Element-by-Element方式有限要素解析における IDR(s) 法とBi IDR(s) 法の実装の改良

柴田, 修作
九州大学大学院システム情報科学府情報学専攻修士課程

尾上, 勇介
九州大学大学院システム情報科学府情報学専攻博士後期課程

藤野, 清次
九州大学情報基盤研究開発センター

樫山, 和男
中央大学理工学部

https://doi.org/10.15017/18988

出版情報：九州大学大学院システム情報科学紀要. 15 (2), pp.85-90, 2010-09-24. 九州大学大学院システム情報科学研究院
バージョン：
権利関係：

九州大学大学院
システム情報科学紀要
第 15 巻　第 2 号　平成 22 年 9 月

Research Reports on Information Science and
Electrical Engineering of Kyushu University
Vol.15, No.2, September 2010

Element-by-Element 方式有限要素解析における IDR(s) 法と Bi_IDR(s) 法の実装の改良

柴田　修作* ・　尾上　勇介** ・　藤野　清次***・　樫山　和男†

Improvement of Implementation of IDR(s) and Bi_IDR(s) Methods in FEM Analysis by Element-by-Element Scheme

Shusaku SHIBATA*, Yusuke ONOUE**, Seiji FUJINO*** and Kazuo KASHIYAMA†

(Received July 23, 2010)

Abstract: Large scale analysis of fluid dynamics needs much computational time and memory even if state-of-the-art computers with multicores are used. This motivates us to incorporate Element-by-Element scheme capability into the employed IDR(s) and Bi_IDR(s) methods which were recently proposed. A way of obtaining a better convergence is insightful for kernel implementation of matrix-vector multiplication. Through numerical experiments, we reveal performance improvement of kernel implementation.

Keywords: FEM, Element-by-Element scheme, IDR(s) method, Bi_IDR(s) method, Implementation

1. は　じ　め　に

流体力学現象を記述する Navier-Stokes 方程式の求解処理を高速化することは，コンピュータの性能が飛躍的に進歩した今日においても、非常に重要な課題の一つとしてよく知られている．そのための有効な解決手段の一つに Element-by-Element(以下, EbE と略す) 方式有限要素法 (Finite ElementMethod) がある[5)7)25)]．1983 年に Hughes らによって考案された EbE 方式有限要素法は，最も計算コストが高い線形方程式の数値解法において，全体行列を作ることなく，領域全体に張った要素ごとの演算を可能にする，メモリ量削減を主目的とした手法[4)]である．EbE 方式有限要素法は，1990 年当時盛んであったベクトル計算機に適合していたため，ベクトル計算機上での解析によく応用された．[1)3)26)]．さらに，2000 年頃には，クラスタ計算機の登場により, EbE 方式はそれへの適用が大きな関心事になりよく研究された[8)9)10)11)18)27)]．さらに, 最近では, マルチコアの CPU の開発と普及により，EbE 方式の有限要素法の再評価が熱心にされている[2)12)13)]．また，EbE 方式の反復法への前処理の適用は依然として大きな研究課題である[16)22)23)]．これについても最近新しい研究進展が見えてきた[24)]．

一方，線形方程式に対する反復法の研究分野においても，最近提案された IDR(s) 法[19)]や Bi_IDR(s) 法[20)] などが大きな旋風を巻き起こしている．これらの反復法は，収束の安定性や頑強性において，従来の反復法にない優れた長所を持っている[14)15)]．そこで，本研究では，線形方程式の解法として IDR(s) 法と Bi_IDR(s) 法を採用し，これらの反復法をさらに有用なものとするために，実装面からの改良策を考察し，有効な手段を探り，高速化を図った．その結果，有益な知見が得られたので報告する．

本論文の構成は以下の通りである．第 2 節で，IDR 定理に基づく反復法である IDR(s) 法と Bi_IDR(s) 法の算法を紹介する．第 3 節で，EbE 方式における行列ベクトル積の計算の高速化について詳細に記述する．第 4 節で，数値実験を通して，IDR(s) 法と Bi_IDR(s) 法に対する高速化の効果を明らかにする．最後に第 5 節でまとめる．

平成 22 年 7 月 23 日受付
*情報学専攻修士課程
**情報学専攻博士後期課程
***情報基盤研究開発センター
†中央大学理工学部

2. EbE 方式有限要素解析と反復法

本研究では, 流れの基礎方程式である Navier-Stokes 方程式を離散化し，得られた連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を IDR(s) 法または Bi_IDR(s) 法で解く．

2.1 IDR(s) 法

解くべき連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ において，行列 A は係数行列, $\boldsymbol{x}$ は解ベクトル, $\boldsymbol{b}$ は右辺ベクトルとする．IDR(s) 法は IDR 定理に基づく解法であり，IDR 定理について簡単に概要を紹介する[19)14)]．

$A \in R^{N\times N}$ とし，行列 P を一次独立なベクトル群 $\boldsymbol{p}_i (i = 1, 2, \ldots, s)$ を列ベクトルに持つ行列 $P = (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_s)$，空間 $\mathcal{G}_0$ を完全 Krylov 空間 $K_N(A; \boldsymbol{r}_0)$ とする．さらに，一連の空間 $\mathcal{G}_j (j = 1, 2, \ldots)$ を，

$$\mathcal{G}_j := (I - \omega_j A)(\mathcal{G}_{j-1} \cap \boldsymbol{N}ull(P^T)) \quad (1)$$

と定義する．パラメータ ω_j は非零のスカラー値, $\boldsymbol{N}ull(P^T)$ を行列 P の転置 P^T の零空間，$I - \omega_j A$ が零でないとき次の定理が成り立つ．

IDR 定理

(i) $\mathcal{G}_j \subseteq \mathcal{G}_{j-1}$　for all $j > 0$

(ii) $\mathcal{G}_j = \{\boldsymbol{0}\}$　for some $j \leq N$

次に，IDR(s) 法について述べる．IDR(s) 法は，IDR 定理で定義される空間 $\mathcal{G}_j$ に，s+1 本の残差ベクトル $\boldsymbol{r_0} := \boldsymbol{b} - A\boldsymbol{x_0}$ が属するように設計される．$\boldsymbol{x_0}$ は初期近似解である．空間 $\mathcal{G}_j$ は，高々 N ステップで零空間になるため，IDR(s) 法が収束する．以下に，IDR(s) 法の算法を示す．

IDR(s) 法の算法

1. Let $\boldsymbol{x}_0$ be a random vector, and put $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
2. For $n = 0, \ldots, s-1$ Do
3. $\quad \boldsymbol{v}_n = A\boldsymbol{r}_n$
4. $\quad \omega = \dfrac{(\boldsymbol{v}_n, \boldsymbol{r}_n)}{(\boldsymbol{v}_n, \boldsymbol{v}_n)}$
5. $\quad \boldsymbol{q}_n = \omega \boldsymbol{r}_n, \quad \boldsymbol{e}_n = -\omega \boldsymbol{v}_n$
6. $\quad \boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \boldsymbol{e}_n,\ \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{q}_n$
7. End Do
8. $E_s = (\boldsymbol{e}_{s-1}, \ldots, \boldsymbol{e}_0),\ \ Q_s = (\boldsymbol{q}_{s-1}, \ldots, \boldsymbol{q}_0)$
9. Do $n = s, s+1, \ldots$
10. $\quad$ Solve $\boldsymbol{c}_n$ from $P^T E_n \boldsymbol{c}_n = P^T \boldsymbol{r}_n$
11. $\quad \boldsymbol{v}_n = \boldsymbol{r}_n - E_n \boldsymbol{c}_n$
12. $\quad$ If $\mathrm{mod}(n, s+1) = s$ then
13. $\quad\quad \boldsymbol{t}_n = A\boldsymbol{v}_n$
14. $\quad\quad \omega = \dfrac{(\boldsymbol{t}_n, \boldsymbol{v}_n)}{(\boldsymbol{t}_n, \boldsymbol{t}_n)}$
15. $\quad\quad \boldsymbol{e}_n = -E_n \boldsymbol{c}_n - \omega \boldsymbol{t}_n$
16. $\quad\quad \boldsymbol{q}_n = -Q_n \boldsymbol{c}_n + \omega \boldsymbol{v}_n$
17. $\quad$ Else
18. $\quad\quad \boldsymbol{q}_n = -Q_n \boldsymbol{c}_n + \omega \boldsymbol{v}_n$
19. $\quad\quad \boldsymbol{e}_n = -A\boldsymbol{q}_n$
20. $\quad$ End If
21. $\quad \boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \boldsymbol{e}_n,\ \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{q}_n$
22. $\quad$ if $||\boldsymbol{r}_{n+1}||_2 / ||\boldsymbol{r}_0||_2 \ \leq\ \epsilon$ then stop
23. $\quad E_{n+1} = (\boldsymbol{e}_n, \ldots, \boldsymbol{e}_{n+1-s}),\ Q_n = (\boldsymbol{q}_n, \ldots, \boldsymbol{q}_{n+1-s})$
24. End Do

2.2 Bi_IDR(s) 法

IDR(s) 法系統の解法の一つである Bi_IDR(s) 法について述べる[20)15)]．残差ベクトル $\boldsymbol{r}_{n+1}$ を空間 $\mathcal{G}_{j+1}$ に属する最初の中間残差ベクトルとする．以下に示す 2 つの直交条件が成立するように Bi_IDR(s) 法の算法を設計する．

$$\boldsymbol{g}_{n+1} \perp \boldsymbol{p}_j \ (i = 2, \ldots s,\ j = 1, \ldots, i-1) \quad (2)$$

$$\boldsymbol{r}_{n+i+1} \perp \boldsymbol{p}_j \ (i = 1, \ldots s,\ j = 1, \ldots, i) \quad (3)$$

上記のベクトル $\boldsymbol{g}_{n+1}$ は，残差ベクトルの差分，即ち，$\boldsymbol{g}_{n+1} := \boldsymbol{r}_{n+1} - \boldsymbol{r}_{n+i+1}$ とする．ベクトル $\boldsymbol{p}_j$ は行列 P の列ベクトルである．上記の 2 条件を課し，計算量の削減と収束安定化を図る．Bi_IDR(s) 法の算法を以下に示す．

Bi_IDR(s) 法の算法

1. Let $\boldsymbol{x}_0$ be a random vector, and put $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$,
2. $\boldsymbol{g}_i = \boldsymbol{q}_i = \boldsymbol{0}, i = 1, \ldots, s, H = I, \omega = 1$
3. $n = 0$
4. While $||\boldsymbol{r}_n||_2 / ||\boldsymbol{r}_0||_2 > \epsilon$ Do
5. $\quad \boldsymbol{f} = P^T \boldsymbol{r}_n, \boldsymbol{f} = (\phi_1, \ldots, \phi_s)$
6. $\quad$ Do $k = 1, \ldots, s$
7. $\quad\quad$ Solve $\boldsymbol{c}$ from $H\boldsymbol{c} = \boldsymbol{f}, \boldsymbol{c} = (\gamma_1, \ldots, \gamma_s)$
8. $\quad\quad \boldsymbol{v} = \boldsymbol{r}_n - \sum_{i=k}^{s} \gamma_i \boldsymbol{g}_i, \boldsymbol{q}_k = \omega \boldsymbol{v} + \sum_{i=k}^{s} \gamma_i \boldsymbol{q}_i$
9. $\quad\quad \boldsymbol{g}_k = A\boldsymbol{q}_k$

   $\quad\quad$ {Make $\boldsymbol{g}_k$ orthogonal to $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{k-1}$ }
10. $\quad\quad$ Do $i = 1, \ldots, k-1$
11. $\quad\quad\quad \alpha = \dfrac{(\boldsymbol{p}_i, \boldsymbol{g}_k)}{\mu_{i,i}}$
12. $\quad\quad\quad \boldsymbol{g}_k = \boldsymbol{g}_k - \alpha \boldsymbol{g}_i, \boldsymbol{q}_k = \boldsymbol{q}_k - \alpha \boldsymbol{q}_i$
13. $\quad\quad$ End Do

    $\quad\quad$ {Compute $\boldsymbol{p}_i^T \boldsymbol{g}_k, i = k, \ldots, s$ update H }
14. $\quad\quad \mu_{i,k} = (\boldsymbol{p}_i, \boldsymbol{g}_k), i = k, \ldots, s, H_{i,k} = \mu_{i,k}$

    $\quad\quad$ {Make residual orthogonal to $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k$ }
15. $\quad\quad \beta = \dfrac{\phi_k}{\mu_{k,k}}$
16. $\quad\quad \boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \beta \boldsymbol{g}_k, \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \beta \boldsymbol{q}_k$

    $\quad\quad$ {Update $\boldsymbol{f} = P^T \boldsymbol{r}$ }
17. $\quad\quad$ If $k < s$ then
18. $\quad\quad\quad \phi_i = 0, i = 1, \ldots, k,$
19. $\quad\quad\quad \phi_i = \phi_i - \beta \mu_{i,k}, i = k+1, \ldots, s$
20. $\quad\quad\quad \boldsymbol{f} = (\phi_1, \ldots, \phi_s)$
21. $\quad\quad$ End If
22. $\quad\quad n = n + 1$
23. $\quad$ End Do
24. $\quad \boldsymbol{t} = A\boldsymbol{r}_n$
25. $\quad \omega = \dfrac{(\boldsymbol{t}, \boldsymbol{r})}{(\boldsymbol{t}, \boldsymbol{t})}$
26. $\quad \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \omega \boldsymbol{r}_n, \boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \omega \boldsymbol{t}$
27. $\quad n = n + 1$
28. End While

3. 行列ベクトル積の計算の高速化

3.1 EbE 方式における行列ベクトル積の計算

一般に，線形方程式の求解において，行列ベクトル積計算

は計算コストが非常に高いことが知られている[17]．そこで，本研究では，EbE 方式を採用し，大きなサイズの元の係数行列 A を使わず，よりサイズの小さい要素係数行列 $\mathrm{A}^{\mathrm{elem.}}$ を使用することを試みる[21]．

要素数を nelm 個とするとき，サイズの小さい要素係数行列 $A^{\mathrm{elem.}}$ の個数も nelm 個になる．このとき，元の大きな係数行列 A は，全部で nelm 個の要素係数行列 $A^{\mathrm{elem.}}$ の重ね合わせから計算することができる．同様に，右辺ベクトル $\boldsymbol{b}$ も，nelm 個の要素右辺ベクトル $\boldsymbol{b}^{\mathrm{elem.}}$ の重ね合わせから作ることができる．このように，EbE 方式では，元の大きな係数行列 A の代わりに小さいサイズの要素係数行列 $A^{\mathrm{elem.}}$ だけを記憶しておけばよい．この点が通常の場合と異なる．すなわち，EbE 方式では，元の係数行列 A とベクトル $\boldsymbol{v}$ の積の計算を一度にする代わりに，要素ごとに nelm 個の要素係数行列 $A^{\mathrm{elem.}}$ とベクトルの積を行い，この手順を要素数の数だけ繰り返すことになる．便宜上，要素係数行列は 9×9 の大きさの小行列とした[6]．

Fig. 1 に，行列ベクトル積を計算するための Fortran 模擬コードを示す．3 次元配列 $\mathrm{A}^{\mathrm{elem.}}$(j, i, m) にはすべての要素係数行列の重ね合わせが格納される．例えば，指標 j と i が m 番目の要素係数行列の行番号と列番号を表すとき，m 番目の要素の情報は配列 $\mathrm{A}^{\mathrm{elem.}}$(j, i, m) に格納される．さらに，配列 x には解ベクトルの値，配列 r には要素係数行列とベクトルの積の計算結果が各々格納される．また，配列 nn には要素に関する結合情報が格納される．

```
r(:) = 0.0d0
do m = 1, nelm
   do i = 1, 9
     do j = 1, 9
     r(nn(j, m)) = r(nn(j, m)) + A^elem.(j, i, m) * x(nn(i, m))
     enddo
   enddo
enddo
```

Fig. 1 A Fortran-like code of matrix-vector multiplication in EbE scheme.

次の小節から，EbE 方式における行列ベクトル積の計算の高速化について述べていく．

3.2 一時的変数の利用によるレジスタの活用

Fig. 2 に，一時的変数 tmp で配列要素 x(nn(i, m)) を最内側ループの外に出した行列ベクトル積の計算の Fortran 模擬コードの例を示す．このコードの最内側ループでは，配列要素 x(nn(i, m)) をメモリにロードするためのロード命令が頻繁に繰り返される．しかし，一般にメモリへのアクセス速度はレジスタへのアクセス速度に比べて相対的に遅いことが知られている．そこで，一時的変数を利用して，アクセス速度が速いレジスタを有効活用する手立てを考える．すなわち，**Fig. 2** のように，2 次元配列 x(nn(i,m)) を変数 tmp に格納し最内側ループの外に出すことができる．コンパイラは，x(nn(i, m)) を配列要素ではなく，最内側ループ中で何度も参照される変数として認識する．このため，x(nn(i, m)) がレジスタに割り当てられる確率が増すと考えられる．

```
r(:) = 0.0d0
{Initialize r(k) (k = 1, ..., N)}
do m = 1, nelm
   do i = 1, 9
     tmp = x(nn(i, m))
     do j = 1, 9
        r(nn(j, m)) = r(nn(j, m)) + A^elem.(j, i, m) * tmp
     enddo
   enddo
enddo
```

Fig. 2 A Fortran-like code of matrix-vector multiplication using temporary variable “tmp”.

3.3 要素係数行列 $A^{\mathrm{elem.}}$ の格納方向の検討

Fig. 2 において，一時的変数 tmp のさらなる利用によりレジスタへのアクセス回数を単純に増やす手立ても考えられる．**Fig. 3** に，使用レジスターの増加に固執し，単純に配列添字を交換した悪い例の行列ベクトル積の計算の Fortran 模擬コードを示す．一見，この変更により効率がよくなったような錯覚に陥り易い．しかし，Fortran における配列の規則では，最も左側の添字で指示された要素から順番にメモリ上に連続に並べられる．そこで，メモリに連続的にアクセスできているかチェックした．しかし，**Fig. 3** では，$\mathrm{A}^{\mathrm{elem.}}$(j, i, m) の添字 i が，添字 j に対して先に変化する．すなわち，$\mathrm{A}^{\mathrm{elem.}}$(j, i, m) はメモリ上で連続的に参照されず，アクセス速度が低下する事態を招く恐れがある．

Fig. 3 に示したコードのままでは速度低下が大きいので，メモリ上に割り当てられた配列 $\mathrm{A}^{\mathrm{elem.}}$ の内容を連続的に参照するために，配列 $\mathrm{A}^{\mathrm{elem.}}$ に対して，要素係数行列の格納方向の変更する．すなわち，要素係数行列の値を列優先から行優先に変更する．**Fig. 4** が **Fig. 1** に対応し，同様に，**Fig. 5** が **Fig. 2** に対応する最終の改良版である．

4. 数 値 実 験

本節では，IDR(s) 法と Bi_IDR(s) 法における高速化手

```
r(:) = 0.0d0
do m = 1, nelm
  do j = 1, 9
    tmp = r(nn(j, m))
    do i = 1, 9
      tmp = tmp + A^elem.(j, i, m) * x(nn(i, m))
    enddo
    r(nn(j, m)) = tmp
  enddo
enddo
```

Fig. 3 A Fortran-like code of matrix-vector multiplication using poorly simple exchange between indices "j" and "i".

```
r(:) = 0.0d0
do m = 1, nelm
  do j = 1, 9
    do i = 1, 9
    r(nn(j, m)) = r(nn(j, m)) + A^elem.(i, j, m) * x(nn(i, m))
    enddo
  enddo
enddo
```

Fig. 4 A Fortran-like code of matrix-vector multiplication according to row-wise reference of $A^{elem.}$.

```
r(:) = 0.0d0
do m = 1, nelm
  do j = 1, 9
    tmp = r(nn(j, m))
    do i = 1, 9
      tmp = tmp + A^elem.(i, j, m) * x(nn(i, m))
    enddo
    r(nn(j, m)) = tmp
  enddo
enddo
```

Fig. 5 A Fortran-like code of matrix-vector multiplication using temporary variable "tmp" and according to row-wise reference of $A^{elem.}$.

法の効果を数値実験で明らかにする．2 種類の計算機を使用した．テスト問題には 2 次元 Cavity 流れ問題を使用した[6]．2 種類の計算機環境を以下に示す．

- 計算機環境 (1)：
  –Primequest 580 (CPU コア：Intel Itanium2，クロック周波数：1.6GHz，メモリ：4Tbytes，OS：RedHat Enterprise Linux AS release 4).
  –コンパイラは Fujitsu Fortran Driver Ver.2.0 を使用.
  –最適化オプションは "-O0"，"-Kfast" を使用.
- 計算機環境 (2)：
  –SR16000(CPU コア：POWER6 プロセッサ，クロック周波数：4.7GHz，メモリ：128Gbytes，OS：IBM AIX 5L V5.3).
  –コンパイラは Hitachi 最適化 Fortran を使用.
  –最適化オプションは "-O0"，"-Oss" を使用.

2 次元 Cavity 流れ問題の境界条件として，上側壁面における流速は $u = 1$, その他の壁面では滑りなし条件 (non-slip 条件：$u, v = 0$) を与えた．解析領域全体に張った解析メッシュは各辺を不等間隔で 32 分割したものを用いた．境界近傍にはメッシュを集中させ，最小メッシュ幅は 1.953×10^{-3} になった． 流れの速さ Reynolds 数は $Re = 10^3$，時間増分値は $\Delta t = 0.01$，最大時間ステップ数は 100 とした．

反復法 IDR(s) 法と Bi_IDR(s) 法のパラメータ s の値は $s = 1,\ 2,\ 4,\ 8$ の場合を調べた．反復法の収束判定条件は相対残差の 2 ノルム：$||\boldsymbol{r}_{n+1}||_2/||\boldsymbol{r}_0||_2 \leq 10^{-8}$ とした．行列は対角スケーリングを施し対角要素をすべて 1 にした．

4.1 IDR(s) 法の場合

ここでは，IDR(s) 法の収束性に基づき高速化技法を比較する．**Table 1** に計算機 Primequest 580 (以下，PQ と略す) 上での IDR(s) 法の結果を示す．同様に，**Table 2** に計算機 SR16000 (以下，SR と略す) 上での IDR(s) 法の結果を示す．計算時間の計測には C 言語の時間計測関数 gettimeofday( ) を使用した．最速であった IDR($s = 4$) 法の結果を表に載せた．**Table 1** と **Table 2** の各欄の内容は以下の通りである．

opt.	自動最適化 option の種類
prio.	要素係数行列格納方向の優先 (priority)
lev.	行列ベクトル積の計算方法のレベル (level)
solv.time	方程式の求解に要した時間の合計
mat.time	行列ベクトル積の計算に要した時間の合計

行列ベクトル積の計算方法の高速化手法の各レベルの内容を以下に示す．また，該当する各高速化レベルに対応する Fortran 模擬コードの図番号を記す．

(1) lev.1：定義通りの計算法 (**Fig. 1** と **Fig. 4** が該当)
(2) lev.2：上の lev.1 の最内側ループに対して 3 段のループアンローリング (loop-unrolling)
(3) lev.3：上の lev.1 に対して，一時的変数を用いてレジスタを積極的に活用する方法 (**Fig. 2** と **Fig. 5** が該当)
(4) lev.4：上の lev.3 の最内側ループに対して 3 段のループアンローリング

Table 1 と **Table 2** 中の，“ratio-1” は最速レベルのときの列優先のときの時間を 1.0 としたときの行優先のときの時間の比を表す．同様に，“ratio-2” と “ratio-3” の値は，要素係数行列を列優先に格納した場合の時間を 1.0 にしたときの行列ベクトル積計算 (lev.1) 使用の合計求解時間の比を表す．表中で太字は最速ケースを表す．

Table 1 Performance improvement of IDR(s) method by some implementations on PQ.

opt.	prio.	lev.	solv.time		mat.time	
			[sec.](ratio-1)	ratio-2	[sec.]	ratio-3
-O0	col.	1	482.90	1.000	390.61	1.000
		2	447.56 (1.00)	0.927	356.61	0.913
		3	430.53	0.892	338.48	0.867
		4	395.07	0.818	304.31	0.779
	row	1	479.57	0.993	388.86	0.996
		2	446.41 (1.00)	0.924	354.99	0.909
		3	373.36	0.773	283.76	0.726
		4	341.06	0.706	249.64	0.639
-Kfast	col.	1	31.66	0.066	33.69	0.086
		2	28.99 (0.065)	0.060	30.82	0.079
		3	31.74	0.066	30.18	0.077
		4	29.09	0.060	27.53	0.070
	row	1	14.98	0.031	13.42	0.034
		2	**7.33** (0.016)	**0.015**	5.79	0.015
		3	14.97	0.031	13.42	0.034
		4	15.06	0.031	13.51	0.035

Table 2 Performance improvement of IDR(s) method by some implementations on SR.

opt.	prio.	lev.	solv.time		mat.time	
			[sec.](ratio-1)	ratio-2	[sec.]	ratio-3
-O0	col.	1	139.48	1.000	116.32	1.000
		2	133.25 (1.00)	0.955	110.42	0.949
		3	121.57	0.872	98.65	0.848
		4	115.17	0.826	92.27	0.793
	row	1	130.86	0.938	107.98	0.928
		2	130.79 (1.00)	0.938	107.64	0.925
		3	89.34	0.641	66.46	0.571
		4	92.24	0.661	69.25	0.595
-Oss	col.	1	8.15	0.058	7.33	0.063
		2	8.11 (0.061)	0.058	7.27	0.063
		3	8.04	0.058	7.23	0.062
		4	8.12	0.058	7.30	0.063
	row	1	6.51	0.047	5.68	0.049
		2	**6.50** (0.050)	**0.047**	5.65	0.049
		3	6.71	0.048	5.87	0.050
		4	6.70	0.048	5.88	0.051

4.2 Bi_IDR(s) 法の場合

前節と同様に，Bi_IDR(s) 法を用いて実験を行う．**Table 3** は PQ 上の Bi_IDR(s) 法の結果, **Table 4** は SR 上の Bi_IDR(s) 法の結果を表す．最速であった Bi_IDR($s = 2$) 法の結果を表に載せる．

Table 3 Performance improvement of Bi_IDR(s) method by some implementations on PQ.

opt.	prio.	lev.	solv.time		mat.time	
			[sec.](ratio-1)	ratio-2	[sec.]	ratio-3
-O0	col.	1	473.68	1.000	439.42	1.000
		2	436.91 (1.00)	0.922	402.60	0.916
		3	416.04	0.878	381.68	0.869
		4	385.84	0.815	350.49	0.798
	row	1	474.30	1.001	439.59	1.000
		2	435.88 (1.00)	0.920	401.09	0.913
		3	355.19	0.750	319.99	0.728
		4	316.69	0.669	282.23	0.642
-Kfast	col.	1	35.53	0.075	34.10	0.078
		2	32.38 (0.074)	0.068	31.06	0.071
		3	35.23	0.074	33.95	0.077
		4	32.35	0.068	31.06	0.071
	row	1	16.39	0.035	15.08	0.034
		2	**7.75** (0.018)	**0.016**	6.48	0.015
		3	17.15	0.036	15.75	0.036
		4	16.60	0.035	15.29	0.035

Table 4 Performance improvement of Bi_IDR(s) method by some implementations on SR.

opt.	prio.	lev.	solv.time		mat.time	
			[sec.](ratio-1)	ratio-2	[sec.]	ratio-3
-O0	col.	1	141.05 (1.00)	1.000	130.97	1.000
		2	134.39	0.953	124.30	0.949
		3	121.14	0.859	111.06	0.848
		4	114.00	0.808	103.92	0.793
	row	1	131.43 (1.00)	0.932	121.36	0.927
		2	131.06	0.929	120.99	0.924
		3	84.83	0.601	74.75	0.571
		4	87.96	0.624	77.89	0.595
-Oss	col.	1	8.72 (0.062)	0.062	8.17	0.062
		2	8.68	0.062	8.13	0.062
		3	8.69	0.062	8.14	0.062
		4	8.68	0.062	8.13	0.062
	row	1	**6.91** (0.053)	**0.049**	6.36	0.049
		2	6.94	0.049	6.38	0.049
		3	7.14	0.051	6.59	0.050
		4	7.17	0.051	6.60	0.050

IDR(s) 法の結果 **Table 1** と **Table 2** そして Bi_IDR(s) 法の結果 **Table 3** と **Table 4** に対する観察から，以下の知見が得られる．

- 要素係数行列を行優先に格納した場合の方が列優先よりも高速である．また，最適化レベルが最も低いとき，最速ケースとなる行列ベクトル積計算の実装レベルは計算機 PQ と SR では異なる．
- IDR(s) 法のとき，最適化レベルが最高のときのレベル lev.2 の行列ベクトル積計算方法が最速である．

さらに，要素係数行列 $A^{elem.}$ の格納方向に関して次の知見が得られる．

- 行優先の $A^{elem.}$ の格納方向により，最適化の効果が増す．
- 計算機 SR では，$A^{elem.}$ の行優先格納により，列優先の格納に比べて，若干の最適化効果があった．しかし，計算機 PQ では，$A^{elem.}$ の行優先格納により，列優先の格納に比べて，非常に大きな最適化効果が得られた．すなわち，**Table 1** では，括弧内の比 (ratio-1) が 0.065 から 0.016 に激減し，同様に **Table 3** でも，同比 0.074 から 0.018 に激減したことからわかる．

5. ま と め

EbE 方式有限要素解析において，連立一次方程式の解法である IDR(s) 法と Bi_IDR(s) 法で最も計算コストが高い行列ベクトル積の計算の高速化方法を検討した．その結果，要素係数行列 $A^{elem.}$ の格納方向を，列優先から行優先に，変更することで行列ベクトル積計算が高速化されることがわかった．さらに，行優先の $A^{elem.}$ の格納方向は，コンパイラによる自動最適化にも大きな効果があることがわかった．

参 考 文 献

1) E. Barragy, G. Carey: A parallel element-by-element solution scheme, Int. J. Num. Meth. Engrg., Vol.26, pp.2367-2382, (1988).
2) G. Beer, I. Smith and C. Duenser: Element-by-element techniques and Parallel Programing in The Boundary Element Method with Programming, Springer Vienna, (2008).
3) J. Erhel, A. Tranard and M. Vidrascu: An element-by-element preconditioned conjugate gradient method on a vector computer, Parallel Computing, Vol.17, pp.1051-1065, (1991).
4) T. Hughes, I. Levit, J. Winget: An Element-by-Element Solution Algorithm for Problems of Structural and Solid Mechanics, Comput. Meths. Appl. Mech. Engng., 36, pp.241-254 (1983).
5) 唐木田泰久，樫山和男：移動境界を考慮した CIVA 安定化有限要素法による高潮氾濫解析，第 19 回数値流体力学シンポジウム，(2005).
6) 樫山和男，野村卓史，藤間昌一：続・有限要素法による流れのシミュレーション，シュプリンガージャパン (2008).
7) 羽田康浩，桜庭雅明，樫山和男：Level Set 法を用いた並列有限要素法による自由表面流れ解析，土木学会第 58 回年次学術講演会，D5-6，pp.1-4，(2003).
8) 三好俊郎, 坂田信二, 吉田有一郎, 斎藤直人：計算力学と CAE シリーズ スーパーコンピューテイング，培風館，(2001).
9) H. Okuda, K. Ohshiro and T. Atsugi: Data-parallel computation of EBE finite element method for air/water/soil coupled systems, Computational Mechanics, Vol.23, pp.158-163, (1999).
10) 奥田洋司，阿南統久：オーダリングによる EBE データ並列有限要素法の効率化，日本機械学会論文集 (B 編)，Vol.65, No.640，pp.3869-3876，(1999).
11) 奥田洋司，工藤真吾：SMP Clusters における Element-by-Element 有限要素法の Hybrid 並列解析，日本機械学会第 14 回計算力学講演会講演論文集，pp.347-348，(2001).
12) 小野正法，吉村忍，河合浩志：線形弾性構造解析における Element-by-Element 演算のマルチコアアーキテクチャ上での性能評価，Transaction of JSCES，Paper No.20070022, (2007).
13) 小野正法，吉村忍，河合浩志：Element-by-Element Matrix Storage Free 法最適化に関する基礎的検討，計算工学講演会論文集，Vol.11，pp.483-484，(2006).
14) 尾上勇介，藤野清次：IDR(s) 法系統の反復法に適用可能な計算量削減の工夫，日本応用数理学会論文誌，Vol.19，No.3, pp.329-350，(2009).
15) 尾上勇介，藤野清次：定常型前処理つき Bi_IDR(s) 法の提案とその性能評価．(投稿中)
16) Y. Saad: Iterative methods for sparse linear systems (2nd edition), SIAM, Philadelphia, (2003).
17) 寒川 光，藤野清次，長嶋利夫，高橋大介：IT Text シリーズ： HPC プログラミング，オーム社，(2009).
18) T. Sheu, C. Fang and S. Tsai: Application of an Element-by-element BiCGStab iterative solver to a monotonic Finite Element Model, Computers and Mathematics with Applications, Vol.37, pp.57-70, (1999).
19) P. Sonneveld, M. van Gijzen: IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems, SIAM J. Sci. Comput., Vol.631, No.2, pp.1035-1062, (2007).
20) P. Sonneveld, M. van Gijzen: An elegant IDR(s) variant that efficiently exploits biorthogonality properties, Depart. of Applied Math. Anal., TR08-21, Delft University of Technology (2008).
21) 牛島省，OpenMP による並列プログラミングと数値計算法，丸善，(2006).
22) M. van Gijzen: Iterative solution methods for linear equations in finite element computations, PhD thesis, Delft University of Technology, (1994).
23) H. van der Vorst: Iterative Krylov methods for large linear systems, Cambridge University Press, (2003).
24) N. Vannieuwenhoven, K. Meerbergen: An element-by-element algebraic multilevel block-ilu preconditioner, Abstract of ICCAM 15th Int. Congress on Computational and Applied Math., Leuven, July 05-09, (2010).
25) M. Wang, T. Sheu: An element-by-element BiCGStab iterative method for three-dimensional steady Navier-Stokes equations, J. of Computational and Applied Math., Vol.79, pp.147-165, (1997).
26) J. Winget, T. Hughes: Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies, Comput. Meths. Appl. Mech. Engng., Vol.52, pp.711-815 (1985).
27) 矢川元基，奥田洋司：計算力学 VII – 計算力学における超並列計算法 –，養賢堂，(2002).