Vol.2014-HPC-145 No.20
2014/7/29

古典グラム・シュミット法に基づく逐次直交化計算の並列実装について

石上 裕之[1,2,a)]　木村 欣司[1,b)]　中村 佳正[1,c)]

概要：逐次再直交化計算向けの古典グラム・シュミット（CGS）法および CGS2 法のスレッド並列実装について，OpenMP に基づく実装法を提案する．提案実装では，OpenMP の指示文を用いた並列化を行うことで，従来の実装よりもデータ再利用性が高くなる．本研究では，提案実装に基づく CGS2 法を実対称 3 重対角行列向け逆反復法の逐次直交化計算に適用した．共有メモリ型マルチコアプロセッサシステム上での数値実験により，従来の逐次直交化アルゴリズムを適用した逆反復法に比べ，高速な固有ベクトル計算が可能となる場合が多くあることを確認した．

キーワード：逐次直交化計算，古典グラム・シュミット法，スレッド並列，OpenMP，逆反復法

1. はじめに

本研究では，共有メモリ型マルチプロセッサシステム上における逐次直交化計算について考える．逐次直交化計算は，連立 1 次方程式の解法である GMRES 法や固有対計算のための Arnoldi 法など，Krylov 部分空間を用いた様々な行列計算において現れる．また，固有ベクトル計算のための逆反復法においても，クラスター固有値に対応する固有ベクトルの計算のため，逐次直交化計算が現れる．これらのいずれの計算においても，逐次直交化計算に必要な計算量は，他の部分の計算に必要な計算量よりも多くなる．

本研究では，固有ベクトル計算のための逆反復法のスレッド並列計算における高速化を主眼に，逐次直交化計算アルゴリズムについて論じる．逆反復法の高速化のためには 2 つの方法がある．1 つ目は，行列乗算によって実装することのできる再直交化付きブロック逆反復法 [9] を用いる方法である．2 つ目は，逆反復法の逐次直交化計算アルゴリズムを高速化することである．前者は高い並列化効率を達成できるが，収束性において後者に劣る．したがって，後者による高速化も重要である．

逐次直交化計算のアルゴリズムとして，古典グラム・シュミット（CGS）法や修正グラム・シュミット（MGS）法，ハウスホルダー変換に基づく直交化法といったアルゴリズムが知られている [6]．CGS 法は行列-ベクトル乗算を用いて実装できるアルゴリズムである．しかしながら，CGS 法による逐次直交化計算では，入力ベクトル次第でベクトルの直交性が失われる場合がある．この問題の解決として，MGS 法や，CGS 法を 2 回繰り返す CGS2 法 [5] が提案されている．MGS 法は CGS 法よりも安定に計算できる一方，ベクトル乗算により実装されるアルゴリズムである．ベクトル乗算は，行列-ベクトル乗算よりも並列化効率が低いため，並列化による高速化が難しい．一方，CGS2 法は CGS 法や MGS 法の 2 倍の計算量を要してしまう．ハウスホルダー変換に基づく直交化は，入力のベクトルに寄らず，安定に計算できることが知られている．更に，compact WY 表現 [13] を用いることで行列-ベクトル乗算による実装が可能である [14] が，CGS 法や MGS 法に比べると計算量は増えてしまう．

本研究では，高速な逐次直交化計算アルゴリズムとして従来と異なる CGS 法および CGS2 法の並列実装を提案する．提案する並列実装法では，従来の CGS 法や CGS2 法の実装のように行列-ベクトル乗算についての並列化を行うのではなく，OpenMP の指示文を用いて各スレッドに割り当てた並列化を行う．その結果，各スレッドにおける計算はベクトル演算となるが，データの再利用性が高まることで CGS 法および CGS2 法の演算性能の向上が期待できる．

本論文の構成は以下の通りである．2 節では，従来の逐次直交化計算アルゴリズムを導入する．3 節では，CGS 法

[1] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University,
Kyoto 606–8501, Japan
[2] 日本学術振興会特別研究員（DC1）
Research Fellow of Japan Society for the Promotion of Science (DC1)
[a)] hishigami@amp.i.kyoto-u.ac.jp
[b)] kkimur@amp.i.kyoto-u.ac.jp
[c)] ynaka@i.kyoto-u.ac.jp

ⓒ 2014 Information Processing Society of Japan

および CGS2 法の OpenMP を用いた並列実装法を提案する．4 節では，実対称 3 重対角行列向け逆反復法を導入する．5 節では，共有メモリ型マルチコアプロセッサシステム上で逆反復法による固有ベクトル計算の数値実験を行い，提案実装の性能評価を行う．6 節はまとめである．

2. 逐次直交化アルゴリズム

本節では，従来の逐次直交化アルゴリズムを導入する．以下では，m 本の n 次元実ベクトル $\boldsymbol{v}_j$, $j = 1, \ldots, m$ $(m \leq n)$ に対して，その正規直交化ベクトル $\boldsymbol{q}_j$, $j = 1, \ldots, m$ を求める場合についてのベクトル逐次添加型の直交化問題について考える．

反復の最中において現れる問題であることから，逐次直交化アルゴリズム内部で並列化を行う必要がある．したがって，本節で導入するアルゴリズムの並列化は，逐次直交化アルゴリズム内部の演算であるベクトル演算や行列-ベクトル乗算をはじめとする BLAS（Basic Linear Algebra Subprograms）内部での並列化をせざるを得ない．一方，BLAS の実装ライブラリには，マルチコアプロセッサシステム向けにチューニングされたライブラリが存在する．本研究では，BLAS の実装ライブラリとして，Intel Math Kernel Library（MKL）を利用することを考える．Intel MKL はスレッド並列化された BLAS ルーチンを提供している．

2.1 古典グラム・シュミット法

古典グラム・シュミット（*CGS*）法 [6] はよく知られた直交化アルゴリズムである．$\boldsymbol{v}_j$ を，$\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}$ と直交するベクトル $\boldsymbol{q}_j$ $(1 \leq j \leq m,\ m \leq n)$ へと計算する場合，CGS 法は次のように定式化される．

$$\boldsymbol{q}_j = \boldsymbol{v}_j - \sum_{k=1}^{j-1} \langle \boldsymbol{q}_k, \boldsymbol{v}_j \rangle \boldsymbol{q}_k. \tag{1}$$

式 (1) は，内積や AXPY といったレベル 1 BLAS によって記述されているが，行列-ベクトル乗算を用いた次式のように再定式化できる．

$$\boldsymbol{q}_j = \boldsymbol{v}_j - Q_{j-1} Q_{j-1}^{\top} \boldsymbol{v}_j, \tag{2}$$

ここで，$Q_{j-1} = \begin{bmatrix} \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_{j-1} \end{bmatrix}$ である．行列-ベクトル乗算のようなレベル 2 BLAS ルーチンは，レベル 1 BLAS ルーチンに比べて並列計算向きである．したがって，CGS 法の並列実装では，通常，式 (2) が適用される．このとき，行列-ベクトル乗算ルーチンである GEMV を 2 回繰り返すことで実現できる．しかしながら，CGS 法によって計算されたベクトルの直交性は，元の行列の条件数に依存して悪化してしまうことが知られている．直交性を改善するため，CGS 法の改良アルゴリズムもまた提案されている．

Algorithm 1 Pseudocode of MGS algorithm

```
1: function MGS(v_j, Q_{j-1})
2:     q_j = v_j
3:     for k = 1 to j - 1 do
4:         s = -<q_k, q_j>              ▷ Call DOT
5:         q_j = q_j + s q_k            ▷ Call AXPY
6:     end for
7:     return q_j
8: end function
```

Algorithm 2 Pseudocode of CGS2 algorithm

```
1: function CGS2(v_j, Q_{j-1})
2:     u_j = v_j - Q_{j-1} Q_{j-1}^T v_j    ▷ Call GEMV×2
3:     q_j = u_j - Q_{j-1} Q_{j-1}^T u_j    ▷ Call GEMV×2
4:     return q_j
5: end function
```

2.2 修正グラム・シュミット法

修正グラム・シュミット（*MGS*）法 [6] は，CGS 法の直交性を改善したアルゴリズムの一つである．Algorithm 1 は，MGS 法によって $\boldsymbol{q}_j$ を計算する擬似コードを示したものである．MGS 法は CGS 法と同じ計算量でありながら，CGS 法よりも高い直交性を達成することが知られている．しかしながら，4 行目および 5 行目にあるように，全ての計算はベクトル演算である，内積および AXPY 演算により構成される．

2.3 CGS2 法

CGS 法の直交性の改善する別の方法として，CGS 法を 2 回繰り返す ***CGS2* 法**が提案されている [5]．Algorithm 2 は，CGS2 法によって $\boldsymbol{q}_j$ を計算する擬似コードを示したものである．2 行目および 3 行目にあるように，CGS 法と同様に，CGS2 法は行列-ベクトル乗算によって構成される．CGS 法と同様に，CGS2 法は行列-ベクトル乗算により構成されるため，レベル 2 BLAS ルーチンである GEMV を利用することができる．しかし，CGS 法や MGS 法に比べて 2 倍の計算量を要するだけでなく，j が大きくなれば行列 Q_{j-1} がキャッシュに収まりきらなくなるため，演算性能に限界がある．

2.4 compact WY 表現に基づく逐次直交化アルゴリズム

グラム・シュミット法の代わりにハウスホルダー変換に基づく直交化を用いるアルゴリズムを適用することもできる．ハウスホルダー変換に基づく直交化の場合，元の行列の条件数に依存しない高い直交性を確保できることが知られている．しかしながら，ハウスホルダー変換に基づく直交化は，ベクトル演算のみで構成されるため，並列化による高速化が難しい．

これに対して Walker は，GMRES 法に対して，行列-ベクトル乗算によって実装可能なハウスホルダー変換に基づく直交化アルゴリズムを提案した．同様に，山本らはハウ

ⓒ 2014 Information Processing Society of Japan

スホルダー変換を compact WY 表現 [13] を用いて再定式化することにより，通常のハウスホルダー変換と同等の数値安定性を持ちながら，行列-ベクトル乗算を用いて逐次直交化計算を実現できることを示した [14]．更に，著者らは compact WY 表現に現れる行列の構造に着目して適切な BLAS ルーチンを使用することにより，計算量を削減した compact WY 表現を用いたハウスホルダー変換に基づく直交化アルゴリズムの実装法を実現した [8].

3. OpenMP を用いた CGS 法の実装法

本節では，2 節で述べたものとは異なる CGS 法および CGS2 法の並列実装法を提案する．CGS 法を表す式 (1) に再度着目すると，$\sum_{k=1}^{j-1}\langle \boldsymbol{q}_k, \boldsymbol{v}_j\rangle \boldsymbol{q}_k$ は k について独立に計算することができる．このことから，この総和計算について，各スレッドに計算を割り当てた並列化が可能である．このようなスレッド並列化は，OpenMP の指示文を用いることで実現できる．

Algorithm 3 は以上のアイディアに基づいた CGS 法の擬似コードである．以下，この実装法を PCGS と呼ぶ．Algorithm 3 では，$\langle \boldsymbol{q}_k, \boldsymbol{v}_j\rangle \boldsymbol{q}_k$ の総和計算を “omp parallel for” 指示節によって k に関して 3 行目から 8 行目において並列化する．ここで，総和の内部計算 $\langle \boldsymbol{q}_k, \boldsymbol{v}_j\rangle \boldsymbol{q}_k$ はそれぞれのスレッドにおいて計算することになる．まず，$s = -\langle \boldsymbol{q}_k, \boldsymbol{v}_j\rangle$ をプライベート変数として，それぞれのスレッドで計算する（5 行目）．そして，$\boldsymbol{q}_j = \boldsymbol{q}_j + s\boldsymbol{q}_k$ を AXPY ルーチンにより計算する（6 行目）．各スレッドで得られた $\boldsymbol{q}_j$ はリダクション演算によって 1 つのベクトルにする必要がある．このような配列リダクションは，Fortran コードにおいては “reduction” 指示節を用いることで簡易に実装できる．

ここで，5 行目および 6 行目における計算は，j の値に関わらず，3 つのベクトル $\boldsymbol{v}_j$，$\boldsymbol{q}_j$，$\boldsymbol{q}_k$ についてキャッシュヒット率の高い計算となる．したがって，PCGS は並列計算において CGS 法の従来実装よりも高いパフォーマンスが得られると期待できる．

PCGS を 2 回繰り返すことによって，CGS2 法と同様の演算が可能となる．以下，提案実装に基づく CGS2 法を PCGS2 と呼ぶ．Algorithm 4 は，PCGS2 の擬似コードを表したものである．PCGS2 においても，PCGS 同様の利点がある．

尚，分散メモリ型計算機向けではあるが，同様の並列化手法が [10], [11] で提案されている．

4. 逆反復法による固有ベクトル計算

本研究では，前節で示した提案実装 PCGS2 の性能評価として，逆反復法による固有ベクトル計算コードへの実装を行った．Algorithm 5 は，n 次実対称 3 重対角行列 T の m 本の固有ベクトルを計算する逆反復法の擬似コードである．Algorithm 5 は LAPACK（Linear Algebra PACKage[1]）に実装された実対称 3 重対角行列向けの逆反復法コードである，xSTEIN [7] を基としたものである．ここで，I は n 次単位行列とし，T の固有値を λ_j ($\lambda_1 < \lambda_2 < \cdots < \lambda_m$, $m \leq n$)，その近似値を $\tilde{\lambda}_j$ としている．

Algorithm 3 Pseudocode of PCGS

```
 1: function PCGS(v_j, Q_{j-1})
 2:     q_j = v_j
 3:     #omp parallel for private(s) reduction(+:q_j)
 4:     for k = 1 to j − 1 do
 5:         s = −⟨q_k, v_j⟩                    ▷ Call DOT
 6:         q_j = q_j + s q_k                  ▷ Call AXPY
 7:     end for
 8:     #omp end parallel for
 9:     return q_j
10: end function
```

Algorithm 4 Pseudocode of PCGS2

```
1: function PCGS2(v_j, Q_{j-1})
2:     [u_j] = PCGS(v_j, Q_{j-1})
3:     [q_j] = PCGS(u_j, Q_{j-1})
4:     return q_j
5: end function
```

隣り合う固有値が十分に離れている場合は，5 行目および 8 行目に表される計算の反復のみで $\boldsymbol{v}_j^{(i)}$ は λ_j の固有ベクトルへと収束する．しかし，隣り合う固有値が近接している，クラスター固有値である場合には，10 行目にあるような逐次的な直交化計算を行うことにより，固有ベクトルの直交性を確保する必要がある．ここで，j_1 はあるクラスターの最小固有値のインデックスである．クラスター固有値であるかどうかの判定は 9 行目において，Peters と Wilkinson によって提案された基準 [12] を用いて行っている．

10 行目の逐次直交化計算には，xSTEIN では MGS 法が採用されている．この逐次直交化計算には，前節までで導入した他のアルゴリズムも適用可能であるが，どのアルゴリズムを採用しても，クラスターに m 個の固有値が含まれる場合，$O(m^2 n)$ の計算量が必要となる．更に，Peters-Wilkinson の判定基準を用いる場合，数千次以上の行列の固有値のほとんどが一つのクラスターと見なされてしまうことが知られている [3]．このため，逐次直交化の対象となるベクトルの本数 k が大きくなり，逆反復法による固有ベクトル計算の計算時間の大半が，逐次直交化計算に占められることとなる他，逐次直交化計算の内部での並列化が必要となる．

5. 性能評価

本節では，共有メモリ型マルチコアプロセッサシステム上での数値実験について報告する．数値実験は，表 1 に表される京都大学学術情報メディアセンターのスーパーコンピュータ Appro Green Blade 8000 を 1 ノード使用した．

実験では，2 節および 3 節で述べた逐次直交化アルゴリズムを 3 重対角行列向け逆反復法の逐次直交化計算ア

Algorithm 5 Inverse iteration algorithm

```
1: function Inv(T, λ̃_1, ..., λ̃_m)
2:     for j = 1, ..., m do
3:         i := 0
4:         Generate v_j^(0) from random numbers
5:         T − λ̃_j I := P_j L_j U_j
6:         repeat
7:             i := i + 1
8:             Solve P_j L_j U_j v_j^(i) = v_j^(i−1)
9:             if |λ̃_{j−1} − λ̃_j| ≤ 10^{−3}||T|| then
10:                Reorthogonalize v_j^(i) against q_{j_1}, ..., q_{j−1}
11:            end if
12:        until converge
13:        Normalization: q_j := v_j^(i)/||v_j^(i)||
14:        Q_j = [Q_{j−1}  q_j]
15:    end for
16:    return Q_m
17: end function
```

表 1: 実験環境
Table 1 Specifications of the experimental environment

CPU	Intel Xeon E5-2670
	2.6GHz, 8cores×2, L3 cache: 20MB
RAM	DDR3-1600 64GB
	Max Memory Bandwidth: 102.4 GB/sec
Compliler	Intel Fortran Compiler 14.0.2
Options	-O3 -xHOST -ipo -no-prec-div
	-mcmodel=medium -shared-intel -openmp
Software	Intel Math Kernel Library 11.1.2

ルゴリズムとして実装した 4 つのコード **Inv-MGS**，**Inv-CGS2**，**Inv-cWY**，**Inv-PCGS2** を比較した．Inv-MGS は，Intel MKL の DSTEIN ルーチンそのもので，逐次直交化計算において MGS 法が実装されている．ここで、Intel MKL の DSTEIN はシリアルスレッドで実装されている．その他のコードは，LAPACK の DSTEIN をベースとして，逐次直交化計算部分が異なるアルゴリズムを実装した．Inv-CGS2 には，Algorithm 2 に表されるような行列-ベクトル乗算による CGS2 法による逐次直交化計算を実装した．Inv-cWY には，[8] で提案した，compact WY 表現を用いたハウスホルダー変換に基づく逐次直交化計算を実装した．Inv-PCGS2 では，3 節で提案した PCGS2（Algorithm 4）を実装した．

BLAS のライブラリとしては，Intel Math Kernel Library (MKL) を用いた．これにより，Inv-CGS2，Inv-cWY の逐次直交化計算では，16 スレッド並列で BLAS 演算が実行される．

逆反復法において，反復回数は 5 回まで許容しているが，いかなる入力行列の場合でも 3 回の反復回数で収束することが確認できている．また，各テスト行列の固有値は，Intel MKL に実装された DSTEBZ ルーチンを利用することにより計算した．DSTEBZ は，実対称 3 重対角行列の固有値を 2 分法によって計算する倍精度演算ルーチンである．

5.1 数値実験 I

数値実験 I では，固有値分布が異なる 2 種類の n 次行列 T_1，T_2 の全固有ベクトル計算に要した実行時間を比較した．

T_1 は，Glued-Wilkinson 行列 [2], [4] である．この行列の固有値は，Peters-Wilkinson の判定基準において，大きさが $n/21$ となるクラスターと $2n/21$ となるクラスターがそれぞれ 7 つ，計 14 のクラスターに分かれることが知られている．T_2 は，全要素を $(0, 1)$ の範囲の乱数により生成した実対称 3 重対角行列である．この行列の固有値は，サイズの小さな行列では数十から数百のクラスターに分かれるが，サイズの大きな行列ではほとんどが一つのクラスターに含まれる．本実験においても，10500 次以上の行列 T_2 の多くは，9 割以上の固有値が一つのクラスターに属していた．

図 1 は，全固有ベクトル計算において各コードが要した実行時間を比較したものである．ここで，図 1a は行列 T_1，図 1b は行列 T_2 の場合を表している．

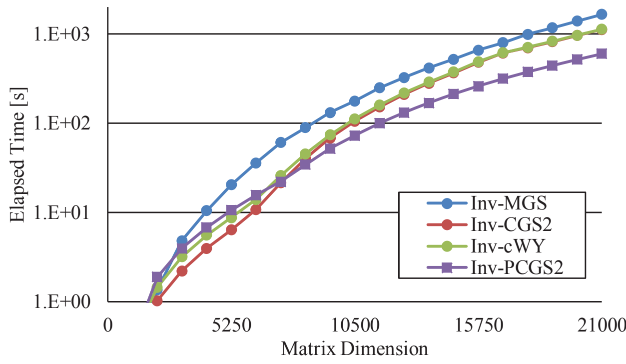
図 1a および 1b から，Inv-PCGS2 はある程度サイズの大きな行列に対しては他のアルゴリズムに対して高速であることが分かる．したがって，Inv-PCGS2 の逐次直交化計算に適用した OpenMP を用いた CGS2 法の実装により，計算性能が向上したということが確認できる．実際，$n = 21000$ の行列 T_1 に対して Inv-PCGS2 は，Inv-MGS の 2.8 倍，Inv-CGS2 の 1.9 倍，Inv-cWY の 1.9 倍高速である．また，$n = 20000$ の行列 T_2 に対して Inv-PCGS2 は，Inv-MGS の 3.0 倍，Inv-CGS2 の 2.1 倍，Inv-cWY の 1.4 倍高速である．その一方で，サイズの小さな行列では，他のコードが Inv-PCGS2 よりも高速である．これは，Inv-PCGS2 の逐次直交化計算において配列リダクション演算を要することから，そのオーバーヘッドによる計算時間の遅延が大きく影響しているものと考えられる．逆に，サイズの大きな行列では，キャッシュヒット率の向上による演算性能の改善が大きいため，リダクションによる遅延の影響が少なくなったと考えられる．

5.2 数値実験 II

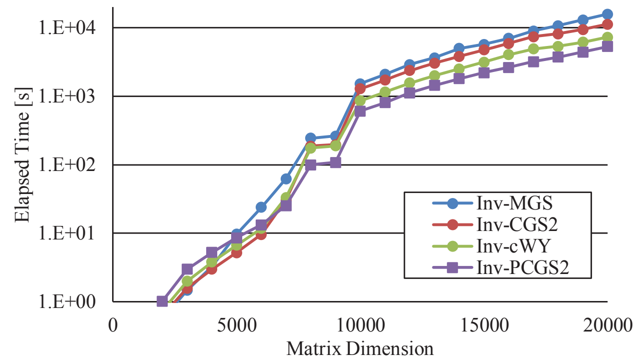
数値実験 II では，最大固有値から m 個の固有値に対応する固有ベクトルのみを計算する場合について各コードの実行時間を比較した．ここでは，テスト行列として全ての対角成分と副対角成分が 1 である行列 T_3 を用いた．行列 T_3 の全固有値は Peters-Wilkinson の判定基準において一つのクラスターに属する．したがって，固有ベクトルを m 本求める場合には，ベクトル m 本の逐次直交化を行うことになる．

図 2 は，行列 T_3 の m 本の固有ベクトルを計算するのに各コードが要した時間を示している．ここで, 図 2a, 2b, 2c, 2d は, それぞれ行列のサイズが $n = 5000, 10000, 15000, 20000$ の場合のものに相当する．

ⓒ 2014 Information Processing Society of Japan



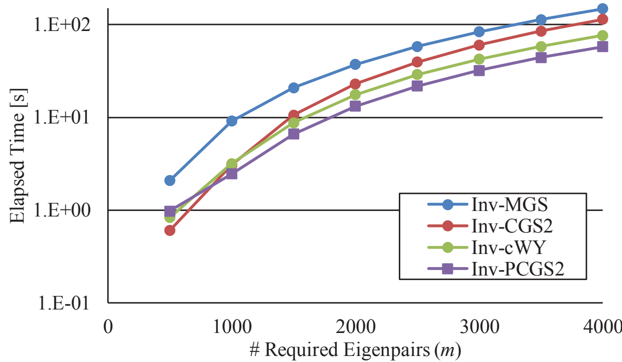
(a) Cases of T_1



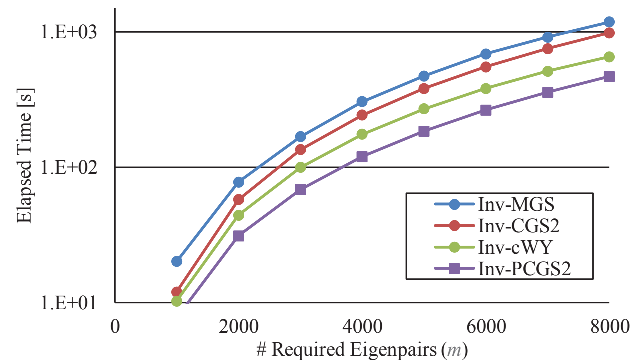
(b) Cases of T_2

図 1: 行列 T_1 および T_2 の全固有ベクトル計算に要した実行時間の比較.

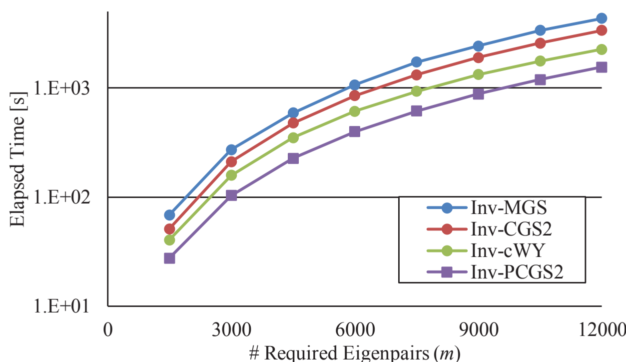
Fig. 1 Comparison of elapsed time for computing all the eigenvectors by each code.



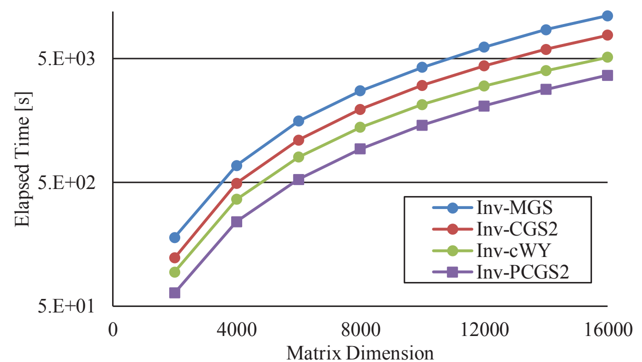
(a) Cases of $n = 5000$



(b) Cases of $n = 10000$



(c) Cases of $n = 15000$



(d) Cases of $n = 20000$

図 2: 行列 T_3 の部分固有対計算に要した実行時間の比較.

Fig. 2 Comparison of elapsed time for computing m eigenvectors of T_3 by each code.

図 2 から，本研究における数値実験では $n = 5000$ かつ $m = 500$ のときを除き，Inv-PCGS2 が他のアルゴリズムよりも高速であることが分かる．特に，求めたい固有ベクトルの本数 m や行列サイズ n の値が大きいときほど，Inv-CGS2 や Inv-MGS に対する性能差は大きい．一方，$n = 5000$ かつ $m = 500$ のときには，Inv-PCGS2 よりも Inv-CGS2 の方が高速であった．この結果も，Inv-PCGS2 に含まれる配列リダクションによる計算時間のオーバーヘッドが起因していると考えられる．以上の結果をまとめると，逐次直交化計算を行うベクトルのサイズが大きく，対象となるベクトルの本数が多くなるほど，Inv-PCGS2 が Inv-CGS2 に対して高速になると期待できることが分かる．

6. まとめ

本研究では，古典グラム・シュミット法に基づく逐次直交化計算のスレッド並列実装について，OpenMP を用いた

ⓒ 2014 Information Processing Society of Japan

実装法を提案した．この実装では，スレッド毎の演算はレベル 1 BLAS ルーチンによるものであるが，データの再利用性が高まることで高速化が期待できる．また，提案実装に基づく CGS2 法を，実対称 3 重対角行列の固有ベクトル計算のための逆反復法に適用した．共有メモリ型マルチコアプロセッサシステム上での数値実験を通して，従来の逐次直交化アルゴリズムを適用した場合に比べ，更に高速な固有ベクトル計算が可能となる場合が多くあることを確認した．

本研究で提案した古典グラム・シュミット法の実装法は，キャッシュ容量やコア数の異なる計算機環境では，本研究で得られた結果とは異なる結果が得られると考えられる．また，実験結果から，逐次直交化するベクトルの次元が小さい，または，ベクトルの本数が少ない場合には，行列-ベクトル乗算で実装した古典グラム・シュミット法が高速になりうる．以上より，計算機環境および入力行列のサイズ，ベクトルの本数といったパラメータに応じて，逐次直交化アルゴリズムを選択できることが望ましい．このようなオートチューニング機能を開発することが，今後の課題の一つである．また，GMRES 法をはじめとする Krylov 部分空間を用いた行列計算アルゴリズムに提案実装の導入することも，今後の課題である．

謝辞　本研究は科学研究費補助金特別研究員奨励費（課題番号：25・2820），基盤研究（B）（課題番号：24360038）の補助を受けている．本研究の結果の一部は，京都大学学術情報メディアセンターのスーパーコンピュータ Appro Green Blade 8000 を利用して得られたものである．

参考文献

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J. W., Dongarra, J., Du Croz, J., Hammarling, S., Greenbaum, A., McKenney, A. and Sorensen, D.: *LAPACK Users' Guide (Third ed.)*, SIAM, Philadelphia, PA, USA (1999).

[2] Demmel, J. W., Marques, O. A., Parlett, B. N. and Vömel, C.: Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers, *SIAM J. Sci. Comput.*, Vol. 30, No. 3, pp. 1508–1526 (2008).

[3] Dhillon, I. S.: A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem, PhD Thesis, EECS Department, University of California, Berkeley (1997).

[4] Dhillon, I. S., Parlett, B. N. and Vömel, C.: Glued matrices and the MRRR algorithm, *SIAM J. Sci. Comput.*, Vol. 27, No. 2, pp. 496–510 (2005).

[5] Giraud, L., Langou, J., Rozloẑnìk, M. and van den Eshof, J.: Rounding error analysis of the classical Gram-Schmidt orthogonalization process, *Numer. Math.*, Vol. 101, No. 1, pp. 87–100 (2005).

[6] Golub, G. H. and van Loan, C. F.: *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA (1996).

[7] Ipsen, I. C. F.: Computing an Eigenvector with Inverse Iteration, *SIAM Review*, Vol. 39, No. 2, pp. 254–291 (1997).

[8] Ishigami, H., Kimura, K. and Nakamura, Y.: On implementation and evaluation of inverse iteration algorithm with compact WY orthogonalization, *IPSJ Transactions on Mathematical Modeling and Its Applications*, Vol. 6, No. 2, pp. 25–35 (2013).

[9] 石上裕之，木村欣司，中村佳正：再直交化付きブロック逆反復法による固有ベクトルの並列計算，2014 年ハイパフォーマンスコンピューティングと計算科学シンポジウム予稿集，pp. 65–75 (2013).

[10] 片桐孝洋，吉瀬謙二，本多弘樹，弓場敏嗣：データ再分散を行う並列 Gram-Schmidt 再直交化，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 45, No. SIG06(ACS6), pp. 75–85 (2004).

[11] Katagiri, T.: Performance Evaluation of Parallel Gram-Schmidt Re-orthogonalization Methods, *High Performance Computing for Computational Science - VECPAR 2002*, Lecture Notes in Computer Science, Vol. 2565, Springer Berlin Heidelberg, pp. 302–314 (2003).

[12] Peters, G. and Wilkinson, J.: *The calculation of specified eigenvectors by inverse iteration*, Handbook for Automatic Computation, pp. 418–439, Springer-Verlag, Berlin (1971).

[13] Schreiber, R. and van Loan, C.: A storage-efficient WY representation for products of Householder transformations, *SIAM J. Sci. Stat. Comput.*, Vol. 10, No. 1, pp. 53–57 (1989).

[14] Yamamoto, Y. and Hirota, Y.: A parallel algorithm for incremental orthogonalization based on the compact WY representation, *JSIAM Letters*, Vol. 3, pp. 89–92 (2011).

© 2014 Information Processing Society of Japan