情報処理学会論文誌　コンピューティングシステム　Vol.8　No.3　1–29 (July 2015)

実対称行列のブロック鏡映変換を用いたブロック三重対角化のOpenMPによる並列化の実験

村上 弘[1,a)]

受付日 2015年1月5日, 採録日 2015年4月25日

概要：ブロック鏡映変換を用いた実対称行列のブロック三重対角形への変換処理の，OpenMP による並列化実装の計算機実験を行った．計算の大部分はブロック小行列（タイル）間の行列積（BLAS3 の xGEMM）になるので，メモリバンド幅を抑えて演算密度が高い計算を行える．きわめて単純な実装でも，普及品のマルチコア CPU を用いた PC 上で演算性能の理論ピーク値の 70%から 80%程度の性能が達成できることを確かめた．

キーワード：ブロックアルゴリズム，ブロック三重対角化，ブロック鏡映変換

Experiments of OpenMP Parallelization of the Block Tridiagonalization of a Real Symmetric Matrix by the Block Reflector Method

Hiroshi Murakami[1,a)]

Received: January 5, 2015, Accepted: April 25, 2015

Abstract: We made experiments of OpenMP parallelization of the block tridiagonalization of a real symmetric matrix by the block reflector method. Since the most part of the calculation is reduced to matrix multiplications (BLAS3's xGEMM) of small matrices (tiles) of the block size, the calculation is rich in arithmetic operations and the requirement of the memory bandwidth is low. By experiments, we examined that even by a very simple implementation of the method, PCs with a multi-core CPU for commodities attained about 70% to 80% of their theoretical peak performances for the calculations.

Keywords: blocked algorithm, tiled algorithm, block tridiagonalization, block reflector

1. はじめに

大規模な（数万元～）実対称密行列の固有値問題を通常の Householder 法を用いて三重対角形を経由して解く方法は BLAS2 であり，記憶走査量が多いことからメモリバンド幅が計算のボトルネックとなる．そこで普通の行列算法に現れる行列要素を通常の数（スカラ）から小行列（タイル）に拡張するブロック化を施せば，計算の粒度を大きくでき，さらに小行列（タイル）どうしの積の計算を BLAS3 の xGEMM を用いて行うことで，計算のボトルネックとなりがちなメモリバンド幅への要求が低減できる．

通常の鏡映変換による実対称行列の Householder 三重対角化法に対してブロック化拡張を施すことで，ブロック鏡映変換によるブロック Householder 三重対角化法が自然に得られる．

いま実対称行列の次数を N とし，ブロック Householder 三重対角化法の計算を小行列（タイル）のサイズを b として行うと，記憶の走査回数は b に反比例して減り，演算量の主要項が $(4/3)N^3$ なので，演算密度は b に比例して向上する．

対称密行列の固有値問題のブロック三重対角化を経由する解法は，通常の三重対角化を経由する方法に類似したも

[1] 首都大学東京数理情報科学専攻
Department of Mathematics and Information Sciences, Tokyo Metropolitan University, Hachioji, Tokyo 192–0397, Japan
[a)] mrkmhrsh@tmu.ac.jp

のになる．1) まず対称密行列 A にブロック鏡映変換を繰り返し適用してブロック三重対角行列 T に変換する．この変換後の T は元の A と固有値の全体が一致する．2) 次に T の固有対を求める．通常は必要な固有値を持つ固有対だけを求める．3) そうして必要な T の固有ベクトルの組に対してブロック三重対角化する際に用いたブロック鏡映変換を逆順に繰り返して作用させることで，元の行列 A の固有ベクトルの組を得る．

全体行列 A はサイズ b の小行列（タイル）に区分けして扱い，タイル内部に対応する記憶領域はメモリ上で連続的に確保するものとする．A の次数 N がブロックサイズ b の倍数でないときは，行と列の各方向の末尾に行あるいは列の個数が b 未満のタイルを割り当てるようにもできるが，今回の実験では記憶量が少し無駄になるが，実装を簡単にするためにすべて同じサイズ b の正方形のタイルだけを用いて，A の末尾の行あるいは列を含むタイルはその一部分だけを使用した．

今回のブロック三重対角化法の実験では特に，以前の論文 [1] で記述したブロック鏡映変換の構成の中で用いていた QR 分解を通常の HQR 法から T-S 行列用の QR 分解（文献 [2], [3]）に置き換える改良を行った．そうして，以前の論文の書かれた 2006 年頃のものと比べて演算性能とメモリ容量が大幅に向上している最近の計算機（PC）を用いて性能測定を行った．

過去の論文 [1] では，

(1) 元の対称密行列 A にブロック鏡映変換を繰り返し適用してブロック三重対角行列 T にする，
(2) 対称三重ブロック対角行列 T を対称帯行列 B に変換して帯幅を半減させる，
(3) 村田の帯 Householder 法を用いて帯行列 B を真の対称三重対角行列へ変換する，
(4) Sturm の二分法を用いて真の対称三重対角行列の必要な範囲（たとえば固有値分布の端付近）にある比較的少数の固有値を求める，
(5) 帯行列 B の比較的少数の固有ベクトルを逆反復法のシフトに固有値を用いることで求める，
(6) 帯行列 B の固有ベクトルを三重ブロック対角行列 T の固有ベクトルに逆変換する，
(7) 三重ブロック対角行列 T の各固有ベクトルに対して，最初のブロック三重対角化に用いたブロック鏡映変換を逆順に適用することで，元の行列 A の固有ベクトルを求める，

という構成により対称密行列の比較的少数の固有対を求める方法，およびその実行例を記述した．

このうち，上記 (2) のステップは文献 [1] に記述した方法を用いて容易に実現できる．

また上記 (3) から (5) までの村田の帯 Householder 法を用いて（帯内が密の）対称帯行列の比較的少数の固有対を求めるステップは文献 [4] の方法を，同書に掲載されたプログラムに修正を加えて計算をしている．

古典的アルゴリズムにより大規模な対称帯行列の固有対の一部を求める方法は文献 [5], [6] にも記述されている．

今回の論文は主に，上記 (1) のステップであるブロック三重対角化の操作だけについて記述する．計算で求める固有対の個数が比較的少数である場合には，このステップが計算全体の主要部になる．

大規模な対称行列のブロック三重対角化法には，本論文で紹介するブロック鏡映変換に基づくもの以外にも，通常の Householder 変換の複数個を集積して作られる WY 表現を用いてブロック化を実現する Bischof の方法などがある [7], [8], [9]．

本論文の新規性：

本論文では以前の論文 [1] で記述したブロック鏡映変換の構成の中で用いていた QR 分解を，通常の HQR 法から T-S 行列用の QR 分解（文献 [2], [3]）に置き換える改良を行った．この点において新規性がある．さらに本論文の後の付録の中で，T-S 行列の直交分解として QR 分解以外のものも利用できることを実験で示した．

2. ブロック三重対角化法

この章の内容は過去の論文 [1] の復習であり，QR 分解と特異値分解を用いてブロック鏡映変換を構成する．

2.1 ブロック鏡映変換の構成法

いま縦長の $m\times b$ 行列 U が $U^TU = I_b$ を満たせば，$H = I_m - 2UU^T$ は，$H^T = H$，$HH = I_m$ を満たすので，H は（U を軸とする）ブロック版の鏡映変換になる．

いま任意に与えられた縦長の $m\times b$ 行列 C に対して U をうまく決めることで HC の先頭の b 行以外をすべて消去できれば，その U を利用してブロック三重対角化ができる．つまり $HC = E_b\beta$ である．ここで E_b は $m\times b$ の拡張単位行列で，β は b 次行列である．そのような U は以下の手順で構成できる（**図 1**，**図 2** を参照）．

(1) $m\times b$ 行列 C の QR 分解を $C \Rightarrow XR$ とする．
$m\times b$ 行列 X の先頭の b 行を集めた b 次行列を $\widehat{X}$ とする．
(2) $\widehat{X}$ の特異値分解を $\widehat{X} \Rightarrow WDV$ とする（この特異値はすべて 1 以下である）．
(3) X の先頭の b 行までが $\widehat{X}$ であったが，X のその位置に b 次行列 WV を加えた $m\times b$ 行列を Y とする．すなわち $Y \Leftarrow X + E_b(WV)$．
(4) $G = V^T\{2(I_b+D)\}^{(-1/2)}$ とし，Y に G を右から乗じた $U \Leftarrow YG$ を作ると，U は $U^TU = I_b$ を満たす．この U が求めるべきブロック鏡映変換の軸になる．
(5) $H = I_m - 2UU^T$ を C に作用させると，先頭の b 行以外がすべて消去されて，$HC = E_b\beta$ となる．ここ

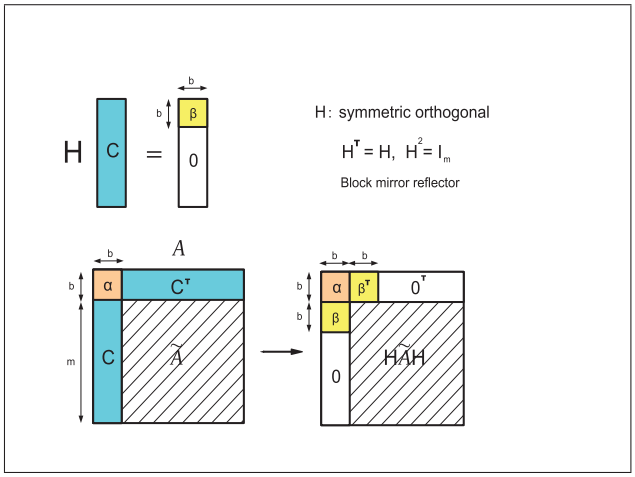


図 1　ブロック両側変換について
Fig. 1　Block two sided transformation.

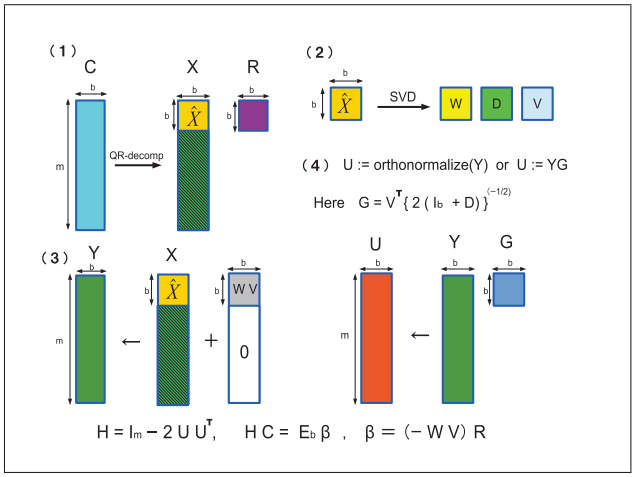


図 2　ブロック鏡映変換の構成法
Fig. 2　Construction of block reflector.

で b 次行列 $\beta = -(WV)\,R$ である．その構成から H は対称な直交行列である．

（注：今回の実験では，C の QR 分解において列ピボット交換も有効階数の決定も行わない上記の構成法を用いた．列ピボット交換や有効階数の決定を行う QR 分解を用いる場合にも同様の構成が可能であるが，省略する．）

（注 2：上記 (1) では「C の QR 分解を $C \Rightarrow XR$ とする」とあるが，実際には分解 $C = XR$ において X が列直交基底であること（$X^TX = I_b$）だけを満たせばブロック鏡映変換を実現するのには十分であることが示せる．つまり行列 R を上三角に限定する必要はないことが分かる．そのことを本論文の後の付録の章でも用いている．）

すると，上記の方法で構成されるブロック鏡映変換を繰り返し適用することで，以下のように任意の実対称行列 A は実対称のブロック三重対角行列へ変換される．

$$H^{(n-2)}\cdots H^{(3)}\,H^{(2)}\,H^{(1)}\,A\,H^{(1)}\,H^{(2)}\,H^{(3)}\cdots H^{(n-2)}$$

$$= \begin{pmatrix} \alpha_1 & \beta_1^T & 0 & \cdots & & & \\ \beta_1 & \alpha_2 & \beta_2^T & 0 & \cdots & & \\ 0 & \beta_2 & \alpha_3 & \beta_3^T & 0 & & \\ \vdots & 0 & \beta_3 & \alpha_4 & \beta_4^T & & \\ & \vdots & & & & & \\ & & 0 & \ddots & \ddots & & \vdots \\ & & & & \ddots & \ddots & 0 \\ & & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1}^T \\ & & & \cdots & 0 & \beta_{n-1} & \alpha_n \end{pmatrix}$$

ただし，N 次行列 A を行と列をサイズ b でブロック分割したときの（ただし最後のブロックのサイズだけは b 以下とする），各方向のブロックの番号は 1 から n の範囲であるとする．そうして第 k 番以下のブロックに含まれる行と列については恒等変換，第 k 番よりも後のブロックに含まれる行と列については上記の方法で第 k 回目に構成されるサイズ $m = N - kb$ のブロック鏡映変換 H，これら 2 つの直和である直交変換を $H^{(k)}$ とする．

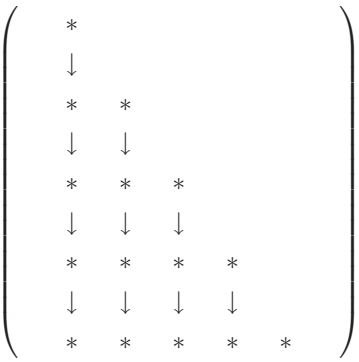
2.2　$H^{(1)}AH^{(1)}$ の実際の計算方法

先頭のブロック列についてはその変換結果を新たに計算して求める必要はなくて，先頭のブロック列の最初のブロックを α として保持する，そうしてブロック鏡映軸 U を作るときに得られた β も保存する．軸 U は後で逆変換に用いるために C の部分に重ね書きして保持する．

いま，A から最初の 1 ブロック分だけ行と列を取り除いたものを $\widetilde{A}$ と表すと，その部分を両側からの鏡映変換で $\widetilde{A} \leftarrow H\widetilde{A}H$ と更新する方法は，補助のブロックベクトル P および対称な小行列 γ を用いて以下の 4 つの式で書ける；

$$\begin{cases} P \leftarrow \widetilde{A}U, & \text{(Kernel 1)} \\ \gamma \leftarrow P^TU, & \\ P \leftarrow 2(U\,\gamma - P), & \\ \widetilde{A} \leftarrow \widetilde{A} + U\,P^T + P\,U^T. & \text{(Kernel 2)} \end{cases}$$

この計算はサイズ b の小行列（タイル）どうしの行列積を豊富に含むので，BLAS3 を用いて実装すれば高速であり記憶参照の局所性も高い．実装では，ブロック行列 A はその対称性を用いることで，対角ブロックを含む下半分だけを（たとえばブロック列ベクトルが並んだ形で）保持する．このブロック鏡映変換を用いた実対称行列のブロック三重対角化法の擬似コードをリスト 1 に示す．各段の両側ブロック鏡映変換において，ブロック行列 A の対角ブロックを含む下三角部分は，上記の「ブロック鏡映を作り，カーネル 1 を処理する」ステップと「カーネル 2 を処理する」ステップのそれぞれで，（並列化をしない場合には）列方向に連続的なアクセス（下図参照）による走査を 2 回受ける．



行列要素がスカラーの場合の「Wilkinson の技巧」[10] を行列要素がタイルであるブロック行列の場合にも同様にして適用することが可能である．それはブロック鏡映変換の第 k 段目での「カーネル 2 を処理する」ステップを実施する際に，同時に第 $k+1$ 段目での「ブロック鏡映を構成し，カーネル 1 を処理する」ステップを重ねながら実施するように計算処理の手順を組み替えることであり，本来個別に行えば 2 回必要になるブロック行列 A の走査回数をまとめることで 1 回に減らす方法である．

リスト 1　ブロック版対称 Householder 三重対角化の擬似コード
List 1　Pseudo code of blocked symmetric Householder tridiagonalization.

```
! NB は行あるいは列のブロック数で，IB，JB などはブロックの添字．
! 行列やベクトルのブロック添字は 1 から NB の範囲をとる．
! 鏡映変換の段数 KB のループの内側には 2 つの計算カーネルがある
DO  KB = 1, NB-2
  「A[KB+1:NB,KB] からブロック鏡映軸 U[KB+1:NB] を作り重ね書きし，
    同時にできる BETA[KB] を格納．ALP[KB]（=A[KB,KB]）も格納．」
  !------------- カーネル 1（ブロック版の行列ベクトル積）
  ! P ← A * U
  P[KB+1:NB] = 0.0
  DO  JB = KB+1, NB
    P[JB] = P[JB] + A[JB,JB] * U[JB]
    DO  IB = JB+1, NB
      P[IB] = P[IB] + A[IB,JB] * U[JB]
      P[JB] = P[JB] + A[IB,JB] * U[IB]
    ENDDO
  ENDDO
  !-------------
  「ブロックベクトル U と P の内積である対称行列 γ を作り，
    P := 2 (U γ - P) を γ の対称性も利用して作る．」
  !------------- カーネル 2（ブロック版の A の階数 2 の更新）
  ! A ← A + P * U^{T} + U * P^{T}
  DO  JB = KB+1, NB
    DO  IB = JB, N
      A[IB,JB] = A[IB,JB] + P[IB] * U[JB] + U[IB] * P[JB]
    ENDDO
  ENDDO
  !-------------
ENDDO
「ALP[NB-1]（=A[NB-1,NB-1]）と BET[NB-1]（=A[NB,NB-1]）と
  ALP[NB]（=A[NB,NB]）を残った A の要素から回収して格納．」
```

2.3　固有ベクトルの逆変換［BCKTRS］

この操作もまた通常の Householder 法と同様にできる．

- 必要な T の固有ベクトルを求めて，それらを逆変換により必要な A の固有ベクトルに戻す．
- T の固有ベクトル $\mathbf{y}$ を A の固有ベクトル $\mathbf{x}$ へ逆変換するには，$\mathbf{y}$ に軸が $U^{(i)}$ のブロック鏡映変換 $H^{(i)}$ を逆順に作用させる：$\mathbf{x} \leftarrow H^{(1)} H^{(2)} \cdots H^{(n-2)} \mathbf{y}$．
- 効率向上のためには，複数の固有ベクトルをまとめて変換する．ベクトルの組 Y に対するブロック鏡映変換 $H^{(i)}$ の適用は，$Y \leftarrow H^{(i)}Y$ の形にまとめることができて BLAS3 を適用できる行列–行列積が豊富になる．ブロック鏡映変換を逆順に適用することにより，T の固有ベクトルの組から対応する A の固有ベクトルの組が得られる．

3.　T-S 型行列用の QR 分解法（TS-QR）について

ブロック鏡映変換の構成には，T-S 型である $m \times b$ 行列（$m \gg b$）の QR 分解を必要とする．この QR 分解に BLAS2 である通常の HQR 法を用いると記憶参照の局所性がないので，メモリバンド幅と並列処理の同期が性能上のボトルネックとなる．ブロック鏡映変換の軸である U を構成する手順の中で BLAS2 である通常の HQR 法を用いると，サイズ b が $\sqrt{m}$ の付近もしくはそれ以上のときにブロック鏡映変換の処理全体の性能が低下する．その理由は，各段のブロック鏡映変換の中で，通常の HQR 法が費やす記憶走査量 $(3/2)mb^2$ が，ブロック鏡映変換の記憶走査量 $(1/2)m^2$ と同程度になるからである．それゆえ，b がある程度大きいときには，ブロック鏡映変換の構成に用いる T-S 型行列用の QR 分解には，通常の HQR 法よりも記憶走査量が少ない別の方法を用いるべきである．

今回の実験に用いた T-S 行列に対する QR 分解（TS-QR）は，Householder 直交変換全体の過程を再帰的に多段階で行うもので，以下のような利点がある：計算の大部分が各部分行列に対する大きい粒度の互いに独立に行える処理で構成されていて並列化に適していること，またデータ参照の局所性が向上するので階層型記憶システムに適しており，また PE 間のデータ転送と同期の頻度が少ないことも並列処理に適している．方法は文献 [11] に記述した．

ただし，今回の実験に用いた実装は，タイルを並べて構成されたブロック列ベクトルの T-S 型 QR 分解をその記憶場所の中で行うのではなくて，それに対応するタイル化されていない通常の縦長の作業用の行列にいったんコピーして，その作業用行列に T-S 行列の QR 分解（TS-QR）を適用し，得られた分解結果の Q を元の対応するブロック列ベクトルにコピーして戻している．このためブロック列ベクトルの全体と作業用領域の間を余分に 2 度走査しながら複製する手間が発生しているし，算法がタイルだけを用いて

組み立てられていないこともプログラム構成上の一貫性を欠いていて美しくない．これは今後改善をしたい．

3.1　TS-QR 法以外の QR 分解法について

T-S 型行列 A の QR 分解を行う方法としては，再帰的な方法で階層的な QR 分解を行う TS-QR 法以外にも，まず A からサイズの小さい対称行列 $S \leftarrow A^TA$ を作り，Cholesky 分解 $S \rightarrow R^TR$ を行って $Q \leftarrow AR^{-1}$ を作ると，$Q^TQ = I$, $A = QR$ であることを用いる Cholesky QR 法などがある（A が悪条件である場合には Cholesky 分解には対角ピボット選択を含める方が良い）．これについてたとえば文献 [12] の第 2 章 5 節には次のように述べられている．

> 「まとめると，TS-QR 法は数値的には Householder QR 法と同程度に正確であるが，Householder QR 法よりも通信量が漸近的に少ない．また精度がはるかに低い Cholesky QR 法と比べて TS-QR 法の通信量はある小さな定数倍多いだけである．」
>
> (原文：“In summary, TSQR is as numerically accurate as Householer QR but communicates asymptotically less than Householder QR. TSQR also communicates only a small constant factor more than the much less accurate Cholesky QR algorithm.”)

あるいは，文献 [13] の第 9 章の最初のあたりには次のように述べられている（ただし行列 A は $m \times n$ で縦長としている）．

> 「並列の場合，Cholesky QR 法と TS-QR 法の通信回数と通信量は同程度であるが，Cholesky QR 法のクリチカルパスにともなう浮動小数演算回数はある定数倍少ない．しかし TS-QR 法で計算した因子 Q はつねに数値的に直交しているが，他方で Cholesky QR 法で計算した因子 Q は $\kappa_2(A)^2$ に比例して直交性が失なわれる．グラム・シュミット系統の方法ではこれらの 2 つの方法よりも通信回数は少なくとも n 倍は必要で，少なくとも $\kappa_2(A)$ に比例して直交性が失なわれる．」
>
> (原文：“In the parallel case, Cholesky QR and TSQR have comparable number of messages and communicate comparable numbers of words, but Cholesky QR requires a constant factor fewer flops along the critical path. However, the Q factor computed by TSQR is always numerically orthogonal, whereas the Q factor computed by Cholesky QR loses orthogonality proportionally to $\kappa_2(A)^2$. The variants of Gram-Schmidt require at best a factor n more messages than these two algorithms, and lose orthogonality at best proportionally to $\kappa_2(A)$.”)

このように，Cholesky QR 法あるいはグラム・シュミット系統の方法の並列化版の直交分解で得られる基底 Q の直交性は，分解する行列の条件数が大きいといくらでも悪くなりうる．するとそれにより構成されたブロック鏡映変換の直交変換としての正確さは保証されない．そこで直交精度の点からは Householder QR 法の並列化には TS-QR 法を選択するのが安全で確実であると考えられる．

なお，Cholesky QR 法やグラム・シュミットの系統の方法で得られる分解の直交性の崩れを修正する反復手法が存在する（たとえば修正を 1 回加える方法を Cholesky $QR2$ などと呼ぶようである）．さらに文献 [14] で SVQB 法という直交基底生成の方法，また SVQB 法と同類の方法を実験した文献 [15] では，Cholesky QR 法に似た方法として縦長の行列 A からまずサイズの小さい対称行列 $S := A^TA$ を BLAS3 の演算を用いて作り，その S の（Cholesky 分解ではなくて）固有値分解を用いて A の直交基底を得る方法が記述されている（T-S 型の行列 A から $S := A^TA$ を構成する計算は，A がタイルを要素とするブロックベクトルであれば，タイル単位の BLAS3 演算を用いて容易に組み立てられる）．

Cholesky QR 法や SVQB 法は Q の直交性の崩れを必要に応じて後から反復処理により補正・修正することができる．しかし，A がきわめて悪条件の場合には，正確な直交性が 1 回の補正だけでは得られずに 2 回以上必要になる場合もある．補正を行う反復回数が増えると記憶の参照量も増えてしまう．通常よりも高精度の数値による演算（たとえば通常用いられている倍精度演算に対する四倍精度演算など）を中間の作業に用いて計算すれば，修正のための反復が省けたり減らしたりできるが，利用ができる場合にも高精度演算はきわめて遅いことが普通なので，実際に利用するうえで困難が予想される．

今回の実験では，とりあえず TS-QR 分解法だけを用いており，Cholesky QR 法に直交性の補正処理を（適応的に）入れた方法やあるいは SVQB 法で基底の直交性の補正も入れる方法などは，採用していない（本論文の付録の章「TS-QR 法以外の直交分解を使った例」の中に，SVQB 法と同類の方法 [15] を用いて行った実験についての記述を追加しておいた）．

4.　ソースコードの例

実験で使ったブロック三重対角化の計算に用いた Fortran90 言語と OpenMP ディレクティブによるソースコードの例を示す（リスト 2 からリスト 6）．

リスト 2 の BHSHLD は，ブロック Householder 三重対角化のルーチンである．行列 A はブロックサイズ b の正方形タイルを並べたものとして対角およびその下半分と対応するタイルだけを保持する．行列 A の次数 N がタイルのサ

イズの倍数でない場合には，A の最後の行あるいは列を含むタイルの内部は A の要素と対応する部分だけが意味を持つ．タイル内部はメモリ上で連続領域にとっている．今回は簡単のため Fortran90 の 3 次元配列を用いていて，配列の第 1 番目の添字と第 2 番目の添字をタイル内部の行列要素をアクセスする添字とし，第 3 番目の添字は行列 A の下半分部分に対応するタイルに列優先順で付けた番号としている．

リスト 3 の MAKE_REFLECT は，ブロック鏡映変換の軸を作成するルーチンとして BHSHLD から呼ばれる．

リスト 4 のルーチン HQR_BLOCK_TS は MAKE_REFLECT から呼ばれ，行列 A のブロック列を TS-QR 法により QR 分解を行う．このルーチンの現状の実装は「とりあえず」のものであり，まずタイルからなるブロック列ベクトルを「通常の格納方式」の配列に複製し，「通常の格納方式」の配列に対する TS-QR 法のルーチン（リスト 5 の HQRF_TS）を用いて QR 分解し，得られた正規直交ベクトルの組 Q をまた再びタイルからなるブロック列ベクトルに複製して書き戻している．

リスト 6 の MAT_SVD は，MAKE_REFLECT の中で使用するルーチンで，タイルサイズの行列の特異値分解を行うものである．名古屋大学数値ライブラリ NUMPAC のルーチン SVDD，あるいは LAPACK のルーチン DGESVD を呼んで処理している．

リスト 2　ブロック三重対角化：BHSHLD
List 2　Block tridiagonalization: BHSHLD.

```
001  !=====================================================================
002  !
003  !    Block Householder Transformation routine.
004  !
005  !=====================================================================
006  SUBROUTINE BHSHLD (N, BLKSZ, NBLK, A, P, R, RANKS)
007  IMPLICIT NONE
008  INTEGER,INTENT(IN) :: N ! The true size of matrix "A".
009  INTEGER,INTENT(IN) :: BLKSZ, NBLK
010  REAL(8),INTENT(INOUT) :: A(BLKSZ,BLKSZ,(NBLK+1)*NBLK/2)
011       ! Matrix "A" will keep the block Householder vectors.
012  REAL(8),INTENT(OUT) :: P(BLKSZ,BLKSZ,NBLK) ! Stores ALP(:,:,:).
013  REAL(8),INTENT(OUT) :: R(BLKSZ,BLKSZ,NBLK) ! Stores BET(:,:,:).
014  INTEGER,INTENT(OUT) :: RANKS(NBLK-2) ! Stores ranks of transformations.
015  !---------------------------------------------------------------------
016  EXTERNAL DSYMM, DGEMM, DSYR2K ! Level-3 BLAS routines.
017  INTEGER BS(NBLK) ! True sizes of the blocks.
018  INTEGER LAST_BS
019  INTEGER KK, RANK
020  REAL(8) R_JB(BLKSZ,BLKSZ), C(BLKSZ,BLKSZ), C_PART(BLKSZ,BLKSZ)
021  INTEGER IB, JB, INDX;  INDX(IB,JB) = IB+(JB-1)*(2*NBLK-JB)/2
022   ! Statement function "INDX" gives the index to (IB,JB)-th block element of
023   ! the symmetric block matrix stored using the symmetry. IB>=JB is assumed.
024  !---------------------------------------------------------------------
025    BS(:NBLK-1) = BLKSZ; BS(NBLK) = N-(NBLK-1)*BLKSZ ! Set block sizes.
026    LAST_BS = BS(NBLK)
027    !---
028    LOOP_KK: DO  KK = 1, NBLK-2
029      !----
030      ! Save ALP(KK).
031      P(:BS(KK),:BS(KK),KK) = A(:BS(KK),:BS(KK),INDX(KK,KK))
032      !----
033      ! Make the block reflector vector into KK-th block column of "A"
034      ! and also BETA(KK).
035      CALL MAKE_REFLECT (KK, BLKSZ, NBLK, A(1,1,INDX(1,KK)), RANK, R(1,1,KK))
036      ! To be secure, unused place of the block vector is zero-filled.
037      A(LAST_BS+1:,:,INDX(NBLK,KK)) = 0.D0
038      !----
039      ! Save the rank of transformation for the later use.
040      RANKS(KK) = RANK
041      !----------
042      ! Skip the transform when the rank of block reflector is nil.
043      IF (RANK == 0) CYCLE LOOP_KK
044      !-------------------------------------------------
045      ! Compute, "R := A * Q"; where "Q" is in the KK-th block column of "A".
046  !$OMP PARALLEL PRIVATE(R_JB,JB,C_PART)
047  !$OMP DO
048      DO  JB = KK+1, NBLK
049        R(:BS(JB),:RANK,JB) = 0.D0
050      ENDDO
051      DO  JB = KK+1, NBLK ! This DO-loop is sequential
052        R_JB(:BS(JB),:RANK) = 0.D0
053  !$OMP DO
054      DO  IB = JB, NBLK
055        IF (IB == JB) THEN ! Diagonal block.
056          CALL DSYMM ('L', 'L', BS(JB), RANK, 1.D0, &
057               A(1,1,INDX(IB,JB)), SIZE(A,1), &
058               A(1,1,INDX(JB,KK)), SIZE(A,1), &
059               1.D0, R_JB, SIZE(R_JB,1))
060        ELSE ! Off-diagonal block (IB > JB).
061          CALL DGEMM ('N', 'N', BS(IB), RANK, BS(JB), 1.D0, &
062               A(1,1,INDX(IB,JB)), SIZE(A,1), &
063               A(1,1,INDX(JB,KK)), SIZE(A,1), &
064               1.D0, R(1,1,IB), SIZE(R,1))
065          CALL DGEMM ('T', 'N', BS(JB), RANK, BS(IB), 1.D0, &
066               A(1,1,INDX(IB,JB)), SIZE(A,1), &
067               A(1,1,INDX(IB,KK)), SIZE(A,1), &
068               1.D0, R_JB, SIZE(R_JB,1))
069        ENDIF
070      ENDDO
071  !$OMP END DO NOWAIT
072  !$OMP CRITICAL
073      R(:BS(JB),:RANK,JB) = R(:BS(JB),:RANK,JB) + R_JB(:BS(JB),:RANK)
074  !$OMP END CRITICAL
075  !$! Barrier required if NOWAIT is used for the OMP-DO above.
076  !$OMP BARRIER
077      ENDDO
078      !-------------------------------------------------
079      ! Compute, "C := (Q^{T}*R)"; Note,"C" is symmetric since "C=Q^{T}AQ".
080  !$OMP SINGLE
081      C(:RANK,:RANK) = 0.D0
082  !$OMP END SINGLE
083  !$! Every thread sets "C_PART" to zero.
084      C_PART(:RANK,:RANK) = 0.D0
085  !$OMP DO
086      DO  JB = KK+1, NBLK
087        CALL DGEMM ('T', 'N', RANK, RANK,BS(JB), 1.D0, &
088                    A(1,1,INDX(JB,KK)), SIZE(A,1), &
089                    R(1,1,JB), SIZE(R,1),          &
090                    1.D0, C_PART, SIZE(C_PART,1))
091      ENDDO
092  !$OMP END DO NOWAIT
093  !$OMP CRITICAL
094      C(:RANK,:RANK) = C(:RANK,:RANK) + C_PART(:RANK,:RANK)
095  !$OMP END CRITICAL
096  !$! Wait for "C" to complete.
097  !$OMP BARRIER
098      !-------------------------------------------------
099      ! Compute, "P:=2*(Q*C-R)"; Note, the symmetry of "C" is used.
100  !$OMP DO
101      DO  IB = KK+1, NBLK
102        CALL DSYMM ('R', 'L', BS(IB), RANK, 1.D0, &
103                    C, SIZE(C,1),                 &
104                    A(1,1,INDX(IB,KK)), SIZE(A,1), &
105                    0.D0, P(1,1,IB), SIZE(P,1))
106        P(:BS(IB),:RANK,IB) = 2 * (P(:BS(IB),:RANK,IB) - R(:BS(IB),:RANK,IB))
107      ENDDO
108      !-------------------------------------
109      !  Compute, "A := A + P * Q^{T} + Q * P^{T}".
110  !$OMP DO  SCHEDULE(DYNAMIC)
111      DO  JB = KK+1, NBLK
112        DO  IB = JB, NBLK
113          IF (IB == JB) THEN ! Diagonal block.
114            CALL DSYR2K ('L', 'N', BS(JB), RANK, 1.D0, &
115                         P(1,1,JB), SIZE(P,1),         &
116                         A(1,1,INDX(JB,KK)), SIZE(A,1), &
117                         1.D0, A(1,1,INDX(JB,JB)), SIZE(A,1))
118          ELSE ! Off-diagonal block (IB > JB).
119            CALL DGEMM ('N', 'T', BS(IB), BS(JB), RANK, 1.D0, &
120                         P(1,1,IB), SIZE(P,1),          &
121                         A(1,1,INDX(JB,KK)), SIZE(A,1), &
122                         1.D0, A(1,1,INDX(IB,JB)), SIZE(A,1))
123            CALL DGEMM ('N', 'T', BS(IB), BS(JB), RANK, 1.D0, &
124                         A(1,1,INDX(IB,KK)), SIZE(A,1), &
125                         P(1,1,JB), SIZE(P,1),          &
126                         1.D0, A(1,1,INDX(IB,JB)), SIZE(A,1))
127          ENDIF
128        ENDDO
129      ENDDO
130  !$OMP END DO NOWAIT
131  !$OMP END PARALLEL
132      !-----------------------------------------
133    ENDDO LOOP_KK
134    IF (NBLK >= 2) THEN
135      ! Save ALP(NBLK-1).
136      P(:,:,NBLK-1) = A(:,:,INDX(NBLK-1,NBLK-1))
137      ! Save BET(NBLK-1).
138      R(:LAST_BS,:,NBLK-1) = A(:LAST_BS,:,INDX(NBLK,NBLK-1))
139      R(LAST_BS+1:,:,NBLK-1) = 0.D0 ! Note: current code requires this.
140    ENDIF
141    IF (NBLK >= 1) THEN
142      ! Save ALP(NBLK).
143      P(:,:,NBLK) = 0.D0 ! Fill zeros to clear unused places.
144      P(:LAST_BS,:LAST_BS,NBLK) = A(:LAST_BS,:LAST_BS,INDX(NBLK,NBLK))
145    ENDIF
146  END SUBROUTINE BHSHLD
```

リスト 3　ブロック鏡映軸の作成：MAKE_REFLECT
List 3　Construction of axis of block reflector: MAKE_REFLECT.

```
001  #define USE_BLAS3
002  !=====================================================================
003  !  From the given block column vector U(:,:,KK+1:NBLK),
004  !  the block reflector vector is formed
005  !  and U(:,:RANK,KK+1:NBLK) is overwritten.
006  !  (The upper blocks U(:,:,1:KK) are never changed.)
007  !
008  !=====================================================================
009  SUBROUTINE MAKE_REFLECT (KK, BLKSZ, NBLK, U, RANK, BET)
010  USE IO_UNIT
011  IMPLICIT NONE
012  INTEGER,INTENT(IN)   :: KK    ! The step of Householder transform.
013  INTEGER,INTENT(IN)   :: BLKSZ ! Size of block.
014  INTEGER,INTENT(IN)   :: NBLK  ! Size of matrix in block.
```

```
015  REAL(8),INTENT(INOUT):: U(BLKSZ,BLKSZ,NBLK) ! Block column vector.
016  INTEGER,INTENT(OUT)  :: RANK  ! Determined rank of the block reflector.
017  REAL(8),INTENT(OUT)  :: BET(BLKSZ,BLKSZ) !
018           ! H * U = E_rank * BET; where BET * PERMU = (-QV)*(R)
019  !-----------------------------------------------------------------------
020  INTEGER PERMU(BLKSZ)    ! PERMU(1:BLKSZ) is a permutation array.
021  REAL(8) QV(BLKSZ,BLKSZ) ! QV(1:RANK,1:RANK) is an orthogonal matrix.
022  REAL(8) R(BLKSZ,BLKSZ)  ! R(1:RANK,1:BLKSZ) is an upper triangle matrix.
023  REAL(8) Q(BLKSZ,BLKSZ), LAMBDA(BLKSZ), W(BLKSZ,BLKSZ)
024  REAL(8) W2(BLKSZ,BLKSZ), TMP(BLKSZ,BLKSZ)
025  INTEGER J, M, N
026  INTEGER IB
027  !-----------------------------------------------------------------------
028    M = (NBLK-KK) * BLKSZ
029    N = BLKSZ
030    !
031    ! Apply TS-HQR for block column vector (without column pivoting).
032    ! For without pivoting, "RANK" will be "N", and "PERMU" will be identity.
033    CALL HQR_BLOCK_TS (M, N, &
034                       U(1,1,KK+1), BLKSZ, NBLK-KK, &
035                       R, SIZE(R,1), &
036                       RANK, PERMU)
037    !
038    IF (RANK == 0) RETURN ! Prevent error for null sized matrix.
039    !-------------------------
040    ! SVD : "U[KK+1] --> Q LAMBDA W^{T}"; (or "Q LAMBDA V", where "V=W^{T}".)
041    CALL MAT_SVD (RANK, RANK,    &
042                  U(1,1,KK+1), SIZE(U,1), &
043                  Q, SIZE(Q,1), &
044                  LAMBDA,       &
045                  W, SIZE(W,1))
046    !-------------------------
047    ! Compute, U[KK+1] := U[KK+1] + Q * W^{T}.
048    !    we first compute "QV := Q * W^{T}" which is also used later.
049  #ifndef USE_BLAS3
050    QV(:RANK,:RANK) = MATMUL(Q(:RANK,:RANK), TRANSPOSE(W(:RANK,:RANK)))
051  #else
052    CALL DGEMM ('N', 'T', RANK, RANK, RANK, 1.D0, &
053         Q, SIZE(Q,1), W, SIZE(W,1), &
054         0.D0, QV, SIZE(QV,1))
055  #endif
056    U(:RANK,:RANK,KK+1) = U(:RANK,:RANK,KK+1) + QV(:RANK,:RANK)
057    !-------------------------
058    ! Compute,  W2 := V^{T} * (2(I+LAMBDA))^(-1/2).
059    DO  J = 1, RANK
060      W2(:RANK,J) = W(:RANK,J) / SQRT(2.D0*(1.D0+LAMBDA(J)))
061    ENDDO
062    !-------------------------
063    ! Compute, U := U * W2.
064  !$OMP PARALLEL DO  PRIVATE(TMP)
065    DO  IB = KK+1, NBLK
066      TMP(:BLKSZ,:RANK) = U(:BLKSZ,:RANK,IB)
067  #ifndef USE_BLAS3
068      U(:BLKSZ,:RANK,IB) = MATMUL(TMP(:BLKSZ,:RANK), W2(:RANK,:RANK))
069  #else
070      CALL DGEMM ('N', 'N', BLKSZ, RANK, RANK, 1.D0, &
071           TMP, SIZE(TMP,1), W2, SIZE(W2,1), &
072           0.D0, U(1,1,IB), SIZE(U,1))
073  #endif
074    ENDDO
075    !-------------------------
076    BET(:RANK,PERMU(:)) = -MATMUL(QV(:RANK,:RANK), R(:RANK,:BLKSZ))
077    BET(RANK+1:BLKSZ,:) = 0.D0 ! Note: current code requires this.
078    !-------------------------
079  END SUBROUTINE MAKE_REFLECT
```

リスト 4　ブロック列ベクトルの H-QR 分解（TS-QR）

List 4　H-QR decomposition of block column vector (TS-QR).

```
001  !=====================================================================
002  !  HQR method for block column vector "A"
003  !  to factorize as  "A P = Q R".
004  !  The effective rank of matirx "A" is returned.
005  !  The store pattern of matrix "A" is blocked.
006  !  Note: with this code, "Q" is overwritten to "A".
007  !=====================================================================
008  SUBROUTINE HQR_BLOCK_TS (M, N, ABLK, BLKSZ, NBLK, R, LDR, RANK, PERMU)
009  IMPLICIT NONE
010  INTEGER, INTENT(IN)  :: M, N ! Matrix "A" is "M" by "N", where "M >= N".
011  INTEGER, INTENT(IN)  :: BLKSZ    ! Size of tile blocking.
012  REAL(8),INTENT(INOUT):: ABLK(BLKSZ,N,NBLK) ! "A" is stored in blocked.
013  INTEGER, INTENT(IN)  :: NBLK     ! Number of blocks.
014  REAL(8),INTENT(OUT)  :: R(LDR,N) ! R(N,N)
015  INTEGER, INTENT(IN)  :: LDR      ! Leading dimension of "R".
016  INTEGER, INTENT(OUT) :: RANK     ! Determined effective rank.
017  INTEGER, INTENT(OUT) :: PERMU(N) ! Permutation array.
018  !-----------------------------------------------------------------------
019  INTEGER J
020  REAL(8),ALLOCATABLE:: A(:,:)
021  INTEGER IFLAG
022  ! Note: The value of "BLKSZ_A" must be by some factor larger than "N".
023  #if 0
024  INTEGER,PARAMETER:: BLKSZ_A = 300
025  #else
026  INTEGER,PARAMETER:: BLKSZ_A = 500
027  #endif
028  !---------------------------------
029    IF (BLKSZ_A < N) STOP 'HQR_BLOCK_TS: ERROR BLKSZ_A < N.'
030    ALLOCATE (A(M,N), STAT=IFLAG)
031       IF (IFLAG /= 0) STOP 'HQR_BLOCK_TS: FAILED TO ALLOCATE "A".'
032    ! Copy the block column to the plain matrix "A".
033    CALL BLKCOL_TO_MAT (M, N, A, SIZE(A,1), ABLK, BLKSZ, NBLK)
034    ! The T-S matrix "A" is QR decomposed by the TS-QR decomposition.
035    CALL HQRF_TS (M, N, A, SIZE(A,1), R, SIZE(R,1), BLKSZ_A)
036    ! Copy the calculated orthonormal basis "Q" to the block column.
037    CALL MAT_TO_BLKCOL (M, N, A, SIZE(A,1), ABLK, BLKSZ, NBLK)
038    RANK = N
039    DO  J = 1, N
040      PERMU(J) = J
041    ENDDO
042    DEALLOCATE (A)
043  END SUBROUTINE HQR_BLOCK_TS
044  !=====================================================================
045  !
046  !  Copy from the block column vector "ABLK" to the matrix "A"
047  !  which stores elements in the usual manner.
048  !
049  !=====================================================================
050  SUBROUTINE BLKCOL_TO_MAT (M, N, A, LDA, ABLK, BLKSZ, NBLK)
051  IMPLICIT NONE
052  INTEGER,INTENT(IN) :: M, N     ! M >= N.
053  REAL(8),INTENT(OUT):: A(LDA,N) ! Stores A(M,N).
054  INTEGER,INTENT(IN) :: LDA      ! Leading dimension of "A".
055  INTEGER,INTENT(IN) :: BLKSZ    ! Tile size.
056  REAL(8),INTENT(IN) :: ABLK(BLKSZ,N,NBLK) ! ABLK(BLKSZ,N,NBLK)
057  INTEGER,INTENT(IN) :: NBLK     ! Number of blocks in the block col vec.
058  !-------------------------------------------------------
059  INTEGER IB, IS, IE
060  !-------------------------------------------------------
061    IF (NBLK * BLKSZ < M) STOP 'BLKCOL_TO_MAT: NBLK*BLKSZ < M.'
062  !$OMP PARALLEL DO PRIVATE (IS,IE)
063    DO  IB = 1, NBLK
064      IS = (IB-1)*BLKSZ+1
065      IE = MIN(IS+BLKSZ-1, M)
066      A(IS:IE,1:N) = ABLK(1:IE-IS+1,1:N,IB)
067    ENDDO
068  END SUBROUTINE BLKCOL_TO_MAT
069  !=====================================================================
070  !
071  !  Copy from the matrix "A" in the usual manner of store
072  !  into the row-partitioned block matrix "ABLK".
073  !
074  !=====================================================================
075  SUBROUTINE MAT_TO_BLKCOL (M, N, A, LDA, ABLK, BLKSZ, NBLK)
076  IMPLICIT NONE
077  INTEGER,INTENT(IN) :: M, N     ! M >= N.
078  REAL(8),INTENT(IN) :: A(LDA,N) ! Stores A(M,N)
079  INTEGER,INTENT(IN) :: LDA      ! Leading dimension of "A".
080  INTEGER,INTENT(IN) :: BLKSZ    ! Tile size.
081  REAL(8),INTENT(OUT):: ABLK(BLKSZ,N,NBLK) ! ABLK(BLKSZ,N,NBLK)
082  INTEGER,INTENT(IN) :: NBLK     ! Number of blocks in block col. vec.
083  !-------------------------------------------------------
084  INTEGER IB, IS, IE
085  !-------------------------------------------------------
086    IF (NBLK * BLKSZ < M) STOP 'MAT_TO_BLKCOL: NBLK*BLKSZ < M.'
087  !$OMP PARALLEL DO PRIVATE (IS,IE)
088    DO  IB = 1, NBLK
089      IS = (IB-1)*BLKSZ+1
090      IE = MIN(IS+BLKSZ-1, M)
091      ABLK(1:IE-IS+1,:,IB) = A(IS:IE,:)
092      IF (IB == NBLK) THEN
093        ABLK(IE-IS+2:BLKSZ,:,NBLK) = 0.D0 ! Fill zeros.
094      ENDIF
095    ENDDO
096  END SUBROUTINE MAT_TO_BLKCOL
```

リスト 5　通常格納の行列の TS-HQR（列交換なし）：HQRF_TS

List 5　TS-HQR for matrix without tiled (without pivoting): HQRF_TS.

```
001  !=====================================================================
002  !  T-S Householder QR-decomposition without column exchange: "A = Q R".
003  !  The place of "A" is overwritten by "Q".
004  !  (Note: this version does not make use the property
005  !  that the intermediate matrix "B" has many zeros.)
006  !=====================================================================
007  RECURSIVE SUBROUTINE HQRF_TS (M, N, A, LDA, R, LDR, BLKSZ_A)
008  IMPLICIT NONE
009  INTEGER,INTENT(IN)   :: M, N     ! M >= N.
010  REAL(8),INTENT(INOUT):: A(LDA,N) ! For A(M,N).
011  INTEGER,INTENT(IN)   :: LDA
012  REAL(8),INTENT(OUT)  :: R(LDR,N) ! For R(N,N).
013  INTEGER,INTENT(IN)   :: LDR
014  INTEGER,INTENT(IN)   :: BLKSZ_A  ! Size of row block.
015  !---------------------------------------------------------------
016  REAL(8),ALLOCATABLE:: B(:,:), PIV(:,:), C(:,:)
017  INTEGER IFLAG
018  INTEGER MB
019  INTEGER NBLK, IB
020  INTEGER NROW     ! Number of rows in the block.
021    NROW(IB) = MIN(BLKSZ_A, M-(IB-1)*BLKSZ_A) ! Statement function.
022  !---------------------------------------------------------------
023    NBLK = (M+BLKSZ_A-1)/BLKSZ_A
024    IF (M < 4*BLKSZ_A) THEN ! The stop criteria for recursion (to be tuned).
025      CALL HQRF (M, N, A, SIZE(A,1), R, SIZE(R,1)) ! Traditional HQR.
026      RETURN
027    ENDIF
028    !-----
029    ! Allocation of work arrays.
030    MB = (NBLK-1)*N + MIN(NROW(NBLK), N)
031    ALLOCATE (B(MB,N), PIV(N,NBLK), STAT=IFLAG)
032      IF (IFLAG/=0) STOP 'HQRF_TS: FAILED TO ALLOCATE B AND PIV.'
033    !-----
034  !$OMP PARALLEL DO
035    DO  IB = 1, NBLK
036      CALL FWD_TRANS (NROW(IB), N, &
037                      A(1+(IB-1)*BLKSZ_A,1), SIZE(A,1), &
038                      PIV(1,IB), &
039                      B(1+(IB-1)*N,1), SIZE(B,1))
040    ENDDO
041    !-----
042    ! RECURSIVE CALL.
043    CALL HQRF_TS (MB, N, B, SIZE(B,1), R, SIZE(R,1), BLKSZ_A)
044    !-----
045  !$OMP PARALLEL  PRIVATE(C,IFLAG)
046    ALLOCATE (C(BLKSZ_A,N), STAT=IFLAG)
047       IF (IFLAG/=0) STOP 'HQRF_TS: FAILED TO ALLOCATE C.'
048  !$OMP DO
049    DO  IB = 1, NBLK
050      CALL BWD_TRANS (NROW(IB), N, &
051                      A(1+(IB-1)*BLKSZ_A,1), SIZE(A,1), &
052                      PIV(1,IB), &
053                      B(1+(IB-1)*N,1), SIZE(B,1), &
054                      C, SIZE(C,1))
055    ENDDO
```

```
056   DEALLOCATE (C)
057 !$OMP END PARALLEL
058   DEALLOCATE (B, PIV)
059 END SUBROUTINE HQRF_TS
060 !=======================================================
061 ! Householder forward decomposition in the block.
062 ! This is level-2 BLAS.
063 !=======================================================
064 SUBROUTINE FWD_TRANS (M, N, A, LDA, PIV, U, LDU)
065 IMPLICIT NONE
066 INTEGER,INTENT(IN)   :: M, N
067 REAL(8),INTENT(INOUT):: A(LDA,N) ! A(M,N)
068 INTEGER,INTENT(IN)   :: LDA
069 REAL(8),INTENT(OUT)  :: PIV(N)
070 REAL(8),INTENT(OUT)  :: U(LDU,N) ! U(N,N)
071 INTEGER,INTENT(IN)   :: LDU
072 !-----------------------------------------------
073 REAL(8) AMX, S, ALPHA, T
074 INTEGER K, J
075 !-----------------------------------------------
076   IF (M <= N) THEN
077     ! Possible case for the last fragment block.
078     U(1:M,1:N) = A(1:M,1:N)
079     RETURN
080   ENDIF
081   DO  K = 1, N
082     U(1:K-1,K) = A(1:K-1,K)
083     ! Renormalize vector using max-norm.
084     AMX = MAXVAL(ABS(A(K:M,K)))
085     IF (AMX /= 0.D0) THEN
086       A(K:M,K) = A(K:M,K) / AMX
087     ENDIF
088     !-----
089     S = SIGN(SQRT(SUM(A(K:M,K)**2)), A(K,K))
090     A(K,K) = A(K,K) + S
091     PIV(K) = - S
092     U(K,K) = PIV(K) * AMX
093     U(K+1:N,K) = 0.D0
094     IF (K == N) EXIT
095     IF (PIV(K) /= 0.D0) THEN
096       ALPHA = 1.D0 / (PIV(K) * A(K,K))
097       DO  J = K+1, N
098         T = DOT_PRODUCT(A(K:M,K), A(K:M,J)) * ALPHA
099         A(K:M,J) = A(K:M,J) + A(K:M,K) * T
100       ENDDO
101     ENDIF
102   ENDDO
103 END SUBROUTINE FWD_TRANS
104 !=======================================================
105 ! Householder backward transformation in the block.
106 ! This is level-2 BLAS.
107 !=======================================================
108 SUBROUTINE BWD_TRANS (M, N, A, LDA, PIV, U, LDU, C, LDC)
109 IMPLICIT NONE
110 INTEGER,INTENT(IN)   :: M, N
111 REAL(8),INTENT(INOUT):: A(LDA,N) ! A(M,N)
112 INTEGER,INTENT(IN)   :: LDA
113 REAL(8),INTENT(IN)   :: PIV(N)
114 REAL(8),INTENT(IN)   :: U(LDU,N) ! U(N,N)
115 INTEGER,INTENT(IN)   :: LDU
116 REAL(8)               C(LDC,N) ! work area.
117 INTEGER,INTENT(IN)   :: LDC
118 !-----------------------------------------------
119 REAL(8) ALPHA, T
120 INTEGER K, J
121 !-----------------------------------------------
122   IF (M <= N) THEN
123     ! Possible case for the last fragment block.
124     A(1:M,1:N) = U(1:M,1:N)
125     RETURN
126   ENDIF
127   DO  J = 1, N
128     C(1:N,J) = U(1:N,J)
129     C(N+1:M,J) = 0.D0
130   ENDDO
131   DO  K = N, 1, -1
132     IF (PIV(K) /= 0.D0) THEN
133       ALPHA = 1.D0 / (PIV(K) * A(K,K))
134       DO  J = 1, N
135         T = DOT_PRODUCT(A(K:M,K), C(K:M,J)) * ALPHA
136         C(K:M,J) = C(K:M,J) + A(K:M,K) * T
137       ENDDO
138     ENDIF
139   ENDDO
140   A(1:M,1:N) = C(1:M,1:N)
141 END SUBROUTINE BWD_TRANS
142 !=============================================================
143 !  Traditional Householder QR factorization
144 !  "without" column exchange as "A = Q R".
145 !  Note: the place of "A" is overwritten by "Q".
146 !=============================================================
147 SUBROUTINE HQRF (M, N, A, LDA, R, LDR)
148 IMPLICIT NONE
149 INTEGER,INTENT(IN)   :: M, N     ! M >= N.
150 REAL(8),INTENT(INOUT):: A(LDA,N) ! For A(M,N).
151 INTEGER,INTENT(IN)   :: LDA      ! Leading dimension of A.
152 REAL(8),INTENT(OUT)  :: R(LDR,N) ! For R(N,N).
153 INTEGER,INTENT(IN)   :: LDR      ! Leading dimension of R.
154 !-------------------------------------------------------------
155 REAL(8) AMX, S, PIV(N), ALPHA, T
156 INTEGER J, K
157 !-------------------------------------------------------------
158   IF (M < N) STOP 'HQRF: ERROR M < N.'
159 !$OMP PARALLEL  PRIVATE(K,ALPHA)
160   !-------------------------------
161   ! Forward-transformations to form "R", and implicitly "Q".
162   DO  K = 1, MIN(N,M)
163 !$OMP SINGLE
164     R(1:K-1,K) = A(1:K-1,K)
165     !-------------------
166     ! Renormalize vector to avoid underflow.
167     AMX = MAXVAL(ABS(A(K:M,K)))
168     IF (AMX /= 0.D0) THEN
169       A(K:M,K) = A(K:M,K) / AMX
170     ENDIF
171     !-------------------
172     ! Determine the Householder vector.
173     S = SIGN(SQRT(SUM(A(K:M,K)**2)), A(K,K))
174     A(K,K) = A(K,K) + S
175     PIV(K) = - S
176 !$OMP END SINGLE
177 !$OMP BARRIER
178     !-------------------
179     IF (PIV(K) /= 0.D0) THEN
180       ALPHA = 1.D0 / (PIV(K) * A(K,K))
181 !$OMP DO  PRIVATE(T)
182       DO  J = K+1, N
183         T = DOT_PRODUCT(A(K:M,K), A(K:M,J)) * ALPHA
184         A(K:M,J) = A(K:M,J) + A(K:M,K) * T
185       ENDDO
186     ENDIF
187 !$OMP SINGLE
188     R(K,K) = PIV(K) * AMX
189     R(K+1:N,K) = 0.D0
190 !$OMP END SINGLE
191 !$OMP BARRIER
192   ENDDO
193   !------------------------
194   ! Back-transform to make the explicit "Q".
195   DO  K = MIN(N,M), 1, -1
196     IF (PIV(K) /= 0.D0) THEN
197       ALPHA = 1.D0 / (PIV(K) * A(K,K))
198 !$OMP DO  PRIVATE(T)
199       DO  J = K+1, N
200         T = DOT_PRODUCT(A(K:M,K), A(K:M,J)) * ALPHA
201         A(K:M,J) = A(K:M,J) + A(K:M,K) * T
202       ENDDO
203 !$OMP SINGLE
204       A(1:K-1,K) = 0.D0
205       T = A(K,K) * ALPHA
206       A(K,K)     = 1.D0 + A(K,K) * T
207       A(K+1:M,K) = A(K+1:M,K) * T
208 !$OMP END SINGLE
209     ELSE
210 !$OMP SINGLE
211       A(1:K-1,K) = 0.D0
212       A(K,K)     = 1.D0
213       A(K+1:M,K) = 0.D0
214 !$OMP END SINGLE
215     ENDIF
216 !$OMP BARRIER
217   ENDDO
218 !$OMP END PARALLEL
219 END SUBROUTINE HQRF
```

リスト **6**　特異値分解作成：MAT_SVD
List 6　Construction of SVD: MAT_SVD.

```
001 !=====================================================================
002 !
003 !  Compute SVD : "A --> U D V^{T}".
004 !
005 !  Note: this routine will not change the array "A"
006 !  unless array "A" is equivalenced to either "U" or "V".
007 !
008 !=====================================================================
009 SUBROUTINE MAT_SVD (M1, M2, A, LDA, U, LDU, D, V, LDV)
010 USE IO_UNIT
011 IMPLICIT NONE
012 INTEGER,INTENT(IN)   :: M1, M2    ! M1 >= M2. The size of the problem.
013 REAL(8),INTENT(INOUT):: A(LDA,M2) ! Mat.of size (M1,M2) to be decomposed.
014 INTEGER,INTENT(IN)   :: LDA       ! The leading dimension of A.
015 REAL(8),INTENT(OUT)  :: U(LDU,M2) ! Left orthog. mat. of size (M1,M2).
016 INTEGER,INTENT(IN)   :: LDU       ! The leading dimension of U.
017 REAL(8),INTENT(OUT)  :: D(M2)     ! The singular values of A.
018 REAL(8),INTENT(OUT)  :: V(LDV,M2) ! Right orthog. mat. of size (M2,M2).
019 INTEGER,INTENT(IN)   :: LDV       ! The leading dimension of V.
020 !---------------------------------------------------------------------
021 #undef  USE_LAPACK
022 #ifndef USE_LAPACK
023   EXTERNAL SVDD ! Numpac routine SVDD.
024   INTEGER,PARAMETER:: ISW=3 ! Switch for NUMPAC SVDD to compute both "U" and "V".
025   REAL(8) WORK(M2) ! Work area for NUMPAC SVDD.
026   INTEGER INFO     ! Return code from NUMPAC SVDD.
027 !-----------------------------------------------------------------
028   IF (M1 < M2) STOP 'ERROR MAT_SVD: ARGUMENTS ERROR. (M1 < M2)'
029   !
030   CALL SVDD (A, SIZE(A,1), &
031              M1, M2, ISW, &
032              D, U, SIZE(U,1), V, SIZE(V,1), &
033              WORK, INFO)
034   IF (INFO /= 0) THEN
035     WRITE(STDERR,*) 'IN MAT_SVD: NUMPAC_SVDD RETURNED CODE=', INFO
036     STOP 'ERROR_SVD'
037   ENDIF
038 #else /* USE_LAPACK */
039 !-----------------------------------------------------------------
040   EXTERNAL DGESVD ! LAPACK ROUTINE.
041   REAL(8) A1(M1,M2), VT(M2,M2)
042   INTEGER LWORK
043   REAL(8),ALLOCATABLE:: WORK(:)
044   INTEGER INFO
045   !--------------------------------
046   LWORK = MAX(3*MIN(M1,M2)+MAX(M1,M2), 5*MIN(M1,M2))
047   ALLOCATE (WORK(LWORK))
048   A1(:M1,:M2) = A(:M1,:M2) ! COPY "A" TO "A1".
049   !
050   CALL DGESVD ('A', 'A', M1, M2, &
051                A1, SIZE(A1,1), &
052                D, U, SIZE(U,1), VT, SIZE(VT,1), &
053                WORK, LWORK, INFO)
054   IF (INFO /= 0) THEN
055     WRITE(STDERR,*) 'IN MAT_SVD: DGESVD RETURNED CODE=', INFO
056     STOP 'ERROR_SVD'
057   ENDIF
058   V(:M2,:M2) = TRANSPOSE(VT)
059   DEALLOCATE (WORK)
060   !--------------------------------
061 #endif /* USE_LAPACK */
062 END SUBROUTINE MAT_SVD
```

5. 計算機実験

以下では「換算性能」とは，N 次の実対称行列をブロック三重対角化する演算量が $(4/3)N^3$ であると見なして計算された演算速度のことであるとする（ブロック三重対角化の算法は，ブロック化を行わない通常の三重対角化の算法と比べて若干余分の演算を行っているが，他方では最後に得られるのはブロック三重対角形であって真の三重対角形には到達していないから楽になっている，という両面があるので比較は簡単ではない）．

プログラムは Fortran90 言語で書き，OpenMP による簡単な並列化をしている．xGEMM などの BLAS3 のルーチンには Intel MKL を用いた．BLAS3 の性能をさらに引き出すには，小規模行列（タイル）に用いる実際のサイズ b に対して高い性能が出るように高度に調整された専用の行列乗算ルーチンを準備して使うべきであろう．

最近の CPU には内部回路の温度が低くて余裕がある場合にはクロック周波数をシステムの基準値よりも上げて動作させ，温度が閾値を超えるとクロックの周波数を下げる機能（インテル社の製品では Turbo Boost と呼称）を有するものがあり，その機能を利用すると CPU の演算性能を上げられる可能性があるが，その反面として外部環境である温度やシステムの冷却能力などによって CPU の演算性能が変化することになり，実験結果の解釈や異なる環境に置かれた計算機システムとの間の公平な比較が困難になる．そこで今回のすべての実験は，そのような機能を無効に設定して行い，CPU が本来の標準クロック周波数を超えた動作をしないようにして測定を行った（なおそのようにしても，CPU を効率良く動作させてその結果 CPU の温度が閾値を超えると，過熱による障害を防ぐためにシステムが CPU のクロック周波数を自動で下げる保護機構があるので，システムの冷却能力や設置環境の違いにより性能測定の結果が変化する可能性は依然として残る）．

また，CPU の各コアについてコアあたり複数のスレッドをハードウェアレベルで自動的に実行できる機能（インテル製品では Hyper Thread と呼称し，たとえば今回使用した CPU で機能を有するものはコアあたり 2 スレッドまで）もまた，それを有効に設定した場合にはメモリの読み書きより浮動小数点演算が中心の計算ではむしろ性能が落ちることがあるので，今回の実験ではこの機能もまた無効に設定して測定を行った．今回用いたそのシステムの CPU についても，Hyper Thread 機能は浮動小数点演算の理論ピーク性能値には影響しない．

以下の（本論文の付録の章を除く）すべての実験において，ブロック鏡映変換の軸の構成に用いた QR 分解法は T-S 型行列用の階層的な方法であり，並列化には OpenMP を用いている．

5.1 Intel Core2 Quad Q6600 のシステムの場合

このシステムはシングル CPU で，CPU は Intel Core2 Quad Q6600 （コア数 4，クロック 2.4 GHz，L1（I/D どちらも）は 32 KB/コア，共有 L2 は 8 MB（4 MB x 2 ダイ））である．この CPU には Hyper Thread 機能や Turbo Boost 機能はない．主記憶の容量は 16 Gbyte（デュアルチャネル動作，DDR3-1600 MHz（PC3-12800）の 4 Gbyte の DIMM が 4 本，しかし実際の動作は DDR3-1067 MHz で，PC3-8500 であろう）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）で，並列化には Intel Fortran の OpenMP の機能のみを利用した．用いたコンパイラオプションは “-fast -openmp” である．この CPU は SSSE3 命令により，1 コアあたりクロックごとに倍精度演算が最大で 4 個実行できるので，CPU の理論ピーク演算性能は 38.4 GFLOPS（倍精度演算，SSE3，SSSE3）である．MKL の BLAS3 ルーチンはリンクのオプション “-mkl=sequential” により，シングルスレッド版だけを使用した．

容量 16 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 6 万までである．**図 3** は行列サイズ N を 1 万から 6 万まで刻み 1 万で変えながら横軸にはブロックサイズ b を 16 から 296 まで刻み 8 でとり，縦軸には換算性能（GFLOPS）の値をとってプロットしたグラフである．**図 4** は同じ測定結果を，縦軸を理論ピーク演算性能に対する換算性能の比率をパーセントで表した値に変更してプロットしたグラフである．**図 5** はブロックサイズ b を 16 から 160 まで 16 刻みでとったそれぞれの場合について，横軸に行列サイズ N をとり，縦軸に換算性能（GFLOPS）をとってプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 4 スレッドで実行した．

5.2 AMD Phenom II X4 940 BE の場合

このシステムはシングル CPU で, CPU は AMD Phenom II X4 940BE（コア数 4，クロック 3 GHz，L1（D/I どちらも）は 64 KB/コア，L2 は 512 KB/コア，共有 L3 は 6 MB，Socket AM2+）である．主記憶の容量は 8 Gbyte（デュアルチャネル動作，DDR2 PC-6400 ECC の 2 Gbyte DIMM を 4 本）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux（Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）を使用した．用いたコンパイラオプションは “-fast -openmp” である．この CPU は MSSE3 命令を持ち, 1 コアあたり CPU クロックごとに倍精度の演算を最大 4 個実行できるので，CPU の理論ピーク演

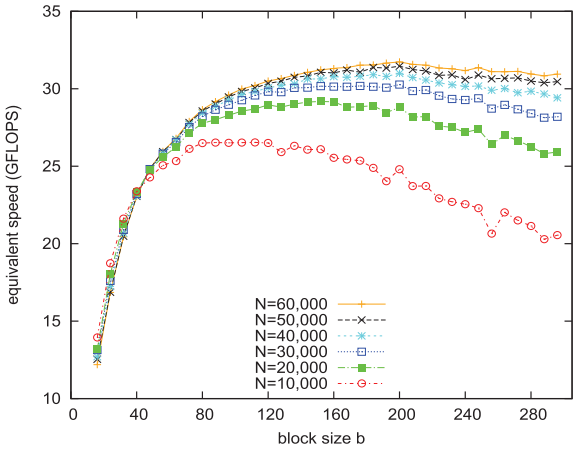


図 3　ブロック三重対角化の換算性能（Core2 Quad Q6600（4 スレッド））

Fig. 3　Equivalent speed of block tridiagonalization (Core2 Quad Q6600 (4 thread)).

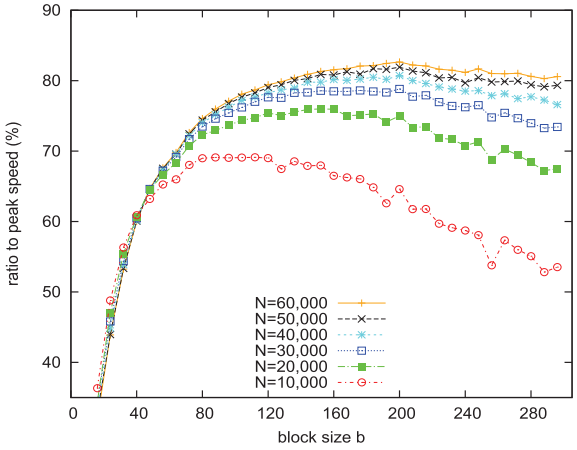


図 4　換算性能の理論ピーク性能に対する比率（Core2 Quad Q6600（4 スレッド））

Fig. 4　Ratio of equivalent speed to theoretical peak performance (Core2 Quad Q6600 (4 thread)).

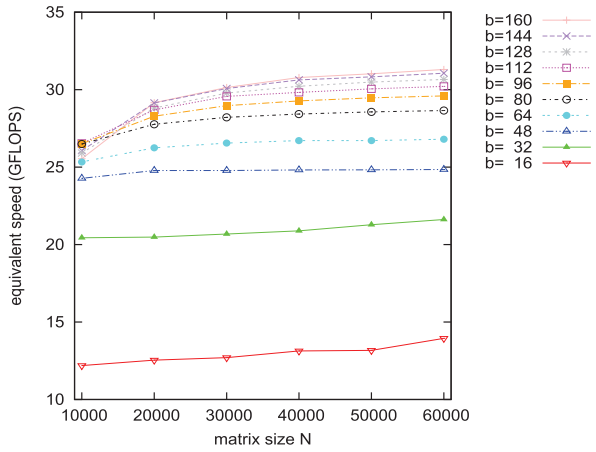


図 5　ブロック三重対角化の換算性能（Core2 Quad Q6600（4 スレッド））

Fig. 5　Equivalent speed of block tridiagonalization (Core2 Quad Q6600 (4 thread)).

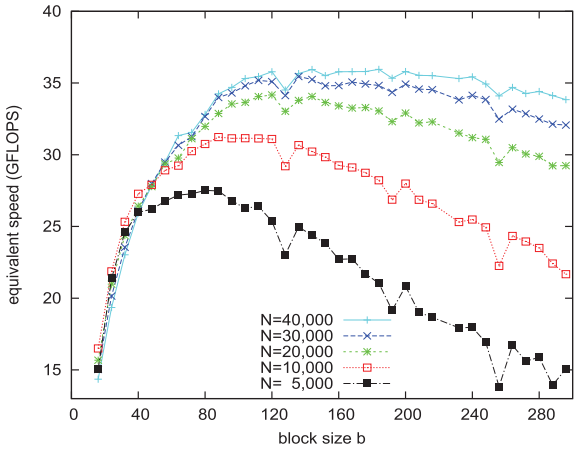


図 6　ブロック三重対角化の換算性能（Phenom II-X4-940BE（4 スレッド））

Fig. 6　Equivalent speed of block tridiagonalization (Phenom II-X4-940BE (4 thread)).

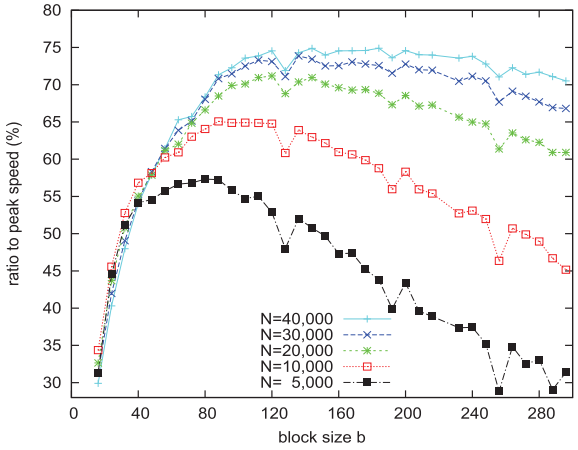


図 7　換算性能の理論ピーク性能に対する比率（Phenom II-X4-940BE（4 スレッド））

Fig. 7　Ratio of equivalent speed to theoretical peak performance (Phenom II-X4-940BE (4 thread)).

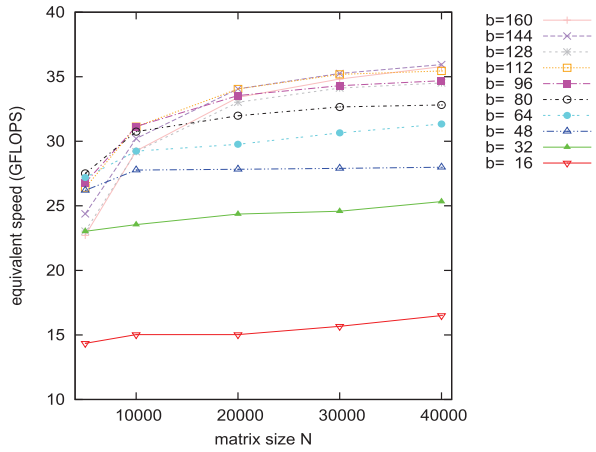


図 8　ブロック三重対角化の換算性能（Phenom II-X4-940BE（4 スレッド））

Fig. 8　Equivalent speed of block tridiagonalization (Phenom II-X4-940BE (4 thread)).

算性能は 48 GFLOPS（倍精度演算）である．MKL の BLAS ルーチンはリンクのオプションとして “-mkl=sequential” を指定してシングルスレッドのものを使用している．

容量 8 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 4 万までである．**図 6** は行列サイズ N が 5 千および 1 万から 4 万までの刻み 1 万のそれぞれの場合について，横軸はブロックサイズ b の値を 16 から 296 まで刻み 8 でとり，縦軸は換算性能（GFLOPS）の値をとってプロットしたグラフである．**図 7** は同じ測定結果を縦軸を理論ピーク演算性能に対する換算性能の比率をパーセントで表した値に変更してプロットしたグラフである．また**図 8** はブロックサイズ b の値を 16 から 160 まで刻み 16 のそれぞれの場合について，横軸に行列 A の次数 N が 5 千および 1 万から 4 万までの刻み 1 万のそれぞれの場合について，縦軸に換算性能（GFLOPS）の値をとってプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 4 スレッドで実行した．

5.3　Intel Core i7-920 のシステムの場合

このシステムはシングル CPU で，CPU は Intel Core i7-920（コア数 4，クロック 2.67 GHz，L1 は（D/I とも）は 32 KB/コア，L2 は 256 KB/コア，共有 L3 は 8 MB）である．CPU の Hyper Thread 機能と Turbo Boost 機能は BIOS の設定でオフにした．主記憶の容量は 48 Gbyte（トリプルチャネル動作，DDR3-1333 MHz（PC3-10600）の 8 Gbyte の DIMM を 6 本．実際の動作は DDR3-1067 MHz，PC3-8500）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux（Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）を用いた．並列化には Intel Fortran の OpenMP の機能だけを利用し，使用したコンパイラオプションは “-fast -openmp” である．この CPU は Nehalem アーキテクチャで SSE4.2 命令を持ち，1 コアあたり CPU クロックごとに倍精度の演算が最大 4 個実行できるので，TB 機能がオフの場合には CPU の理論ピーク演算性能は 42.72 GFLOPS（倍精度演算，SSE4.2）である．MKL の BLAS ルーチンはリンクのオプション “-mkl=sequential” を指定してシングルスレッド版だけを使用した．

対称性を用いてブロック三重対角化の計算を容量 48 Gbyte の主記憶上で行える行列のサイズ N は 1 万の倍数であれば 11 万までであるが，今回の計算では 9 万までを扱った．**図 9** は行列サイズ N を 1 万から 9 万まで刻み 1 万で変えながら横軸にはブロックサイズ b を 16 から 304 まで刻み 8 でとり，縦軸には換算性能（GFLOPS）の値をとってプロットしたグラフである．**図 10** は同じ測定結果を，縦軸を変更して理論ピーク演算性能に対する換算

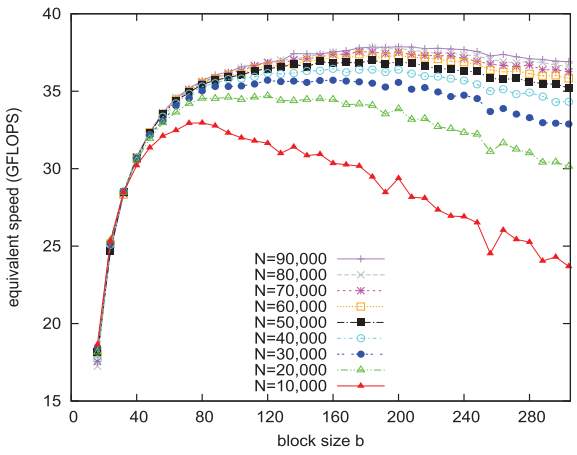


図 9　ブロック三重対角化の換算性能（Core i7-920（4 スレッド））
Fig. 9　Equivalent speed of block tridiagonalization (Core i7-920 (4 thread)).

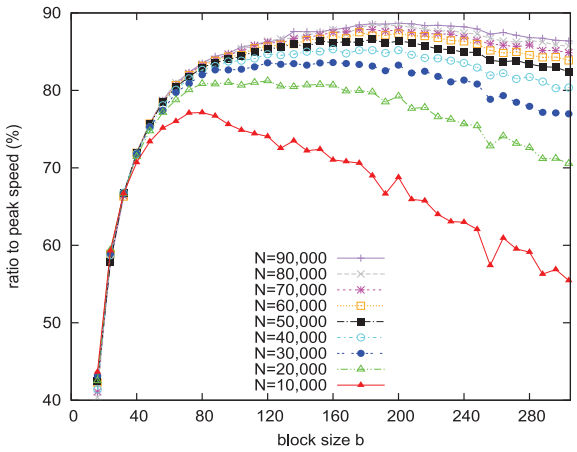


図 10　換算性能の理論ピーク性能に対する比率（Core i7-920（4 スレッド））
Fig. 10　Ratio of equivalent speed to theoretical peak performance (Core i7-920 (4 thread)).

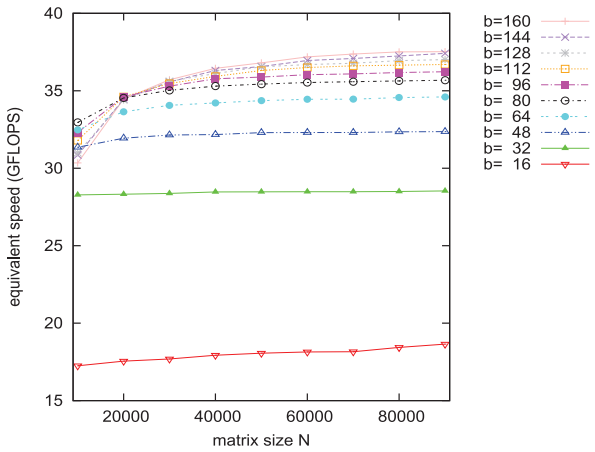


図 11　ブロック三重対角化の換算性能（Core i7-920（4 スレッド））
Fig. 11　Equivalent speed of block tridiagonalization (Core i7-920 (4 thread)).

性能の比率をパーセントで表した値にしてプロットしたグラフである．**図 11** はブロックサイズ b を 16 から 160 まで刻み 16 で変えたそれぞれの場合について，横軸に行列サイズを 1 万から 9 万まで刻み 1 万でとり，横軸に換算性能（GFLOPS）をとってプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 4 スレッドで実行した．

5.4　AMD Phenom II X6 1090T のシステムの場合

このシステムはシングル CPU で，CPU は AMD Phenom II X6 1090T B.E.（コア数 6，クロック 3.2 GHz，L1（D/I ともに）は 64 KB/コア，L2 は 512 KB/コア，共有 L3 は 6 MB，Socket AM3）である．主記憶の容量は 32 Gbyte（デュアルチャネル動作，DDR3-1333 MHz（PC3-10600）CL9 の 8 Gbyte DIMM を 4 本）である．OS は CentOS 6.5 for x86_64 であり，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）である．並列化には Intel Fortran の OpenMP の機能のみを利用し，用いたコンパイラオプションは “-fast -openmp” である．この CPU は SSE4a 命令を持ち，1 コアあたり CPU クロックごとに倍精度の演算が最大 4 個実行できるので，Tubo Core Technology がオフの場合の CPU の理論ピーク演算性能は 76.8 GFLOPS（倍精度演算，SSE3, SSE4a）である．MKL の BLAS ルーチンはリンクのオプションに “-mkl=sequential” を指定してシングルスレッド版だけを使用している．

容量 32 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 9 万までである．**図 12** は行列サイズ N を 1 万から 9 万まで刻み 1 万で変えながら，横軸にブロックサイズ b を 16 から 296 まで刻み 8 でとり，縦軸に換算性能の値をとりプロットしたグラフである．**図 13** は同じ測定結果を，縦軸を理論ピーク演算性能に対する換算性能の比率をパーセントで表した値に変更してプロットしたグラフである．**図 14** はブロックサイズ b を 16 から 160 まで刻み 16 で変えたそれぞれの場合について，横軸に行列サイズを 1 万から 9 万まで刻み 1 万でとり，縦軸に換算性能をとりプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 6 スレッドで実行した．

5.5　Intel Core i7-2600K のシステムの場合

このシステムはシングル CPU で，CPU は Intel Core i7-2600K （コア数 4，クロック 3.4 GHz，L1（D/I ともに）は 32 KB/コア，L2 は 256 KB/コア，共有 L3 は 8 MB）である．CPU の Hyper Thread 機能と Turbo Boost 機能は BIOS の設定でオフにした．主記憶の容量は 32 Gbyte（デュアルチャネル動作，DDR3-1600 MHz（PC3-10600 動作），

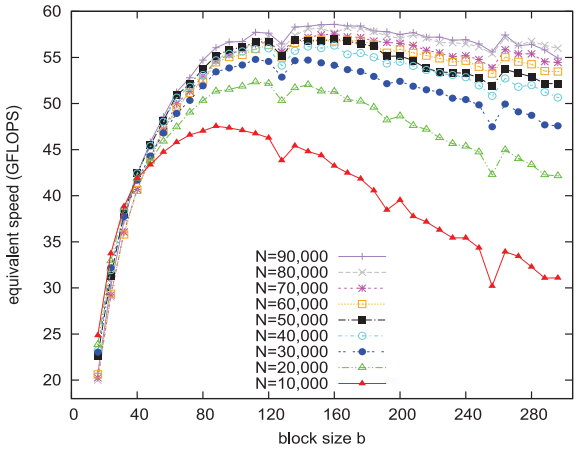


図 12　ブロック三重対角化の換算性能（Phenom II X6 1090T（6 スレッド））

Fig. 12　Equivalent speed of block tridiagonalization (Phenom II X6 1090T (6 thread)).

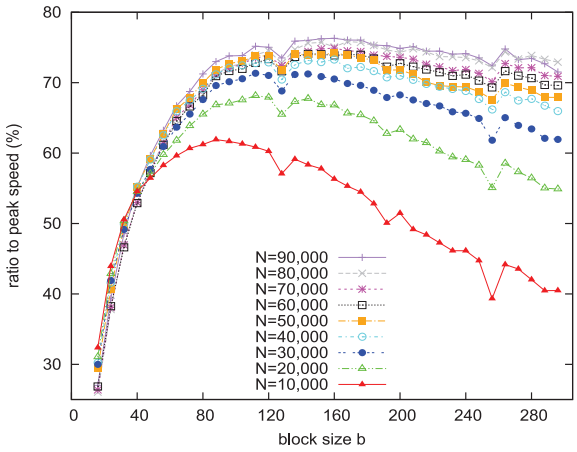


図 13　換算性能の理論ピーク性能に対する比率（Phenom II X6 1090T（6 スレッド））

Fig. 13　Ratio of equivalent speed to theoretical peak performance (Phenom II X6 1090T (6 thread)).

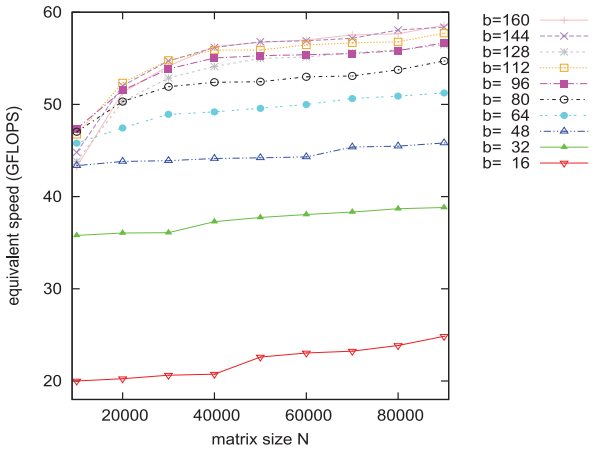


図 14　ブロック三重対角化の換算性能（Phenom II X6 1090T（6 スレッド））

Fig. 14　Equivalent speed of block tridiagonalization (Phenom II X6 1090T (6 thread)).

CL10 の 8 Gbyte DIMM を 4 本）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）で，用いたコンパイラオプションは “-fast -openmp” である．この CPU は Sandy Bridge アーキテクチャの AVX 命令を持ち，1 コアあたり CPU クロックごとに倍精度の演算を最大 8 個実行できるので，TB 機能がオフの場合の CPU の理論ピーク演算性能は 108.8 GFLOPS （倍精度演算，AVX 命令）である．MKL の BLAS ルーチンはリンクのオプション “-mkl=sequential” を指定し，シングルスレッド版だけを使用した．

容量 32 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 9 万までである．**図 15** は行列サイズ N が 1 万から 9 万まで刻み 1 万のそれぞれの場合について，横軸にブロックサイズ b の値を 16 から 288 まで刻み 16 でとり，縦軸に換算性能（GFLOPS）の値をとってプロットしたグラフであり，**図 16** は同じ測定結果を縦軸を変更して，理論ピーク演算性能に対する換算性能の比率をパーセントで表した値にしてプロットしたグラフである．また**図 17** はブロックサイズ b の値を 16 から 160 まで刻み 16 で変えたそれぞれの場合について，横軸に行列 A の次数 N を 1 万から 9 万まで刻み 1 万でとり，縦軸に換算性能（GFLOPS）の値をとってプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 4 スレッドで実行した．

5.6　Intel Core i7-3770K のシステムの場合

このシステムはシングル CPU で，CPU は Intel Core i7-3770K （コア数 4，クロック 3.5 GHz，L1（I/D ともに）は 32 KB/コア，L2 は 256 KB/コア，共有 L3 は 8 MB）である．CPU の Hyper Thread 機能と Turbo Boost 機能は BIOS の設定でオフにした．主記憶の容量は 32 Gbyte（デュアルチャネル，DDR3-1600 MHz（PC3-12800 動作）の 8 Gbyte DIMM を 4 本）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）である．用いたコンパイラオプションは “-fast -openmp” である．この CPU は Ivy-Bridge アーキテクチャの AVX 命令を持ち，1 コアあたり CPU クロックごとに倍精度の演算を最大 8 個実行できるので，TB 機能がオフの場合の CPU の理論ピーク演算性能は 112.0 GFLOPS（倍精度演算，AVX 命令）である．MKL の BLAS ルーチンはリンクのオプション “-mkl=sequential” によりシングルスレッド版だけを使用した．

容量 32 Gbyte の主記憶上で対称性を用いてブロック三

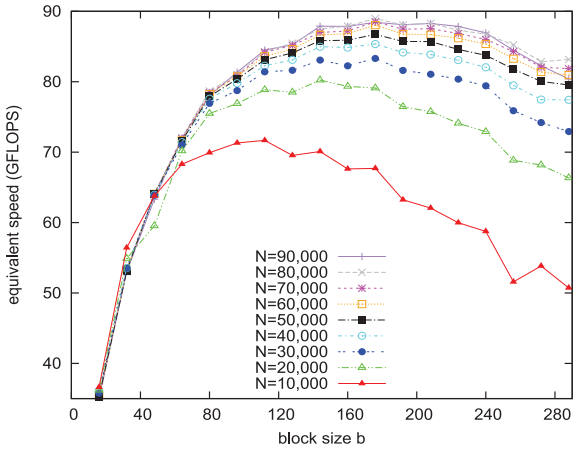


図 15　ブロック三重対角化の換算性能（Core i7-2600K（4 スレッド））
Fig. 15　Equivalent speed of block tridiagonalization (Core i7-2600K (4 thread)).

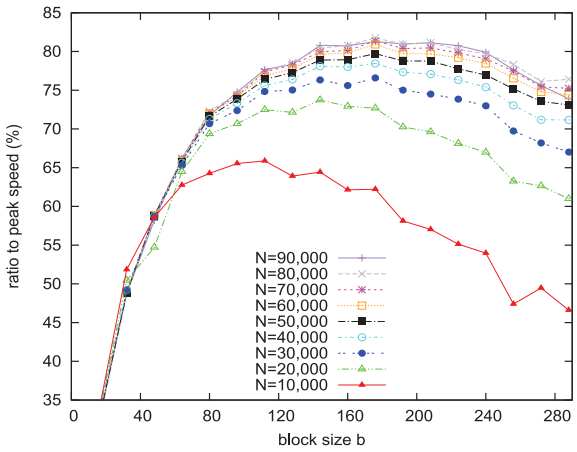


図 16　換算性能の理論ピーク性能に対する比率（Core i7-2600K（4 スレッド））
Fig. 16　Ratio of equivalent speed to theoretical peak performance (Core i7-2600K (4 thread)).

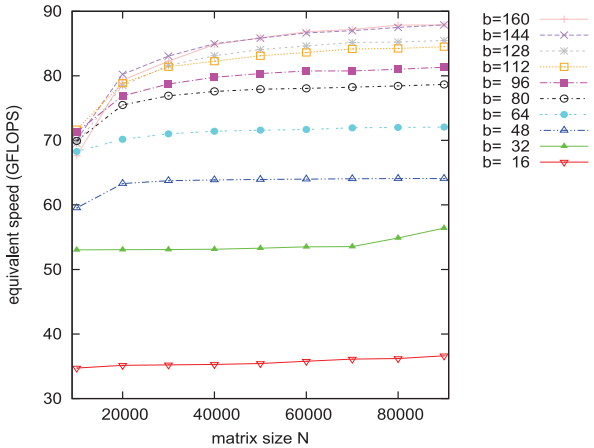


図 17　ブロック三重対角化の換算性能（Core i7-2600K（4 スレッド））
Fig. 17　Equivalent speed of block tridiagonalization (Core i7-2600K (4 thread)).

重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 9 万までである．**図 18** は行列サイズ N を 1 万から 9 万まで刻み 1 万で変えながら，横軸にブロックサイズ b を 16 から 288 まで刻み 16 で，縦軸に換算性能（GFLOPS）の値をとってプロットしたグラフである．**図 19** は同じ測定結果を，縦軸を理論ピーク演算性能に対する換算性能の比率をパーセントで表した値に変更してプロットしたグラフである．**図 20** は，ブロックサイズ b を 16 から 160 まで刻み 16 で変えたそれぞれの場合について，横軸に行列サイズを 1 万から 9 万まで刻み 1 万でとり，縦軸には換算性能（GFLOPS）をとりプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 4 スレッドで実行した．

5.7　Intel Core i7-3930K のシステムの場合

このシステムはシングル CPU で，CPU は Intel Core i7-3930K（コア数 6，クロック 3.2 GHz，L1（I/D ともに）は 32 KB/コア，L2 は 256 KB/コア，共有 L3 は 12 MB）である．CPU の Hyper Thread 機能と Turbo Boost 機能は BIOS の設定でオフにした．主記憶の容量は 64 Gbyte（クアドチャネル，DDR3-1600（PC3-12800），CL10 の 8 Gbyte DIMM を 8 本）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）で，用いたコンパイラオプションは “-fast -openmp” である．この CPU は Ivy-Bridge アーキテクチャの AVX 命令により，1 コアあたり CPU クロックごとに倍精度の演算を最大で 8 個実行できるので，TB 機能がオフの場合には CPU の理論ピーク演算性能は 153.6 GFLOPS （倍精度演算，AVX 命令）である．MKL の BLAS ルーチンはリンクオプション “-mkl=sequentual” によりシングルスレッド版だけを使用した．

容量 64 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 12 万までである．**図 21** は行列サイズ N を 1 万，2 万，3 万，5 万，7 万，9 万，12 万と変えたそれぞれの場合について，横軸にブロックサイズ b を 16 から 288 まで刻み 16 でとり，縦軸には得られた換算性能（GFLOPS）をとってプロットしたグラフであり，**図 22** は同じ測定結果を縦軸を変更して，CPU の理論ピーク演算性能に対する換算性能の比率をパーセント値でプロットしたグラフである．**図 23**）ではブロックサイズ b を 16 から 160 まで刻み 16 で変えたそれぞれの場合について，横軸には行列サイズ N を 1 万から 12 万まで刻み 1 万でとり，縦軸には換算性能（GFLOPS）をとりプロットしたグラフである．図 21 から図 23 はいずれも OpenMP により並列化し，CPU のコア数と等しい 6 スレッドで実行した結果である．

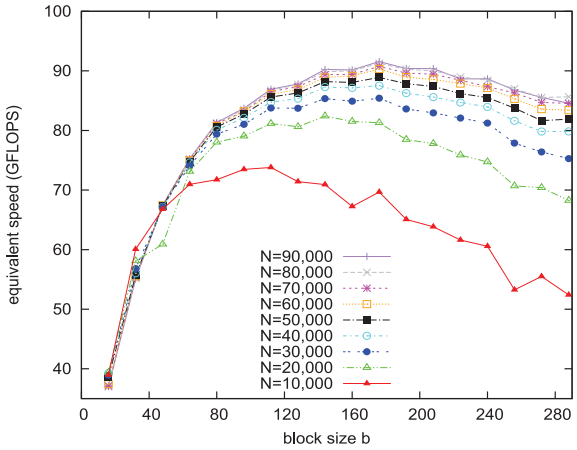


図 18　ブロック三重対角化の換算性能（Core i7-3770K（4 スレッド））
Fig. 18　Equivalent speed of block tridiagonalization (Core i7-3770K (4 thread)).

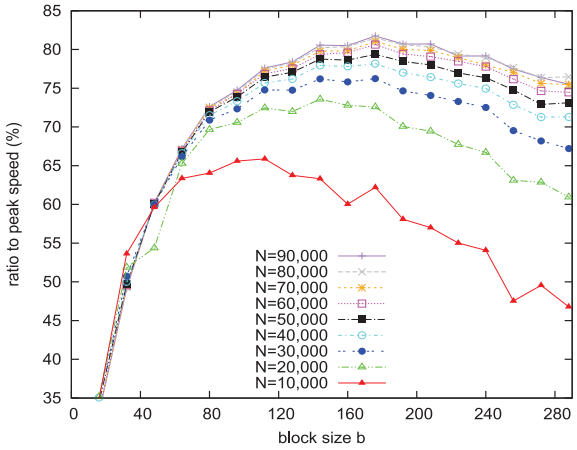


図 19　換算性能の理論ピーク性能に対する比率（Core i7-3770K（4 スレッド））
Fig. 19　Ratio of euivalent speed to theoretical peak performance (Core i7-3770K (4 thread)).

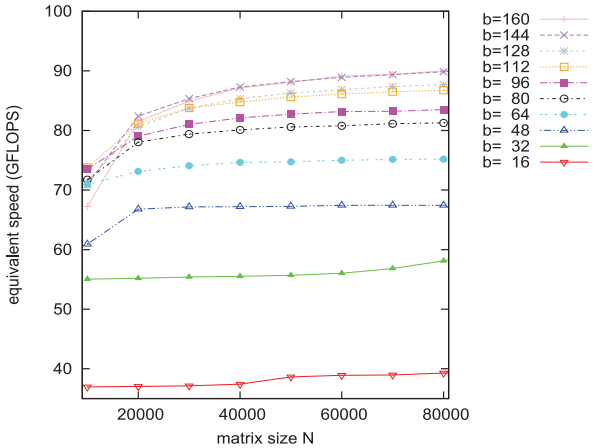


図 20　ブロック三重対角化の換算性能（Core i7-3770K（4 スレッド））
Fig. 20　Equivalent speed of block tridiagonalization (Core i7-3770K (4 thread)).

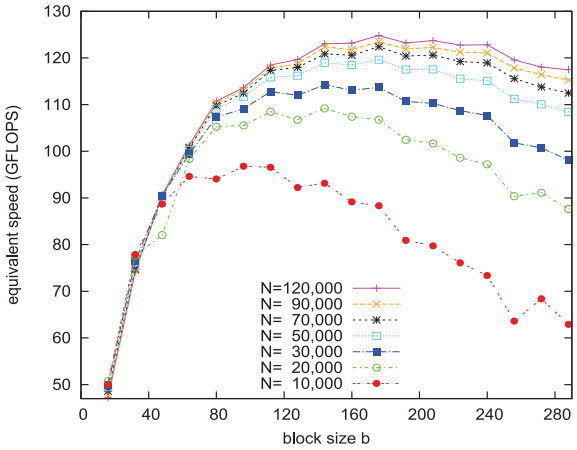


図 21　ブロック三重対角化の換算性能（Core i7-3930K（6 スレッド））

Fig. 21　Equivalent speed of block tridiagonalization (Core i7-3930K (6 thread)).

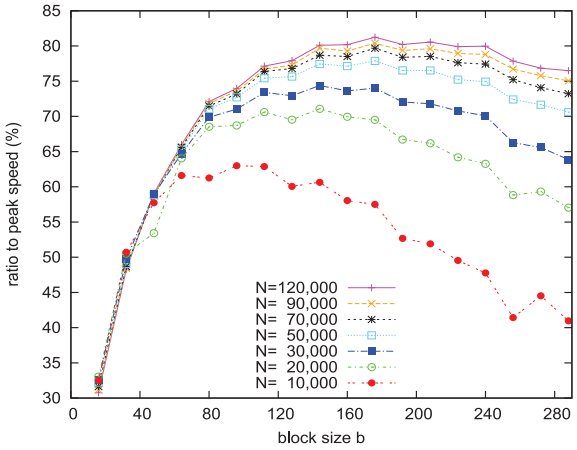


図 22　換算性能の理論ピーク性能に対する比率（Core i7-3930K（6 スレッド））

Fig. 22　Ratio of equivalent speed to theoretical peak performance (Core i7-3930K (6 thread)).

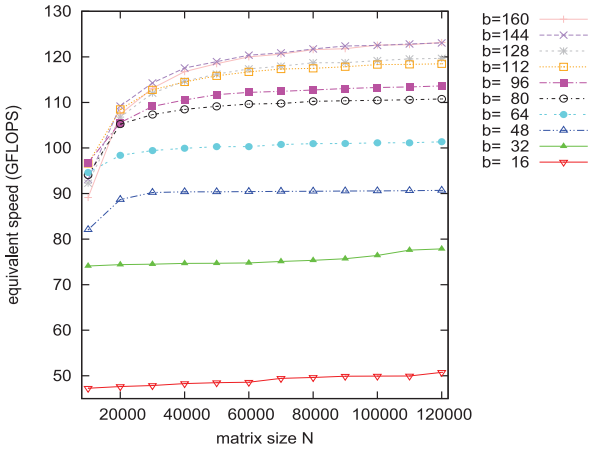


図 23　ブロック三重対角化の換算性能（Core i7-3930K（6 スレッド））

Fig. 23　Equivalent speed of block tridiagonalization (Core i7-3930K (6 thread)).

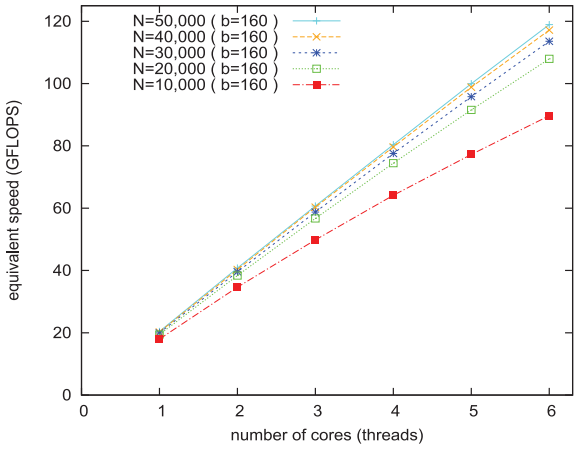


図 24　ブロック三重対角化の換算性能（Core i7-3930K）（使用スレッド数と性能の直線性，$b = 160$）

Fig. 24　Equivalent Performance of block tridiagonalization (Core i7-3930K) (Linearity of performance to the number of threads, for $b = 160$).

図 24 ではブロックサイズ b を 160 に固定して，行列サイズ N を 1 万から 5 万まで刻み 1 万で変えたそれぞれの場合に対して，横軸には並列計算に用いたコア数（スレッド数）を 1 から 6 まで 1 ずつ増やしてとり，縦軸には得られた換算性能の値をとってプロットしたグラフである．このグラフから，この実験では換算性能が使用スレッド数にほぼ比例していることが分かる．

5.8　Intel Core i7-4770 のシステムの場合

このシステムはシングル CPU で，CPU は Intel Core i7-4770（コア数 4，クロック 3.4 GHz，L1（I/D ともに）は 32 KB/コア，L2 は 256 KB/コア，共有 L3 は 8 MB）である．CPU の Hyper Thread 機能と Turbo Boost 機能は BIOS の設定でオフにした．主記憶の容量は 32 Gbyte（デュアルチャネル，DDR3-1600 MHz（PC3-12800 動作）の 8 Gbyte DIMM を 4 本）である．この CPU の内蔵 GPU の機能は使わないように BIOS で設定した．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux（Intel Fortran コンパイラ version 15.0.0, MKL ライブラリ version 11.2）で，用いたコンパイラオプションは “-fast -openmp” である．この CPU は Haswell アーキテクチャの AVX2 命令を持ち，1 コアあたり CPU クロックごとに倍精度演算を最大 16 個実行できるので，TB 機能がオフの場合には CPU の理論ピーク演算性能は 217.6 GFLOPS（倍精度演算，AVX2 命令）である．MKL の BLAS はリンクのオプション “-mkl=sequential” によりシングルスレッド版だけを使用した．

容量 32 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 9 万までである．**図 25** は行列サイズ N を 1 万から 9 万まで刻み 1 万で変えながら横軸にブロックサイズ b

を 16 から 296 まで刻み 8 で，縦軸に換算性能（GFLOPS）の値をとってプロットしたグラフである．**図 26** は同じ測定結果を，縦軸を理論ピーク演算性能に対する換算性能の比率をパーセントで表した値に変更してプロットしたグラフである．**図 27** は，ブロックサイズ b をそれぞれ 16 から 160 まで刻み 16 で変えた場合について，横軸に行列サイズを 1 万から 9 万まで刻み 1 万でとり，縦軸に換算性能（GFLOPS）をとりプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 4 スレッドで実行した．

5.9　Intel Core i7-4770R のシステムの場合

このシステム（Gigabyte BRIX Pro GB-BXi7-4770R）はシングル CPU で，CPU は Intel Core i7-4770R（コア数 4，クロック 3.2 GHz，L1（I/D ともに）は 32 KB/コア，L2 は 256 KB/コア，共有 L3 は 6 MB，共有 L4 は 128 MB）である．このシステムは BIOS からでは CPU の Hyper Thread 機能がオフにできず，OS の起動時パラメータの設定で “maxcpus=4” とすることにより，Hyper Thread 機能が動作しない状態にして作動させた．また Turbo Boost 機能もオフにしている．このシステムの CPU の Core i7-4770 との違いは，CPU の標準クロックが 3.4 GHz から 3.2 GHz に下がっていること，共有 L3 のサイズが 8 Mbyte から 6 Mbyte に減っていること，本来はグラフィックメモリである Iris Pro Graphics 5200 機能用の 128 Mbyte の eDRAM が CPU のパッケージ内に含まれ，それを CPU 側からは大きな容量の共有 L4 キャッシュとして利用できることである．また主記憶の容量は CPU では 32 Gbyte までサポートできる仕様になっているが，このシステムのマザーボード上にはメモリのソケットが 2 個しかないので 16 Gbyte（デュアルチャネル，DDR3L-1600 MHz（PC3-12800 動作）の 8 Gbyte SO-DIMM を 2 本）に制限される．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux（Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）で，用いたコンパイラオプションは “-fast -openmp” である．この CPU は（Corei7-4700 と同様に）Haswell アーキテクチャの AVX2 命令を持ち，1 コアあたり CPU クロックごとに倍精度演算を最大 16 個実行できるので，TB 機能がオフの場合には CPU の理論ピーク演算性能は 204.8 GFLOPS（倍精度演算，AVX2 命令）である．MKL の BLAS はリンクのオプション “-mkl=sequential” によりシングルスレッド版だけを使用した．

容量 16 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 6 万までである．**図 28** は行列サイズ N を 1 万から 6 万まで刻み 1 万で変えながら横軸にブロックサイズ b を 16 から 296 まで刻み 8 で，縦軸に換算性能（GFLOPS）

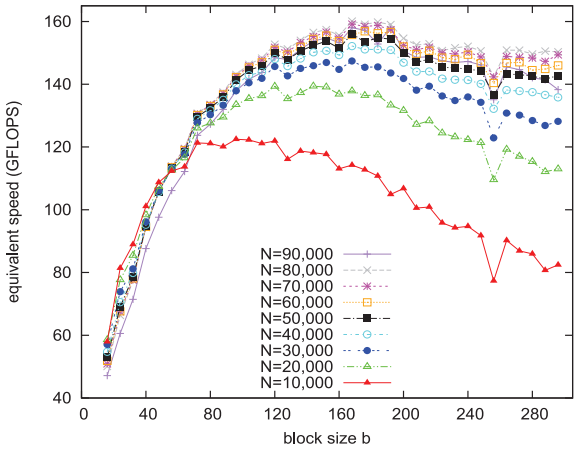


図 25　ブロック三重対角化の換算性能（Core i7-4770（4 スレッド））
Fig. 25　Equivalent speed of block tridiagonalization (Core i7-4770 (4 thread)).

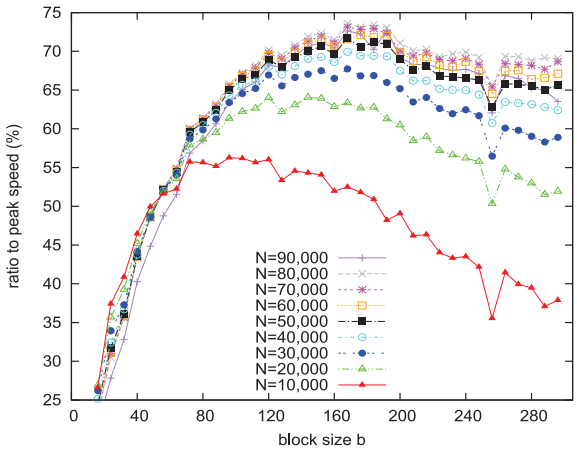


図 26　換算性能の理論ピーク性能に対する比率（Core i7-4770（4 スレッド））
Fig. 26　Ratio of equivalent speed to theoretical peak performance (Core i7-4770 (4 thread)).

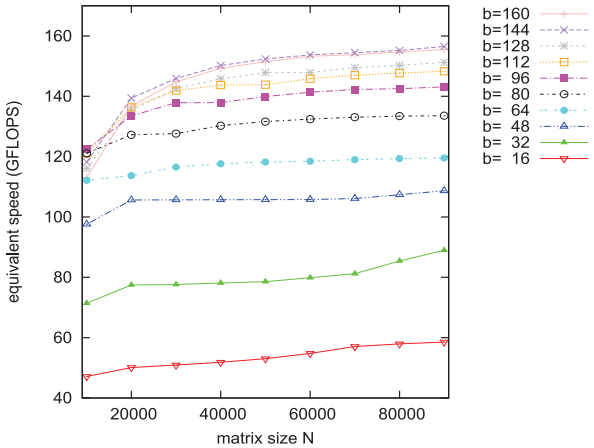


図 27　ブロック三重対角化の換算性能（Core i7-4770（4 スレッド））
Fig. 27　Equivalent speed of block tridiagonalization (Core i7-4770 (4 thread)).

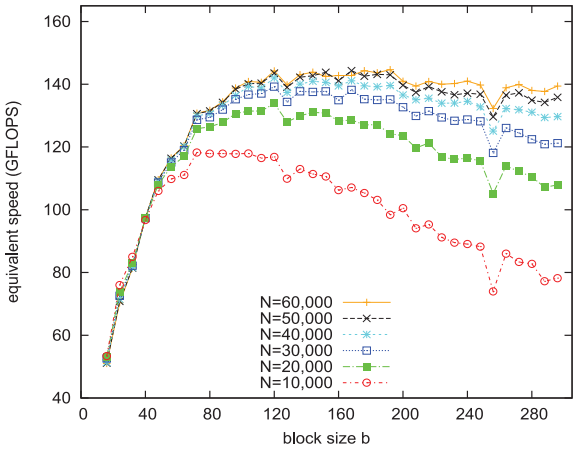


図 28　ブロック三重対角化の換算性能（Core i7-4770R（4 スレッド））
Fig. 28　Equivalent speed of block tridiagonalization (Core i7-4770R (4 thread)).

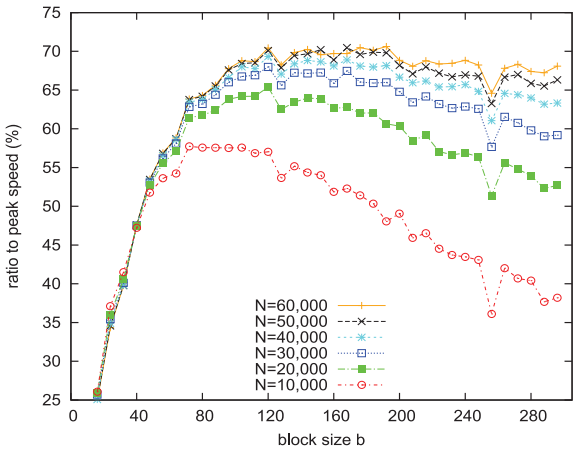


図 29　換算性能の理論ピーク性能に対する比率（Core i7-4770R（4 スレッド））
Fig. 29　Ratio of equivalent speed to theoretical peak performance (Core i7-4770R (4 thread)).

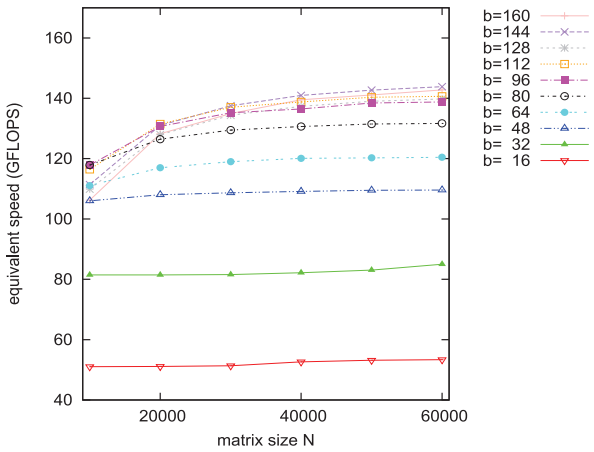


図 30　ブロック三重対角化の換算性能（Core i7-4770R（4 スレッド））
Fig. 30　Equivalent speed of block tridiagonalization (Core i7-4770R (4 thread)).

の値をとってプロットしたグラフである．**図 29** は同じ測定結果を，縦軸を理論ピーク演算性能に対する換算性能の比率をパーセントで表した値に変更してプロットしたグラフである．**図 30** は，ブロックサイズ b をそれぞれ 16 から 160 まで刻み 16 で変えた場合について，横軸に行列サイズを 1 万から 6 万まで刻み 1 万でとり，縦軸に換算性能（GFLOPS）をとりプロットしたグラフである．すべての計算は OpenMP を用いて並列化し，CPU のコア数と同じ 4 スレッドで実行した．

5.10　AMD A10-7850K のシステムの場合

このシステムはシングル CPU で，CPU は AMD A10-7850K （コア数は 4（Steamroller モジュールが 2 個），クロックは 3.7 GHz，データ L1 は 64 KB（16 KB/コア），L2 は 4 MB（2 MB/モジュール））である．TURBO 機能は BIOS の設定でオフにした．主記憶の容量は 32 Gbyte（デュアルチャネル，DDR3-2133（PC3-17000），CL10 の 8 Gbyte DIMM を 4 本）である．OS は CentOS 7.0 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）で，用いたコンパイラオプションは “-fast -openmp” である．この CPU は内蔵する Steamroller モジュールごとに倍精度演算を CPU クロックごとに最大 8 個実行できるので，TURBO 機能がオフの場合の CPU 機能の理論ピーク演算性能は 59.2 GFLOPS（倍精度演算）である．この CPU の特徴である HSA 機能による内蔵 GPU 機能の演算能力は使用していない．このシステムでは，ブロックサイズ b の値が 12 の倍数のときにその前後の値と比べて性能がかなり良いことが実験の結果および MKL の xGEMM による行列乗算単独の試験から分かったので，計算は b が 12 の倍数の場合についてだけ行った．

ブロック三重対角化の計算を対称性を用いて容量 32 Gbyte の主記憶上で行える行列のサイズ N は，1 万の倍数であれば 9 万までである．**図 31** は行列サイズ N を 1 万から 9 万まで刻み 1 万で変えたそれぞれの場合について，横軸にブロックサイズ b を 12 から 300 まで刻み 12 でとり，縦軸に得られた換算性能（GFLOPS）をとってプロットしたグラフであり，**図 32** は同じ測定結果を縦軸を変更して，CPU 機能の理論ピーク演算性能に対する換算性能の比率をパーセント値でプロットしたグラフである．**図 33** はブロックサイズ b を 12 から 300 まで刻み 12 で変えたそれぞれの場合について，横軸に行列サイズ N を 1 万から 9 万まで刻み 1 万でとり，縦軸に換算性能（GFLOPS）をとりプロットしたグラフである．図 31 から図 33 はいずれも OpenMP による並列化で CPU 機能のコア数と等しい 4 スレッド（Steamroller モジュールあたり 2 スレッド）で実行した結果である．

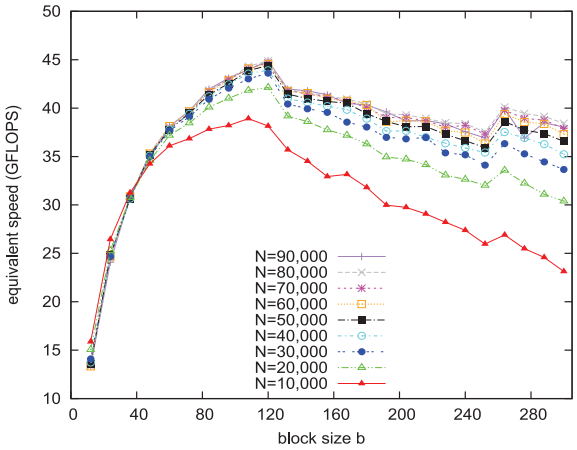


図 31　ブロック三重対角化の換算性能（A10-7850K（4 スレッド））
Fig. 31　Equivalent speed of block tridiagonalization (A10-7850K (4 thread)).

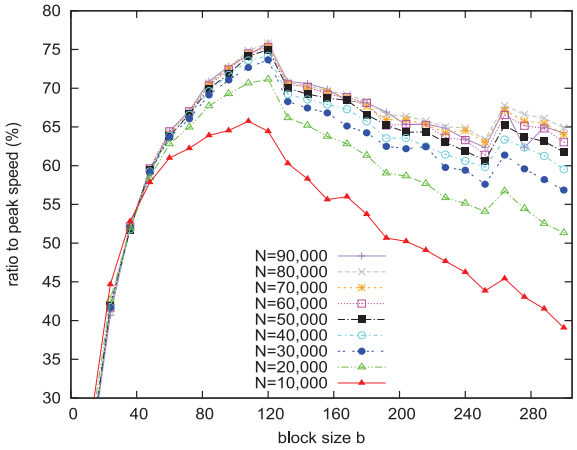


図 32　換算性能の理論ピーク性能に対する比率（A10-7850K（4 スレッド））
Fig. 32　Ratio of equivalent speed to theoretical peak performance (A10-7850K (4 thread)).

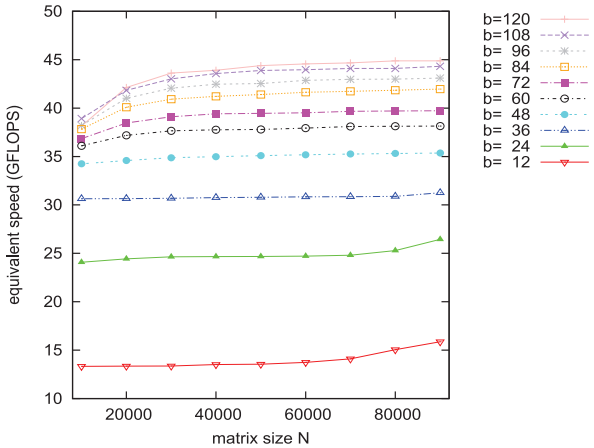


図 33　ブロック三重対角化の換算性能（A10-7850K（4 スレッド））
Fig. 33　Equivalent speed of block tridiagonalization (A10-7850K (4 thread)).

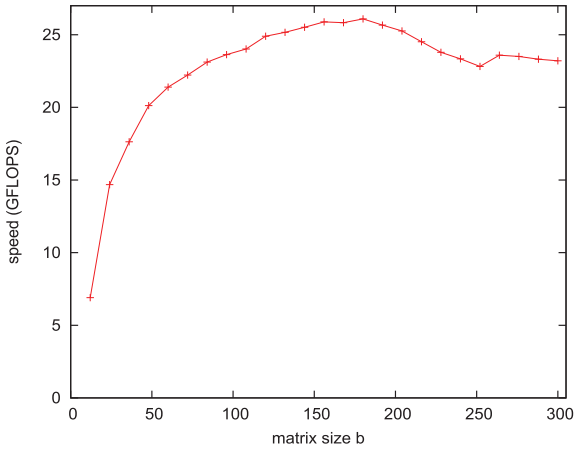


図 34　b 次の正方行列の DGEMM による積和の演算速度（A10-7850K（1 スレッド）），b は 12 の倍数の場合
Fig. 34　Speed of multiply-and-add of small size b matrices (A10-7850K (1 thread)), for b is multiple of 12.

性能のピークが他の CPU の場合よりも小さい $b = 120$ の付近にあり，しかも明瞭であるのは，このシステムの最外側の L2 キャッシュの容量が 4 MB であり他の CPU に比べて小さいためであろう．これについて調査するために小さい正方行列どうしの積和（DGEMM）の演算性能を測ってみた．**図 34** は，横軸に行列の次数 b の値を 12 から 300 まで 12 刻みで，縦軸に MKL で 1 コア（1 スレッド）を用いた場合の行列の積和 $S := S + X * Y$（DGEMM）の演算速度（GFLOPS，倍精度）をそれぞれグラフにプロットしたものである．1 コアでの理論ピーク演算性能は 29.6 GFLOPS（倍精度）である．すでに $b = 120$ 付近で演算速度が飽和している傾向が見え，$b = 170$ 付近から上では b を増すと演算速度が減少している．このことからも，この CPU では行列積を MKL を用いて計算する場合には，キャッシュを奪い合うことのない 1 コア（1 スレッド）の場合であっても行列の次数 $b = 200$ を超えるぐらいで，演算器へのデータの供給能力が少し不足するようである．

5.11　Intel Atom c2570 のシステムの場合

このシステムはシングル CPU で，CPU は Intel Atom C2570（コア数 8 = 4 モジュール，クロック 2.4 GHz，L1（I）は 32 KB/コア，L1（D）は 24 KB/コア，L2 は 1 MB/モジュール，L3 はなし）である．この CPU には Hyper Thread 機能がなく，Turbo Boost 機能は BIOS の設定でオフにした．この CPU はたとえば 2 コア単位でモジュールとなっていて L2 キャッシュを共有しているほか，L1（D）のサイズが 32 KB/コアから 24 KB/コアに減少していることなど，これまでの Intel Core アーキテクチャとはかなり異なっている．システムの主記憶の容量は 32 Gbyte（デュアルチャネル，DDR3-1600 MHz（PC3-12800），CL9，8 Gbyte DIMM を 4 本）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE

2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）で，用いたコンパイラオプションは “-fast -openmp” である．この Silvermont アーキテクチャの CPU には SSE4.2 演算のできる 128 bit 演算機がコアあたり 1 個あるとされる．この CPU の理論上の浮動小数点数演算の最大性能値については資料を探したが，現時点では明確に述べている資料を見つけることができなかった．次数が千～数千の大きな行列に対して MKL version 11.2 の SGEMM，DGEMM を用いて行列乗算（$S := S + A * B$ や $S := S + A^T * B$）を行って得られた演算性能の最高値は 62 GFLOPS（単精度），18.1 GFLOPS（倍精度）であり，両者の違いは 3.4 倍である．仮に CPU コアあたりで可能な倍精度演算の速度を 1 FP/CLK であるとすれば，クロックが 2.4 GHz で 8 コアのときには 19.2 GFLOPS となるので，今回はとりあえずこの値を（真の値は現時点では不明であるが）理論上の最大演算性能値であると仮定する（これはあくまでも推定値である）．MKL の BLAS ルーチンはリンクオプション “-mkl=sequentual” によりシングルスレッド版だけを使用した．

容量 32 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 9 万までである．**図 35** は行列サイズ N を 1 万から 9 万まで 1 万刻みで変えたそれぞれの場合について，横軸にブロックサイズ b を 16 から 200 まで刻み 8 でとり，縦軸には得られた換算性能（GFLOPS）をとってプロットしたグラフであり，**図 36** は同じ測定結果を縦軸を変更して，CPU の理論ピーク演算性能（あくまでも仮定した値 19.2 GFLOP）に対する換算性能の比率をパーセント値でプロットしたグラフである．**図 37**）ではブロックサイズ b を 16 から 160 まで刻み 16 で変えたそれぞれの場合について，横軸には行列サイズ N を 1 万から 9 万まで刻み 1 万でとり，縦軸には換算性能（GFLOPS）をとりプロットしたグラフである．図 35 から図 37 はいずれも OpenMP により並列化し，CPU のコア数と等しい 8 スレッドで実行した結果である．

5.12　Intel Core i7-5960X のシステムの場合

このシステムはシングル CPU で，CPU は Intel Core i7-5960X（コア数 8，クロック 3.0 GHz，L1（I/D ともに）は 32 KB/コア，L2 は 256 KB/コア，共有 L3 は 20 MB）であり，CPU の Hyper Thread 機能と Turbo Boost 機能は BIOS の設定でオフにした．主記憶の容量は 64 Gbyte（クアドチャネル，DDR4-2133 MHz（PC4-17000）の 8 Gbyte DIMM を 8 本）である．OS は CentOS 7.0 for x86_64 で，コンパイラと BLAS ライブラリは Intel Parallel Studio XE 2015 for 64 bit Linux （Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）で，用いたコンパ

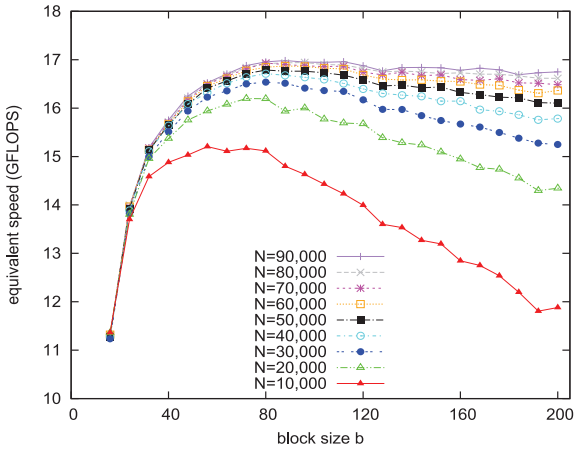


図 35　ブロック三重対角化の換算性能（Atom C2570 (8 スレッド)）
Fig. 35　Equivalent speed of block tridiagonalization (Atom C2570 (8 thread)).

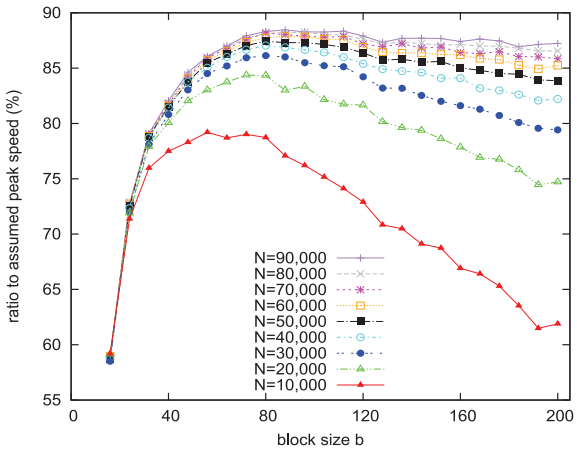


図 36　換算性能の理論ピーク性能（仮定した値）に対する比率（Atom C2570（8 スレッド））
Fig. 36　Ratio of equivalent speed to theoretical peak performance (Atom C2570 (8 thread)).

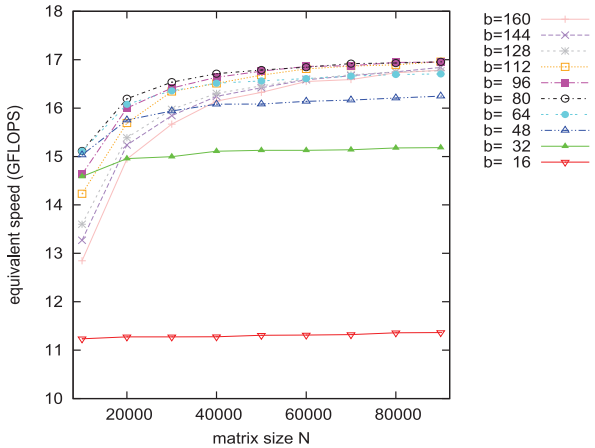


図 37　ブロック三重対角化の換算性能（Atom C2570 (8 スレッド)）
Fig. 37　Equivalent speed of block tridiagonalization (Atom C2570 (8 thread)).

イラオプションは “-fast -openmp” である．この CPU は Haswell-E アーキテクチャで AVX2 命令により，1 コアあたり CPU クロックごとに倍精度の演算を最大で 16 個実行できるので，TB 機能がオフの場合の CPU の理論ピーク演算性能は 384.0 GFLOPS （倍精度演算，AVX2 命令）である．MKL の BLAS ルーチンはリンクオプション “-mkl=sequentual” によりシングルスレッド版だけを使用した．

容量 64 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える行列のサイズ N は，1 万の倍数であれば 12 万までである．**図 38** は行列サイズ N を 1 万，2 万，3 万，5 万，7 万，9 万，12 万と変えたそれぞれの場合について，横軸にブロックサイズ b を 16 から 296 まで刻み 8 でとり，縦軸には得られた換算性能（GFLOPS）をとってプロットしたグラフであり，**図 39** は同じ測定結果を縦軸を変更して，CPU の理論ピーク演算性能に対する換算性能の比率をパーセント値でプロットしたグラフである．**図 40** はブロックサイズ b を 16 から 160 まで刻み 16 で変えたそれぞれの場合について，横軸には行列サイズ N を 1 万から 12 万まで刻み 1 万でとり，縦軸には換算性能（GFLOPS）をとりプロットしたグラフである．図 38 から図 40 はいずれも OpenMP により並列化し，CPU のコア数と等しい 8 スレッドで実行した結果である．

図 41 はブロックサイズ b を 160 に固定して，行列サイズ N を 1 万から 5 万まで刻み 1 万で変えたそれぞれの場合に対して，横軸には並列計算に用いたコア数（スレッド数）を 1 から 8 まで 1 ずつ増やしてとり，縦軸には得られた換算性能の値をとってプロットしたグラフである．このグラフから，この実験では換算性能が使用スレッド数にほぼ比例していることが分かる．また，**図 42** は，経過時間から TS-QR 分解に費やした部分を引き去ったことに相当する「仮想的な場合」の換算性能を，図 41 と同様にプロットしたグラフである．

5.13　得られた近似固有対の検証例

N 次の実対称な Frank 行列の行列要素は $a_{i,j} = N + 1 - \max(i, j)$ で与えられる．このテスト行列の厳密な固有値は昇順に，簡単な数式 $\lambda_k = 0.25 \Big/ \sin^2 \left\{ \dfrac{(N-k+0.5)\,\pi}{2N+1} \right\}$, $k=1, 2, \ldots, N$ となることを用いて計算できて，次数 N が大きいほど最小側の固有値が下界 0.25 の近くに密集する性質がある．

表 1 に，次数 $N = 10,000$ の Frank 行列のブロック三重対角化（ブロックサイズ $b = 100$）を経由することで求めた最小側 100 個の近似固有値（Solved Value）とそれに対応する厳密値（Exact Value），近似固有値の誤差（Error：計算値から厳密値を引いた値）を掲げる．ブロック三重対角行列の固有値は，帯化の後に村田の帯 Householder 法を用いて真

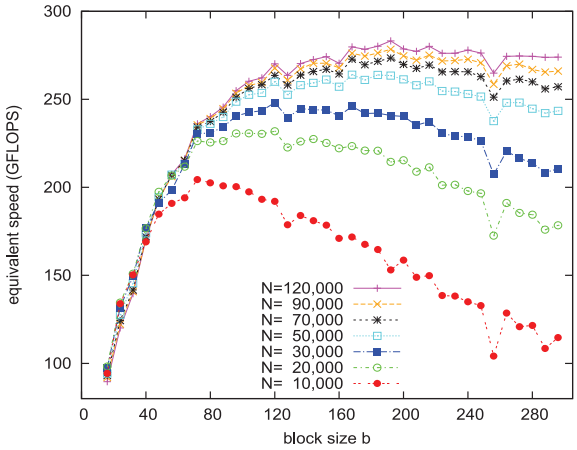


図 38　ブロック三重対角化の換算性能（Core i7-5960X（8 スレッド））
Fig. 38　Equivalent speed of block tridiagonalization (Core i7-5960X (8 thread)).

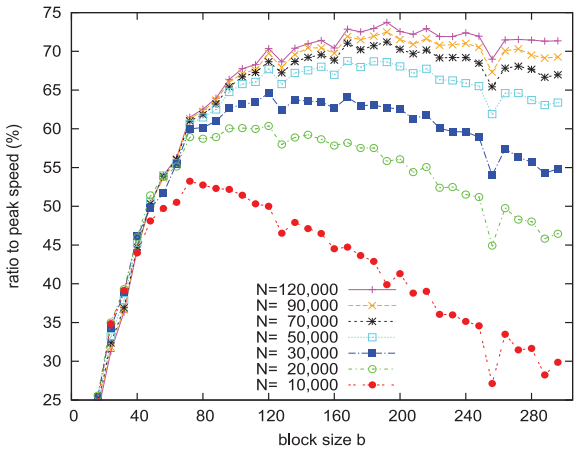


図 39　換算性能に対する理論ピーク性能の比率（Core i7-5960X（8 スレッド））
Fig. 39　Ratio of equivalent speed to theoretical peak performance (Core i7-5960X (8 thread)).

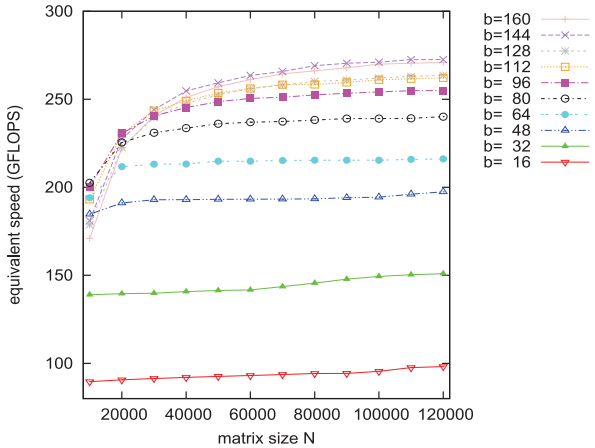


図 40　ブロック三重対角化の換算性能（Core i7-5960X（8 スレッド））
Fig. 40　Equivalent speed of block tridiagonalization (Core i7-5960X (8 thread)).

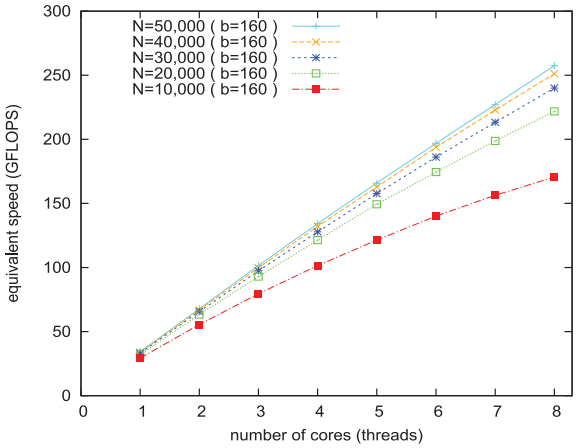


図 41　ブロック三重対角化の換算性能（Core i7-5960X）（使用スレッド数と性能の直線性，$b = 160$）

Fig. 41　Equivalent Performance of block tridiagonalization (Core i7-5960X) (Linearity of performance to the number of threads, for $b = 160$).

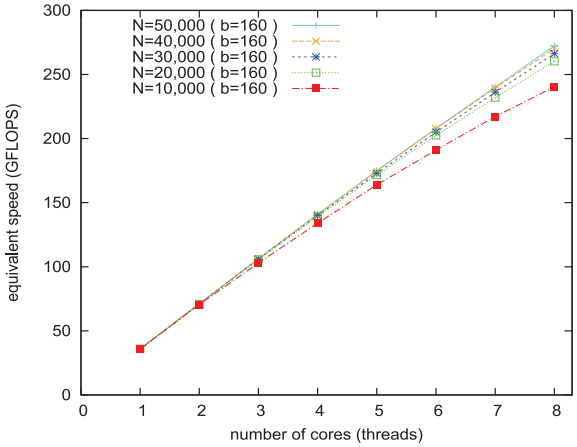


図 42　TS-QR 分解の処理を仮想的に除いた場合のブロック三重対角化の換算性能（Core i7-5960X）
（使用スレッド数と性能の直線性，$b = 160$）

Fig. 42　Equivalent Performance of block tridiagonalization when TS-QR decomposition is virtually removed (Core i7-5960X) (Linearity of performance to the number of threads, for $b = 160$).

の三重対角行列へ変換して，Sturm 二分法により求めている（使用した計算システムは CPU が Corei7-3930K で，コンパイラは Intel Fortran version 14.0.2，BLAS3 には Intel MKL versio 11.1（update2）を用いた．OpenMP による 6 スレッド並列の実行結果である．MKL の BLAS ルーチンはライブラリのリンク時の指定 “-mkl=sequential” によりシングルスレッド動作のものだけを利用している）．表から 100 個の近似固有値の誤差は 10^{-12} 程度であることが分かる．Frank 行列の要素の絶対値の最大値は N で，今の場合それは 10^4 であるから，固有値の精度は倍精度演算による計算結果としては十分良いといえる．また，これら 100 個の固有対のベクトルの間の正規直交性の精度の崩れの大きさである $\max_{i,j} |\mathbf{v}_i^T \mathbf{v}_j - \delta_{i,j}|$ は 10^{-12} から 10^{-11} の程度であった．

同様に次数 $N = 10,000$ の Frank 行列のブロック三重対角化（ブロックサイズ $b = 100$）を経由して近似固有対 $(\lambda_k, \mathbf{v}_k)$ を固有値の最大側から 100 個（$N-k+1 = 1, 2, \ldots, 100$）求めた．これら 100 個の固有対のベクトルに対する正規直交性の精度の崩れの大きさ $\max_{i,j} |\mathbf{v}_i^T \mathbf{v}_j - \delta_{i,j}|$ は 2.7×10^{-15} であった．近似対に対する残差ベクトル $\mathbf{r}_k = (A - \lambda_k I)\mathbf{v}_k$ の最大ノルムを残差の大きさ μ_k とし，そうして相対残差の大きさ ρ_k を残差の大きさの固有値の大きさに対する比 $\mu_k/|\lambda_k|$ と定義して近似固有対の誤差の目安とする．**表 2** にそれらの値（$N-k+1$，λ_k，μ_k，ρ_k）を掲げる．これら 100 個の近似対は相対残差の大きさ ρ_k が 10^{-12} 程度以下で求まっていることが分かる．

5.14　実験で見られたブロックサイズ b の増大に対する性能低下についての考察

今回の実験では，ブロックサイズ b を増していく場合に性能の低下傾向が，特に行列の次数 N が 1 万のように小さい場合には，顕著になる．その主な原因は，ブロック鏡映軸を構成するために今回用いた T-S 型行列の QR 分解（TS-QR 分解）の算法が，その再帰的な構成の内側で用いている HQR 分解法が単純な BLAS2 だからである．そのため，列数 b が大きくなると内側の HQR 分解法で扱う行列が CPU の最外側の共有キャッシュ（スレッド並列処理の場合はスレッド間で共有もしている）の容量に収まらなくなり，キャッシュと主記憶の間で記憶の転送の頻繁なやりとり（スラッシング）が生じて計算性能が低下する（HQR 分解が扱う小さい行列の行の数は列の数 b よりもある程度の比率で大きくなければ，再帰処理が有効にはならないので，b を増やすときには行の数も b に比例する程度には増やす必要がある）．そのため今回の実験で使用した（BLAS2 の HQR 分解を再帰的に用いている）TS-QR 分解は，列の数 b が大きくなって HQR 分解の処理がキャッシュに収まらなくなると処理速度が低下する．

実際に今回の実装による TS-QR 分解がブロック三重対角化の処理の（換算）性能の低下を起こしていることを確かめる簡単な方法は，計算時間に TS-QR 分解を含めて求めた「換算性能」と，含めずに求めた「換算性能」とを比較してみることである．経過時間に含めない場合の測定は，TS-QR 分解のルーチンをダミーのルーチンで差し替えて（特別な関係を持たない）適当な「偽の値」の Q や R を処理をしたふりをして時間をほとんどかけずに返すことで実現できる（もちろんこれは単に経過時間を計測し比較するためだけの計算であって，計算で得られるブロック三重対角行列の固有対などは意味がないものになる）．

以下では例として，Core2 Quad Q6600 のシステムだけを取り上げて示すが，他のシステムについても TS-QR 分解を省いた場合の「換算性能」のグラフは同じような傾

表 1　100 個の最小側固有値と厳密値（Frank 行列，N=10,000，ブロックサイズ 100）

Table 1　Solved smallest 100 eigenvalues and exact ones (Frank Matrix, N=10,000, block size 100).

k	Solved Value	Exact Value	Error
1	0.25000 00061 692609	0.25000 00061 678860	1.4E-12
2	0.25000 00246 721549	0.25000 00246 715454	6.1E-13
3	0.25000 00555 132418	0.25000 00555 109818	2.3E-12
4	0.25000 00986 905135	0.25000 00986 862011	4.3E-12
5	0.25000 01541 989320	0.25000 01541 972121	1.7E-12
6	0.25000 02220 444571	0.25000 02220 440256	4.3E-13
7	0.25000 03022 290092	0.25000 03022 266549	2.4E-12
8	0.25000 03947 441301	0.25000 03947 451161	-9.9E-13
9	0.25000 04996 021814	0.25000 04995 994273	2.8E-12
10	0.25000 06167 884255	0.25000 06167 896092	-1.2E-12
11	0.25000 07463 165081	0.25000 07463 156849	8.2E-13
12	0.25000 08881 791201	0.25000 08881 776799	1.4E-12
13	0.25000 10423 764134	0.25000 10423 756224	7.9E-13
14	0.25000 12089 063146	0.25000 12089 095428	-3.2E-12
15	0.25000 13877 770408	0.25000 13877 794738	-2.4E-12
16	0.25000 15789 824589	0.25000 15789 854508	-3.0E-12
17	0.25000 17825 280318	0.25000 17825 275116	5.2E-13
18	0.25000 19984 057043	0.25000 19984 056963	8.0E-15
19	0.25000 22266 200470	0.25000 22266 200475	-5.0E-16
20	0.25000 24671 729524	0.25000 24671 706103	2.3E-12
21	0.25000 27200 588683	0.25000 27200 574322	1.4E-12
22	0.25000 29852 819131	0.25000 29852 805630	1.4E-12
23	0.25000 32628 404925	0.25000 32628 400551	4.4E-13
24	0.25000 35527 378547	0.25000 35527 359634	1.9E-12
25	0.25000 38549 697088	0.25000 38549 683450	1.4E-12
26	0.25000 41695 399557	0.25000 41695 372595	2.7E-12
27	0.25000 44964 473930	0.25000 44964 427691	4.6E-12
28	0.25000 48356 853263	0.25000 48356 849382	3.9E-13
29	0.25000 51872 643527	0.25000 51872 638339	5.2E-13
30	0.25000 55511 832225	0.25000 55511 795255	3.7E-12
31	0.25000 59274 322725	0.25000 59274 320849	1.9E-13
32	0.25000 63160 221058	0.25000 63160 215864	5.2E-13
33	0.25000 67169 468336	0.25000 67169 481064	-1.3E-12
34	0.25000 71302 158542	0.25000 71302 117245	4.1E-12
35	0.25000 75558 102203	0.25000 75558 125220	-2.3E-12
36	0.25000 79937 549007	0.25000 79937 505829	4.3E-12
37	0.25000 84440 284490	0.25000 84440 259936	2.5E-12
38	0.25000 89066 405657	0.25000 89066 388431	1.7E-12
39	0.25000 93815 878193	0.25000 93815 892227	-1.4E-12
40	0.25000 98688 822437	0.25000 98688 772261	5.0E-12
41	0.25001 03685 030822	0.25001 03685 029495	1.3E-13
42	0.25001 08804 676391	0.25001 08804 664915	1.1E-12
43	0.25001 14047 699116	0.25001 14047 679532	2.0E-12
44	0.25001 19414 100316	0.25001 19414 074380	2.6E-12
45	0.25001 24903 856389	0.25001 24903 850520	5.9E-13
46	0.25001 30517 019147	0.25001 30517 009033	1.0E-12
47	0.25001 36253 589020	0.25001 36253 551030	3.8E-12
48	0.25001 42113 536196	0.25001 42113 477641	5.9E-12
49	0.25001 48097 043577	0.25001 48096 790023	2.5E-11
50	0.25001 54203 761025	0.25001 54203 489358	2.7E-11

表 1（続き）

Table 1 (Continued)

k	Solved Value	Exact Value	Error
51	0.25001 60433 710860	0.25001 60433 576851	1.3E-11
52	0.25001 66787 257477	0.25001 66787 053731	2.0E-11
53	0.25001 73264 049605	0.25001 73263 921253	1.3E-11
54	0.25001 79864 334650	0.25001 79864 180695	1.5E-11
55	0.25001 86588 059163	0.25001 86587 833361	2.3E-11
56	0.25001 93435 134211	0.25001 93434 880576	2.5E-11
57	0.25002 00405 486602	0.25002 00405 323694	1.6E-11
58	0.25002 07499 358710	0.25002 07499 164090	1.9E-11
59	0.25002 14716 552093	0.25002 14716 403163	1.5E-11
60	0.25002 22057 175627	0.25002 22057 042339	1.3E-11
61	0.25002 29521 257237	0.25002 29521 083067	1.7E-11
62	0.25002 37108 651452	0.25002 37108 526820	1.2E-11
63	0.25002 44819 662585	0.25002 44819 375097	2.9E-11
64	0.25002 52653 860216	0.25002 52653 629418	2.3E-11
65	0.25002 60611 508857	0.25002 60611 291332	2.2E-11
66	0.25002 68692 494254	0.25002 68692 362407	1.3E-11
67	0.25002 76896 852398	0.25002 76896 844240	8.2E-13
68	0.25002 85224 789599	0.25002 85224 738450	5.1E-12
69	0.25002 93676 084182	0.25002 93676 046682	3.7E-12
70	0.25003 02250 767262	0.25003 02250 770603	-3.3E-13
71	0.25003 10948 944987	0.25003 10948 911907	3.3E-12
72	0.25003 19770 510064	0.25003 19770 472310	3.8E-12
73	0.25003 28715 520827	0.25003 28715 453553	6.7E-12
74	0.25003 37783 943115	0.25003 37783 857404	8.6E-12
75	0.25003 46975 843832	0.25003 46975 685651	1.6E-11
76	0.25003 56291 090212	0.25003 56290 940110	1.5E-11
77	0.25003 65729 755171	0.25003 65729 622619	1.3E-11
78	0.25003 75291 948525	0.25003 75291 735042	2.1E-11
79	0.25003 84977 508733	0.25003 84977 279267	2.3E-11
80	0.25003 94786 374807	0.25003 94786 257206	1.2E-11
81	0.25004 04718 757659	0.25004 04718 670795	8.7E-12
82	0.25004 14774 571303	0.25004 14774 521993	4.9E-12
83	0.25004 24953 878226	0.25004 24953 812789	6.5E-12
84	0.25004 35256 615556	0.25004 35256 545190	7.0E-12
85	0.25004 45682 727088	0.25004 45682 721232	5.9E-13
86	0.25004 56232 330044	0.25004 56232 342971	-1.3E-12
87	0.25004 66905 392238	0.25004 66905 412491	-2.0E-12
88	0.25004 77701 939974	0.25004 77701 931900	8.1E-13
89	0.25004 88621 908222	0.25004 88621 903327	4.9E-13
90	0.25004 99665 318268	0.25004 99665 328931	-1.1E-12
91	0.25005 10832 233701	0.25005 10832 210890	2.3E-12
92	0.25005 22122 552830	0.25005 22122 551410	1.4E-13
93	0.25005 33536 365788	0.25005 33536 352720	1.3E-12
94	0.25005 45073 630800	0.25005 45073 617073	1.4E-12
95	0.25005 56734 353283	0.25005 56734 346747	6.5E-13
96	0.25005 68518 546985	0.25005 68518 544043	2.9E-13
97	0.25005 80426 251668	0.25005 80426 211290	4.0E-12
98	0.25005 92457 398557	0.25005 92457 350838	4.8E-12
99	0.25006 04612 046286	0.25006 04611 965063	8.1E-12
100	0.25006 16890 080611	0.25006 16890 056364	2.4E-12

表 2　最大側固有値を持つ近似対 100 個の絶対残差と相対残差
Table 2　Absolute and relative residuals of 100 eigenpairs whose values are largest.

$N-k+1$	λ_k	μ_k	ρ_k
1	4.0532526488935396E+07	2.2E-07	5.5E-15
2	4.5036141284002308E+06	3.4E-08	7.5E-15
3	1.6213011395574124E+06	1.2E-08	7.4E-15
4	8.2719449977419653E+05	6.6E-09	7.9E-15
5	5.0040164389632968E+05	5.4E-09	1.1E-14
6	3.3497964040443662E+05	4.2E-09	1.2E-14
7	2.3983751768602489E+05	2.9E-09	1.2E-14
8	1.8014464513603126E+05	3.6E-09	2.0E-14
9	1.4025103975413094E+05	7.7E-09	5.5E-14
10	1.1227854983088264E+05	3.7E-09	3.3E-14
11	9.1910574049029630E+04	2.9E-09	3.1E-14
12	7.6621116236226851E+04	3.3E-09	4.3E-14
13	6.4852125582360895E+04	2.2E-09	3.4E-14
14	5.5600256729296132E+04	3.4E-09	6.1E-14
15	4.8195715206906170E+04	2.3E-09	4.8E-14
16	4.2177530165484219E+04	2.1E-09	5.0E-14
17	3.7220034119122211E+04	1.9E-09	5.2E-14
18	3.3087859991093588E+04	2.8E-09	8.4E-14
19	2.9607480269633375E+04	2.1E-09	7.0E-14
20	2.6648687150453530E+04	1.9E-09	7.1E-14
21	2.4112234675327774E+04	1.7E-09	7.1E-14
22	2.1921406430117248E+04	1.6E-09	7.3E-14
23	2.0016145756060945E+04	4.1E-09	2.0E-13
24	1.8348895649360369E+04	2.3E-09	1.3E-13
25	1.6881602036454911E+04	4.4E-09	2.6E-13
26	1.5583522935908295E+04	1.6E-09	1.0E-13
27	1.4429605015929694E+04	2.5E-09	1.7E-13
28	1.3399265616488077E+04	2.7E-09	2.0E-13
29	1.2475468500057621E+04	2.6E-09	2.1E-13
30	1.1644015079052420E+04	3.1E-09	2.6E-13
31	1.0892995563117307E+04	2.9E-09	2.7E-13
32	1.0212360079924802E+04	1.7E-09	1.7E-13
33	9.5935807078746639E+03	2.1E-09	2.2E-13
34	9.0293830454460149E+03	1.5E-09	1.7E-13
35	8.5135314341383128E+03	1.6E-09	1.9E-13
36	8.0406559197674669E+03	1.6E-09	2.0E-13
37	7.6061119331684968E+03	3.3E-09	4.4E-13
38	7.2058658060186426E+03	1.7E-09	2.3E-13
39	6.8364008251896448E+03	1.4E-09	2.0E-13
40	6.4946397200672791E+03	1.8E-09	2.8E-13
41	6.1778803779953414E+03	1.7E-09	2.7E-13
42	5.8837422693880308E+03	2.2E-09	3.7E-13
43	5.6101215909068696E+03	2.9E-09	5.2E-13
44	5.3551535422766410E+03	2.1E-09	3.9E-13
45	5.1171804690551435E+03	1.4E-09	2.8E-13
46	4.8947248515862102E+03	1.7E-09	3.4E-13
47	4.6864663155616363E+03	1.7E-09	3.5E-13
48	4.4912219941616759E+03	1.7E-09	3.7E-13
49	4.3079296947642233E+03	1.6E-09	3.6E-13
50	4.1356334216384039E+03	1.6E-09	3.9E-13

表 2（続き）
Table 2 (Continued)

$N-k+1$	λ_k	μ_k	ρ_k
51	3.9734708851715050E+03	2.0E-09	5.0E-13
52	3.8206626921016536E+03	1.7E-09	4.4E-13
53	3.6765029630928557E+03	2.1E-09	5.6E-13
54	3.5403511662514038E+03	1.8E-09	5.1E-13
55	3.4116249897691750E+03	1.6E-09	4.8E-13
56	3.2897941052843739E+03	1.4E-09	4.3E-13
57	3.1743746969769036E+03	1.6E-09	4.9E-13
58	3.0649246508065244E+03	2.6E-09	8.6E-13
59	2.9610393144032769E+03	2.9E-09	9.7E-13
60	2.8623477515395366E+03	1.4E-09	4.9E-13
61	2.7685094263351393E+03	1.7E-09	6.2E-13
62	2.6792112617577386E+03	1.6E-09	5.9E-13
63	2.5941650248981687E+03	1.4E-09	5.5E-13
64	2.5131049981821443E+03	1.5E-09	6.0E-13
65	2.4357859013323059E+03	1.9E-09	7.7E-13
66	2.3619810336933147E+03	1.7E-09	7.1E-13
67	2.2914806106113010E+03	1.6E-09	7.1E-13
68	2.2240902710424343E+03	1.4E-09	6.5E-13
69	2.1596297365416558E+03	1.3E-09	5.9E-13
70	2.0979316043327135E+03	1.8E-09	8.5E-13
71	2.0388402593549113E+03	2.2E-09	1.1E-12
72	1.9822108920692190E+03	1.7E-09	8.6E-13
73	1.9279086104342036E+03	1.5E-09	7.9E-13
74	1.8758076358741819E+03	1.1E-09	5.7E-13
75	1.8257905742806647E+03	1.2E-09	6.3E-13
76	1.7777477541502963E+03	1.4E-09	8.0E-13
77	1.7315766248851319E+03	1.2E-09	7.1E-13
78	1.6871812090852377E+03	1.7E-09	1.0E-12
79	1.6444716033670684E+03	1.5E-09	9.1E-13
80	1.6033635228553430E+03	3.4E-09	2.1E-12
81	1.5637778850356249E+03	1.1E-09	7.1E-13
82	1.5256404291280896E+03	1.4E-09	9.5E-13
83	1.4888813675591850E+03	1.5E-09	9.8E-13
84	1.4534350664740343E+03	1.6E-09	1.1E-12
85	1.4192397525574786E+03	1.0E-09	7.1E-13
86	1.3862372437160718E+03	1.0E-09	7.5E-13
87	1.3543727014273538E+03	9.6E-10	7.1E-13
88	1.3235944027870164E+03	9.8E-10	7.4E-13
89	1.2938535304830154E+03	9.9E-10	7.6E-13
90	1.2651039791041233E+03	2.1E-09	1.7E-12
91	1.2373021763460129E+03	1.0E-09	8.4E-13
92	1.2104069178206403E+03	1.1E-09	9.0E-13
93	1.1843792142980869E+03	8.7E-10	7.3E-13
94	1.1591821503229867E+03	8.1E-10	7.0E-13
95	1.1347807532481079E+03	1.0E-09	9.0E-13
96	1.1111418718168457E+03	8.6E-10	7.7E-13
97	1.0882340635069797E+03	9.8E-10	9.0E-13
98	1.0660274899209364E+03	1.1E-09	1.0E-12
99	1.0444938195719944E+03	1.8E-09	1.7E-12
100	1.0236061374751637E+03	1.5E-09	1.4E-12

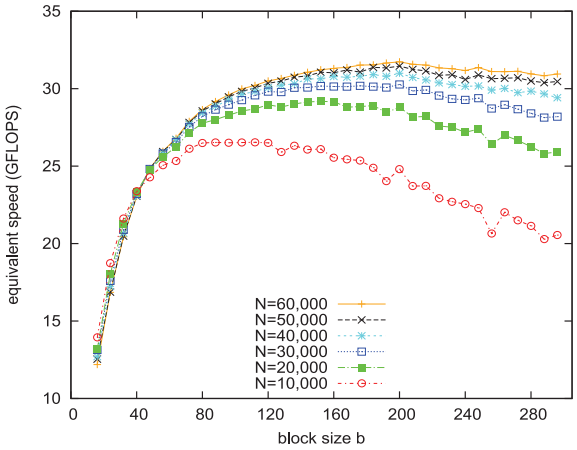


図 43　TS-QR 分解を計算する場合の，ブロック三重対角化の換算性能（Core2 Quad Q6600（4 スレッド））

Fig. 43　Equivalent speed of block tridiagonalization with TS-QR (Core2 Quad Q6600 (4 thread)).

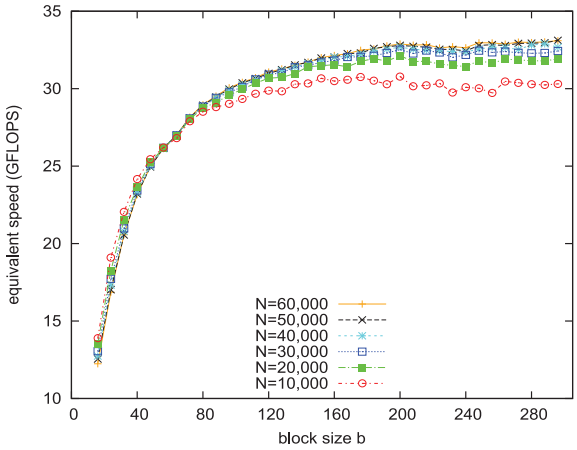


図 44　TS-QR 分解を省略した場合の，ブロック三重対角化の換算性能（Core2 Quad Q6600（4 スレッド））

Fig. 44　Equivalent speed of block tridiagonalization without TS-QR (Core2 Quad Q6600 (4 thread)).

向を示した．計算システムはシングル CPU で，CPU は Intel Core2 Quad Q6600 （コア数 4，クロック 2.4 GHz，L1(I/D)：32 KB，共有 L2:8 MB（4 MB x 2 ダイ））であり，主記憶は 16 Gbyte（デュアルチャネル動作，DDR3-1600 MHz（PC3-12800）の 4 Gbyte の DIMM が 4 本，しかし実際の動作は DDR3-1067 MHz で，PC3-8500 であろう）である．OS は CentOS 6.5 for x86_64 で，コンパイラと BLAS ライブラリは（本論文執筆時の最新バージョンであった）Intel Parallel Studio XE 2015 for 64 bit Linux（Intel Fortran コンパイラ version 15.0.0，MKL ライブラリ version 11.2）である．並列化には Intel Fortran の OpenMP の機能のみを利用した．用いたコンパイラオプションは “-fast -openmp” である．MKL の BLAS ルーチンはリンクのオプション “-mkl=sequential” により，シングルスレッド版だけを使用した．この CPU の 4 コアでの理論ピーク演算性能は 38.4 GFLOPS（倍精度演算）である．

容量 16 Gbyte の主記憶上で対称性を用いてブロック三重対角化の計算が行える最大の行列のサイズ N は，1 万の倍数であれば 6 万である．**図 43** は TS-QR 分解を行った場合の計算であり，**図 44** は TS-QR 分解を省略した場合の計算である．どちらも行列サイズ N を 1 万から 6 万まで刻み 1 万で変えながら横軸にはブロックサイズ b を 16 から 296 まで刻み 8 でとり，縦軸には換算性能（GFLOPS）の値をとってプロットしたグラフである．すべての計算は OpenMP で並列化し，CPU のコア数と同じ 4 スレッドで実行した．

TS-QR 分解を含めた計算の場合には，b の増加に対する換算性能の劣化が N が小さいほどより顕著に現れている．たとえば，$N = 10,000$ の場合に $b = 80$ では換算性能が 26.5 GFLOPS であるのに対して $b = 200$ では 24.8 GFLOPS，$b = 240$ では 22.5 GFLOPS，$b = 280$ では 21.1 GFLOPS などと低下している．TS-QR 分解を省略した計算の場合には，b の増加に対する換算性能の劣化傾向は見られない．$N = 10,000$ の場合には $b = 80$ では換算性能は 28.5 GFLOPS，$b = 200$ では 30.8 GFLOPS，$b = 240$ では 30.1 GFLOPS，$b = 280$ では 30.3 GFLOPS とほぼ一定値を保っている．つまり，$N = 10,000$ の場合には TS-QR 分解を含めない場合に比べて換算性能の低下が $b = 80$ で 7%，$b = 200$ では 20%程度，$b = 240$ では 25%程度，$b = 280$ では 30%程度となっている．その理由は，現在使用している TS-QR 分解の方法では，その内部で用いている BLAS2 の HQR 分解の性能は列の数（今の場合は b である）が大きくなるとかなり低下してしまうからである．これに対する最も簡単な改善法は HQR 分解のワーキングセットサイズを減らしてキャッシュとメモリの間の記憶の入れ替えの転送量をなるべく減らすことである．たとえば HQR 分解の内部で毎回の Householder 変換のたびにピボット列以降の列を最後まで追いかけて更新するのをやめて，列方向をブロックに分割して扱い，変換による更新範囲をその列ブロック内にとどめて蓄積し，変換が進んで新しい列ブロックに移ったときにそれまで当該ブロックへの適用を延期していた過去の変換をブロック単位で適用する「蓄積型の HQR 分解」を採用するか，あるいは列ブロック内部の Householder 変換を束ねてブロック Householder 変換を作り，BLAS3 の行列積を用いてそれ以降の列ブロックに適用していく方式の（ブロック）HQR 分解がさらに効率的であろう．このような改善の追加は今後の課題である．

6.　今後に改良すべきこと

現状のブロック三重対角化の計算には性能を向上させる余地がまだ残っている．たとえば，ブロック鏡映変換の軸の構成において T-S 行列の QR 分解に用いた今回の算法は，適切なサイズによる行分割により階層的再帰的に処理しているが，分割されたローカルな行列に対する QR 分

解は（ブロックサイズ b があまり大きくないことを仮定して）BLAS2 の算法だけで組み立てられているので，行列の列幅であるブロックサイズ b がある程度大きくなるとローカルな行列が CPU のキャッシュに収まらなくなり，ローカルな行列に対する QR 分解の計算速度が低下する．よって T-S 行列の QR 分解全体の速度も落ちる．この問題に対処するには，列方向についてもまたブロッキングを行い，ローカルな行列の QR 分解の内部で BLAS3 を用いる処理に置き換える必要がある．

今回のブロック三重対角化のカーネル部分（2.2 節のリスト 1）では，対称性から下三角部分だけを保持したブロック行列に対する鏡映変換 1 段階分の処理は，「ブロック版の行列ベクトル積」（カーネル 1）の箇所では列方向には逐次的とし，対象とする列を複数スレッドで行方向に分割して並列化を行った．また，「ブロック版の階数 2 の更新」（カーネル 2）の計算では各スレッドがそれぞれ相異なる列を前から後に向かって 1 つずつ処理していくように並列化を行った（Fortran+OpenMP によるソースコードの例はリスト 2 の BHSHLD である）．いずれも各スレッドに割り当てる作業量を完全に均等にはできなかったり，変換が進んだ終盤では行列サイズが小さくなり作業の割当てを受けないスレッドが生じたりする．元の行列のサイズが最初から小さい場合や，計算に使用できるスレッド数が多いときには特に影響が大きくなる．そのような影響を減らすためには，単純に行あるいは列に基づいてスレッドを割り当てるのではなくて，たとえば表を用いて作業の割当てを管理するなどにより，なるべく各スレッドの待ち状態が生じないようにする改良が必要であろう．

サイズ b の行列積を行う BLAS3 のルーチン xGEMM の性能は，その実装内部でのブロッキング処理やハードウェアの性質などに依存し，性能は必ずしも b の増加につれて単調には増加せず（たとえば b がある特定の値や特定の値の倍数のときに性能が非常に良いというような）複雑な挙動を示す．もしも実際の計算に用いるブロックサイズ b の値が固定あるいは数通りしかなければ，その値に対してチューニングした専用の行列乗算系のルーチンを作成して用いれば，汎用のルーチン xGEMM よりも高い性能が出せるであろう．

なお，仮にブロック三重対角化の処理単独ではブロックサイズ b により大きい値を採用すれば処理の経過時間を短縮できる場合であっても，それに続いて行うブロック三重対角行列 T に対する固有値問題解法の処理の経過時間は長くなる．そのため固有値問題の計算全体としての性能を最適にするブロックサイズ b の値は，問題の行列のサイズ N や必要な固有対の個数にも依存する．

7. まとめ

今回の実験では，実対称密行列のブロック三重対角化を行う際に，ブロックサイズ b を適切な範囲である程度大きく選ぶことにより，行列の次数 N が大きい場合に，コア数が 4 から 8 程度の小規模なマルチコア型の CPU1 個で構成された計算機（PC）上では，CPU の浮動小数点数演算性能の理論ピーク値の 70%から 80%程度の「換算性能」を達成できることを確認した．

以前の報告 [10] で紹介したいわゆる「Wilkinson の技巧」は，通常の Householder 変換に基づく三重対角化に適用すると CPU とメモリとの間のメモリ転送量を 2/3 倍程度に減らせる．その方法はブロック化（タイル化）された Householder 変換に基づくブロック三重対角化の算法にもまったく同様に適用できるので，マルチコアの CPU 上に実装して OpenMP で並列化して実験をしてみたところ，行列が主記憶に収まる場合については，ブロックサイズ b が非常に小さいところでは「Wilkinson の技巧」を採用すると性能がごくわずかに向上する場合もあるが，ブロックサイズ b が少し大きくなると，むしろ性能が少し落ちるという結果が得られたので，今回の報告には含めていない．その理由として考えられることは，まずブロックサイズ b がある程度大きくなると行列積の BLAS3 ルーチン xGEMM の記憶参照の局所性が十分に高まるので「行列を走査するためのメモリバンド幅」のボトルネックが解消されてしまうこと，さらに「Wilkinson の技巧」を採用すると作業用のブロックベクトルの個数が増えるのでそれによりむしろキャッシュとメモリ間のデータの入れ替えの転送量が増えるデメリットが生じるためであろうと思われるが，今後により詳しい考察や分析をすべきである．

今回の実験で用いた計算システムの CPU は（比較的高価な Intel Corei7-5960X を別とすれば）どれも日用品レベルの安価なものであり，CPU 内部のコア数（使用スレッド数）は 4 から 6 あるいは 8 で比較的少ないものである．このため，行列要素がスカラーである場合の算法をほぼ忠実にタイルに置き換えた形の算法のプログラムをあまり書き換えずに OpenMP の指示行を挿入する比較的簡単な並列化の方法だけで，CPU の持つ演算性能の理論ピーク値との比で 70%から 80%程度のかなり良い（換算）性能値を得ることができたのであろうと思われる．そこで今後はさらに，最近のより高価な計算サーバ用のコア数が多いマルチコア CPU （たとえばコア数が 10 から 18 など）や，特にメニーコア（たとえばコア数が数十から数百）の CPU や GPU についても，同様のブロック化（タイル化）算法を用いて高いピーク性能比を出せるかについての調査研究を行うつもりである．

参考文献

[1] Murakami, H.: An Implementation of the Block Householder Method, 情報処理学会論文誌：コンピューティングシステム, Vol.47, No.SIG 7 (ACS 14), pp.61–80 (2006).

[2] Langou, J.: Allreduce Householder factorizations, *Proc. 2007 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications, Météopole Toulouse, France: ENSEEIHT-IRIT RT/AP0/07/10*, pp.103–106 (July 2007).
[3] Langou, J.: All Reduce Algorithms: Application to Householder QR Factorization, *Sparse Days at CERFACS Meeting*, Toulouse, France (Aug. 2007).
[4] 村田健朗，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).
[5] 長谷川秀彦：古典的アルゴリズムによる大規模対称帯行列の固有値計算，計算工学講演会論文集，Vol.10, No.1, pp.247–248 (2005).
[6] 長谷川秀彦：古典的アルゴリズムによる大規模対称帯行列の固有値計算，計算工学講演会論文集，Vol.14, No.1, pp.205–206 (2009).
[7] 山本有作：キャッシュマシン向け対称帯行列固有値解法の性能・精度評価，情報処理学会論文誌：コンピューティングシステム，Vol.46, No.SIG 3 (ACS 8), pp.81–91 (2005).
[8] Yamamoto, Y.: Performance modeling and optimal block size selection for a BLAS-3 based tridiagonalization algorithm, *Proc. 8th International Conference on High-Performance Computing in Asia-Pacific Region* (*HPCASIA'05*) (2005).
[9] Bai, Y. and Ward, R.C.: Parallel block tridiagonalization of real symmetric matrices, *J. Parallel Distrib. Comput.*, Vol.68, No.5, pp.703–715 (2008).
[10] 村上　弘：両側ハウスホルダ変換に対する Wilkinson の著書の AEP 中の『技巧』について，情報処理学会研究報告，Vol.2008-HPC-118, No.12, pp.67–72 (2008).
[11] 村上　弘：マルチコア CPU システムおよび小規模 SMP 並列システム上での Tall Skinny 型 QR 分解法の実験，情報処理学会論文誌：コンピューティングシステム，Vol.2, No.SIG 2 (ACS 26), pp.19–29 (2009).
[12] Demmel, J., Grigori, L., Hoemmen, M. and Langou, J.: Communication-optimal parallel and sequential QR and LU factorizations, *SIAM J. Sci. Comput.*, Vol.34, No.1, pp.A206–A239 (2012).
[13] Demmel, J.W., Grigori, L., Hoemmen, M.F. and Langou, J.: Communication-optimal parallel and sequential QR and LU factorizations, Tech. Report UCB/EECS-2008-89 (Also, LAPACK Working Note #204), Univ. California at Berkeley (Aug. 2008).
[14] Stathopoulos, A. and Wu, K.: A block orthogonalization procedure with constant synchronization requirements, *SIAM J. Sci. Comput.*, Vol.23, No.6, pp.2165–2182 (2002).
[15] 村上　弘：非常に細長い大規模行列に対する記憶参照局所性が高い正規直交化法の実験，日本応用数理学会論文誌，Vol.17, No.4, pp.399–454 (2007).

付　　録

A.1　TS-QR 法以外の直交分解を使用した例

今回の実験に用いた TS-QR 法の実装は BLAS2 だけで構成されているので，列数の多くなった場合には性能が悪化すること，およびそれを今後改良する方法については，すでに本論文中に述べた．本付録ではその改良とは別の新たな試みとして，BLAS3 で構成された T-S 行列用の簡単な（列）直交分解の方法により TS-QR 法を置き換えてみる．

リスト 7　固有値分解を用いる列直交分解法
List 7　Orthogonal decomposition method using eigen-decomposition.

```
V ← I ; /* n 次単位行列 */
k ← 0 ; /* ループカウンタ */
BEGIN LOOP:
  S ← A^T A ;/* 行列 S は n 次対称 */
  k ← k + 1 ; /* カウントを増す */
  もしも k > 1 且つ「S が十分に対角行列とみなせる」
  または k が kmx（例：3 回）に達したら EXIT LOOP ;
  S → U D U^T ; /* 固有値が降順の固有値分解 */
  A ← A U , V ← V U ;
END LOOP
D ← √diag(S) ; /* S の対角の平方根を D に */
r ← rank_ε(D) ; /* 対角行列 D の有効階数を r に */
Q ← A D^† ; /* A に D の一般逆を乗じて Q に */
R ← D V^T ;
```

BLAS3 を使って T-S 行列の直交分解を行う Cholesky QR 法に類似した文献 [14] の SVQB 法があり，それと同類の方法として，我々が以前の文献 [15] ですでに記述したものがある．それは $m \times n$ の T-S 行列 A の直交分解 $A = QR$ を行うのに，まず $S = A^T A$ を作り，S の（Cholesky 分解ではなくて）固有値分解を用いて（反復改良を適応的に入れて）行うものである．その算法を（リスト 7）に簡単に示す．

行列 A は上書きで変更されることに注意する．元の A の場所に Q を上書きして格納することもできる．この計算は本来は T-S 行列 A の（近似）特異値分解 $A = QDV^T$ であって，分解を $A = QR$ としたときの行列 Q は列直交で，右側の行列 $R = DV^T$ は上三角ではないことに注意する．得られた（列）階数 r の Q と（行）階数 r の R が，元の A の（近似）直交分解 $A \to QR$ を与える（なお，アンダフローによる精度低下のリスクをなるべく避けるためには，直交分解の構成の最初に A の各列をその絶対値の最大値に近い値で割ってスケーリングする前処理を加えた方が望ましいが，今回は省略している）．

リスト 7 の算法中の $S \leftarrow A^T A$ と $A \leftarrow AU$ の計算が演算量について主要部分であり，それぞれ BLAS3 を用いて計算ができる．T-S 行列 A が行方向にブロック分割あるいはタイル分割されていれば，その各分割ごとに独立した BLAS3 の演算を行って構成すると並列化も容易にできる．

ループの脱出条件「S が十分に対角行列とみなせる」の判定には「すべての $i < j$ に対して，$|s_{i,j}| < 10\,\varepsilon_{\mathrm{MAC}}\,\max(s_{i,i}, s_{j,j})$」を用いた（不等式は $s_{i,j}^2 < 10\,\varepsilon_{\mathrm{MAC}}\,s_{i,i}s_{j,j}$ とする方が自然であると思われるが，S が悪条件の場合には二乗の大きさの値を作るのでアンダフローを起こすリスクが高くなりそうである）．

リスト 7 の算法で用いる小規模な n 次対称行列 S の固有値分解 $S \to U\,D\,U^T$ には，分解の直交性の品質を重視

して（Rutishauser の）Jacobi 法を用いた．Jacobi 法の系統以外の算法を用いることも考えられるが，その場合に固有値に縮退があるときや S が悪条件である場合に得られる U の直交精度が悪化している可能性がある（固有値が降順に並ぶ分解としているのは，階数が落ちている場合の処理の利便性や悪条件の場合の精度上の利点による）．

算法中の対角行列 D の有効階数 r の決定は，D の対角要素の最大値が 0 のときは $r = 0$ とし，そうでないときは D の最大要素の ε 倍よりも大きい値を持つ対角要素の個数とする．ここで相対比 ε は丸め誤差の単位程度の正数とした．D の一般逆行列 $D^{\dagger}$ は，D の各対角要素に対して，有効階数のカウントに含めた要素にはその逆数を，含めなかった微小な要素には 0 をそれぞれ対応させた対角行列である．

A.1.1　リスト 7 の直交分解の算法を用いた実験例

ブロック Householder 三重対角化法のブロック鏡映軸の構成に直交分解の算法（リスト 7）を用いて行った実験の例を示す．ブロックサイズ $b = 100$ で $N = 10,000$ 次の Frank 行列をブロック三重対角化し，それにより得られたブロック三重対角行列 T の固有値を通常の Householder 変換とそれに続く QR 反復法で求めた．その最小側の固有値 100 個について，得られた近似値と，厳密値，厳密値からの誤差を**表 A·1** に示す．これら 100 個の固有値が絶対誤差 10^{-11} 程度以下で求まっていることが分かる．

A.1.2　直交分解に Jacobi 法を利用した結果とその考察

N 次対称行列のブロック三重対角化の計算において（ブロック三重対角化のタイルのサイズ b がここでの n である），ブロック鏡映軸をリスト 7 の算法による直交分解を用いて構成するとき，N がたとえば 1 万程度であまり大きくない場合にはブロックのサイズ b を大きくすると「換算性能」がかなり劣化することが分かった．その原因は S の固有値分解を行うために現在採用している Jacobi 法の経過時間が全体に占める割合の増大である．これは Jacobi 法のルーチンを S の偽の固有値分解を素早く返すダミーのルーチンに置き換えて比較することで確認できた（その場合には，正しい Jacobi 法のルーチンを用いる場合とダミールーチンを用いた場合をそれ以外の条件を揃えて比較するために，リスト 7 の直交分解の算法中での適応的な動作は止めて Jacobi 法がつねに 1 回だけ呼び出されるようにして計測をした）．

その方針による比較を（すでに紹介した）CPU が Core i7-5960X のシステムを用いて行った．**図 A·1** は，リスト 7 の直交分解の算法で Jacobi 法を 1 回だけ用いた場合，**図 A·2** はその直交化の中の Jacobi 法をダミーのルーチンで置き換えて計算した場合の「換算性能」である．どちらも行列サイズ N を 1 万から 4 万まで刻み 1 万で変えながら横軸にはブロックサイズ b を 16 から 296 まで刻み 8 でとり，縦軸には換算性能（GFLOPS）の値をとってプロットしたグラフである．計算は OpenMP で並列化し，CPU のコア数と同じ 8 スレッドで実行しているが，Jacobi 法は並列化されていない．

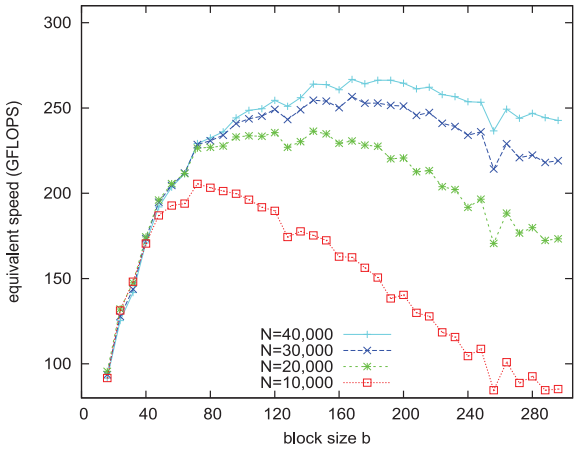


図 A·1　直交分解で利用する固有値分解に Jacobi 法を用いた場合の，ブロック三重対角化の換算性能（Core i7-5960x（8 スレッド））

Fig. A·1　Equivalent speed of block tridiagonalization with Jacobi orthogonalization (Core i7-5960x (8 thread)).

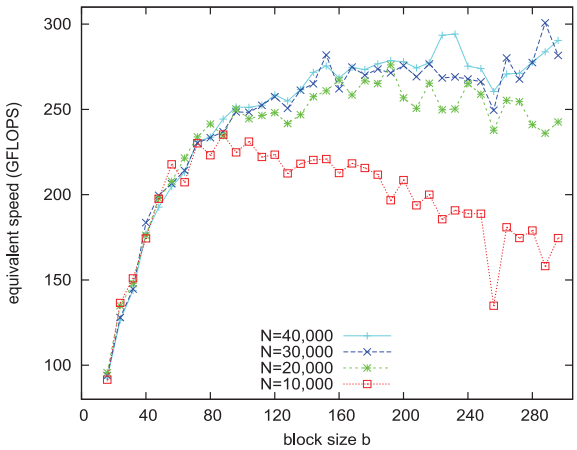


図 A·2　直交分解で利用する固有値分解に Jacobi 法を省略した場合の，ブロック三重対角化の換算性能（Core i7-5960x（8 スレッド））

Fig. A·2　Equivalent speed of block tridiagonalization with dummy Jacobi orthogonalization (Core i7-5960x (8 thread)).

Jacobi 法の記憶参照の特性

Jacobi 法の算法は記憶参照の局所性がなく，計算の最内側でつねに行列要素を行および列の両方向に順番に参照するので不連続なパターンのアクセスを多く含むことが欠点の 1 つである．そのため計算機の階層記憶システムの転送能力を十分有効に利用できない．特に今の場合，次数 n（サイズ b）が大きくなり S（の上半分）と U が CPU のコアのキャッシュに収まらなくなり，さらに CPU の共有

表 A·1　100 個の最小側固有値と厳密値（Frank 行列，N=10,000, ブロックサイズ 100）

Table A·1　Solved smallest 100 eigenvalues and exact ones (Frank Matrix, N=10,000, block size 100).

k	Solved Value	Exact Value	Error
1	0.25000 00061 678717	0.25000 00061 678860	-1.4E-14
2	0.25000 00246 714834	0.25000 00246 715454	-6.2E-14
3	0.25000 00555 120093	0.25000 00555 109818	1.0E-12
4	0.25000 00986 862653	0.25000 00986 862011	6.4E-14
5	0.25000 01541 998539	0.25000 01541 972121	2.6E-12
6	0.25000 02220 447211	0.25000 02220 440256	7.0E-13
7	0.25000 03022 272930	0.25000 03022 266549	6.4E-13
8	0.25000 03947 494641	0.25000 03947 451161	4.3E-12
9	0.25000 04996 048852	0.25000 04995 994273	5.5E-12
10	0.25000 06167 952559	0.25000 06167 896092	5.6E-12
11	0.25000 07463 210125	0.25000 07463 156849	5.3E-12
12	0.25000 08881 784622	0.25000 08881 776799	7.8E-13
13	0.25000 10423 750802	0.25000 10423 756224	-5.4E-13
14	0.25000 12089 109597	0.25000 12089 095428	1.4E-12
15	0.25000 13877 833264	0.25000 13877 794738	3.9E-12
16	0.25000 15789 887558	0.25000 15789 854508	3.3E-12
17	0.25000 17825 282186	0.25000 17825 275116	7.1E-13
18	0.25000 19984 077923	0.25000 19984 056963	2.1E-12
19	0.25000 22266 208392	0.25000 22266 200475	7.9E-13
20	0.25000 24671 718042	0.25000 24671 706103	1.2E-12
21	0.25000 27200 595542	0.25000 27200 574322	2.1E-12
22	0.25000 29852 816623	0.25000 29852 805630	1.1E-12
23	0.25000 32628 445963	0.25000 32628 400551	4.5E-12
24	0.25000 35527 478340	0.25000 35527 359634	1.2E-11
25	0.25000 38549 690873	0.25000 38549 683450	7.4E-13
26	0.25000 41695 376900	0.25000 41695 372595	4.3E-13
27	0.25000 44964 455708	0.25000 44964 427691	2.8E-12
28	0.25000 48356 849328	0.25000 48356 849382	-5.4E-15
29	0.25000 51872 652864	0.25000 51872 638339	1.5E-12
30	0.25000 55511 812118	0.25000 55511 795255	1.7E-12
31	0.25000 59274 335225	0.25000 59274 320849	1.4E-12
32	0.25000 63160 381665	0.25000 63160 215864	1.7E-11
33	0.25000 67169 556885	0.25000 67169 481064	7.6E-12
34	0.25000 71302 166688	0.25000 71302 117245	4.9E-12
35	0.25000 75558 297759	0.25000 75558 125220	1.7E-11
36	0.25000 79937 598631	0.25000 79937 505829	9.3E-12
37	0.25000 84440 287738	0.25000 84440 259936	2.8E-12
38	0.25000 89066 472155	0.25000 89066 388431	8.4E-12
39	0.25000 93816 023525	0.25000 93815 892227	1.3E-11
40	0.25000 98688 930077	0.25000 98688 772261	1.6E-11
41	0.25001 03685 093724	0.25001 03685 029495	6.4E-12
42	0.25001 08804 725206	0.25001 08804 664915	6.0E-12
43	0.25001 14047 856077	0.25001 14047 679532	1.8E-11
44	0.25001 19414 136809	0.25001 19414 074380	6.2E-12
45	0.25001 24904 018721	0.25001 24903 850520	1.7E-11
46	0.25001 30517 205062	0.25001 30517 009033	2.0E-11
47	0.25001 36253 640908	0.25001 36253 551030	9.0E-12
48	0.25001 42113 493542	0.25001 42113 477641	1.6E-12
49	0.25001 48096 806339	0.25001 48096 790023	1.6E-12
50	0.25001 54203 635464	0.25001 54203 489358	1.5E-11

表 A·1（続き）

Table A·1 (Continued)

k	Solved Value	Exact Value	Error
51	0.25001 60433 641792	0.25001 60433 576851	6.5E-12
52	0.25001 66787 114957	0.25001 66787 053731	6.1E-12
53	0.25001 73263 950383	0.25001 73263 921253	2.9E-12
54	0.25001 79864 196922	0.25001 79864 180695	1.6E-12
55	0.25001 86587 884782	0.25001 86587 833361	5.1E-12
56	0.25001 93434 914174	0.25001 93434 880576	3.4E-12
57	0.25002 00405 392726	0.25002 00405 323694	6.9E-12
58	0.25002 07499 228533	0.25002 07499 164090	6.4E-12
59	0.25002 14716 526296	0.25002 14716 403163	1.2E-11
60	0.25002 22057 044851	0.25002 22057 042339	2.5E-13
61	0.25002 29521 130842	0.25002 29521 083067	4.8E-12
62	0.25002 37108 670114	0.25002 37108 526820	1.4E-11
63	0.25002 44819 427389	0.25002 44819 375097	5.2E-12
64	0.25002 52653 677377	0.25002 52653 629418	4.8E-12
65	0.25002 60611 365469	0.25002 60611 291332	7.4E-12
66	0.25002 68692 382862	0.25002 68692 362407	2.0E-12
67	0.25002 76896 874625	0.25002 76896 844240	3.0E-12
68	0.25002 85224 758088	0.25002 85224 738450	2.0E-12
69	0.25002 93676 045860	0.25002 93676 046682	-8.2E-14
70	0.25003 02250 786462	0.25003 02250 770603	1.6E-12
71	0.25003 10948 930536	0.25003 10948 911907	1.9E-12
72	0.25003 19770 540345	0.25003 19770 472310	6.8E-12
73	0.25003 28715 505536	0.25003 28715 453553	5.2E-12
74	0.25003 37783 861002	0.25003 37783 857404	3.6E-13
75	0.25003 46975 698470	0.25003 46975 685651	1.3E-12
76	0.25003 56290 970864	0.25003 56290 940110	3.1E-12
77	0.25003 65729 806537	0.25003 65729 622619	1.8E-11
78	0.25003 75291 751306	0.25003 75291 735042	1.6E-12
79	0.25003 84977 300399	0.25003 84977 279267	2.1E-12
80	0.25003 94786 345906	0.25003 94786 257206	8.9E-12
81	0.25004 04718 677232	0.25004 04718 670795	6.4E-13
82	0.25004 14774 580111	0.25004 14774 521993	5.8E-12
83	0.25004 24953 833931	0.25004 24953 812789	2.1E-12
84	0.25004 35256 624944	0.25004 35256 545190	8.0E-12
85	0.25004 45682 804635	0.25004 45682 721232	8.3E-12
86	0.25004 56232 369917	0.25004 56232 342971	2.7E-12
87	0.25004 66905 451133	0.25004 66905 412491	3.9E-12
88	0.25004 77701 936512	0.25004 77701 931900	4.6E-13
89	0.25004 88621 930450	0.25004 88621 903327	2.7E-12
90	0.25004 99665 336900	0.25004 99665 328931	8.0E-13
91	0.25005 10832 207199	0.25005 10832 210890	-3.7E-13
92	0.25005 22122 605821	0.25005 22122 551410	5.4E-12
93	0.25005 33536 514709	0.25005 33536 352720	1.6E-11
94	0.25005 45073 733136	0.25005 45073 617073	1.2E-11
95	0.25005 56734 485253	0.25005 56734 346747	1.4E-11
96	0.25005 68518 642430	0.25005 68518 544043	9.8E-12
97	0.25005 80426 214424	0.25005 80426 211290	3.1E-13
98	0.25005 92457 388408	0.25005 92457 350838	3.8E-12
99	0.25006 04611 968936	0.25006 04611 965063	3.9E-13
100	0.25006 16890 056443	0.25006 16890 056364	7.9E-15

キャッシュにも収まらなくなると主記憶とキャッシュの間で入れ替えが頻発して計算が遅くなる．n が大きい場合は，Jacobi 法にブロック化を施し，さらに並列化した「並列化ブロック Jacobi 法」を採用すれば性能の劣化を軽減できるはずであるが，その実装はあまり簡単ではないので今後の課題にして，並列化やブロック化は施していない普通の Jacobi 法を今回は用いた．

村上 弘　（正会員）

1960 年生．1992 年北海道大学大学院理学博士号（化学第二学専攻）取得．現在，首都大学東京数理情報科学専攻の准教授．数理的な問題の数値的あるいは記号的方法による効率あるいは精度の良い解法やその並列化手法の研究に従事．日本応用数理学会，日本数式処理学会，SIAM，AMS，ACM，IEEE 各会員．