ハイパフォーマンス
コンピューティング 90－2
（２００２．５．２７）

共有メモリ型並列計算機向け線形演算ライブラリにおける並列化手法の評価

舘野 諭司[†] 西村 成司[††] 重原 孝臣[†] 長谷川 秀彦[†††] 桧山 澄子[†]

[†]埼玉大学 [††]日本SGI株式会社 [†††]図書館情報大学

概要

本稿では，共有メモリ型並列計算機であるHITACHI SR8000 1ノード上において，連立一次方程式，固有値問題を例にNAG Fortran SMPライブラリ，ベンダ提供のLAPACKの性能を比較し，それぞれのライブラリで採用されている並列化手法の特徴・問題点を明確にする．

Evaluation of Tuning Techniques in Parallelized Linear Algebra Libraries on Shared Memory Parallel Computers

Satoshi Tateno,[†] Seiji Nishimura,[††] Takaomi Shigehara,[†]
Hidehiko Hasegawa[†††] and Sumiko Hiyama[†]

[†]Saitama University, [††]SGI Japan Ltd.,
[†††]University of Library and Information Science

Abstract

In this paper, we evaluate efficiency of tuning techniques in parallelized linear algebra libraries, LAPACK tuned by vendors and NAG Fortran SMP Library, on a single node of HITACHI SR8000, which is a representative shared memory parallel computer. Direct solution of linear systems and eigenvalue problems are considered.

1 はじめに

ライブラリ使用は，ユーザが自ら処理ルーチンを作成する手間が省けるという点だけでなく，精度の保証・汎用性といった面でも優れている．特にライブラリの実効性能は，プログラムの実行に数十時間以上かかる大規模問題を扱うユーザやリアルタイム処理が必要なユーザにとって最も関心のある部分である．しかし，ライブラリの利用によりすべての問題について実効性能が改善されるという保証はない．また，同一の処理を実現できるライブラリは一つとは限らず，施されているチューニングの差異により各ライブラリの実効性能は異なる．特に並列計算機上では，並列化手法も性能に大きな影響を与える．

本研究ではHITACHI SR8000 1ノードを共有メモリ型並列計算機として使用し，並列化のアプローチが異なる線形演算ライブラリの性能評価をおこない，それぞれのライブラリが採用している並列化手法の特徴・問題点を明らかにする．性能測定をおこなう問題は，連立一次方程式の直接解法，対称行列/非対称行列の固有値問題である．比較の対象としたライブラリは，LAPACK(Linear Algebra Package)[1, 2] とその互換ルーチンを含むライブラリである．前者には演算処理に用いるBLAS(Basic Linear Algebra Subprograms) [3, 4, 5, 6, 7] を重点的に並列化した並列版のLAPACK，後者にはBLASではなくルーチン本体の並列性のある部分をOpenMP[8] により並列化したThe Numerical Algorithm Group社のNAG Fortran SMPライブラリ (以下，*NAG*と称す)[9] を使用する．また，ライブラリの使用による性能向上の程度を調べるため，何もチューニングを施していないユーザ作成のプログラムとの比較もおこなう．

本稿の構成は以下の通りである．数値実験の計算環境および比較をおこなった各ライブラリの特徴を2節に示す．3節では，まず実験方法および測定に用いたアルゴリズムを提示し，次に連立一次方程式，対称行列の固有値問題，非対称行列の固有値問題の順に実験データの提示・解析をおこない，各ライブラリの特徴を明らかにする．最後に4節にて本稿のまとめをおこなう．

表 1: コンパイラのバージョンとコンパイルオプション

Compiler	Compile Option	
Optimizing FORTRAN 90 Compiler V01-03-/A	-nolimit -noscope -64 -omp -save -O4 -parallel +SBTLB	
Optimizing FORTRAN 77 Compiler V01-03-/A	共通	-nolimit -noscope -64 -pvfunc=3
	8CPU	-W0,'OPT(O(4)),MP(P(3))' -procnum=8 -parallel
	1CPU	-W0,'OPT(O(4))' -noparallel

表 2: ライブラリのバージョンとリンクするライブラリ

		Netlib	*Vendor*	*NAG*
バージョン		Version 3.0	V01-03	Release 2
ライブラリ	8CPU	並列版	並列版	NAG Fortran SMP ライブラリ
	1CPU	シングル版	シングル版	
BLAS	8CPU	並列版 BLAS		シングル版 BLAS
	1CPU	シングル版 BLAS		

2 計算環境

2.1 計算機のスペック

数値実験はすべて HITACHI SR8000 1 ノード (以下，SR8000 と称す) 上でおこなう．SR8000 は演算 CPU 8 台と制御 CPU 1 台で 1 ノードを構成しており，16GByte のメインメモリを共有する．CPU は PowerPC ベースであり，二次キャッシュを利用せずに高い性能を発揮する擬似ベクトル機構 [10, 11] を備えている．また，各 CPU の一次キャッシュは命令 64KByte，データ 128KByte であるが，並列処理時には一次キャッシュも共有され，容量が CPU 台数倍に増加する．測定に用いた CPU 台数は 1 台および 8 台である．

測定プログラムは FORTRAN77 で作成した．使用したコンパイラのバージョンとコンパイルオプションを表 1 に示す．*NAG* では OpenMP の使用が前提であるが，今回使用した FORTRAN77 コンパイラは OpenMP をサポートしていない．そのため，*NAG* の測定には OpenMP をサポートし，性能が FORTRAN77 コンパイラとほとんど変わらない FORTRAN90 コンパイラを用いた．

2.2 比較対象のライブラリ

比較対象は，Netlib[12] で配布されている LAPACK(以下，*Netlib* と称す)，ベンダ提供の HITACHI LAPACK(以下，*Vendor* と称す) および *NAG* である．各ライブラリとも核演算には BLAS を用いている．BLAS のチューニングによる性能変化を調査するため，Netlib で配布されている未チューニングの BLAS (以下，**NetlibBLAS** と称す) およびベンダがチューニングした BLAS(以下，**VendorBLAS** と称す) を用いる．BLAS には複数の CPU 上で並列処理される並列版と単一の CPU 上で逐次処理されるシングル版が存在する．性能評価をおこなったライブラリのバージョンと 1CPU/8CPU においてリンクするライブラリ・BLAS のオブジェクトを表 2 に示す．

また，ライブラリの使用による性能改善の程度を調べるため，標準的アルゴリズムに基づくユーザ作成のプログラム (以下，*User* と称す) も比較の対象に加える．

並列版 LAPACK (*Netlib*, *Vendor*)

並列化は BLAS ルーチン内部等の局所的な演算処理に対して重点的におこなわれている．*Netlib*, *Vendor* 共に，並列処理をおこなうかどうかで並列版とシングル版の 2 種類が存在し，並列版ライブラリを利用する場合には BLAS も並列版をリンクする必要がある．*Vendor* はベンダによるチューニングが施されており，*Netlib* はコンパイラによる最適化・自動並列化機能のみ用いている．*Netlib*, *Vendor* 共に，バージョンは LAPACK Version 3.0 相当である．

NAG Fortran SMP ライブラリ (*NAG*)

並列化は演算処理 (BLAS) に対してではなく，互いにデータ依存性のない独立なルーチンを繰り返し呼び出すループやセクション処理等，ルーチン本体の並列性のある部分に対して OpenMP を用いておこなう．*NAG* を利用するプログラムは並列版・シングル版の区別はなく，並列処理に利用する CPU 台数を環境変数に指定することで並列処理

をおこなう．また，並列化された処理の内部で呼び出される BLAS は 1 つの CPU 上で処理される．そのため，シングル版 BLAS のみをリンクしさらに並列化されることを防ぐ．*NAG* には LAPACK では提供されていないルーチンも含まれているが，今回はアルゴリズムによる差をできるだけ少なくするため *NAG* に含まれる LAPACK 互換のルーチンを用いる．

ユーザ作成のプログラム (*User*)

ループアンローリング，ブロック化等の特別なチューニングを施さず，標準的アルゴリズムに従って作成する．配列のメモリアクセスだけは FORTRAN77 の仕様に合わせてある．また，BLAS, OpenMP の並列化指示文は使用しない．並列化にはコンパイラの自動並列化機能のみを用いる．

3 実験結果

数値実験は連立一次方程式，対称/非対称行列の固有値問題に対しておこなう．すべての問題において行列サイズ n は $n = 5000$ とする．OS の time コマンドを用いてプログラムの経過時間 (real) と各プロセッサの CPU 時間の合計 (user) を計測し，主に real の比較により性能を評価する．表 3〜5 中，real の単位は時:分:秒，user の単位は秒である．また，Speed-Up は 1CPU の real を 8CPU の real で割った値である．なお，実験に用いたプログラムのソースコードは [13] にて公開する．

3.1 連立一次方程式

本節では，部分軸選択を伴う LU 分解による連立一次方程式の直接解法について議論する．係数行列は区間 $[0,1)$ の一様乱数を成分に持つ密行列とする．*User* ではまず係数行列の LU 分解をおこない，前進消去後進代入により解を求める．LAPACK ルーチンは dgetrf,dgetrs を用いる．精度は絶対誤差の 2 乗ノルムが 10^{-12} 未満となるようにした．

表 3 に実験結果を示す．LU 分解後の求解部分の演算量 $[O(n^2)]$ は分解の演算量 $[O(n^3)]$ よりはるかに少なく，実質的に LU 分解に要する時間を反映している．VendorBLAS を用いた場合，1CPU の real は，*NAG*(0:01:25)，*Vendor*(0:05:31)，*Netlib*(0:07:15)，*User*(0:10:54) の順で性能が良い．この結果から *NAG* は *Vendor* よりも高度なチューニングが施されていることがわかる．8CPU の real も *NAG*(0:00:18) が最も速く，次いで *Vendor*(0:00:23)，*Netlib*(0:00:28)，*User*(0:03:21) の順となっている．*NAG* の測定には 1CPU と 8CPU とで同一のオブジェクトを用いていることから，LU 分解ルーチンに施されたチューニングは 8CPU の性能にも影響を与えていると考えられる．これらのことから，ルーチン本体の並列性のある部分に対して並列化をおこなう *NAG* のアプローチは，基盤となるルーチン (この場合 LU 分解のルーチン) のチューニングが重要であると考えられる．

NetlibBLAS を用いた場合においても 8CPU の real は *NAG*(0:01:11)，*Vendor*(0:01:15)，*Netlib*(0:01:15)，*User* の順となり，ライブラリ使用のプログラムが *User* に勝る．これは，LAPACK 互換のライブラリでは LU 分解ルーチンにブロック化アルゴリズム [1, 14] が採用されアルゴリズムの並列性が一層高められていることが主な要因である．OpenMP による並列化はルーチンのアルゴリズムに対しておこなわれるため，ブロック化アルゴリズムの採用により *NAG* は並列化されやすくなり，性能が改善されると考えられる．また，ブロック化アルゴリズムでは核演算にチューニング・並列化の効果が現れやすい Level 3 BLAS[7] が用いられている．そのため，主に演算処理 (BLAS) を並列化している *Vendor*，*Netlib* でも性能が改善される．

なお，*Vendor*，*Netlib* では Speed-Up が CPU 台数倍 (8 倍) 以上になる場合がある．主な原因は 1CPU，8CPU で使用するライブラリ・BLAS のオブジェクトが異なることである．特に SR8000 の並列処理では一次キャッシュが共有され容量が増えるため，ワークメモリの取り方やメモリアクセスの仕方が並列版，シングル版で異なる．

3.2 対称固有値問題

本節では，フランク行列を用いて実対称行列の全固有値・固有ベクトルの計算を評価する．n 次フランク行列の (i,j) 成分は

$$a_{ij} = n - \max(i,j) + 1, \quad i,j = 1,\cdots,n$$

である．*User* では，ハウスホルダー変換により行列を三重対角化し，二分法と逆反復法により全固有値・固有ベクトルを求める．LAPACK ルーチンは，Relatively Robust Representation[15] による dsyevr，分割統治法による dsyevd，QR 法による dsyev を用いる．*NAG* では dsyevd と同等のルーチンを用いる．これは，dsyevr,dsyev に相当する LAPACK 互換ルーチンが *NAG* に含まれていないためである．固有値の収束判定は相対残差が 10^{-12} 未満とした．

表 4 に実験結果を示す．VendorBLAS を用いた場合，1CPU の real は *NAG*(0:09:05)，dsyevr の *Vendor*(0:14:01)，dsyevd の *Vendor*(0:18:29)，dsyevr の *Netlib*(0:21:06)，dsyevd の *Netlib*(0:24:28)，*User*(0:27:18) の順に性能が良く，ライブラリ自体のチューニングは *NAG* が優れていることがわか

表 3: 連立一次方程式の解法の実行時間 (単位は real 時:分:秒, user 秒)．行列サイズは $n = 5000$.

Library	VendorBLAS			NetlibBLAS		
	1CPU	8CPU	Speed-Up	1CPU	8CPU	Speed-Up
NAG	real 0:01:25 user 670	real 0:00:18 user 140	4.72	real 0:08:07 user 3870	real 0:01:11 user 140	6.86
Vendor	real 0:05:31 user 180	real 0:00:23 user 176	14.39	real 0:08:25 user 503	real 0:01:15 user 592	6.73
Netlib	real 0:07:15 user 276	real 0:00:28 user 213	15.35	real 0:08:22 user 499	real 0:01:15 user 591	6.69
	Without BLAS					
User	real 0:10:54 user 653	real 0:03:21 user 1600	3.25			

る．しかし，8CPU の real は dsyevd を用いた *Vendor*(0:01:24)，dsyevr の *Vendor*(0:01:27)，dsyevd の *Netlib*(0:01:39)，dsyevr の *Netlib*(0:01:43)，*NAG*(0:02:21)，*User*(0:04:14) の順に性能が良い．これは，対称固有値問題の並列処理においては局所的な演算を高速に処理できる *Netlib*，*Vendor* のアプローチのほうが高い性能を発揮できることを意味する．また，dsyev を用いた *Vendor*，*Netlib* の real は 8CPU でそれぞれ 0:04:43，0:04:54 と，*User* の性能よりも劣るという結果を得た．これは dsyev の演算量が dsyevr，dsyevd よりも多く，アルゴリズムの逐次性が高いためである．このことから固有値問題では逐次性が低く，演算量の少ないアルゴリズムを用いることが重要であるといえる．

NetlibBLAS を用いた場合には，BLAS を用いる *NAG*，*Vendor*，*Netlib* の性能は大幅に低下し，8CPU の場合は dsyevr の *Netlib*(0:10:47) が最速，1CPU の場合は *NAG*(0:33:55) が最速となり，*User* よりも劣る．これは，BLAS の高速化が重要であることを示すと同時に，ブロック化アルゴリズムの採用等，アルゴリズム自体の改良がなければ BLAS を利用するライブラリの大幅な性能向上は期待できないことを示している．

3.3 非対称固有値問題

本節では，非対称行列の固有値問題について検討する．固有値は全て求めるが，固有ベクトルは求めない．非対称固有値問題の一例として，実験には

$$b_{ij} = \mathrm{mod}(n + j - i, n) + 1, \quad i, j = 1, \cdots, n$$

を (i, j) 成分とする巡回行列を用いた．*User* では，ピボット選択つきガウス消去法と同等の手順により係数行列をヘッセンベルグ行列へ変換し，シフト付き QR 法により全固有値を求める [16]．LAPACK ルーチンは dgeev を使用する．*NAG* には dgeev に相当するルーチンが実装されていないため，dgeev 内部で呼び出されるヘッセンベルグ行列への変換ルーチン (dgehrd 相当)，ヘッセンベルグ行列に対して QR 法により固有値を求めるルーチン (dhseqr 相当) を組み合わせて測定をおこなう．dgeev では，ヘッセンベルグ行列への変換にハウスホルダー変換を用いているため，変換には *User* の約 2 倍の演算量が必要となる [16]．*NAG* のヘッセンベルグ行列への変換ルーチン内でどのようなアルゴリズムが用いられているかは不明である．また，*NAG* では非対称行列の固有値問題に関するルーチンは並列化されていない [9]．固有値の収束判定は相対残差が 10^{-12} 未満とした．なお dgeev では，固有値の収束を速めるため，ヘッセンベルグ型に変換する前にバランシング [1] をおこなうが，実験に用いた巡回行列ではバランシングは効かない．

表 5 に実験結果を示す．*NAG* のルーチンは並列化されておらず，8CPU の並列処理時においても 1CPU と同じ処理をおこなうため，*NAG* の Speed-Up は 1.00 となっている．VendorBLAS を用いた場合，1CPU の real は *NAG*(0:24:31) が *User*(0:25:13) よりわずかに優れ，次いで *User*，*Netlib*(0:47:05)，*Vendor*(1:00:07) の順となる．この結果より，*NAG* は *User* と同等のアルゴリズムを採用していると推測できる．8CPU の real は，*User* が 0:08:16 と最も良く，*Vendor*(0:13:12) が *Netlib*(0:10:01) に劣るという結果を得た．これらの結果を見る限り，ヘッセンベルグ行列への変換に要する演算量の違いを考慮に入れても，非対称固有値問題においてはライブラリの使用により性能が改善されているとはいえない．また，連立一次方程式や対称固有値の結果と比較すると，非対称固有値問題での各プログラムの性能は著しく低く，並列化も大した効果を発揮できていない．このことから，非対称固有値問題では各プログラムとも計算機の特性を十分に活用しているとはいえず，性能改善にはアルゴリズムの改良もしくは新

表 4: 対称固有値問題の実行時間 (単位は real 時:分:秒, user 秒)．行列サイズは $n = 5000$.

Library	VendorBLAS			NetlibBLAS		
	1CPU	8CPU	Speed-Up	1CPU	8CPU	Speed-Up
NAG	real 0:09:05 user 4319	real 0:02:21 user1107	3.87	real 0:33:55 user 16205	real 0:11:41 user 5550	2.90
Vendor dsyevr	real 0:14:01 user 594	real 0:01:27 user 670	9.67	real 0:35:09 user 2097	real 0:11:20 user 5384	3.10
Vendor dsyevd	real 0:18:29 user 705	real 0:01:24 user 642	13.20	real 0:43:16 user 2583	real 0:12:04 user 5752	3.59
Vendor dsyev	real 0:27:26 user 1472	real 0:04:43 user 2237	5.82	real 0:45:31 user 2710	real 13:40 user 6507	3.33
Netlib dsyevr	real 0:21:06 user 961	real 0:01:43 user 788	12.29	real 0:34:06 user 2030	real 0:10:47 user 5130	3.16
Netlib dsyevd	real 0:24:28 user 1043	real 0:01:39 user 764	14.83	real 0:42:20 user 2521	real 0:11:32 user 5495	3.67
Netlib dsyev	real 0:31:08 user1687	real 0:04:54 user 2309	6.35	real 0:45:09 user 2688	real 0:13:11 user 6275	3.42
	Without BLAS					
User	real 0:27:18 user 1631	real 0:04:14 user 2016	6.45			

しいアルゴリズムの開発が不可欠である．

また，NetlibBLAS を用いた場合の real は，8CPU の場合 *Vendor* は 0:22:41，*Netlib* は 0:20:02，1CPU の場合 *NAG* は 0:48:35，*Vendor* は 1:24:47，*Netlib* は 1:12:39 となる．これは，VendorBLAS を用いても最大で 2 倍程度しか性能が改善されないことを意味する．連立一次方程式や対称固有値問題の場合は VendorBLAS を用いると最大約 8 倍，平均で約 3 倍性能が向上することを考慮に入れると，非対称固有値問題では BLAS のチューニングだけでは大幅な性能向上は期待できないといえる．

4 まとめ

本稿では SR8000 1 ノード上において，主に BLAS 内部等の局所的な演算処理を並列化するベンダ提供の LAPACK，Netlib 上の LAPACK，ライブラリルーチン本体の並列性のある部分を OpenMP を用いて並列化する NAG Fortran SMP ライブラリ，および自動並列化機能を利用したユーザ作成プログラムについて性能比較をおこない，各ライブラリで用いられている並列化手法の特徴・問題点を明らかにした．数値実験の結果より，NAG Fortran SMP ライブラリのように BLAS の並列化に頼らず OpenMP によりルーチン本体の並列性のある部分を重点的に並列化する方法は，連立一次方程式の場合のようにルーチン本体が十分にチューニングされ適度な並列性を備えているようなアルゴリズムに対しては計算機の性能を十分に引き出せる最適な方法であるが，固有値問題のように並列性が低いアルゴリズムでは最適な方法とはいえないことが明らかとなった．また，ベンダ提供の LAPACK のように局所的な演算処理 (BLAS) を重点的に並列化する方法は，アルゴリズムの並列性が低い場合でも十分な効果を得られるが，性能の大部分が演算処理すなわち BLAS のチューニングによるものであるため，アルゴリズムを改良しなければ大幅な性能改善が期待できないことが明らかとなった．これらのことから，今回比較した 2 種類の並列化手法は大きな性能差があるわけではなく，状況に応じて使い分けることで計算機を最大限に活用することが重要であるといえる．非対称固有値問題の結果は，ライブラリの使用だけではプログラムの性能が改善されない一例といえる．

本研究で得た結論を踏まえると，性能の面で今後のライブラリに期待されることは，並列性の高いアルゴリズムを採用し常に並列計算機の特性を活用できる並列化手法を備えていること，どのような問題に対しても高速に解を得られる性能を提供できることであるといえる．

参考文献

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J. J., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney,

表 5: 非対称固有値問題の実行時間 (単位は real 時:分:秒, user 秒). 行列サイズは $n = 5000$.

Library	VendorBLAS			NetlibBLAS		
	1CPU	8CPU	Speed-Up	1CPU	8CPU	Speed-Up
NAG	real 0:24:31 user 1464	real 0:24:33 user 1464	1.00	real 0:48:35 user 2903	real 0:48:43 user 2903	1.00
Vendor	real 1:00:07 user 3598	real 0:13:12 user 6271	4.55	real 1:24:47 user 5074	real 0:22:41 user 10830	3.74
Netlib	real 0:47:05 user 2817	real 0:10:01 user 4765	4.70	real 1:12:39 user 4347	real 0:20:02 user 9566	3.63
	Without BLAS					
User	real 0:25:13 user 1507	real 0:08:16 user 3942	3.05			

A. and Sorensen, D.: *LAPACK Users' Guide (3rd Ed.)*, SIAM (2000).

[2] 山本喜一, 榊原進, 野寺隆志, 長谷川秀彦: これだけは知っておきたい数学ツール, 共立出版 (1999).

[3] 小国力, 村田健郎, 三好俊郎, Dongarra, J. J., 長谷川秀彦: 行列計算ソフトウエア, 丸善 (1991).

[4] Dongarra, J. J., Duff, I. S., Sorensen, S. C. and van der Vorst, H. A.: *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM (1991).

[5] Lawson, C., Hanson, R., Kincaid, D. and Krogh, F.: Basic Linear Algebra Subprograms for FORTRAN Usage, *ACM Trans. Math. Software*, Vol. 5, pp. 308–325 (1979).

[6] Dongarra, J. J., DuCroz, J., Hammarling, S. and Hanson, R.: An Extended Set of FORTRAN Basic Linear Algebra Subprograms, *ACM Trans. Math. Software*, Vol. 14, pp. 1–17 (1988).

[7] Dongarra, J. J., DuCroz, J., Duff, I. S. and Hammarling, S.: A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Software*, Vol. 16, pp. 1–17 (1990).

[8] OpenMP Architecture Review Board: `http://www.OpenMP.org/`.

[9] NAG Fortran SMP Library Documentation: `http://www.nag.co.uk/numeric/FL/manual/html/FSlibrarymanual.asp` .

[10] Nishiyama, H., Motokawa, K., Kyushima, I. and Kikuchi, S.: Pseudo-vectorizing Compiler for the SR8000, *Proc. Euro-Par 2000, LNCS 1900*, Springer-Verlag, pp. 1023–1027 (2000).

[11] Brehm, M., Bader, R., Heller, H. and Ebner, R.: Pseudovectorization, SMP, and Messege Passing on the Hitachi SR8000-F1, *Proc. Euro-Par 2000, LNCS 1900*, Springer-Verlag, pp. 1351–1361 (2000).

[12] Netlib: `http://www.netlib.org/`.

[13] プログラムのソースコード: `http://www.me.ics.saitama-u.ac.jp/~sigehara/hps/hps.html`.

[14] Dongarra, J. J., Duff, I. S., Sorensen, D. C. and van der Vorst, H. A.: *Numerical Linear Algebra for High-Performance Computers*, SIAM (1998).

[15] Dhillon, I. S.: *A New $O(n^2)$ Algorithm for Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*, PhD Thesis, University of California at Berkeley (1997).

[16] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P.: *Numerical Recipes in FORTRAN Second Edition*, Cambridge University Press (1992).