情報処理学会論文誌　コンピューティングシステム　Vol.7　No.3　1–12 (Aug. 2014)

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

石上 裕之[1,2,a)]　木村 欣司[1,b)]　中村 佳正[1,c)]

受付日 2013年12月6日, 採録日 2014年1月22日

概要：本論文では，並列計算機向けの実対称 3 重対角行列の固有ベクトル計算アルゴリズムとして再直交化付きブロック逆反復法を提案する．逆反復法による固有ベクトル計算における再直交化計算では，従来，ベクトル演算や行列–ベクトル乗算といった並列化粒度の比較的小さい演算を用いたアルゴリズムが中心であった．また，逆反復法の改良アルゴリズムとして Multiple Relatively Robust Representation（MRRR）法が提案されているが，計算対象の行列の固有値分布によっては，固有ベクトルの直交性が失われてしまうことが知られている．本論文で提案する再直交化付きブロック逆反復法は，行列乗算中心の実装が可能な同時逆反復法を基にした，大粒度の並列性を持つアルゴリズムである．さらに，提案アルゴリズムにより，MRRR 法で計算が破綻してしまうような固有ベクトルを行列乗算を用いて再計算することも可能になる．共有メモリマルチコアプロセッサシステム上での数値実験において，提案アルゴリズムにより，逆反復法や同時逆反復法と同等の計算精度が達成され，より高速な並列計算が実現されることを示す．

キーワード：固有ベクトル計算，逆反復法，同時逆反復法，再直交化，マルチコア計算

Reorthogonalized Block Inverse Iteration Algorithm for Parallel Computation of Eigenvectors

Hiroyuki Ishigami[1,2,a)]　Kinji Kimura[1,b)]　Yoshimasa Nakamura[1,c)]

Received: December 6, 2013, Accepted: January 22, 2014

Abstract: A reorthogonalized block inverse iteration algorithm is proposed for parallel computation of eigenvectors for symmetric tridiagonal matrices. The reorthogonalization process in the inverse iteration algorithm for computing eigenvectors is mainly based on the vector operations or the matrix-vector multiplications, whose parallel granularity is relatively small. Multiple Relatively Robust Representations (MRRR) algorithm is also proposed for computing eigenvectors, but the MRRR algorithm occasionally loses orthogonality. The proposed algorithm is derived from the simultaneous inverse iteration algorithm, which enables us to implement matrix-matrix multiplications and then has large parallel granularity. The proposed algorithm helps us to modify eigenvectors of the matrix, which the MRRR algorithm fails to compute with good orthogonality. Numerical experiments on shared memory multi-core processors show that the proposed algorithm achieves high accuracy as and is faster than both the inverse iteration algorithm and the simultaneous inverse iteration algorithm.

Keywords: eigenvector computation, inverse iteration, simultaneous inverse iteration, reorhtogonalization, multi-core processing

[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

1. はじめに

$n \times n$ 実対称密行列 A に対する標準固有対問題

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i, \quad i = 1, \ldots, n$$

を考える．ここで，$\lambda_i \in \mathbb{R}$ は A の固有値 $(\lambda_1 < \cdots < \lambda_n)$，$\boldsymbol{v}_i \in \mathbb{R}^n$ は λ_i に対応する固有ベクトルである．実対称密

行列の固有対計算は多くの科学技術計算で現れる基本的な線形計算である．振動解析における固有モード計算や蛋白質などの構造解析で用いられる分子軌道計算では，全固有対ではなく，一部の固有対のみが必要とされる．以下では，このような問題を部分固有対問題と呼ぶ．近年，これらのシミュレーションは大規模かつ複雑なものになり，部分固有対問題の対象となる行列も大規模化している．このような大規模な問題を高速に解くため，近年では並列計算機の利用が一般的となっており，並列計算機向きの部分固有対計算法の需要が高まっている．

一般に，実対称行列の固有対計算は，前処理として直交変換によって 3 重対角化した後，3 重対角行列の固有対を計算する．元の行列の固有ベクトルは，3 重対角化の逆変換を施すことによって得られる．3 重対角化や逆変換の手法については，高い並列化効率を達成する様々なアルゴリズムが提案されている [4], [12], [15], [22].

実対称 3 重対角行列の一部の固有対のみを計算する手法としては，二分法と逆反復法を組み合わせる方法（BI 法）[17] や MRRR（Multiple Relatively Robust Representations）法 [7], [9] がある．どちらの手法も，数値計算ライブラリ LAPACK（Linear Algebra PACKage [2]）に採用されており，STEVX，STEGR としてルーチンが提供されている．

BI 法では，二分法で求めたい固有値を計算し，得られた固有値を基に，対応する固有ベクトルを逆反復法によって計算する．二分法による固有値計算は，大粒度の並列計算が可能である一方，従来の逆反復法はマルチコア CPU 上においても並列化しにくいことが知られている．これは，クラスタ固有値に対応する固有ベクトル計算において，ベクトル演算中心の再直交化計算を行っており，並列化による高速化のボトルネックとなっているからである．この問題に対して，行列–ベクトル乗算を用いた再直交化アルゴリズムを適用することで，固有ベクトルの並列計算を高速化ができることが示されている [14], [16]．しかし，キャッシュヒット率や並列化効率の点で，行列–ベクトル乗算は行列乗算に及ばない．行列乗算を用いた逆反復アルゴリズムがあれば，さらに高い並列化効率を達成できると期待される．

一方，MRRR 法もまた逆反復計算に基づくアルゴリズムであるが，逆反復法とは異なり再直交化計算を回避するため，逆反復法に比べて計算量を抑えることができる．また，文献 [19] では，並列計算機向けの MRRR 法の実装法も提案されており，高い並列化効率を達成することが示されている．しかし，Glued-Wilkinson 行列 [6], [8] のような非常に密集した固有値を持つ行列に対しては，MRRR 法の計算時間は増大してしまうことが報告されている [6]．また，行列の固有値分布によっては，MRRR 法による固有ベクトル計算では直交性を失ってしまう例があることも知られている．実際，下位ルーチンとして 3 重対角行列向けの MRRR 法を実装した LAPACK の SYEVR ルーチンに対して，特定の行列に対するバグ報告がなされており，いまだ解決がなされていない [1]．このように，計算量の意味で有利な MRRR 法でも対応できない行列があることから，二分法と逆反復法の組合せによる固有対計算法の並列実装についても研究を進める必要がある．また，MRRR 法による固有ベクトル計算が破綻してしまうケースについて，逆反復法で再計算することで固有ベクトルの直交性を改善する方法も考えられる．もちろんその際にも，行列乗算での実装が可能なアルゴリズムが求められる．

本研究では，行列乗算を用いた並列化効率の高い固有ベクトル計算法として，再直交化つきブロック逆反復法を提案する．このアルゴリズムは，行列乗算を中心とした実装が可能な同時逆反復法を基にした，大粒度の並列性を持つアルゴリズムである．提案アルゴリズムを導入することにより，MRRR 法で計算が破綻してしまうような固有ベクトルを，行列乗算中心のアルゴリズムで再計算することも可能になる．本論文の構成は以下のとおりである．2 章では，固有ベクトル計算法の従来アルゴリズムとして逆反復法および同時逆反復法について述べる．3 章では，提案アルゴリズムである再直交化付きブロック逆反復法について述べる．4 章では，本研究の性能評価で用いる実装コードについて述べる．実装コードはマルチコアプロセッサ向けのもので，スレッド並列化の方法についても述べる．5 章では，共有メモリマルチコアプロセッサシステム上での性能評価の結果を示す．6 章はまとめである．

2. 従来アルゴリズム

本章では，従来アルゴリズムとして，逆反復法および同時逆反復法について述べる．また，本研究では，倍精度演算による数値計算について論じる．

2.1 逆反復法

n 次実対称 3 重対角行列 T の固有ベクトル計算について考える．T の固有値を $\lambda_j \in \mathbb{R}$（$\lambda_1 < \lambda_2 < \cdots < \lambda_n$）とし，$\boldsymbol{q}_j \in \mathbb{R}^n$ を λ_j に対応する T の固有ベクトルとする．逆反復法による固有ベクトル計算では，まず，λ_j の近似固有値 $\tilde{\lambda_j}$，一様乱数により生成した n 次元ベクトル $\boldsymbol{v}_j^{(0)}$ を用意する．このとき，以下の連立方程式を反復して解くことによりベクトル $\boldsymbol{v}_j^{(i)}$ が得られる．

$$\left(T - \tilde{\lambda}_j I\right) \boldsymbol{v}_j^{(i)} = \boldsymbol{v}_j^{(i-1)}, \quad i = 1,\ 2,\ \ldots. \tag{1}$$

ここで，I は n 次単位行列である．$i \to \infty$ において，$\boldsymbol{v}_j^{(i)}$ は固有ベクトル $\boldsymbol{q}_j$ に収束する．m（$\leq n$）本の固有ベクトルを逆反復法で計算する場合，計算量は $O(mn)$ となる．実装上では，反復計算の過程におけるオーバフローの回避のため，反復ごとに $\boldsymbol{v}_j^{(i)}$ を正規化しなければならない．

Algorithm 1 逆反復法（Inv）

```
1: function Inv(T, λ̃_1, ..., λ̃_{m_c})
2:     for j := 1, ..., m_c do
3:         i := 0
4:         Generate v_j^(0) from random numbers
5:         T − λ̃_j I := P_j L_j U_j    ▷ 部分ピボット選択付き LU 分解
6:         repeat
7:             i := i + 1
8:             Solve P_j L_j U_j v_j^(i) = v_j^(i−1)    ▷ 連立方程式の求解
9:             v_j^(i) ⊥ Q_{j−1} となるよう v_j^(i) を再直交化
10:        until converge
11:        Normalize q_j := v_j^(i)/‖v_j^(i)‖
12:    end for
13:    return Q_{m_c} = [q_1  ···  q_{m_c}]
14: end function
```

T の固有値が互いに十分離れている場合，上記の逆反復計算により直交性を保ったまま固有ベクトルを得ることができる．しかし，固有値どうしが近接している場合には，逆反復計算のみでは直交性が失われてしまうことが知られており，この場合それらの固有ベクトルについて再直交化計算を行う手法が提案されている [13]．以下では，近接している固有値をクラスタと呼ぶ．隣り合う固有値どうしが同じクラスタに属するかどうかについては，Peters-Wilkinson の判定基準 [18] が広く使われている．この判定基準では，$|\tilde{\lambda}_{j-1} - \tilde{\lambda}_j| \leq 10^{-3}\|T\|$（$2 \leq j \leq m$）を満たすとき，$\lambda_{j-1}$ と λ_j は同じクラスタに属すると見なす．ここで，行列ノルム $\|T\|$ としてはどのノルムをとってもよいとされているが，実装上は，1-ノルムを用いる．以下の議論においても，クラスタの判定基準として，Peters-Wilkinson の判定基準を用いるものとする．

Algorithm 1 はクラスタ固有値に対して再直交化計算を施す逆反復法の擬似コードを表している．Algorithm 1 は LAPACK に提供されている実対称 3 重対角行列向け逆反復法の倍精度実装コード DSTEIN に基づいている．式 (1) を数値的に解くため，$T - \tilde{\lambda}_j I$ を部分ピボット選択付き LU 分解し，得られた $P_j L_j U_j$ を用いて連立方程式の求解を行う．ここで，ある固有ベクトルの計算過程において LU 分解の結果は不変であることから，5 行目に 1 度だけ行い，結果をメモリに保存しておくことで計算量を削減できる．連立方程式の求解は，反復ループ内部にあたる 8 行目において行う．9 行目は再直交化プロセスにあたり，DSTEIN では修正グラム・シュミット（MGS）法が採用されている．m_c 個のクラスタ固有値に対応する固有ベクトルを再直交化する場合，計算量は $O\left(m_c^2 n\right)$ となる．このため，クラスタ固有値の数が多くなれば，固有ベクトル計算にかかる計算時間の大半が再直交化に費やされることになる．

Peters-Wilkinson の判定基準を用いた場合，数千以上のサイズの行列の固有値の大半が 1 つの固有値クラスタに属してしまうことが知られている [7]．このため，逆反復法による固有ベクトル計算を行う場合，固有値クラスタに着目した並列化ではなく，反復内の各演算において並列化しなくてはならない．MGS 法は，AXPY 演算や内積計算といった演算内での並列化には不向きな演算のみで実装されているため，MGS 法を再直交化プロセスに採用している DSTEIN を並列計算でうまく高速化することは難しい．

逆反復法を並列計算において高速化するため，再直交化プロセスにおいて行列–ベクトル乗算中心のアルゴリズムを用いる方法が提案されている [14], [16]．1 つは，行列–ベクトル乗算が利用可能な古典グラム・シュミット（CGS）法を用いる方法である．CGS 法は MGS 法と代数的に等価であるため，計算量は同じである．行列–ベクトル乗算で実装可能であるため，MGS 法よりも演算性能を向上させることができる一方，CGS 法による再直交化計算では直交性が悪化する可能性があることが知られている．これを回避するため，CGS 法を 2 回繰り返す CGS2 法 [11] が提案されているが，計算量は MGS 法の 2 倍必要となる．もう 1 つは，Householder 変換による再直交化計算に compact WY 表現 [20] を導入する方法である [23]．この方法について，文献 [14] では計算量を削減した実装が提案されているが，最小でも MGS 法の 1.5 倍の計算量が必要となる．このように，行列–ベクトル乗算を利用できる再直交化アルゴリズムは，いずれも MGS 法より多くの計算量を必要とする．このため，計算機環境や使用するライブラリによっては，MGS 法を用いた逆反復法よりも高速な並列計算ができるとは限らない．

2.2　同時逆反復法

同時逆反復法 [5] は，クラスタ固有値に対応する固有ベクトルの計算に行列乗算を利用できるアルゴリズムである．文献 [5] では，行列 T の重複固有値 σ に対する定式化が示されている．本論文では，行列 T のあるクラスタ固有値 $\lambda_1, \ldots, \lambda_{m_c}$ に対して，それぞれの近似値 $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_{m_c}$ を用いて定式化したアルゴリズムを同時逆反復法とする．

同時逆反復法では，まず，$n \times m_c$ 行列 $V^{(0)}$ を乱数により生成し，QR 分解 $V^{(0)} := Q^{(0)}R^{(0)}$ により正規直交行列 $Q^{(0)}$ を計算する．$Q^{(0)}$ を初期行列とした以下の 2 式で表される計算を反復することで，$\lambda_1, \ldots, \lambda_{m_c}$ に対応する固有ベクトル $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{m_c}$ を計算することができる．

$$\left(T - \tilde{\lambda}_k I\right) \boldsymbol{v}_k^{(i-1)} = \boldsymbol{q}_k^{(i-1)},\ k = 1, \ldots, m_c, \tag{2}$$

$$V^{(i)} := Q^{(i)} R^{(i)}. \tag{3}$$

ここで，$\boldsymbol{v}_k^{(i)}$ と $\boldsymbol{q}_k^{(i)}$ はそれぞれ $V^{(i)}$ と $Q^{(i)}$ の第 k 列ベクトルである．同時逆反復法は，式 (2) において，逆反復法の式 (1) で表される連立方程式の求解を m_c 本同時に行い，再直交化計算の代わりに式 (3) の QR 分解をすることにより固有ベクトル計算を行う．

Algorithm 2 は，同時逆反復法を表す擬似コードである．同時逆反復法においても，連立方程式 (2) の求解は，部分

Algorithm 2 同時逆反復法（SI）

```
1: function SI(T, λ̃_1, ..., λ̃_{m_c})
2:     Generate V^(0) = [v_1^(0) ... v_{m_c}^(0)]
3:     V^(0) := Q^(0) R^(0)                          ▷ QR 分解
4:     for k := 1, ..., m_c do
5:         T − λ̃_k I := P_k L_k U_k   ▷ 部分ピボット選択付き LU 分解
6:     end for
7:     i := 0
8:     repeat
9:         i := i + 1
10:        for k = 1, ..., m_c do
11:            Solve P_k L_k U_k v_k^(i) = q_k^(i−1)   ▷ 連立方程式の求解
12:        end for
13:        V^(i) := Q^(i) R^(i)                      ▷ QR 分解
14:    until converge
15:    return Q := [Q^(i)]
16: end function
```

ピボット選択付き LU 分解（5 行目）の結果をメモリに保存し，それらを用いて連立方程式の求解（11 行目）を行う．ここで，5 行目および 11 行目の演算は，k について同期なしで並列化が可能である一方，保存しなければならない P_k，L_k，U_k の要素数は，逆反復法の m_c 倍必要になる．13 行目における $n \times m_c$ 行列 $V^{(i)}$ の QR 分解は行列乗算中心の実装が可能で，逆反復法における再直交化計算と同程度の計算量で行える．

以上のように，同時逆反復法では，大半の計算を行列乗算を中心とした実装が可能なだけでなく，行列乗算で実装できない計算についても，同期コストの少ない並列化を施すことが可能である．したがって，同時逆反復法は逆反復法よりも高速な並列計算が期待できる．

3. 提案アルゴリズム

本章では，同時逆反復法にブロックサイズパラメータ r（$< m_c$）を導入した，再直交化付きブロック逆反復法を提案する．以下では簡単のため，r を m_c の約数とする．

3.1 再直交化付きブロック逆反復法

再直交化付きブロック逆反復法では，ある m_c 本の固有ベクトルを r 本ごとのブロックと見なし，ブロック内のすべての固有ベクトルを計算した後，次のブロックについての固有ベクトル計算を行う．固有ベクトルのうち，$\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{(j-1)r}$ の計算が終了し，$\boldsymbol{q}_{(j-1)r+1}, \ldots, \boldsymbol{q}_{jr}$ の r 本の計算を行う場合について示す．ここで，$j = 1, 2, \ldots, m_c/r$ とする．

再直交化付きブロック逆反復法では，$n \times r$ 行列 $V_{j,r}^{(0)}$ を乱数により生成し，QR 分解 $V_{j,r}^{(0)} := Q_{j,r}^{(0)} R_{j,r}^{(0)}$ により正規直交行列 $Q_{j,r}^{(0)}$ を得る．$Q_{j,r}^{(0)}$ を初期行列として次の 3 ステップからなる反復計算を行うことにより，$\lambda_{(j-1)r+1}, \ldots, \lambda_{jr}$ に対応する固有ベクトル $\boldsymbol{q}_{(j-1)r+1}, \ldots, \boldsymbol{q}_{jr}$ が得られる．ここで，$V_{j,r}^{(i)} = \left[\boldsymbol{v}_{(j-1)r+1}^{(i)} \ \cdots \ \boldsymbol{v}_{jr}^{(i)}\right]$，$Q_{j,r}^{(i)} = \left[\boldsymbol{q}_{(j-1)r+1}^{(i)} \ \cdots \ \boldsymbol{q}_{jr}^{(i)}\right]$ とする．

(i) 以下の r 本の連立方程式を解く：

$$\left(T - \tilde{\lambda}_k I\right) \boldsymbol{v}_k^{(i)} = \boldsymbol{q}_k^{(i-1)},\ k = (j-1)r+1,\ \ldots,\ jr. \tag{4}$$

(ii) $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{(j-1)r}$ と直交するように $V_{j,r}^{(i)}$ を再直交化する．

(iii) (ii) で得られたベクトルを並べた $n \times r$ 行列を QR 分解し，$Q_{j,r}^{(i)}$ とする．

同時逆反復法と異なる点は，まず，(i) において解くべき連立方程式が r 本になることである．この計算も同期なしの並列化が可能で，メモリ使用量は逆反復法の r 倍必要である．しかしながら，$r \ll m_c$ である場合，同時逆反復法に比べてメモリ使用量は非常に少なくなる．

また，同時逆反復法における QR 分解の代わりに，(ii) の再直交化計算と (iii) の QR 分解を行う．(ii) は，j ステップ目において $O\left((j-1)r^2 n\right)$ の計算量を要するが，行列乗算によって実装が可能である．(iii) は $n \times r$ 行列の QR 分解であることから，計算量は $O\left(r^2 n\right)$ で，やはり行列乗算を中心としたアルゴリズムで計算可能である．したがって，同時逆反復法における QR 分解や逆反復法における再直交化計算と同程度の計算量を行列乗算による計算で代替することができる．これら 2 段階の再直交化計算はブロック再直交化計算とも呼ばれ，CGS 法や CGS2 法のブロック版アルゴリズムであるブロック CGS（BCGS）法 [21] や BCGS2 法 [3] を適用することができる．

以上の (i)，(ii)，(iii) の計算を反復することで固有ベクトルを計算するのが再直交化付きブロック逆反復法である．このアルゴリズムは，$r = m_c$ のときは同時逆反復法，$r = 1$ のときは逆反復法と等価なアルゴリズムとなる．

Algorithm 3 は，BCGS2 法を適用した再直交化付きブロック逆反復法の擬似コードである．14 行目から 17 行目が BCGS2 法を適用した部分で，14 行目と 16 行目が (ii) に，15 行目と 17 行目が (iii) に対応する計算となる．

3.2 提案アルゴリズムの収束性

同時逆反復法の収束性から，提案アルゴリズムである再直交化付きブロック逆反復法の収束性を示す．

同時逆反復法において，最も収束しにくい悪条件な場合は，T の近接固有値すべてが重複してしまう場合である．したがって，この場合に収束性が保証されるのであれば，本研究で用いた同時逆反復法により得られるベクトル列が固有ベクトルに収束することは明らかである．一方，文献 [5] で定義された同時逆反復法は，行列 T の代数的重複度 l（$< n$）の固有値 σ に対して，$V^{(i)}, Q^{(i)} \in \mathbb{R}^{n \times l}$ とすると以下の 2 式の反復となる．

$$(T - \sigma I)\, V^{(i)} = Q^{(i-1)},$$
$$V^{(i)} := Q^{(i)} R^{(i)}.$$

Algorithm 3 再直交化付きブロック逆反復法（RBI）

function RBI(T, r, $\tilde{\lambda}_1$, …, $\tilde{\lambda}_{m_c}$)
  for $j := 1, \ldots, m_c/r$ **do**
    $i := 0$
    Generate $V_{j,r}^{(0)} := \left[\boldsymbol{v}_{(j-1)r+1}^{(0)} \quad \cdots \quad \boldsymbol{v}_{jr}^{(0)}\right]$
    $V_{j,r}^{(0)} := Q_{j,r}^{(0)} R_{j,r}^{(0)}$ ▷ QR 分解
    for $k := (j-1)r+1, \ldots, jr$ **do**
      $T - \tilde{\lambda}_k I := P_k L_k U_k$ ▷ 部分ピボット選択付き LU 分解
    end for
    repeat
      $i := i + 1$
      for $k = (j-1)r+1, \ldots, jr$ **do**
        Solve $P_k L_k U_k \boldsymbol{v}_k^{(i)} = \boldsymbol{q}_k^{(i-1)}$ ▷ 連立方程式の求解
      end for
      $V_{j,r}^{(i)} := V_{j,r}^{(i)} - Q_{(j-1)r} Q_{(j-1)r}^{\top} V_{j,r}$
      $V_{j,r}^{(i)} := \overline{Q}_{j,r}^{(i)} R_{j,r}^{(i)}$ ▷ QR 分解
      $\overline{Q}_{j,r}^{(i)} := \overline{Q}_{j,r}^{(i)} - Q_{(j-1)r} Q_{(j-1)r}^{\top} \overline{Q}_{j,r}^{(i)}$
      $\overline{Q}_{j,r}^{(i)} := Q_{j,r}^{(i)} R_{j,r}^{(i)}$ ▷ QR 分解
    until converge
    $Q_{jr} = \left[Q_{(j-1)r} \quad Q_{j,r}^{(i)}\right]$ $\left(Q_r = \left[Q_{1,r}^{(i)}\right]\right)$
  end for
  return $Q_{m_c} = \left[\boldsymbol{q}_1 \quad \cdots \quad \boldsymbol{q}_{m_c}\right]$
end function

以上の計算により $Q^{(i)}$ が重複固有値 σ に対応する固有ベクトルが張る空間に収束することは，文献 [5] の Lemma 5.9.2 において証明されている．したがって，T の近接固有値に対する同時逆反復法の収束性は保証される．

提案アルゴリズムでは，以上の演算に加えて，計算済みの固有ベクトルに関する再直交化計算を反復の過程で行っている．この操作は，$V^{(i)}$ に含まれている計算済みの固有ベクトルの成分を抜き取ることにすぎないため，固有ベクトルの収束に悪影響を及ぼすことはない．

以上の議論により，提案アルゴリズムにより得られるベクトル列は固有ベクトルに収束する．

4. マルチコア CPU 向け実装コード

本章では，5 章の性能評価に用いる，逆反復法，同時逆反復法，提案アルゴリズムである再直交化付きブロック逆反復法の実装コードについて説明する．それぞれのコードはマルチコア CPU 向けに実装され，その並列化方法についても説明する．

本研究では，実装の多くに既存の LAPACK ルーチンや BLAS（Basic Linear Algebra Subprograms）を利用できる．これらは，Intel Math Kernel Library（MKL）において並列実装が提供されており，各ルーチンの内部でスレッド並列化できる．ただし，一部の LAPACK ルーチンや，BLAS のベクトル演算にあたるものは並列化されていないが，CPU 向けに高度にチューニングされたルーチンとして提供されている．実装コードにおいて並列化可能なものについては，OpenMP の指示文を使用してスレッド並列化を施した．

4.1 逆反復法

逆反復法の実装コードは，Algorithm 1 の 9 行目の再直交化計算を除き，LAPACK の DSTEIN ルーチンを基に作成した．連立方程式 (1) の求解においては，LAPACK ルーチンを用いることができる．5 行目における部分ピボット選択付き LU 分解は DLAGTF，8 行目の計算では DLAGTS を用いることができる．これらのルーチンは，内部演算の並列化が難しく，Intel MKL においても並列実装は提供されていない．

9 行目の再直交化計算には，MGS 法と CGS2 法を適用した．これらの実装コードをそれぞれ，**Inv-MGS**，**Inv-CGS2** とする．MGS 法の内部演算はすべてベクトル演算であるため，Inv-MGS は並列計算を行わない実装コードとなる．一方，CGS2 法は行列–ベクトル乗算ルーチン DGEMV によって実装できるため，Inv-CGS2 は DGEMV についてスレッド並列計算を行う実装コードとなる．

4.2 同時逆反復法

同時逆反復法は，各固有値クラスタに Algorithm 2 を適用するような実装を施した．このため，同時逆反復法による固有ベクトル計算の前に，固有値を各クラスタに分けるルーチンが必要となる．このルーチンは並列化を施さないが，m 個の固有値に対して $O(m)$ 程度の計算量で済む．

逆反復法と同様に，連立方程式の求解では 5 行目の計算に DLAGTF，11 行目の計算に DLAGTS を使用できる．同時逆反復法では，1 回の反復において連立方程式を m_c 本解くことになるが，これらの連立方程式は独立に計算できるため，OpenMP により 4 行目と 10 行目の for ループについてスレッド並列化した．

3 行目と 13 行目の QR 分解には，再直交化計算同様，Householder 変換に基づくアルゴリズムと CGS 法に基づくアルゴリズム両方を適用することができる．以下では，同時逆反復法の QR 分解に適用した実装について述べる．

4.2.1 Householder 変換に基づく QR 分解

Householder 変換による QR 分解はよく知られている方法で，compact WY 表現を用いることで，行列乗算中心の実装が可能である．特に文献 [10] で提案された再帰的実装は，並列化効率の高い演算が可能であることが知られており，LAPACK においても DGEQRT3 ルーチンとして実装されている．DGEQRT3 により計算された直交行列は compact WY 表現に分解された要素のままでしか得られないため，これら要素から直交行列を復元する必要がある．この復元計算は非常に容易で，行列乗算を用いることで達成できる．

このように，DGEQRT3 ルーチンで compact WY 表現を作り，直交行列を行列乗算で復元する操作による QR 分

Algorithm 4 BCGS algorithm

1: **function** BCGS($V_{1,r},\ \ldots,\ V_{k/r,r}$)
2: $Q_0 = \boldsymbol{O}$
3: **for** $j = 1,\ \ldots,\ k/r$ **do**
4: $V_{j,r} := V_{j,r} - Q_{(j-1)r} Q_{(j-1)r}^{\top} V_{j,r}$
5: $V_{j,r} := Q_{j,r} R_{j,r}$ ▷ QR 分解
6: $Q_{jr} = \begin{bmatrix} Q_{(j-1)r} & Q_{j,r} \end{bmatrix}$ $(Q_r = [Q_{1,r}])$
7: **end for**
8: **return** $Q_k = \begin{bmatrix} \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_k \end{bmatrix}$
9: **end function**

解を HQR 法と呼ぶ．HQR 法の計算は，DGEQRT3 においても，直交行列の復元計算においても，行列乗算ルーチンの DGEMM および DTRMM を中心とした BLAS により実装できる．したがって，HQR 法は Intel MKL を利用することで内部の BLAS について並列化できる．

以下では，HQR 法を QR 分解に実装した同時逆反復法のコードを **SI-HQR** とする．

4.2.2 CGS2 法の再帰的実装に基づく QR 分解

再直交化計算における CGS 法は，行列–ベクトル乗算による実装であった．しかし，QR 分解に CGS 法を適用する場合，計算順序を入れ替えることで，行列–ベクトル乗算で表現されていた計算を行列乗算に置き換えることができ，行列乗算の割合を高めることができる．この考えに則り，横澤らは文献 [24] において再帰的分割法に基づく CGS 法の実装を提案している．本研究では，この再帰的実装の CGS 法を，rCGS 法とする．rCGS 法は行列乗算ルーチン DGEMM を中心とした BLAS を用いて実装できるため，Intel MKL を利用することで BLAS 内でのスレッド並列化がなされる．

同時逆反復法の実装には，rCGS 法を 2 回繰り返した rCGS2 法による QR 分解を適用し，その実装コードを **SI-rCGS2** とする．ここで，rCGS2 法と HQR 法の計算量は同程度であるため，SI-HQR と SI-rCGS2 の計算量もまた同程度である．

4.2.3 BCGS2 法による QR 分解

文献 [21] では，QR 分解における CGS 法の行列乗算の割合を増やす方法として，BCGS 法が提案されている．BCGS 法は CGS 法にブロックサイズパラメータ r を導入したものである．Algorithm 4 は，$n \times k$ 行列の QR 分解を行う場合の BCGS 法を表す擬似コードである．4 行目は，すでに計算済みのベクトルと直交するように，ブロック内部のベクトルの計算を行っており，行列乗算を用いることができる．5 行目は，ブロック内部での QR 分解を行っており，rCGS 法あるいは HQR 法を用いることで，大半の計算を行列乗算による演算で実装することが可能である．

CGS 法と同様に，BCGS 法も計算対象の行列によっては，ベクトルの直交性が悪化してしまう場合がある．このため，BCGS 法を 2 回繰り返す BCGS2 法 [3] が提案されている．BCGS2 法も，行列乗算ルーチン DGEMM を用いた実装ができるため，Intel MKL を利用することで演算内のスレッド並列化が可能になる．

BCGS2 法により QR 分解を行う同時逆反復法の実装コードのうち，内部の QR 分解に HQR 法を実装したものを **SI-BCGS2(H)**，rCGS 法を実装したものを **SI-BCGS2(G)** とする．ここで，HQR 法の計算量は rCGS 法に比べて多くなるため，SI-BCGS2(H) は SI-BCGS2(G) に比べて多くの計算量を要する．また，rCGS 法を QR 分解に実装した BCGS2 法と rCGS2 法は同程度の計算量である．

4.3 再直交化付きブロック逆反復法

再直交化付きブロック逆反復法についても，同時逆反復法と同様の実装を施す．4.2 節で述べたように，QR 分解の方法には Householder 変換に基づくものと CGS 法の再帰的実装に基づくものの 2 通りがある．したがって，再直交化付きブロック逆反復法についても 2 通りの実装を施す．以下では，Householder 変換に基づく QR 分解を利用した実装コードを **RBI-BCGS2(H)**，CGS 法の再帰的実装に基づく QR 分解を利用した実装コードを **RBI-BCGS2(G)** とする．

両コード共通の計算についての実装について説明する．まず，固有ベクトル計算の前に，固有値を各クラスタに分けるルーチンを加えた．次に，Algorithm 3 の 7 行目の計算に DLAGTF，12 行目の計算に DLAGTS，それぞれの LAPACK ルーチンが適用できる．再直交化付きブロック逆反復法では，1 回の反復で r 本の連立方程式を解くため，OpenMP によって 6 行目と 11 行目の for 文をスレッド並列化した．以上の実装は，同時逆反復法にも共通する計算であるが，再直交化付きブロック逆反復法独自の実装として，14 行目と 16 行目に，それぞれ，行列乗算ルーチン DGEMM を 2 回用いた．DGEMM は，Intel MKL を利用することで，内部演算においてスレッド並列化できる．

最後に，RBI-BCGS2(H) と RBI-BCGS2(G) それぞれの QR 分解の実装について説明する．ここで，5 行目は $n \times r$ 長方行列の QR 分解であるため，BCGS2 法ではなく，再帰的実装に基づく QR 分解を利用する．RBI-BCGS2(H) では，5 行目，15 行目，17 行目すべての QR 分解の実装について，HQR 法を実装した．RBI-BCGS2(G) では，5 行目には rCGS2 法，15 行目および 17 行目の QR 分解には rCGS 法を実装した．HQR 法は rCGS 法よりも計算量が多いため，RBI-BCGS2(H) の方が RBI-BCGS2(G) よりも計算量が多くなってしまう．これら QR 分解の計算部分については，Intel MKL を利用することで，演算内でのスレッド並列化することができる．

表 1 は本章で述べた，逆反復法，同時逆反復法および再直交化付きブロック逆反復法の実装コードについて，実装したルーチンやアルゴリズム，それらの並列化方法をまとめたものである．

表 1　各コードにおいて使用した主なルーチンとその並列化方法
Table 1　Main routines and parallelism in each code.

	Inv-MGS	Inv-CGS2	SI-HQR	SI-rCGS2	SI-BCGS2(H)	SI-BCGS2(G)	RBI-BCGS2(H)	RBI-BCGS2(G)
初期行列の QR 分解			HQR 法	rCGS2 法	BCGS2 法	BCGS2 法	HQR 法	rCGS2 法
連立方程式の求解	DLAGTF	DLAGTF	DLAGTF	DLAGTF	DLAGTF	DLAGTF	DLAGTF	DLAGTF
	DLAGTS	DLAGTS	DLAGTS	DLAGTS	DLAGTS	DLAGTS	DLAGTS	DLAGTS
再直交化計算	MGS 法	CGS2 法						
QR 分解			HQR 法	rCGS2 法	BCGS2 法	BCGS2 法		
ブロック再直交化							BCGS2 法	BCGS2 法
BCGS2 法内部					HQR 法	rCGS 法	HQR 法	rCGS 法

黒字：並列化なし
青字：OpenMP による for ループのスレッド並列化
赤字：Intel Math Kernel Library による，演算内部の各 BLAS におけるスレッド並列化

5. 数値実験

数値実験では，**表 2** に示される共有メモリマルチコアプロセッサシステム 1 ノードを使用し，スレッド並列計算を行った．実対称 3 重対角行列 T_1 および T_2 の全固有ベクトルを計算することにより，提案アルゴリズムの性能を評価する．評価においては，4 章において説明した実装コードである，Inv-MGS，Inv-CGS2，SI-HQR，SI-rCGS2，SI-BCGS2(H)，SI-BCGS2(G)，RBI-BCGS2(H)，RBI-BCGS2(G) を用いる．ここで，Inv-MGS は LAPACK の DSTEIN ルーチンそのものであるが，本実験においては，Intel Fortran Compiler で改めてコンパイルしたものを使用した．

性能評価のテスト行列には，固有値分布の異なる 2 つの実対称 3 重対角行列 T_1，T_2 を用いた．テスト行列 T_1 は Glued-Wilkinson 行列 [6], [8] 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 $n/21$ となるクラスタと $2n/21$ となるクラスタがそれぞれ 7 つ，計 14 のクラスタに分かれることが知られている．さらに，それぞれのクラスタに属する固有値が非常に密集しており，条件数の悪い問題として知られている．テスト行列 T_2 は，全要素を $(0, 1)$ の範囲の一様乱数により生成した実対称 3 重対角行列を用いた．この行列の固有値は，小次元行列ではある程度の数のクラスタに分かれる一方，大次元行列ではほとんどが 1 つのクラスタに含まれる．数値実験において実際に用いた行列でも，n が 10000 以上の場合においては，9 割以上の固有値が 1 つのクラスタをなす行列となった．

また，各テスト行列の固有値は，Intel MKL に実装された LAPACK の 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二分法によって計算する倍精度演算ルーチンである．

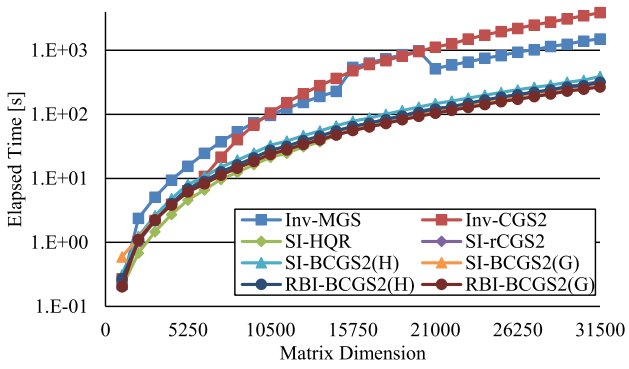
最後に，各コードにおいて，許容する反復回数は 5 回である．しかし，すべての実験において，いかなる入力行列の場合でも 3 回の反復回数で収束することが確認できている．

表 2　実験環境（Appro Green Blade 8000）
Table 2　Specifications of Appro Green Blade 8000.

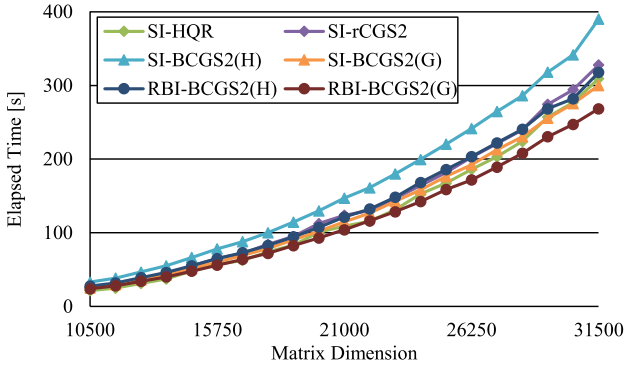
CPU	Intel Xeon E5-2670 (2.6 GHz, 8 cores × 2)
RAM	DDR3-1600 64 GB
Compiler	Intel Fortran Compiler 13.1.3
Options	-O3 -xHOST -ipo -no-prec-div -static -openmp
Libraries	Intel Math Kernel Library 11.0 update 1

5.1 各コードの性能比較

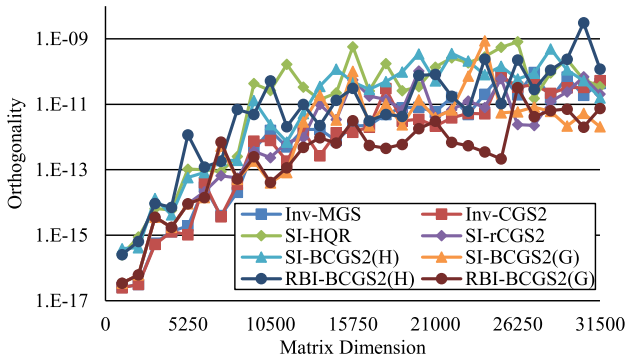
図 1 および**図 2** は，それぞれテスト行列 T_1 および T_2 に対する 16 スレッド並列計算での数値実験を行った結果を示すものである．この比較においては，SI-BCGS2(G) および RBI-BCGS2(G) のブロックサイズパラメータは $r = 256$ とした．図 1 (a) と図 2 (a) では，全固有ベクトル計算に要した計算時間の比較を行っている．これらの図から，特に大次元の行列に対して，行列乗算を中心とした固有ベクトル計算法である同時逆反復法および再直交化付きブロック逆反復法の実装コードが，逆反復法の実装コードである Inv-MGS，Inv-CGS2 よりも高速な並列計算を実現していることが分かる．また，図 1 (b) および図 2 (b) では，図 1 (a) と図 2 (a) で示した結果のうち，SI-HQR，SI-rCGS2，SI-BCGS2(H)，SI-BCGS2(G)，RBI-BCGS2(H)，RBI-BCGS2(G) の計算時間のみを比較したものである．これらの図から，特に大次元の行列に対しては，CGS 法を QR 分解に実装した再直交化付きブロック逆反復法コード RBI-BCGS2(G) が最も高速であることが分かる．実際 RBI-BCGS2(G) は，$n = 31500$ の T_1 の全固有ベクトル計算について，Inv-MGS に対して約 6 倍，Inv-CGS2 に対して約 15 倍，$n = 30000$ の T_2 の全固有ベクトル計算でも，Inv-MGS に対して約 12 倍，Inv-CGS2 に対して約 31 倍高速であった．また，RBI-BCGS(G) は，同時逆反復法の実装コード SI-rCGS2 や SI-BCGS2(G) に比べて，T_1 の全固有ベクトル計算では 10%，T_2 の全固有ベクトル計算で



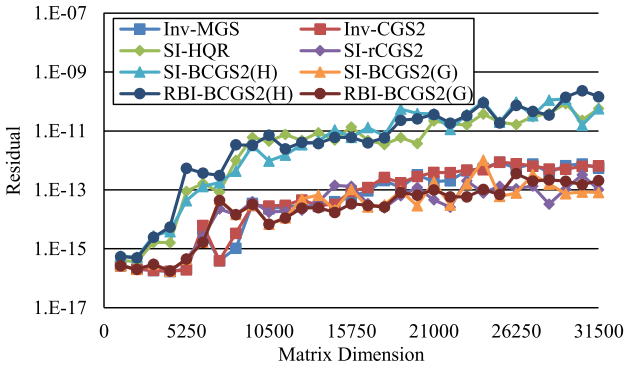
(a) Elapsed time of all code



(b) Elapsed time of SI-rCGS2, SI-BCGS2(G), and RBI-BCGS2(G)



(c) Orthogonality $\|QQ^\top - I\|_\infty/n$



(d) Residual $\|TQ - QD\|_\infty/n$

図 1　固有ベクトル計算における逆反復法，同時逆反復法，再直交化付きブロック逆反復法の比較（$r = 256$，テスト行列 T_1）

Fig. 1　Comparisons of Inv, SI, and RBI code in computing eigenvectors of T_1. Note that $r = 256$.

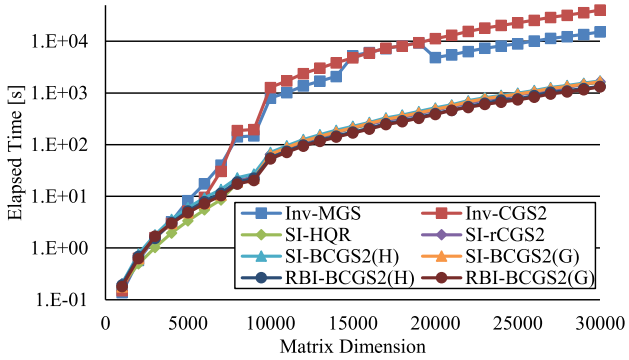
は 20%近い性能向上が確認できている．同じく CGS 法を QR 分解に実装した同時逆反復法コード SI-rCGS2 および SI-BCGS2(G) と比べると，RBI-BCGS2(G) で使用されるメモリ空間は少ないため，より高い演算性能が得られたと考えられる．Householder 変換を CGS 法を QR 分解に実装した SI-HQR，SI-BCGS2(H)，RBI-BCGS2(H) は，それぞれ，SI-rCGS2，SI-BCGS2(G)，RBI-BCGS2(G) と比較すると，計算量の大小に応じた計算時間となっており，CGS 法による QR 分解を実装したコードの方が総じて高速であるという結果が得られた．

図 1 (c) および図 2 (c) では各コードによって得られた固有ベクトルの直交性 $\|QQ^\top - I\|_\infty/n$，図 1 (d) および図 2 (d) では各コードによって得られた固有ベクトルの残差 $\|TQ - QD\|_\infty/n$ を表している．ここで，D は対角要素に固有値が並ぶ対角行列である．これら固有ベクトルの計算精度を表すグラフから，同じ反復回数の固有ベクトル計算において，Householder 変換に基づく QR 分解を用いた実装コードである SI-HQR，SI-BCGS2(H)，RBI-BCGS2(H) は，他のコードに比べて顕著に精度が劣るという結果が得られた．その一方で，SI-rCGS2，SI-BCGS2(G)，RBI-BCGS2(G) は，従来アルゴリズムである Inv-MGS や Inv-CGS2 と同等の計算精度を得られることが分かる．しかしながら，T_1 の直交性を表すグラフにおいて，SI-rCGS2(G) の $n = 19550$，SI-BCGS2(G) の $n = 24150$ など，ところどころ精度の劣化が見られる以上，同時逆反復法の実装コードは CGS 法に基づく QR 分解を用いた場合でも再直交化付きブロック逆反復法よりも精度面での信頼性に欠ける．

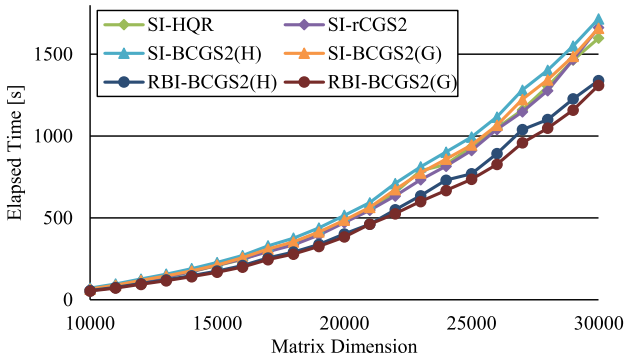
以下では，同時逆反復法および再直交化付きブロック逆反復法の実装コードについてさらなる性能評価を行う．特に，CGS 法に基づいた QR 分解を利用した実装コードの方が速度と精度両面において良い結果が得られていることから，SI-rCGS2，SI-BCGS2(G)，RBI-BCGS2(G) について性能評価を行う．

5.2　強スケーリングの評価

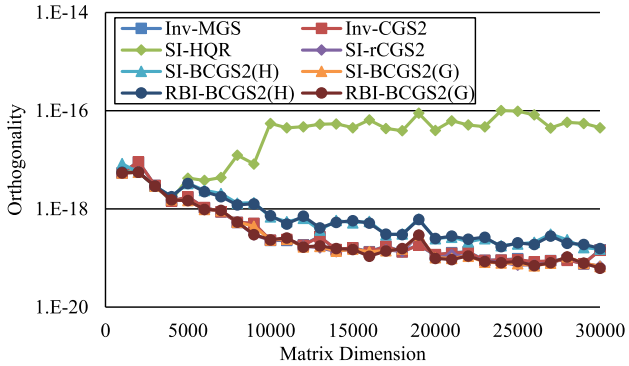
図 3 および**図 4** は，行列 T_1 および T_2 の問題サイズを固定したときの，SI-rCGS2，SI-BCGS2(G)，RBI-BCGS2(G) の並列化効率を評価した結果である．すべてのグラフにおいて，各手法を 1 スレッドで計算したときに要した計算時間を基準に，スレッド数を変化させたときどの程度の高速化が見られるかを示している．この比較においても，SI-BCGS2(G) および RBI-BCGS2(G) のブロックサイズパ



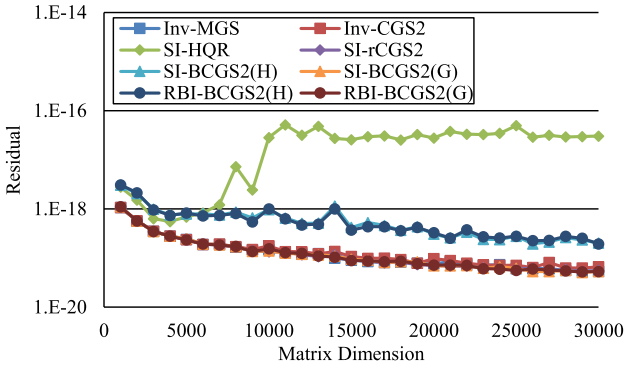
(a) Elapsed time of all code



(b) Elapsed time of SI-rCGS2, SI-BCGS2(G), and RBI-BCGS2(G)



(c) Orthogonality $\|QQ^\top - I\|_\infty/n$



(d) Residual $\|TQ - QD\|_\infty/n$

図 2　固有ベクトル計算における逆反復法，同時逆反復法，再直交化付きブロック逆反復法の比較（$r = 256$，テスト行列 T_2）

Fig. 2　Comparisons of Inv, SI, and RBI code in computing eigenvectors of T_2. Note that $r = 256$.

ラメータは $r = 256$ とした．

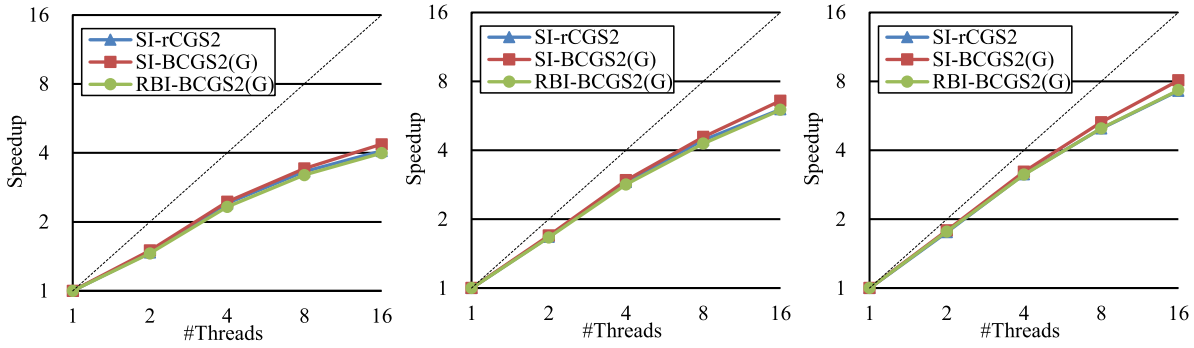
16 スレッドを使用することで，RBI-BCGS2(G) は，テスト行列 T_1 の $n = 10500$ において約 4 倍，$n = 21000$ において約 6 倍，$n = 31500$ において約 7 倍の並列化効率を達成した．また，テスト行列 T_2 の $n = 10000$ において約 11 倍，$n = 20000$ において約 13 倍，$n = 30000$ において約 14 倍の並列化効率を達成し，T_1 に対する結果よりも良い結果が得られた．これは，固有値が大規模なクラスタをなすテスト行列 T_2 の方が，全体の計算量に対する再直交化計算や QR 分解を行う割合が大きくなるからであると考えられる．これらの計算は行列乗算中心のキャッシュヒット率の高い計算であり，このことによる演算性能の向上が起因して，T_2 に対して高い並列化効率が得られたと考えられる．また，どちらのテスト行列に対しても，問題サイズが小さいときよりも大きいときの方が高い並列化効率を得られた．また，RBI-BCGS2(G) は SI-rCGS2 と比較するとほとんど同程度の並列化効率を達成しているが，SI-BCGS2(G) には少し劣るという結果が得られた．

5.3　異なるブロックサイズパラメータにおける性能

図 5 は，行列 T_1 および T_2 の問題サイズを固定し，異なるブロックサイズ r を与えた場合の SI-BCGS2(G)，RBI-BCGS2(G) の計算時間を比較している．強スケーリングによる性能評価と同様に，問題サイズには，行列 T_1 に対しては 10500，21000，31500，行列 T_2 に対しては 10000，20000，30000 を選んだ．

図 5 からは，SI-BCGS2(G) と RBI-BCGS2(G) の両方について，同様の傾向が観察される．まず，行列の種類やサイズにかかわらず，ある程度の大きさまでは，ブロックサイズ r に反比例するように計算時間が削減される．これは，r の値を大きくすることにより，QR 分解や再直交化計算における行列乗算の割合が増えるからである．その一方で，どの行列サイズにおいても $r = 2048$ のときに最も計算時間が短くなるわけではなく，r を大きくとれば必ず高速になるというわけではない．

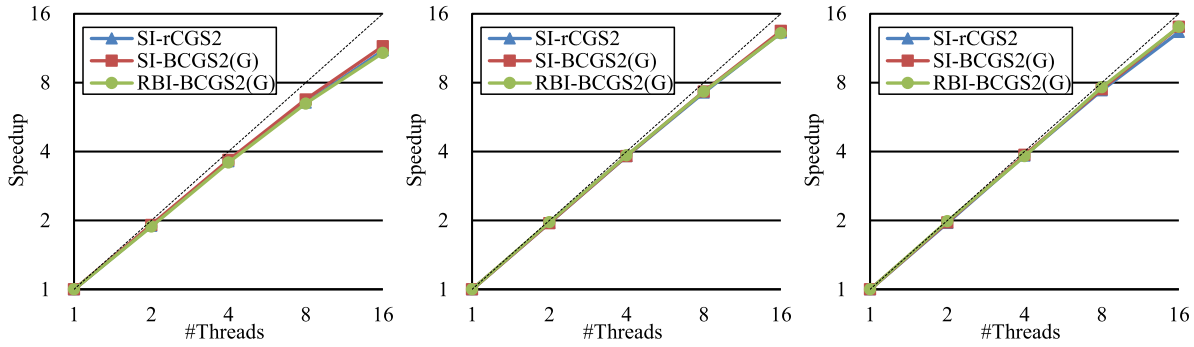
以上の性能評価により，ブロックサイズ r をうまく設定することで計算時間の削減ができることは明らかである．しかし，r の値が大きい場合の計算時間の変化幅は徐々に小さくなっているため，他の性能評価において設定した $r = 256$ のような値であれば，最適に近い性能での並列計算ができていると見なせる．



(a) n = 10500　(b) n = 21000　(c) n = 31500

図 3　T_1 に対する SI-rCGS2，SI-BCGS2(G)，RBI-BCGS2(G) の強スケーリング評価（$r = 256$）

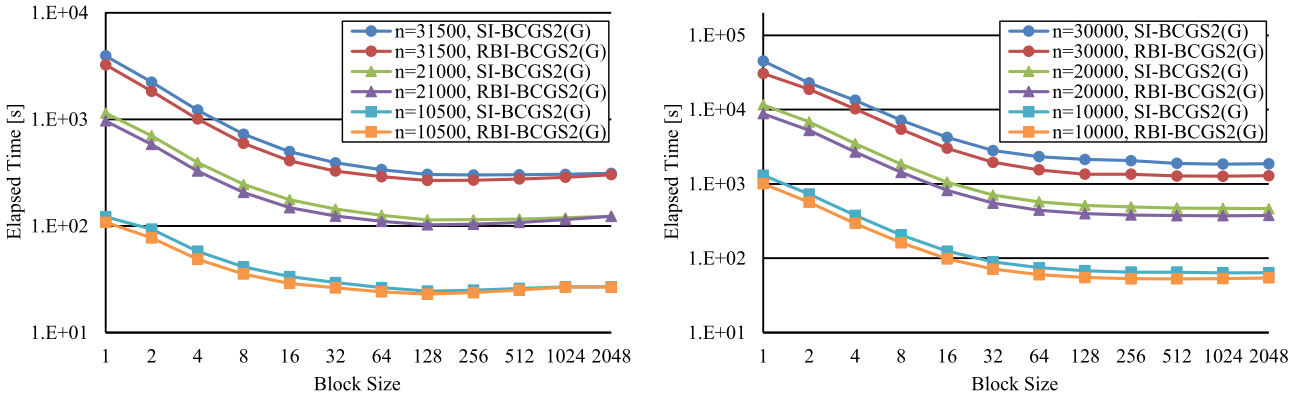
Fig. 3　Strong scalability of SI-rCGS2, SI-BCGS2(G), and RBI-BCGS2(G) for T_1 and $r = 256$.



(a) n = 10000　(b) n = 20000　(c) n = 30000

図 4　T_2 に対する SI-rCGS2，SI-BCGS2(G)，RBI-BCGS2(G) の強スケーリング評価（$r = 256$）

Fig. 4　Strong scalability of SI-rCGS2, SI-BCGS2(G), and RBI-BCGS2(G) for T_2 and $r = 256$.



(a) Case of T_1　(b) Case of T_2

図 5　異なるブロックサイズ r に対する SI-BCGS2(G) および RBI-BCGS2(G) の計算時間

Fig. 5　Elapsed time of SI-BCGS2(G) and RBI-BCGS2(G) for each block size parameter r.

6. まとめと今後の課題

本論文では，並列計算機向けの実対称 3 重対角行列の固有ベクトル計算アルゴリズムとして再直交化付きブロック逆反復法を提案した．提案アルゴリズムは，行列乗算を中心とした固有ベクトル計算法である同時逆反復法にブロックパラメータを導入したもので，大粒度の並列性を持つ．共有メモリマルチコアプロセッサシステム上での性能評価

では，内部の QR 分解に再帰的実装に基づく CGS 法を実装した提案アルゴリズムが，速度と精度両面において優れた並列計算を達成することを示した．また，異なるブロックサイズパラメータの値を選択した場合の性能評価では，ある程度のサイズまで大きくすることで最適に近い性能が得られることを示した．

今後の課題としては，1 章において述べた，MRRR 法と提案アルゴリズムを組み合わせた新しいアルゴリズムの実装が考えられる．このアルゴリズムにより，MRRR 法で計算が破綻してしまうような行列に対しても，高速高精度な固有ベクトル計算が可能になると期待できる．また，より大規模な問題への適用のため，提案アルゴリズムの分散並列環境での実装および性能評価も重要な課題である．

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] See the past records in LAPACK BUG LIST Homepage, available from ⟨http://www.netlib.org/lapack/bug_list.html⟩.
[2] 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).
[3] Barlow, J.L. and Smoktunowicz, A.: Reorthogonalized block classical Gram-Schmidt, *Numer. Math.*, Vol.123, No.3, pp.1–29 (2012).
[4] Bischof, C.H., Marques, M. and Sun, X.: Parallel bandreduction and tridiagonalization, *Proc. 6th SIAM Conference on Parallel Processing for Scientific Computing*, pp.22–24 (1993).
[5] Chatelin, F.C. and Ahués, M.: *Eigenvalues of Matrices*, SIAM, Philadelphia, PA, USA (2012).
[6] 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).
[7] Dhillon, I.S.: A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem, Ph.D. Thesis, EECS Department, University of California, Berkeley (1997).
[8] 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).
[9] Dhillon, I.S., Parlett, B.N. and Vömel, C.: The design and implementation of the MRRR algorithm, *ACM Trans. Math. Softw.*, Vol.32, No.4, pp.533–560 (2006).
[10] Elmroth, E. and Gustavson, F.G.: Applying recursion to serial and parallel QR factorization leads to better performance, *IBM J. Res. Dev.*, Vol.44, No.4, pp.605–624 (2000).
[11] 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).
[12] Imamura, T., Yamada, S. and Machida, M.: Development of a high performance eigensolver on the peta-scale next generation supercomputer system, *Prog. Nuclear Science and Technology*, Vol.2, pp.643–650 (2011).
[13] Ipsen, I.C.F.: Computing an Eigenvector with Inverse Iteration, *SIAM Review*, Vol.39, No.2, pp.254–291 (1997).
[14] Ishigami, H., Kimura, K. and Nakamura, Y.: On implementation and evaluation of inverse iteration algorithm with compact WY orthogonalization, *IPSJ Trans. Mathematical Modeling and Its Applications*, Vol.6, No.2, pp.25–35 (2013).
[15] Katagiri, T. and Itoh, S.: A massively parallel dense symmetric eigensolver with communication splitting multicasting algorithm, *High Performance Computing for Computational Science – VECPAR 2010*, Lecture Notes in Computer Science, Vol.6449, pp.139–150, Springer Berlin Heidelberg (2011).
[16] 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, pp.302–314, Springer Berlin Heidelberg (2003).
[17] Parlett, B.N.: *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, PA, USA (1998).
[18] Peters, G. and Wilkinson, J.: The calculation of specified eigenvectors by inverse iteration, *Handbook for Automatic Computation*, pp.418–439, Springer-Verlag, Berlin (1971).
[19] Petschow, M. and Bientinesi, P.: MR^3-SMP: A symmetric tridiagonal eigensolver for multi-core architectures, *Parallel Computing*, Vol.37, No.12 (2011).
[20] 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).
[21] Stewart, G.: Block Gram-Schmidt orthogonalization, *SIAM Journal on Scientific Computing*, Vol.31, No.1, pp.761–775 (2008).
[22] Wu, Y.-J.J., Alpatov, P.A., Bischof, C.H. and van de Geijn, R.A.: A parallel implementation of symmetric band reduction using PLAPACK, *Proc. Scalable Parallel Libraries Conference, Mississippi State University* (1996).
[23] 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).
[24] 横澤拓弥，高橋大介，朴　泰祐，佐藤三久：行列積を用いた古典 Gram-Schmidt 直交化法の並列化，情報処理学会論文誌　コンピューティングシステム（ACS），Vol.1, No.1, pp.61–72 (2008).

石上 裕之　（学生会員）

1988 年生．2011 年京都大学工学部情報学科卒業．2013 年同大学大学院情報学研究科数理工学専攻修士課程修了．現在，同博士後期課程在学中．2013 年 4 月より日本学術振興会特別研究員（DC1）．並列計算機向け特異値・固有値分解アルゴリズムの研究開発に従事．日本応用数理学会学生会員．

木村 欣司　（正会員）

1976 年生．2004 年神戸大学大学院自然科学研究科情報メディア科学専攻博士課程修了．博士（理学）．2006 年京都大学大学院情報学研究科特定有期雇用助手．2007 年新潟大学大学院自然科学研究科助教．2008 年京都大学大学院情報学研究科特定講師．2009 年京都大学大学院情報学研究科特定准教授．離散可積分系，計算機代数，数値解析に関する研究に従事．日本応用数理学会，日本数理式処理学会各会員．

中村 佳正　（正会員）

1955 年生．1983 年京都大学大学院工学研究科博士課程修了．工学博士．岐阜大学助教授，同志社大学教授，大阪大学大学院基礎工学研究科教授等を経て，2001 年より京都大学大学院情報学研究科数理工学専攻教授．専門は応用数学，とりわけ，応用可積分系や計算数学，高速高精度の特異値分解法 I-SVD を開発している．主著に「可積分系の機能数理」（共立出版，2006 年）がある．日本学術会議連携会員，日本応用数理学会，日本数学会，SIAM，AMS 各会員．