**，**図 11** に示す．コレスキー分解の時と同じく，分割数を増やすにつれ，逐次部分が増え，並列性能が落ちているのがわかる．

ⓒ 2017 Information Processing Society of Japan

表 5: Computational time of forward and backward substitution

Method	Number of sub-regions	Number of threads(sec.) 1	2	4	8	16
NORMAL	16	1.4344	0.7216	0.3672	0.1951	0.1277
	64	0.4094	0.2089	0.1082	0.0592	0.0415
	256	0.1878	0.0990	0.0547	0.0327	0.0227
	1024	0.0912	0.0651	0.0524	0.0457	0.0428
CCS	16	0.1895	0.0969	0.0500	0.0261	0.0156
	64	0.0667	0.0349	0.0188	0.0110	0.0076
	256	0.0574	0.0338	0.0218	0.0161	0.0134
	1024	0.0736	0.0577	0.0501	0.0461	0.0444
SKYLINE	16	0.1334	0.0667	0.0352	0.0196	0.0113
	64	0.0555	0.0291	0.0154	0.0089	0.0059
	256	0.0471	0.0271	0.0170	0.0121	0.0098
	1024	0.0530	0.0419	0.0363	0.0335	0.0323

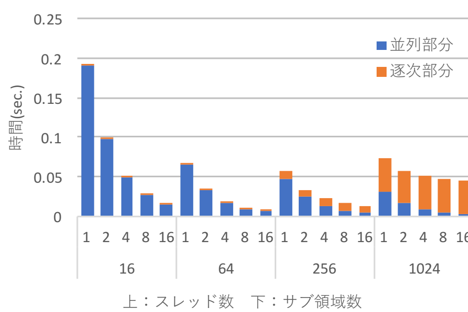


図 10: Breakdown of the computational time of the forward and backward substitution (using CCS format)

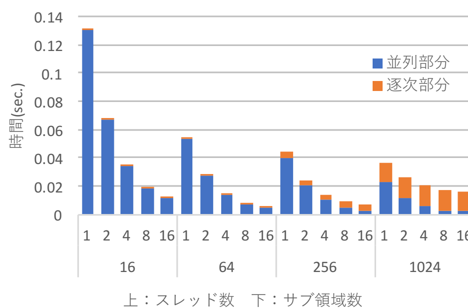


図 11: Breakdown of the computational time of the forward and backward substitution (using skyline storage format)

7. まとめ

本稿では，導電性高分子のシミュレーションに現れる連立一次方程式に one-way dissection オーダリングを用いた疎行列直接解法を適用するための前段階として，ポアソン方程式を 5 点差分近似した場合に生ずる連立一次方程式に同手法を適用し，疎行列の格納形式の違いに着目した性能評価を行った結果について報告した．具体的には，CCS 形式，スカイライン（とスカイラインライク）格納形式を用いたプログラムを作成し，密行列形式で格納する場合と比較した．また，格納形式の比較とともに，one-way dissection オーダリングにおけるサブ領域への分割数と性能の関係についても調査を行った．

数値実験の結果より，今回の評価では，スカイライン格納形式を用いる場合が，コレスキー分解（数値分解）と前進後退代入の両方において最速となった．また，分割数を多くすると，ブロック部分の行列サイズや帯幅が小さくなるので，並列処理が可能な部分の計算時間が短縮するが，一方で，逐次計算時間部分の時間が増加してしまうことが確認され，適切な分割数を設定する必要があることが示された．

最後に，今後の課題について述べる．今回は，FX10 システム（神戸大学の π-computer）のみで評価を行ったので，その他の環境（一般的な Xeon 環境や KNL のようなメニーコア CPU 環境）で評価を行う必要がある．また，性能評価の結果を踏まえて，スカイライン（やスカイラインライク）格納方式をベースにより効率的な格納形式を検討する余地も残っている．一方，並列数が非常に多い環境ではサブ領域の数を多くする必要が生じるが，その場合には，逐次計算の部分が増加してしまい，性能が低下する．そこて，逐次計算部分の計算方法を変更し，並列性をより高めるオーダリング方法について検討する必要もある．以上の課題を検討すると同時に，実際に導電性高分子に関する時間発展のシュレーディンガー方程式に適用し，従来の手法と我々が検討している手法の性能を比較することが今後の主な課題である．

謝辞　本研究について助言を賜りました神戸大学大学院システム情報学研究科の小柳義夫特命教授，谷口隆晴准教授，鳥取大学大学院工学研究科の星健夫准教授に厚く御礼申し上げます．本研究成果の一部は，JSPS 科研費 JP17H02828，JP15K16000 及び学際大規模情報基盤共同利用・共同研究拠点（課題番号：jh170053-NAJ）の助成を受けたものです．

参考文献

[1] 白川英樹，廣木一亮：導電性高分子の基礎，材料化学の基礎 8，シグマ アルドリッチ (2012).
[2] Imachi, H.: Numerical Methods for Large-scale Quantum Material Simulations, PhD Thesis, Tottori University (2017).
[3] Hoshi, T., Yamamoto, S., Fujiwara, T., sogabe, T. and Zhang, S.-L.: An order-N electronic structure theory with generalized eigenvalue equations and its application to a ten-million-atom system, *Journal of Physics: Condensed Matter* (2012).
[4] 井町宏人：100 ナノ量子電気伝導計算むけの並列量子波束ダイナミクスソルバー開発，スーパーコンピューティングニュース，Vol. 18, No. 6, pp. 1–6 (2016).
[5] George, A.: An Automatic One-Way Dissection Algorithm for Irregular Finite Element Problems, *SIAM Journal on Numerical Analysis*, Vol. 17, No. 6, pp. 740–751 (1980).

ⓒ 2017 Information Processing Society of Japan

[6] フレデリック・マグレス，フランソワ=グザヴィエ・ルー，桑原拓也：並列計算の数理とアルゴリズム，森北出版株式会社 (2015).
[7] var Grondelle, J.: Symbolic Sparse Cholesky Factorisation Using Elimination Trees, Master's thesis, Utrecht University (1999).
[8] 山本有作：疎行列連立一次方程式の直接解法，計算工学，Vol. 11, No. 4, pp. 1458–1462 (2006).
[9] 村田健郎，J.J. ドンガラ，小国　力，三好俊郎，長谷川秀彦：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).

ⓒ 2017 Information Processing Society of Japan