キャッシュマシン向け対称密行列固有値解法の性能・精度評価

山 本 有 作†

キャッシュマシン上での固有値計算では，Dongarra らの提案した多段化ハウスホルダ法による三重対角化アルゴリズムが広く使われている．しかし，この方法では全演算の半分がデータ再利用性の低い行列ベクトル積となり，通常はピークの 10〜25%程度の性能しか達成できない．そこで本論文では，演算のほとんどを行列乗算で行える Bischof らおよび Wu らのアルゴリズムについて，性能と精度の詳しい評価を行った．Pentium 4 Xeon，Opteron，Alpha 21264A，Ultra SPARC III の 4 種のプロセッサで性能を評価したところ，3840 元の行列の三重対角化を行う場合に Dongarra らのアルゴリズムの約 2 倍，ピークの最大 50%程度の性能を達成できた．また，4 種のテスト行列で固有値の精度を評価したところ，Dongarra らのアルゴリズムと比べて固有値の相対誤差はやや大きくなるが，増大の程度は多くの場合 2〜3 倍以内であり，最大でも 1 桁程度に抑えられることを明らかにした．

Performance and Accuracy of Algorithms for Computing the Eigenvalues of Real Symmetric Matrices on Cache-based Microprocessors

Yusaku Yamamoto†

Dongarra's tri-diagonalization algorithm has been widely used to compute the eigenvalues of real symmetric matrices on cache-based microprocessors. However, it is known that this algorithm can attain only 10 to 25% of the peak performance because half of the total arithmetic operations are done in the form of matrix-vector multiplication, which has low rate of data reuse. In this paper, we evaluate the performance and accuracy of Bischof's and Wu's algorithms, which can perform most of the operations as matrix-matrix multiplication. Experiments on the Pentium 4 Xeon, Opteron, Alpha 21264A and Ultra SPARC III microprocessors show that these algorithms can attain twice the performance of Dongarra's algorithm when tri-diagonalizing a matrix of order 3840, achieving 50% of the peak performance. The accuracy of computed eigenvalues was found to be slightly lower, but the maximum relative errors lie within 2 to 3 times that of Dongarra's algorithm in most cases, and within 10 times even in the worst case.

1. はじめに

実対称密行列の固有値計算は広い分野で使われる基本的な線形計算の 1 つであり，分子計算，統計計算などに幅広い応用を持つ[15),17),20)]．近年では，シミュレーションの大規模化・精密化により解くべき行列のサイズが大型化するとともに，PC やワークステーション上で計算を行うケースが増えつつあり，これらのマシン上で大規模な固有値計算を効率的に実行できるソルバの開発が重要な課題となっている．

PC やワークステーションで使われているマイクロプロセッサは Intel Pentium 4，AMD Opteron，DEC Alpha など多種にわたるが，これらはいずれも数百 KB〜数 MB のキャッシュを持ち，データの再利用性を高めてキャッシュを有効利用することがプログラムの性能向上のために決定的に重要である．

実対称密行列の固有値を計算する場合，最もよく使われるアルゴリズムは，入力行列 $\mathbf{A}$ をハウスホルダ法により実対称三重対角行列 $\mathbf{T}$ に変換し，$\mathbf{T}$ の固有値を二分法あるいは QR 法により求める方法である[11),15),17)]．これらの処理のうち，二分法あるいは QR 法は演算量が $O(N^2)$（N は行列のサイズ）と小さいうえ，計算対象のデータが三重対角行列のため，行列サイズが大きくても全データがキャッシュに収まりやすく，データ再利用性は比較的高い．一方，ハウスホルダ法による三重対角化の部分は，演算量が約 $\frac{4}{3}N^3$ と大きく，かつオリジナルの形ではデータ再利

† 名古屋大学大学院工学研究科計算理工学専攻
Department of Computational Science and Engineering, Nagoya University

用性が低いことが知られている[8),9)]．

そのため，ハウスホルダ法をキャッシュマシン向けに最適化したアルゴリズムとして，Dongarra らのアルゴリズム[8),9)]，Bischof らのアルゴリズム[4),5)]，この 2 つを組み合わせた Wu らのアルゴリズム[10),18)] などが提案されている．このうち，Dongarra らのアルゴリズムはハウスホルダ変換における行列の rank-2 更新と呼ばれる処理を複数ステップ分まとめ，データ再利用性の高い行列乗算を用いて計算する方法である．このアルゴリズムは LAPACK[1)]，ScaLAPACK[6)] などに実装されて広く利用され，計算精度・安定性は十分に確認されている．しかし，このアルゴリズムでは行列乗算の形で計算できるのは全演算量の半分のみであり，残りの半分はデータ再利用性の低い行列ベクトル積の形となるため，多くのキャッシュマシンではピーク性能の 10〜25%程度しか達成できないことが知られている[9),16)]．

一方，Bischof らのアルゴリズムは行列 $\mathbf{A}$ をブロックハウスホルダ変換と呼ばれる方法によりいったん帯行列 $\mathbf{B}$ に変換し，$\mathbf{B}$ をさらに三重対角行列に変換する方法である．この方法では $O(N^3)$ の演算量を持つ部分すべてを行列乗算で行えるため，原理的には Dongarra らのアルゴリズムより高い性能を実現することが可能である．しかし，本アルゴリズムは Dongarra らのアルゴリズムに比べてあまり知られておらず，多様な計算機環境・例題を用いて体系的に評価を行った結果はまだ報告されていない．そのため，実用性を保証するには，より多くのマシンでの性能評価，およびより多くの例題での精度評価が必要である．

そこで本論文では，Dongarra らのアルゴリズム，Bischof らのアルゴリズム，Wu らのアルゴリズムの 3 つに対し，Pentium 4，Opteron，Alpha 21264，Ultra SPARC III などの最近のマイクロプロセッサ上での性能比較を行う．また，乱数行列，構造解析より得られる行列，解析的に固有値の求まる行列などを用いて，得られる固有値の精度評価を行う．これにより，Bischof らのアルゴリズムとそれを改良した Wu らのアルゴリズムの実用性を評価することを目的とする．

以下では，まず 2 章で基本的なハウスホルダ法のアルゴリズムを示し，その変形として Dongarra らのアルゴリズム，Bischof らのアルゴリズム，Wu らのアルゴリズムを説明する．次に 3 章で，上記の各マイクロプロセッサ上で 3 つのアルゴリズムの性能を比較するとともに，4 種のテスト例題を用いて 3 つのアルゴリズムの精度を評価する．4 章では固有ベクトルの計算が必要な場合について，各アルゴリズムのメリット/デメリットを簡単に考察する．最後に 5 章でまとめを述べる．

2. 従来の固有値計算法とその問題点

2.1 固有値計算の処理フロー

実対称行列（またはエルミート行列）$\mathbf{A}$ に対して，

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{1}$$

を満たす λ を固有値，ベクトル $\mathbf{x}$ を固有ベクトルと呼ぶ．$\mathbf{A}$ を $N \times N$ 行列とするとき，固有値は重複度も含めてちょうど N 個存在する．固有値の計算は，分子計算，統計計算，構造解析などの分野で幅広い応用を持つ基本的な線形計算である．

$\mathbf{A}$ が密行列の場合，固有値を計算するには，まず $\mathbf{A}$ をハウスホルダ法と呼ばれるアルゴリズムにより三重対角行列 $\mathbf{T}$ に相似変換し，次に二分法あるいは QR 法により $\mathbf{T}$ の固有値 $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ を求める．行列 $\mathbf{A}$ のサイズを N とするとき，ハウスホルダ法による三重対角化の演算量は約 $\frac{4}{3}N^3$，二分法あるいは QR 法の演算量は $O(N^2)$ であり，大規模問題では前者が計算時間のほとんどを占める．そこで，本論文では三重対角化部分のみを対象とする．

2.2 基本的なハウスホルダ法

ハウスホルダ法による三重対角化のアルゴリズムをアルゴリズム 1 に示す[11),17)]．なお，以下で英大文字の太文字は行列，英小文字の太文字はベクトル，英大文字，英小文字およびギリシャ文字の細文字はスカラを表す．また，ベクトル $\mathbf{a}$ に対して，a_i は $\mathbf{a}$ の第 i 要素を示す．

［アルゴリズム 1：ハウスホルダ法］
do $k = 1, N-2$
［鏡像変換ベクトル $\mathbf{u}^{(k)}$ の作成］
$\sigma^{(k)} = \sqrt{\mathbf{d}^{(k)t}\mathbf{d}^{(k)}}$
$\mathbf{u}^{(k)} = (d_1^{(k)} - \mathrm{sgn}(d_1^{(k)})\sigma^{(k)}, d_2^{(k)}, \ldots, d_{N-k}^{(k)})$
$\alpha^{(k)} = 2/\parallel \mathbf{u}^{(k)} \parallel_2$
［行列ベクトル積］
$\mathbf{p}^{(k)} = \alpha^{(k)}\mathbf{C}^{(k)}\mathbf{u}^{(k)}$
$\beta^{(k)} = \alpha^{(k)}\mathbf{u}^{(k)t}\mathbf{p}^{(k)}/2$
$\mathbf{q}^{(k)} = \mathbf{p}^{(k)} - \beta^{(k)}\mathbf{u}^{(k)}$
［行列の rank-2 更新］
$\mathbf{C}^{(k)} := \mathbf{C}^{(k)} - \mathbf{u}^{(k)}\mathbf{q}^{(k)t} - \mathbf{q}^{(k)}\mathbf{u}^{(k)t}$
end do

計算は $k = 1$ から $k = N-2$ までの $N-2$ 段からなる．**図 1** に示すとおり，第 k 段（$1 \le k \le N-2$）において，全体行列の第 k 列目の第 $k+1$ 行目以降

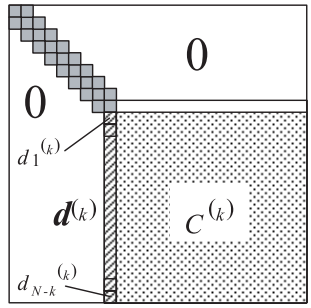

図 1 ハウスホルダ法の第 k 段における行列
Fig. 1 Matrix at the k-th stage of the Householder method.

の要素からなるベクトルを $\mathbf{d}^{(k)}$，第 $(k+1,k+1)$ 要素を左上隅の要素とする部分行列を $\mathbf{C}^{(k)}$ とする．

第 k 段では，まず $\mathbf{d}^{(k)}$ から鏡像変換ベクトル $\mathbf{u}^{(k)}$ を作成した後，それを行列 $\mathbf{C}^{(k)}$ に掛けることによってベクトル $\mathbf{p}^{(k)}$，$\mathbf{q}^{(k)}$ を作成する．$\mathbf{u}^{(k)}$ を第 k 段におけるピボット列，$\mathbf{q}^{(k)}$ をピボット行と呼ぶ．最後に，$\mathbf{u}^{(k)}$ と $\mathbf{q}^{(k)}$ を使って，行列 $\mathbf{C}^{(k)}$ の更新を行う．この演算は，行列 $\mathbf{C}^{(k)}$ にランクが 1 の行列を 2 個加えるので，rank-2 更新と呼ばれる．以上の処理は，行列 $\mathbf{C}^{(k)}$ に対し，ハウスホルダ変換行列

$$\mathbf{H}^{(k)} \equiv \mathbf{I} - \alpha^{(k)}\mathbf{u}^{(k)}\mathbf{u}^{(k)t} \tag{2}$$

を用いて相似変換

$$\mathbf{C}^{(k)} := \left(\mathbf{H}^{(k)}\right)^{-1}\mathbf{C}^{(k)}\mathbf{H}^{(k)} \tag{3}$$

を行ったことに相当する[11),17)]．この処理を第 1 段から第 $N-2$ 段まで繰り返すことにより，行列の三重対角化が完了する[11),17)]．

ハウスホルダ法では，行列ベクトル積と行列の rank-2 更新にそれぞれ約 $\frac{2}{3}N^3$ の演算量がかかり，それぞれが全演算量のほぼ半分を占める．しかし，これらの演算では，行列 $\mathbf{C}^{(k)}$ のサイズを n とするとき，ともに $O(n^2)$ 回の演算に対して $O(n^2)$ 回のメモリアクセスが必要であり，データの再利用性が低いという問題点がある．

2.3 Dongarra らのアルゴリズム

そこで Dongarra らは，rank-2 更新の部分のデータ再利用性を高めるため，ハウスホルダ法を多段化したアルゴリズムを開発した[8),9)]．これをアルゴリズム 2 に示す．ここで，$(\mathbf{X})_i$ は行列 $\mathbf{X}$ の第 i 列目からなる列ベクトルを表し，$[\mathbf{A}|\mathbf{B}]$ は行列 $\mathbf{A}$ と行列 $\mathbf{B}$ とを並べてできる行列を表す．また，簡単のため，N は L で割り切れると仮定する．

このアルゴリズムでは，1 段ごとに行列の rank-2 更新を行うのではなく，ある整数 L を決め，L 段に 1 回，L 段分の rank-2 更新をまとめて行う．これは，行列 $\mathbf{C}^{(K*L)}$ に対し，L 段分の相似変換

$$\begin{aligned}\mathbf{C}^{(K*L)} := &\left(\mathbf{H}^{(K*L)}\right)^{-1}\cdots\left(\mathbf{H}^{((K-1)*L+1)}\right)^{-1}\\ &\times\mathbf{C}^{((K-1)*L)}\\ &\times\mathbf{H}^{((K-1)*L+1)}\cdots\mathbf{H}^{(K*L)}\end{aligned} \tag{4}$$

をまとめて行ったことに相当する．これにより，rank-2 更新は，rank-2L 更新で置き換えられる．この処理は一種の行列乗算であるため，データの再利用性は高く，キャッシュの有効利用が可能となる．しかし，演算量の残りの半分を占める行列ベクトル積については，データ再利用性が低いままで残り，この部分が性能上のネックとなるという問題点がある．

［アルゴリズム 2：Dongarra らのアルゴリズム］
do $K = 1,\ N/L$
  $\mathbf{U}^{((K-1)*L)} = \phi,\ \mathbf{Q}^{((K-1)*L)} = \phi$
  do $k = (K-1)*L+1,\ K*L$
    ［部分ハウスホルダ変換］
    $\mathbf{d}^{(k)} := \mathbf{d}^{(k)} - \mathbf{U}^{(k-1)}(\mathbf{Q}^{(k-1)t})_{k-(K-1)*L}$
    $\quad - \mathbf{Q}^{(k-1)}(\mathbf{U}^{(k-1)t})_{k-(K-1)*L}$
    ［鏡像ベクトル $\mathbf{u}^{(k)}$ の作成］
    $\sigma^{(k)} = \sqrt{\mathbf{d}^{(k)t}\mathbf{d}^{(k)}}$
    $\mathbf{u}^{(k)} = (d_1^{(k)} - \mathrm{sgn}(d_1^{(k)})\sigma^{(k)}, d_2^{(k)}, \ldots, d_{N-k}^{(k)})$
    $\alpha^{(k)} = 2/\parallel\mathbf{u}^{(k)}\parallel_2$
    ［行列ベクトル積］
    $\mathbf{p}^{(k)} = \alpha^{(k)}(\mathbf{C}^{(k)} - \mathbf{U}^{(k-1)}\mathbf{Q}^{(k-1)t}$
    $\quad - \mathbf{Q}^{(k-1)}\mathbf{U}^{(k-1)t})\mathbf{u}^{(k)}$
    $\beta^{(k)} = \alpha^{(k)}\mathbf{u}^{(k)t}\mathbf{p}^{(k)}/2$
    $\mathbf{q}^{(k)} = \mathbf{p}^{(k)} - \beta^{(k)}\mathbf{u}^{(k)}$
    $\mathbf{U}^{(k)} = [\mathbf{U}^{(k-1)}|\mathbf{u}^{(k)}]$
    $\mathbf{Q}^{(k)} = [\mathbf{Q}^{(k-1)}|\mathbf{q}^{(k)}]$
  end do
  ［行列の rank-2L 更新］
  $\mathbf{C}^{(K*L)} := \mathbf{C}^{((K-1)*L)} - \mathbf{U}^{(K*L)}\mathbf{Q}^{(K*L)t}$
  $\quad - \mathbf{Q}^{(K*L)}\mathbf{U}^{(K*L)t}$
end do

2.4 Bischof らのアルゴリズム

この問題を解決するため，Bischof らは行列 $\mathbf{A}$ をまず帯行列 $\mathbf{B}$ に変換し，次に $\mathbf{B}$ を三重対角行列に変換するという 2 段階のアルゴリズムを提案した[4),5)]．$\mathbf{B}$ の半帯幅を L とするとき，前半の帯行列への変換はサイズ $L\times L$ の行列乗算のみを用いて約 $\frac{4}{3}N^3$ の演算量

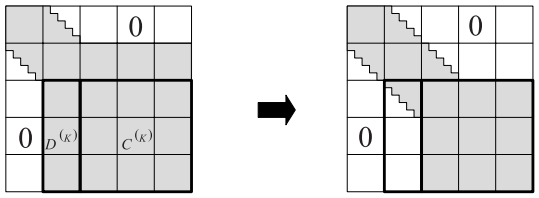

図 2 Bischof らのアルゴリズムの第 K 段における行列
Fig. 2 Matrix at the K-th stage of Bischof's algorithm.

で実行でき，後半の三重対角行列への変換は村田法[14]あるいは Rutishauser 法[17]と呼ばれる方法により約 $6N^2L$ の演算量で実行できる．したがって $L \ll N$ ならば演算量はハウスホルダ法とほぼ等しく，かつ演算のほとんどが行列乗算で行えるため，Bischof らのアルゴリズムはキャッシュマシン上で高い性能を発揮できると考えられる．

Bischof らのアルゴリズムのうち，前半の帯行列への変換の部分をアルゴリズム 3 に示す[4),5)]．ここで，$\boldsymbol{\alpha}$ などのギリシャ文字の太文字は行列を表すと新たに定義する．また，簡単のため，N は半帯幅 L で割り切れると仮定する．さらに，行列の要素を大きさ $L \times L$ のブロックに区分けし，ブロックを単位として参照する場合は大文字の添字を使って「第 K ブロック列」，「第 (K, K) ブロック要素」のように呼ぶ．

［アルゴリズム 3：Bischof らのアルゴリズム］
do $K = 1, N/L - 1$
［ブロックハウスホルダ変換の作成］
$\mathbf{D}^{(K)}$ を第 1 ブロックが上三角行列で第 2 ブロック以下がゼロ行列であるブロックベクトルに変換するブロックハウスホルダ変換 $\mathbf{I} - \mathbf{U}^{(K)}\boldsymbol{\alpha}^{(K)}\mathbf{U}^{(K)t}$ を求める．
［行列-ブロックベクトル積］
$\mathbf{P}^{(K)} = \mathbf{C}^{(K)}\mathbf{U}^{(K)}\boldsymbol{\alpha}^{(K)}$
$\boldsymbol{\beta}^{(K)} = \boldsymbol{\alpha}^{(K)t}\mathbf{U}^{(K)t}\mathbf{P}^{(K)}/2$
$\mathbf{Q}^{(K)} = \mathbf{P}^{(K)} - \mathbf{U}^{(K)}\boldsymbol{\beta}^{(K)}$
［行列の rank-2L 更新］
$\mathbf{C}^{(K)} := \mathbf{C}^{(K)} - \mathbf{U}^{(K)}\mathbf{Q}^{(K)t} - \mathbf{Q}^{(K)}\mathbf{U}^{(K)t}$
end do

計算は $K = 1$ から $K = N/L - 1$ までの $N/L - 1$ 段からなる．第 K 段における行列の変形の様子とブロックベクトル $\mathbf{D}^{(K)}$，行列 $\mathbf{C}^{(K)}$ の定義を **図 2** に示す．各段における処理の流れは 2.2 節で説明したハウスホルダ法とほぼ同一であり，ベクトルが幅 L のブロックベクトル（すなわち行列）に，スカラがサイズ $L \times L$ の行列に置き換わる形となる．特に，ベクトル $\mathbf{d}^{(k)}$，$\mathbf{u}^{(k)}$，$\mathbf{p}^{(k)}$，$\mathbf{q}^{(k)}$ はそれぞれ幅 L のブロックベクトル $\mathbf{D}^{(K)}$，$\mathbf{U}^{(K)}$，$\mathbf{P}^{(K)}$，$\mathbf{Q}^{(K)}$ に，スカラ $\alpha^{(k)}$，$\beta^{(k)}$ はそれぞれ $L \times L$ 行列 $\boldsymbol{\alpha}^{(K)}$，$\boldsymbol{\beta}^{(K)}$ に置き換わる．

第 K 段 $(1 \le K \le N/L-1)$ の処理では，図 2 に示すとおり，まず全体行列の第 K ブロック列の第 $K+1$ 番目以降の要素からなるブロックベクトル $\mathbf{D}^{(K)}$ に着目し，$\mathbf{D}^{(K)}$ を，第 1 ブロックが上三角行列でそれ以外の部分がゼロであるようなブロックベクトルに変換するブロックハウスホルダ変換 $\mathbf{I} - \mathbf{U}^{(K)}\boldsymbol{\alpha}^{(K)}\mathbf{U}^{(K)t}$ を求める．このような $\mathbf{U}^{(K)}$，$\boldsymbol{\alpha}^{(K)}$ は $\mathbf{D}^{(K)}$ の QR 分解および WY-representation と呼ばれる方法により容易に求めることができる[3)]．次に $\mathbf{U}^{(K)}$ を行列 $\mathbf{C}^{(K)}$ に掛けることにより，ブロックベクトル $\mathbf{P}^{(K)}$，$\mathbf{Q}^{(K)}$ を作成する．$\mathbf{U}^{(K)}$ を第 K 段におけるブロックピボット列，$\mathbf{Q}^{(K)}$ をブロックピボット行と呼ぶ．最後に，$\mathbf{U}^{(K)}$ と $\mathbf{Q}^{(K)}$ を使って行列 $\mathbf{C}^{(K)}$ の更新を行う．この演算は rank-2L 更新となる．以上の処理は，行列 $\mathbf{C}^{(K)}$ に対し，ブロックハウスホルダ変換行列

$$\tilde{\mathbf{H}}^{(K)} \equiv \mathbf{I} - \mathbf{U}^{(K)}\boldsymbol{\alpha}^{(K)}\mathbf{U}^{(K)t} \tag{5}$$

を用いて相似変換

$$\mathbf{C}^{(K)} := \left(\tilde{\mathbf{H}}^{(K)}\right)^{-1}\mathbf{C}^{(K)}\tilde{\mathbf{H}}^{(K)} \tag{6}$$

を行ったことに相当する．これにより，図 2 に示すように第 K ブロック列の帯行列化が完了する[4),5)]．

上記の処理においては，行列-ブロックベクトル積と行列の rank-2L 更新が，それぞれ演算量のほぼ半分を占める．これらは両方ともサイズ $L \times L$ の行列乗算を用いて行うことができ，L が大きいほど 1 回の計算におけるデータの再利用性が高くなる．したがって，キャッシュサイズに応じて L を適切にとることにより，キャッシュマシン上で高い性能が期待できる．

2.5 Wu らのアルゴリズム

Bischof らのアルゴリズムでは，前半の帯行列化部分の性能を最大化する L はキャッシュサイズにより定まるが，一方で後半の三重対角化部分の演算量は L に比例して増大する．そのため，全体の実行時間を最小化しようとすると L を十分大きくとれず，前半部においてキャッシュマシンの性能を十分発揮できない場合が生じうる．

この問題を解決するため，Wu らは Bischof らのアルゴリズムをさらに多段化したアルゴリズムを提案した[18]．このアルゴリズムでは，Bischof らのアルゴリズムにおいて，1 段ごとに行列の rank-2L 更新を行う

のではなく，ある整数 L' を決め，L' 段に 1 回，L' 段分の更新をまとめて行う．これは，行列 $\mathbf{C}^{(K*L')}$ に対し，L' 段分の相似変換

$$\begin{aligned}&\mathbf{C}^{(K*L')}\\ &:= \left(\tilde{\mathbf{H}}^{(K*L')}\right)^{-1}\cdots\left(\tilde{\mathbf{H}}^{((K-1)*L'+1)}\right)^{-1}\\ &\quad\times\mathbf{C}^{((K-1)*L')}\\ &\quad\times\tilde{\mathbf{H}}^{((K-1)*L'+1)}\cdots\tilde{\mathbf{H}}^{(K*L')}\end{aligned}\tag{7}$$

をまとめて行ったことに相当する．これにより得られるアルゴリズムをアルゴリズム 4 に示す．ここで，$\mathsf{U}^{(K)}$ など Sans Serif 体の記号は行列を要素とする行列を表し，$(\mathsf{U}^{(K)})_i$ はその第 i ブロック列のみからなる行列を表す．ただし，K のみは整数を表すとする．

［アルゴリズム 4：Wu らのアルゴリズム］
do $\mathsf{K}=1, N/(LL')-1$
  $\mathsf{U}^{((\mathsf{K}-1)*LL')}=\phi$, $\mathsf{Q}^{((\mathsf{K}-1)*LL')}=\phi$
  do $K=(\mathsf{K}-1)*L'+1, \mathsf{K}*L'$
    ［部分ハウスホルダ変換］
    $\mathbf{D}^{(K)} := \mathbf{D}^{(K)}$
      $-\mathsf{U}^{(K-1)}(\mathsf{Q}^{(K-1)t})_{K-(\mathsf{K}-1)*L'}$
      $-\mathsf{Q}^{(K-1)}(\mathsf{U}^{(K-1)t})_{K-(\mathsf{K}-1)*L'}$
    ［ブロックハウスホルダ変換の作成］
    $\mathbf{D}^{(K)}$ を第 1 ブロックが上三角行列で第 2 ブロック以下がゼロ行列であるブロックベクトルに変換するブロックハウスホルダ変換 $\mathbf{I}-\mathbf{U}^{(K)}\boldsymbol{\alpha}^{(K)}\mathbf{U}^{(K)t}$ を求める．
    ［行列-ブロックベクトル積］
    $\mathbf{P}^{(K)}=(\mathbf{C}^{(K)}-\mathsf{U}^{(K-1)}\mathsf{Q}^{(K-1)t}$
      $-\mathsf{Q}^{(K-1)}\mathsf{U}^{(K-1)t})\mathbf{U}^{(K)}\boldsymbol{\alpha}^{(K)}$
    $\boldsymbol{\beta}^{(K)}=\boldsymbol{\alpha}^{(K)t}\mathbf{U}^{(K)t}\mathbf{P}^{(K)}/2$
    $\mathbf{Q}^{(K)}=\mathbf{P}^{(K)}-\boldsymbol{\beta}^{(K)}\mathbf{U}^{(K)}$
    $\mathsf{U}^{(K)}=[\mathsf{U}^{(K-1)}|\mathbf{U}^{(K)}]$
    $\mathsf{Q}^{(K)}=[\mathsf{Q}^{(K-1)}|\mathbf{Q}^{(K)}]$
  end do
  ［行列の rank-2LL' 更新］
  $\mathbf{C}^{(K*L')} := \mathbf{C}^{((K-1)*L')}-\mathsf{U}^{(\mathsf{K}*L')}\mathsf{Q}^{(\mathsf{K}*L')t}$
    $-\mathsf{Q}^{(\mathsf{K}*L')}\mathsf{U}^{(\mathsf{K}*L')t}$
end do

多段化の適用により，行列の rank-2L 更新の部分は，rank-2LL' 更新となる．したがって Wu らのアルゴリズムでは，Bischof らのアルゴリズムこの部分に対して L の値を L' 倍にしたのと同様の効果があり，後半の三重対角化部分の演算量を増やさずに帯行列化部分の性能を向上できると考えられる．

なお，Wu らのアルゴリズムにおいて $L'=1$ とすると Bischof らのアルゴリズムとなる．また，$L=1$ とすると Dongarra らのアルゴリズムとなる．したがって，Wu らのアルゴリズムはこの両者の組合せであり，特別な場合として両方のアルゴリズムを含むと見なせる．

3. 性能と精度の評価

3.1 性能評価

3.1.1 測定条件

前章で述べた Dongarra らのアルゴリズム，Bischof らのアルゴリズム，Wu らのアルゴリズムを実装し，様々なマイクロプロセッサ上で性能評価を行った．評価に使ったプロセッサは Intel Pentium 4 Xeon（2.0 GHz，L1 データキャッシュ8 KB/L2 データキャッシュ256 KB），AMD Opteron（1.6GHz, 64 KB/1 MB），DEC Alpha 21264A（750 MHz, 64 KB/4 MB），Sun Ultra SPARC III（750 MHz, 64 KB/4 MB）の 4 種類である．プログラムは FORTRAN で作成し，コンパイラは Ultra SPARC III では Sun の Fortran 77 コンパイラ，それ以外のマシンでは g77 を用いた．最適化オプションはすべて-O4 である．また，計算の中心部となる行列乗算には，すべてのマシン上で ATLAS3.6.0[2] のルーチンを用いた．ルーチンとしては，行列-ブロックベクトル積では DSYMM，行列の更新では DSYR2K，その他の行列乗算はすべて DGEMM を使用した．

三重対角化部分の性能は入力行列によらないため，入力行列としては Frank 行列 $a_{ij}=\min(i,j)$ を用いた．行列サイズは N=480，960，1920，3840 の 4 通りとし，Dongarra らのアルゴリズムにおける多段化の段数は L'=1，2，4，8，16，32，64 の 7 通り，Bischof らのアルゴリズムにおけるブロックサイズ（半帯幅）は L=1，3，6，12，24，48，96 の 7 通りに変えて測定した．また，Wu らのアルゴリズムでは，L'，L をそれぞれ 1 以外の 6 通りの値に変えて性能を測定した．

3.1.2 L と L' を変えた場合の性能評価

まず，Opteron 上で L と L' を変えた場合の三重対角化の実行時間を**表 1**（$N=960$ の場合）と**表 2**（$N=1920$ の場合）に示す．表中で $L=1$ の列が Dongarra らのアルゴリズム，$L'=1$ の行が Bischof らのアルゴリズムに相当し，それ以外の部分が Wu らのアルゴリズムに相当する．また，各 L，L' に対する 4 つの数字は，一番上が帯行列化の時間，2 番目の括弧中の数字が帯行列化における行列乗算部分のみの時間，3 番目が帯行列から三重対角行列への変換時間，4

表 1 L と L' を変えたときの実行時間（Opteron 1.6 GHz, $N = 960$）
Table 1 Execution times as a function of L and L' (Opteron 1.6 GHz, $N = 960$).

L'	L =1	3	6	12	24	48	96
1	8.08	3.48	1.96	1.23	0.92	0.77	0.81
	(0.0)	(3.45)	(1.94)	(1.21)	(0.86)	(0.66)	(0.59)
	–	0.07	0.10	0.16	0.26	0.44	0.96
	8.08	**3.55**	**2.06**	**1.39**	**1.18**	**1.21**	**1.77**
2	4.70	2.23	1.37	0.98	0.80	0.75	0.85
	(3.71)	(2.19)	(1.35)	(0.95)	(0.74)	(0.64)	(0.63)
	–	0.07	0.10	0.16	0.26	0.44	0.96
	4.70	**2.30**	**1.47**	**1.14**	**1.06**	**1.19**	**1.81**
4	2.92	1.63	1.11	0.86	0.78	0.79	
	(1.97)	(1.62)	(1.10)	(0.83)	(0.73)	(0.69)	
	–	0.07	0.10	0.16	0.26	0.45	
	2.92	**1.70**	**1.21**	**1.02**	**1.04**	**1.24**	
8	2.05	1.39	1.01	0.85	0.84		
	(1.13)	(1.37)	(1.00)	(0.83)	(0.78)		
	–	0.07	0.10	0.17	0.26		
	2.05	**1.46**	**1.11**	**1.02**	**1.10**		
16	1.64	1.32	1.04	0.94			
	(0.71)	(1.31)	(1.02)	(0.91)			
	–	0.07	0.10	0.16			
	1.64	**1.39**	**1.14**	**1.10**			
32	1.48	1.43	1.20				
	(0.55)	(1.41)	(1.17)				
	–	0.07	0.09				
	1.48	**1.50**	**1.29**				
64	1.64	1.71					
	(0.64)	(1.68)					
	–	0.07					
	1.64	**1.78**					

表 2 L と L' を変えたときの実行時間（Opteron 1.6 GHz, $N = 1920$）
Table 2 Execution times as a function of L and L' (Opteron 1.6 GHz, $N = 1920$).

L'	L=1	3	6	12	24	48	96
1	49.31	22.79	12.97	8.28	6.24	5.21	5.33
	(0.0)	(22.75)	(12.87)	(8.15)	(6.01)	(4.73)	(4.25)
	–	0.31	0.42	0.69	1.08	2.16	4.93
	49.31	**23.10**	**13.39**	**8.97**	**7.32**	**7.37**	**10.26**
2	30.13	15.48	9.55	6.79	5.50	5.04	5.47
	(22.02)	(15.45)	(9.48)	(6.68)	(5.27)	(4.58)	(4.39)
	–	0.31	0.42	0.68	1.09	2.16	4.93
	30.13	**15.79**	**9.97**	**7.47**	**6.59**	**7.20**	**10.40**
4	19.96	12.08	8.09	6.08	5.34	5.19	5.87
	(11.93)	(12.04)	(8.04)	(5.98)	(5.12)	(4.73)	(4.80)
	–	0.31	0.42	0.69	1.09	2.16	4.93
	19.96	**12.39**	**8.51**	**6.77**	**6.43**	**7.35**	**10.80**
8	14.92	10.64	7.47	6.01	5.55	5.66	
	(6.90)	(10.62)	(7.38)	(5.90)	(5.33)	(5.20)	
	–	0.32	0.42	0.68	1.09	2.16	
	14.92	**10.96**	**7.89**	**6.69**	**6.64**	**7.82**	
16	12.55	10.17	7.51	6.36	6.17		
	(4.50)	(10.15)	(7.47)	(6.24)	(5.95)		
	–	0.31	0.42	0.68	1.08		
	12.55	**10.48**	**7.93**	**7.04**	**7.25**		
32	11.84	10.60	8.19	7.26			
	(3.67)	(10.57)	(8.13)	(7.15)			
	–	0.31	0.42	0.68			
	11.84	**10.91**	**8.61**	**7.94**			
64	12.76	11.86	9.66				
	(4.53)	(11.82)	(9.58)				
	–	0.31	0.42				
	12.76	**12.17**	**10.08**				

番目の太字が合計の実行時間を示す．また，3 つのアルゴリズムそれぞれに対し，最も短かった実行時間を四角で囲んである．本来，この表はすべてのプロセッサ，すべての N に対して載せるべきであるが，ここでは紙面の制約から，この 2 つの場合のみを載せた．

表より，この 2 つのケースでは，ともに Dongarra らのアルゴリズムでは $L' = 32$，Bischof らのアルゴリズムでは $L = 24$ が最適であることが分かる．Bischof らのアルゴリズムでは，前半の帯行列化の部分だけを取り出すと $L = 48$ のほうが高速であるが，L が増えると後半部の実行時間が増加するため，全体としては $L = 24$ が最適になっている．これに対し，Wu らのアルゴリズムは L を増やさずに前半部における rank-2L 更新のブロックサイズを実効的に大きくできるため，全実行時間の短縮に効果的であることが分かる．なお，Wu らのアルゴリズムでは，Bischof らのアルゴリズムに対する最適な L あるいは 1 つ下の L を使い，$L' = 4$ または $L' = 8$ とした場合が最も効果的であり，この傾向は他のプロセッサでも同様であった．

また，括弧の中の数字を見ることにより，たとえば $N = 1920$ の場合，$L' = 32$ の Dongarra らのアルゴリズムで行列乗算の占める割合は全実行時間 11.84 秒中の 3.67 秒（31%）にすぎず，それ以外の部分が実行時間のネックとなっていることが分かる．一方，Bischof らのアルゴリズムでは 7.32 秒中の 6.01 秒（82%），Wu らのアルゴリズムでは 6.43 秒中の 5.12 秒（79%）が行列乗算の実行時間となっており，行列乗算の占める割合を増やして高速化を行うという意図は確かに達成されていることが分かる．

3.1.3 3 つのアルゴリズムの性能比較

次に，各プロセッサ上で 3 つのアルゴリズムの性能を比較した結果を**図 3**，**図 4**，**図 5**，**図 6** に示す．性能は，通常のハウスホルダ法の演算量 $\frac{4}{3}N^3$ を三重対角化の全実行時間で割ることにより算出した．なお，L，L' は各 N に対して最適な値を用い，その値はグラフ上に記入してある．

グラフより，Bischof らのアルゴリズムは Dongarra らのアルゴリズムに対してほぼ一貫して高速であり，

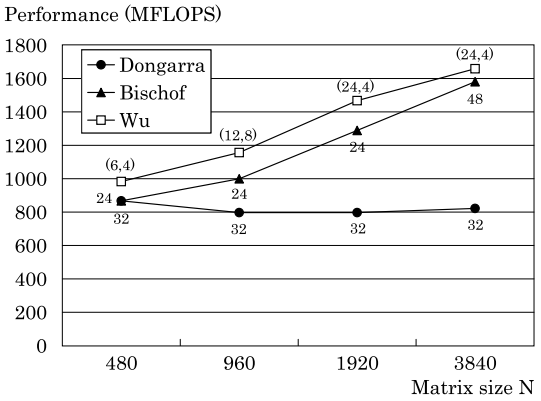

図 3 Opteron（1.6 GHz）上での各アルゴリズムの性能
Fig. 3 Performance of the three algorithms on the Opteron processor (1.6 GHz).

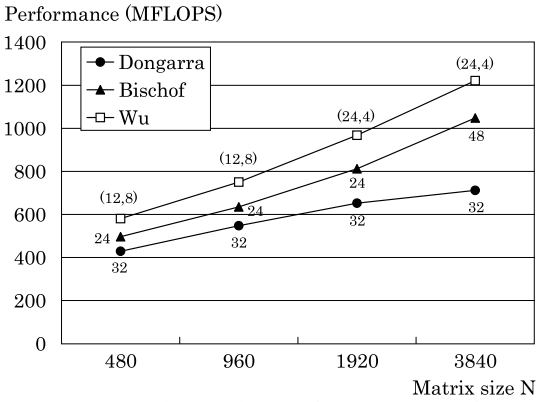

図 4 Xeon（2.0 GHz）上での各アルゴリズムの性能
Fig. 4 Performance of the three algorithms on the Xeon processor (2.0 GHz).

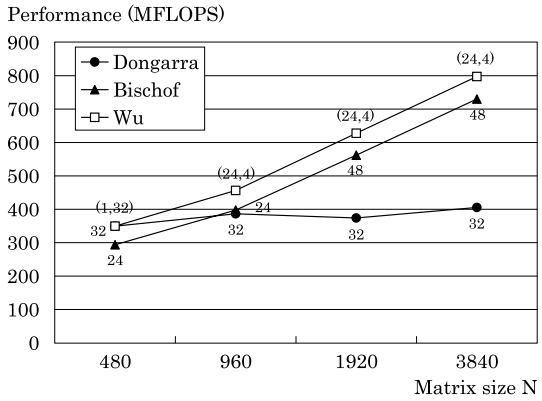

図 5 Alpha 21264A（750 MHz）上での各アルゴリズムの性能
Fig. 5 Performance of the three algorithms on the Alpha 21264A processor (750 MHz).

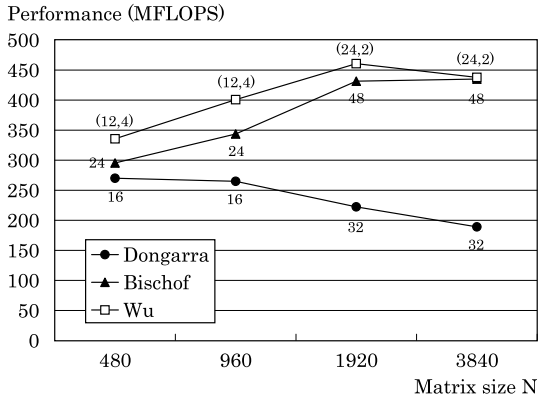

図 6 Ultra SPARC III（750 MHz）上での各アルゴリズムの性能
Fig. 6 Performance of the three algorithms on the Ultra SPARC III processor (750 MHz).

その差は N が大きくなるにつれて開くことが分かる．また，Wu らのアルゴリズムを適用することにより Bischof らのアルゴリズムをさらに 10〜15%程度高速化でき，$N = 3840$ の場合にはすべてのプロセッサに対して Dongarra らのアルゴリズムに比べ 2 倍程度の高速化が達成できることが分かる．なお，$N = 3840$ の場合，Opteron および Alpha 上での性能はそれぞれピークの 52%，53%を達成しており，最大でも 25%程度しか達成できなかった Dongarra らのアルゴリズムに比べ，大きく優位に立つ．以上より，Bischof らのアルゴリズム，およびそれを改良した Wu らのアルゴリズムは，大規模問題を解く場合に広い範囲のマイクロプロセッサに対して有効であることが明らかになった．

3.2 精度評価

次に，3 つのアルゴリズムにより得られる固有値の精度を比較するため，4 種類のテスト行列を用いて精度評価を行った．用いた行列は，(a) Frank 行列，(b) ラプラス方程式の固定境界値問題を 2 次元 5 点差分により離散化して得られる行列，(c) Harwell-Boeing Sparse Matrix Collection [12] から取ってきた 5 個の構造解析の行列（**表 3** 参照），(d) 区間 [0,1] の一様乱数を要素とする対称行列，の 4 種類である．三重対角行列の固有値を求めるには二分法を使った．また，比較対照の固有値 λ_i^c としては，(a)，(b) は解析解，(c)，(d) は通常のハウスホルダ法により求めた固有値を用いた．固有値の精度は，相対誤差の最大値

$$\max_{1 \le i \le N} \left| \frac{\lambda_i - \lambda_i^c}{\lambda_i^c} \right| \tag{8}$$

を用いた．L，L' としては，多くのプロセッサで最適であった値として，Dongarra らのアルゴリズムでは $L' = 32$，Bischof らのアルゴリズムでは $L = 24$，Wu らのアルゴリズムでは $L = 24$，$L' = 4$ を採用した．

表 3 構造解析の行列に関する緒元
Table 3 Parameters for the structural analysis matrices.

名称	N	非ゼロ要素数	条件数（推定値）
bcsstk08	1074	7017	4.7×10^{7}
bcsstk12	1473	17857	5.3×10^{8}
bcsstk13	2003	42943	4.6×10^{10}
bcsstk23	3134	24156	6.9×10^{12}
bcsstk24	3562	81736	65

なお，本節の計算はすべて Opteron（1.6 GHz）上で行った．

(a)，(b)，(c) に対する結果をそれぞれ**表 4**，**表 5**，**表 6** に示す．また，(d) の乱数行列に対しては，各 N に対してそれぞれ 10 通りの乱数行列を用いて評価を行った結果を**図 7** に示す．

これらの結果より，Bischof らのアルゴリズムと Wu らのアルゴリズムによる固有値の相対誤差は，Dongarra らのアルゴリズムに比べてやや大きくなる傾向が見られる．しかし，増大の程度は多くの場合 2〜3 倍以内であり，最大でも 1 桁程度に抑えられている．したがって，この程度の誤差増大が許容できる応用においては，Bischof らのアルゴリズムおよび Wu らのアルゴリズムは有効な選択肢となりうると考えられる．なお，L，L' の値をそれぞれ 3.1.1 項に述べた 7 通りに変化させた場合の精度評価も行ったが，この範囲内では両アルゴリズムとも誤差の大きな増大は見られず，Dongarra らのアルゴリズムに比べた誤差の増大は 1 桁以内に収まった．

4. 固有ベクトルの計算が必要な場合についての考察

以上では固有値計算を行う場合を対象として各アルゴリズムの性能・精度評価を行ったが，次に固有ベクトルの計算も必要な場合について，各アルゴリズムのメリット/デメリットを簡単に考察する．

4.1 固有ベクトルの計算法と演算量

以下では，計算したい固有ベクトルの本数 M が N に比べて小さいとし，固有ベクトルの計算には逆反復法[11),17)] を用いるとする．また，逆反復法における直交化の演算量は，固有値の分布や直交化のアルゴリズム[7),13),20),21)] により大きく変わるため，今回は考慮の対象から外すとする．

従来のハウスホルダ法および Dongarra らのアルゴリズムで固有ベクトルを求める場合は，まず入力行列 $\mathbf{C}$ を変換して得られた三重対角行列 $\mathbf{T}$ の固有ベクトルを求め，これに対して逆変換を行うことにより，$\mathbf{C}$ の固有ベクトルを求める[11),17)]．このときの演算量は，逆反復法部分が $O(NM)$，逆変換部分が約 $2N^2M$ である．

一方，Bischof らのアルゴリズムおよび Wu らのアルゴリズムでは，$\mathbf{C}$ をまず半帯幅 L の帯行列 $\mathbf{B}$ に変換してから，$\mathbf{B}$ を三重対角行列 $\mathbf{T}'$ に変換するという 2 段階の変換を行う．この場合，固有ベクトルの計算法としては次の 2 通りが考えられる．

表 4 Frank 行列に対する固有値の最大相対誤差
Table 4 Maximum relative errors of the computed eigenvalues for the Frank matrices.

N	Dongarra ら	Bischof ら	Wu ら
480	3.87×10^{-11}	4.03×10^{-11}	4.03×10^{-11}
960	1.59×10^{-10}	2.45×10^{-10}	2.45×10^{-10}
1920	7.39×10^{-10}	7.72×10^{-10}	7.90×10^{-10}
3840	1.91×10^{-9}	3.65×10^{-10}	3.65×10^{-9}

表 5 2 次元 5 点差分の行列に対する固有値の最大相対誤差
Table 5 Maximum relative errors of the computed eigenvalues for the 2-dimensional 5-point finite difference matrices.

N	Dongarra ら	Bischof ら	Wu ら
480	4.24×10^{-15}	3.69×10^{-14}	1.14×10^{-14}
960	1.94×10^{-14}	8.87×10^{-14}	3.03×10^{-14}
1920	4.43×10^{-14}	1.90×10^{-13}	9.11×10^{-14}
3840	1.14×10^{-13}	1.68×10^{-13}	3.42×10^{-13}

表 6 構造解析の行列に対する固有値の最大相対誤差
Table 6 Maximum relative errors of the computed eigenvalues for the structural analysis matrices.

N	Dongarra ら	Bischof ら	Wu ら
bcsstk08	1.38×10^{-9}	2.91×10^{-9}	1.37×10^{-9}
bcsstk12	1.46×10^{-8}	1.39×10^{-8}	2.58×10^{-8}
bcsstk13	1.34×10^{-6}	1.16×10^{-6}	1.26×10^{-6}
bcsstk23	1.78×10^{-4}	1.53×10^{-4}	1.09×10^{-4}
bcsstk24	1.36×10^{-5}	3.60×10^{-5}	3.09×10^{-5}

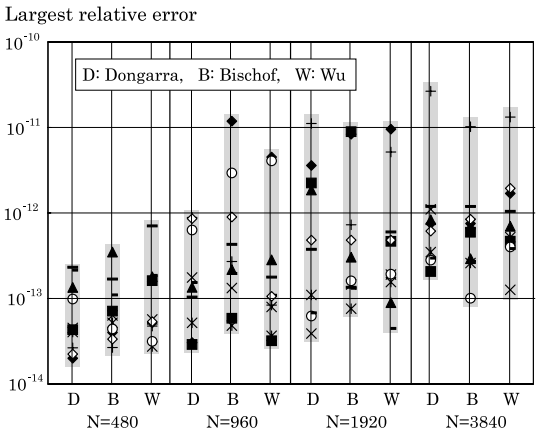

図 7 乱数行列に対する固有値の最大相対誤差
Fig. 7 Maximum relative errors of the computed eigenvalues for the random matrices.

(1) まず $\mathbf{T}'$ の固有ベクトルを求め，これに対して2段階の逆変換を行うことにより，$\mathbf{C}$ の固有ベクトルを求める．
(2) 帯行列 $\mathbf{B}$ の固有ベクトルを直接求め，これに対して1段階の逆変換を行うことにより，$\mathbf{C}$ の固有ベクトルを求める．

演算量は，(1) の場合，逆反復法部分が $O(NM)$，逆変換部分が約 $4N^2M$ となる[10]．(2) の場合は，逆反復法部分が（半帯幅 L の帯行列に対する連立一次方程式を解く必要があることから）$O(NL^2M)$，逆変換部分が約 $2N^2M$ となる．通常は $L \ll N$ であり，(2) のほうが演算量が小さいため，以下では (2) の計算法を採用した場合を考える．

4.2 性能に関する考察

Dongarra らのアルゴリズム，Bischof らのアルゴリズム，Wu らのアルゴリズムの 3 つにおける固有ベクトル計算の演算量を比較すると，逆変換の演算量は3つとも同じであるが，逆反復法の演算量は Dongarra らのアルゴリズムの $O(NM)$ に対し，他の2つは $O(NL^2M)$ と L^2 倍の演算量が必要である．一方，前章の結果より，三重対角化の性能は，N が数千程度の場合，Wu らのアルゴリズムが Dongarra らのアルゴリズムに比べて 2 倍程度優っている．

いま，M/N と L を固定して N を大きくした場合を考えると，逆反復法の演算量は $O(N^2)$，三重対角化の演算量は $O(N^3)$ で増大するから，十分大きい N に対しては三重対角化の時間が支配的となり，Bischof らのアルゴリズムと Wu らのアルゴリズムは，固有ベクトルの計算が必要な場合でも Dongarra らのアルゴリズムより高速になると考えられる．なお，Bischof らのアルゴリズムと Wu らのアルゴリズムとを比べると，後者は三重対角化の性能が優っており，固有ベクトル計算部分の演算量はまったく同じであることから，Wu らのアルゴリズムのほうがつねに高速となる．

以上は N が大きい場合の漸近的な議論であるが，与えられた N，M に対して Dongarra らと Wu らのどちらのアルゴリズムが高速か，また，L をどう選ぶのが最適かを知るには，実験あるいは性能評価モデルによる解析が必要であり，今後の課題である．

4.3 精度に関する考察

逆反復法で第 i 番目の固有ベクトル $\mathbf{v_i}$ を求める場合を考える．第 i 番目の真の固有値を λ_i^c，計算した固有値を λ_i とすると，逆反復法では，反復ベクトル中の $\mathbf{v_i}$ 成分が反復ごとに $|\lambda_i - \lambda_i^c|^{-1}$ 倍になることを利用して計算を行っている[11),17)]．前節の結果より，Bischof らのアルゴリズムと Wu らのアルゴリズムによる固有値の相対誤差は，Dongarra らのアルゴリズムに比べ，最大 1 桁程度大きくなる傾向が見られるが，この結果，拡大率 $|\lambda_i - \lambda_i^c|^{-1}$ も 1/10 程度にまで小さくなりうる．

これにより収束までに必要な反復回数が増える可能性が生じるが，拡大率は通常 10 の数十乗という大きな値であることを考えると，反復回数の増加はたかだか 1 回で済むと予想される．これに関しては，詳しい数値実験により検証を行う予定である．

5. おわりに

5.1 本研究のまとめ

本研究では，キャッシュマシン向けに提案された三重対角化アルゴリズムである Bischof らのアルゴリズムおよび Wu らのアルゴリズムの実用性を評価することを目的とし，Pentium 4 Xeon，Opteron，Alpha 21264A，Ultra SPARC III の 4 種のマイクロプロセッサ上での性能評価と，Frank 行列，2 次元 5 点差分の行列，構造解析から得られた行列，乱数行列の 4 種類のテスト行列による精度評価とを行った．その結果，次の点を明らかにした．

(1) Bischof らのアルゴリズムは Dongarra らのアルゴリズムに比べてほぼ一貫して高速であり，その差は N が大きくなるにつれ増大する．
(2) Wu らのアルゴリズムは Bischof らのアルゴリズムよりさらに 10〜15%高速であり，$N = 3840$ の場合には Dongarra らのアルゴリズムに比べ 2 倍程度の高速化が達成できる．また，$N = 3840$ の場合，Opteron，Alpha 21264A ではピークの 50%以上の性能が達成できる．
(3) Bischof らのアルゴリズムと Wu らのアルゴリズムによる固有値の相対誤差は，Dongarra らのアルゴリズムに比べてやや大きくなる傾向が見られる．しかし，増大の程度は多くの場合 2〜3 倍以内であり，最大でも 1 桁程度である．

以上より，この程度の誤差増大が許容できる応用においては，Bischof らのアルゴリズムおよびそれを改良した Wu らのアルゴリズムは，広く使われている Dongarra らのアルゴリズムに代わる有力な選択肢となりうると考えられる．

5.2 今後の課題

今後は，今回開発したプログラムを分子計算などの実問題に適用するとともに，固有ベクトル計算部分の実装と性能・精度評価，および分散メモリ向けの並列化を行い，本プログラムを PC クラスタ向けの固有値ソルバへと発展させていく予定である．また，Wu ら

のアルゴリズムではプロセッサの種類や問題サイズ N に応じた適切なブロックサイズ L，L' の決定が重要なため，これらを自動的に決定するための自動チューニング技術の開発もあわせて行っていく予定である．

謝辞 本研究に対して有益な助言をくださった査読者の方々に感謝いたします．また日頃からご指導いただいている名古屋大学大学院工学研究科の杉原正顯教授に感謝いたします．なお，本研究は名古屋大学 21 世紀 COE プログラム「計算科学フロンティア」および科学研究費補助金若手研究（B）（課題番号 16760053）の補助を受けている．

参 考 文 献

1) Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S. and Sorensen, D.: *LAPACK Users' Guide*, 2nd edition, SIAM, Philadelphia (1995).
2) http://math-atlas.sourceforge.net
3) Bischof, C. and van Loan, C.F.: The WY Representation for Products of Householder Matrices, *SIAM Journal on Scientific and Statistical Computing*, Vol.8, No.1, pp.s2–s13 (1987).
4) Bischof, C., Marques, M. and Sun, X.: Parallel Bandreduction and Tridiagonalization, Technical Report 8, *PRISM Working Note* (1993).
5) Bischof, C., Lang, B. and Sun, X.: Parallel Tridiagonalization through Two-step Band Reduction, Technical Report 17, *PRISM Working Note* (1994).
6) Blackford, L.S, Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D. and Whaley, R.C.: *ScaLAPACK Users' Guide*, SIAM, Philadelphia (1997).
7) Dhillon, I.: A New $O(n^2)$ Algorithm for the Symmetric Tri-diagonal Eigenvalue/Eigenvector Problem, Ph.D. Thesis, Computer Science Division, University of California, Berkeley (1997).
8) Dongarra, J.J., Hammarling, S.J. and Sorensen, D.C.: Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations, *Journal of Computational and Applied Mathematics*, Vol.27, pp.215–227 (1989).
9) Dongarra, J.J. and van de Geijn, R.A.: Reduction to Condensed Form for the Eigenvalue Problem on Distributed Architectures, *Parallel Computing*, Vol.18, No.9, pp.973–982 (1992).
10) Gansterer, W.N., Kvasnicka, D.F., Schwarz, K. and Ueberhuber, C.W.: Blocking Techniques in Symmetric Eigensolvers, *Proc. 9th SIAM Conference on Parallel Processing and Scientific Computing 1999*, SIAM, Philadelphia (1999).
11) Golub, G.H. and van Loan, C.F.: *Matrix Computations*, 3rd edition, Johns Hopkins University Press (1996).
12) http://math.nist.gov/MatrixMarket/collections/hb.html
13) 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，Vol.45, No.SIG 6 (ACS 6)，pp.75–85 (2004).
14) 村田健郎，小国　力，唐木幸比古：スーパーコンピュータ：科学技術計算への適用，丸善 (1985).
15) Parlett, B.N.: *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia (1998).
16) Salvini, S.A. and Mulholland, L.S.: The NAG FORTRAN Library, *Proc. 9th SIAM Conference on Parallel Processing and Scientific Computing 1999*, SIAM, Philadelphia (1999).
17) Wilkinson, J.H. and Reinsch, C. (Eds.): *Linear Algebra*, Springer-Verlag (1971).
18) 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* (1996).
19) 山本有作，大河内俊夫：ガウス消去法の超並列機向け最適化，並列処理シンポジウム JSPP '95 論文集，pp.217–224 (1995).
20) 山本有作，猪貝光祥，直野　健：共有メモリ型並列計算機向けの高並列固有ベクトル解法と SR8000 での評価，情報処理学会論文誌，Vol.42, No.4, pp.771–778 (2001).
21) Yamamoto, Y., Igai, M. and Naono, K.: A New BLAS-3 Based Parallel Algorithm for Computing the Eigenvectors of Real Symmetric Matrices, Yang, L.T. and Pan, Y. (Eds.), *High Performance Scientific and Engineering Computing — Hardware/Software Support*, Kluwer Academic Publishers (2004).

(平成 16 年 5 月 13 日受付)
(平成 16 年 8 月 31 日採録)

山本 有作（正会員）

1966 年生．1990 年東京大学工学部計数工学科（数理工学コース）卒業．1992 年同大学院工学系研究科物理工学専攻修士課程修了．同年（株）日立製作所中央研究所入所．2003 年名古屋大学大学院工学研究科計算理工学専攻助手．現在，同講師．並列計算機向け行列計算アルゴリズムおよび金融工学向け高速計算アルゴリズムの研究開発に従事．高性能計算とその応用に興味を持つ．博士（工学）．SIAM，INFORMS 各会員．