Vol.2019-HPC-168 No.20
2019/3/7

高い演算密度をもつヤコビ回転カーネルの構成手法

工藤 周平[1,a)]　今村 俊幸[1]

概要：ヤコビの固有値計算手法 (ヤコビ法) は密行列に対して平面回転 (Givens 回転、またはヤコビ回転) を繰り返し適用することで行列を徐々に対角行列へと近づけていく反復法である．ヤコビ回転は行列のうち 2 つの行・列ベクトルに対する計算であり，データ再利用性が悪いため，演算効率が低い．そこで一般的には，ヤコビ回転を集積することで行列積に書き換えることが行われるが，この手法には演算量が倍増するという欠点がある．この問題を解決するため，我々はヤコビ法の巡回順序自体が持つデータ再利用性を活用し，Level-3 BLAS で用いられる階層的なループブロック化手法を適用する，ヤコビ回転カーネルの構成手法を考案した．本手法を基にした我々の実装は，いくつかの CPU 環境上でピーク演算性能比 80% 以上となる高性能を示している．本稿では，この構成手法の詳細と，性能測定結果を示す．

1. はじめに

ヤコビの固有値計算手法 (ヤコビ法) [5] は古くからある対称行列固有値計算手法の 1 つであり，平面回転 (Givens 回転，本稿ではヤコビ回転と呼ぶ) による直交変換を繰り返すことで行列を徐々に対角行列へと近づける手法である．ヤコビ法は密行列に対する反復法であるため，他手法と比べて演算量が大きいという欠点がある．しかし我々は，ヤコビ法が持つ他手法にはない次のような特徴に興味を持っている．1) ブロック化を適用することで演算密度を高められるため，メモリバンド幅がボトルネックになりにくいこと，2) 安定かつ高精度な計算手法であり，さらに特別な構造を持った行列に対して相対誤差の意味で固有値を計算できること，3) 反復法であるため，精度をコントロールしたり，解に近い初期値を与えることで高性能化する可能性があること．そこで我々はヤコビ法の高性能実装手法を確立することで，他の固有値計算手法にはない機能を提供する固有値計算ライブラリを構築することを目的としている．

本稿ではヤコビ法のヤコビ回転を行うループの構造に着目し，Level-3 BLAS で用いられる階層的なループブロック化手法 [4] を適用することで，高い演算密度を実現する計算カーネルを構築する手法を示す．我々の手法は第一に，Block-Oriented (BO) 巡回ヤコビ法を出発点とし，階層的なループブロック化を適用することで，計算カーネルを構築するのに適したループ構造を抽出すること，第二に，Anda & Park の Self-scaling Rotation (SSR) [1] に対して分岐を排除する改良を行い，演算量削減と分岐排除の両方を実現することである．これらの結果生成された計算カーネルは，多くの積和演算と，少ない数のデータ移動命令，そして最小限のループ制御命令で構成されるため，Level-3 カーネルに近い，高い演算密度を持つ．本稿ではヤコビ法についてのみ取り扱うが，同様の構造を持つヤコビ法の応用手法 (例えばヤコビ SVD 法) にも適用可能である．

我々の手法は Zee ら [7] の QR 反復高速化手法をヤコビ法向けに拡張したものである．ただし Zee らのものとは重要な部分で大きな違いがある．第一に，QR 反復とヤコビ法ではヤコビ回転適用順序が大きく異なる．第二に，我々は高速平面回転の技術を使って演算量を削減している．第三に，我々は 1 つのアーキテクチャのみを対象にするのではなく，多数のアーキテクチャ上で性能を出すための構成を検討し実証もしている．以上のように，彼らの成果と共通点はあるが，我々の研究は重要な独自成果を含む．

ヤコビ法のブロック化手法として，BO 手法の他にフルブロック化手法が存在する．フルブロック化手法はヤコビ回転の代わりにブロック回転と呼ばれる直交変換を行うため，計算の大部分を行列積に置き換えることができる．しかしフルブロック化手法は数式上ヤコビ法と異なる手法であり，収束性や誤差解析など未知な部分が残されている．そこで本研究の手法を用いることで，フルブロック手法を用いずとも行列積に近い高性能を実現できれば，この問題を回避できる．また，BO 手法において，ヤコビ回転をそのまま行列に適用するのではなく，ヤコビ回転のみを集積した直交行列を生成することで，ヤコビ回転を行列積に変換する手法がある．この手法は，ヤコビ回転の集積自体が

[1] 理化学研究所 計算科学研究センター
R-CCS, Kobe, Hyogo 657–0046, Japan
[a)] shuhei.kudo@riken.jp

追加コストとなること，また，行列積はヤコビ回転と比べて演算量が約 2 倍となることという問題点がある．そこで本研究の手法によって実行効率 50%程度以上を達成できれば，計算性能上のメリットがある．

対称行列向け固有値計算手法としては，三重対角化を用いた手法が一般的である．この手法は演算量が小さいという利点があるが，データ再利用性の低い計算を含むため，近年の演算性能と比べてメモリバンド幅の小さい計算機では低性能となる．また，この手法は密行列に対しては特別な場合を除いてほぼ一定の演算量であるため，入力行列の特徴を利用することが不可能である．三重対角化手法の改良手法として，帯行列化手法が存在し，演算密度の問題を避けることが可能だが，オーバーヘッドが大きい手法であるため巨大な行列でなければ特徴を活かせないこと，また，入力行列の特徴を活かせない点は三重対角化手法と共通すること，などの問題を持っている．そこでヤコビ法は他手法とは異なる状況で有用となる可能性がある．

本稿は次の構成となっている．本節では本稿の概要と研究背景を示した．次節ではヤコビ法のアルゴリズムの詳細と，BO 手法，高速平面回転，また Anda & Park の Self-scaling Rotation (SSR) を解説する．第 3 節ではスケーリングによる SSR の分岐排除手法を示す．第 4 節ではヤコビ回転カーネルの構成手法を示す．第 5 節では性能測定結果を示す．最後の節で本稿をまとめる．

2. ヤコビ法のアルゴリズム

本節ではヤコビ法に関する既存手法を解説する．はじめにヤコビ法の基本的なアルゴリズムを示し，BO 手法によるデータ再利用性の向上手法と，BO 手法を用いた場合の演算量の中心となるサブルーチンを示す．また，ヤコビ回転の演算量を削減する既存手法である高速平面回転 (Fast Plane Rotation, FPR) について解説する．

2.1 ヤコビ法

ヤコビ法は対称行列 $A \in \mathbb{R}^{n\times n}$ に対してヤコビ回転 $G(p,q,\theta), p \neq q$ による相似変換を繰り返すことで，行列を徐々に対角行列へと近づけていく反復法である．ヤコビ回転はある軸 p, q に関する平面回転である:

$$G(p,q,\theta)_{i,j} = \begin{cases} \cos\theta & \text{if } i=j=p \cup i=j=q \\ \sin\theta & \text{if } i=p \cap j=q \\ -\sin\theta & \text{if } i=q \cap j=p \\ \delta_{i,j} & \text{otherwise} \end{cases} . \quad (1)$$

ヤコビ法は初期値 $A^{(0)}=A$, $V^{(0)}=I$ とおき n 番目のヤコビ回転を $G^{(k)} = G(p^{(k)}, q^{(k)}, \theta^{(k)})$ とおくとき，

$$A^{(k+1)} = \left(G^{(k)}\right)^{\top} A^{(k)} G^{(k)}, \quad (2)$$

$$V^{(k+1)} = V^{(k)} G^{(k)} \quad (3)$$

の 2 つの反復式を計算する．ただし実際の計算は値の変化する一部の列・行 ($p^{(k)}$, $q^{(k)}$ 列・行) のみを書き換えるよう実装する．また，式 (2) は対称変換であるため，上 (下) 三角要素のみを計算することで演算量を削減する．最終的にある M 反復目に行列 $A^{(M)}$ が充分に対角行列へと近づいたとき，その対角成分を並べた対角行列を $\Lambda^{(M)}$ とすると，$AV^{(M)} \approx V^{(M)}\Lambda^{(M)}$ は A の近似的な固有値分解となっている．有限精度計算においてはこの打切り誤差に加え，各反復において累積誤差が発生する．詳細な誤差解析については Demmel らの論文 [2] を参照されたい．

この反復式において $p^{(k)}$ と $q^{(k)}$, $\theta^{(k)}$ の設定がアルゴリズムの数学的性質や性能を変化させる．Jacobi のオリジナルの手法 [5] は，行列の要素に基づきこれらを動的に決定する．しかし探索のための追加コストが大きいため，通常はより簡単な巡回ヤコビ法を用いる．巡回ヤコビ法は固定的順序によって選択した $(p^{(k)}, q^{(k)})$ を用いる手法である．具体的には，行列の上 (下) 三角部分の非対角要素を一度ずつ選択するような組の数列を 1 つ決め，その順序に従って繰り返し同じ順序を用いて $(p^{(k)}, q^{(k)})$ を選ぶ．この数列のことを巡回順序と呼ぶ．巡回順序については次小節で詳しく述べる．一方，$\theta^{(k)}$ については，$A^{(k+1)}$ の $(p^{(k)}, q^{(k)})$ 要素が 0 となるよう定める．これを満たすものは無数にあるが，$\left|\theta^{(k)}\right| \leq \frac{\pi}{4}$ という収束のための十分条件 [3], [6] が知られており，ここではこの条件を用いる．

以上の手順を Matlab 言語を用いて図 **1** に示す．このアルゴリズムの演算量の大部分を占めるものは，n が大きい場合，列・行に対するヤコビ回転の適用であり，1 反復当たり $6(n-1)$ の演算量となる．行列の上三角部分にある非対角要素の数は $\frac{n(n-1)}{2}$ 個であるため，1 つの巡回当たりのこの演算量は $3n(n-1)^2$ となる．

2.2 Block-Oriented 手法

Block-Oriented (BO) 手法を理解するには図を見るのが最も簡単だろう．図 **2** は行巡回順序と BO 順序の 2 つについての具体的な例を示している．この図では升目の位置の組が数列の何番目に現れるかを示しており，例えば，図中 $(2,5)$ の位置に 14 という数字がある場合，数列の 14 番目の組が $(2,5)$ だと意味する．図左の行巡回順序は，C 言語の配列のように “行優先” に組を選ぶものであり，理論的によく解析されている．図右の BO 順序は，行列をブロック分割し，“ブロック内部優先” に組を選ぶ．ブロック内部やブロック間の順序は任意だが，ここではブロック内部に列優先，ブロック間に行優先の順序を用いている．またこのブロック化は再帰的に適用できる．

```
function b = checktol(x,y,z,tol)
  ctol = tol
  if x ne 0.
    ctol *= sqrt(abs(x))
  end
  if y ne 0.
    ctol *= sqrt(abs(y))
  end
  b = (abs(z) > ctol)
end
function [c,s,t] = eig2x2(x,y,z)
  g = 0.5*(y-x)/z
  if abs(g) >= RROOTEPS
    t = s = 0.5/g
    c = 1.
  else
    t =copysign(1./(abs(g) + sqrt(1. + g*g)),g)
    c = sqrt(1./(1. + t*t))
    s = c*t
  end
end
function b = jevdsweep(A,V,tol,pivs)
   b = false
   for P = pivs
     [p,q] = [P[0], P[1]]
     [x,y,z] = [A(p,p),A(q,q),A(p,q)]
     if checktol(x,y,z,tol)
       b = true
       [c,s,t] = eig2x2(x,y,z)
       A(p,q) = 0.
       A(p,p) -= z * t
       A(p,p) += z * t
       u = A(1:p-1,p)
       A(1:p-1,p) = c*u - s*A(1:p-1,q)
       A(1:p-1,q) = s*u + c*A(1:p-1,q)
       u = A(p,p+1:q-1)
       A(p,p+1:q-1) = c*u - s*A(p+1:q-1,q)'
       A(p+1:q-1,q) = s*u' + c*A(p+1:q-1,q)
       u = A(p,q+1:)
       A(p,q+1:) = c*u - s*A(q,q+1:)
       A(q,q+1:) = s*u + c*A(q,q+1:)
       u = V(:,p)
       V(:,p) = c*u - s*V(:,q)
       V(:,q) = s*u + c*V(:,q)
     end
   end
end
```

図 1　ヤコビ法のアルゴリズム．`checktol` は収束判定を行う．`eig2x2` は余弦と正弦，正接を計算する．`jevsweep` はヤコビ回転を 1 巡回分実行する．プログラム中，`RROOTPES` は計算機イプシロンの逆数平方根，`tol` は収束判定の閾値，`pivs` は巡回順序を表す大きさ $2 \times \frac{n(n-1)}{2}$ の行列．

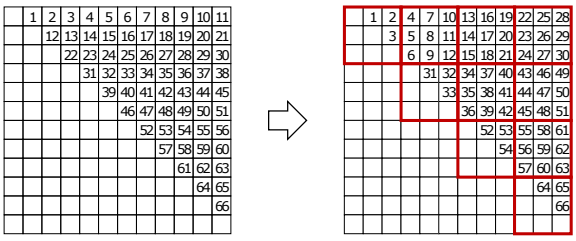

図 2　巡回順序の例．図左は行巡回順序，右は Block-Oriented 順序．

一般に，行巡回順序と，ブロック内部を列または行巡回，ブロック間を列または行巡回にした Block-Oriented 順序

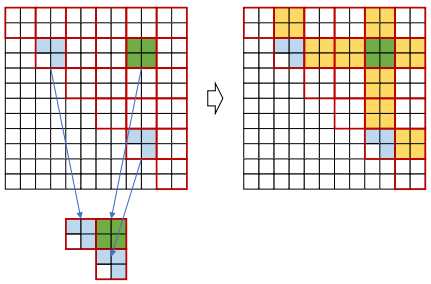

図 3　Block-Oriented 順序における非対角ブロック更新の様子．緑色のブロックを巡回する場合，水色のブロックのデータが必要になるが，黄色のブロックの更新は遅延実行できる．

は同値関係にある．ヤコビ法は，列・行単位の変換であるため，図中において同じ行，同じ列にある数字間の順序関係が変わっていないことが同値の条件であり[*1]，実際に図を例にして確認できる．同値関係にある順序によるヤコビ法は数式上同一の結果が得られる．

BO 手法の利点はアクセスの局所性である．図 **3** に非対角ブロック $(2,5)$ 内部を巡回するときの，行列更新の様子を図示する．ヤコビ法は $\theta^{(k)}$ の計算と行列の更新を交互に繰り返す計算だが，データ間の依存関係を考えると，計算順序を交換できる部分がある．とくにブロック単位で考えた場合に，あるブロック内部の巡回において $\theta^{(k)}$ を計算するために必要なブロックはごく少数であり，他のブロックの更新は遅延実行できる．例えば $(2,5)$ ブロック内部を巡回するときは，$(2,2)$, $(2,5)$, $(5,5)$ ブロックのデータのみが必要である．これは部分行列のヤコビ法を計算し，結果を全体に適用していると理解することもできる．

本稿では，部分行列に対するヤコビ法実行のことをピボットブロック更新，その他のブロックの更新のことを非対角ブロック更新と呼ぶ．この順序で計算を行った場合の利点は，逐次的で高性能化が難しいピボットブロック更新は小さな部分行列の中で行われるため，キャッシュメモリーを利用できることである，また，演算量の多い非対角ブロック更新においては $\theta^{(k)}$ が事前に求まっているためより簡単な計算構造となることも利点である．

非対角ブロック更新は演算量の大部分を占めるため重要である．図 **4** に非対角ブロック更新のアルゴリズムを示す．本稿ではこの部分に高性能化手法を適用することでヤコビ法全体の高速化を目指すものである．

2.3　高速平面回転

高速平面回転 (FPR) はヤコビ回転 $G = G(p,q,\theta)$ の適用における演算量を削減する手法である．ヤコビ法では右側・左側適用の両方があるが，転置すれば同じであるため，この節では右側適用の場合のみを考慮する．ヤコビ回転は

[*1] 2 つのヤコビ回転 $G_1 = G(p_1,q_1,\theta_1), G_2 = G(p_2,q_2,\theta_2)$ について p_1,q_1,p_2,q_2 がすべて相異なるとき $G_1G_2 = G_2G_1$ が成り立つこと（ヤコビ回転の並列性）を用いる．

```
function offdupd(b,transA,A,transB,B,cs)
  if transA
    A = A'
  end
  if transB
    B = B'
  end
  for i=1:b
    for j=1:b
      [c,s] = [cs[0,j,i],cs[1,j,i]]
      u = A(:,j)
      A(:,j) = c*u - s*B(:,i)
      B(:,i) = s*u + c*B(:,i)
    end
  end
  if transA
    A = A'
  end
  if transB
    B = B'
  end
end
```

図 4　非対角ブロック更新のアルゴリズム．`trasA`, `transB` は転置の制御変数，`cs` はピボットブロック更新で得られた余弦と正弦を並べたもの．

1 つの要素を除いて単位行列と一致するため，右側適用の場合，行列の 2 つの列のみが変化する．この 2 つの列をそれぞれ x_p, x_q とおくと，ヤコビ回転の更新式は

$$\begin{bmatrix} x'_p & x'_q \end{bmatrix} = \begin{bmatrix} x_p & x_q \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} cx_p - sx_q & sx_p + cx_q \end{bmatrix}. \tag{4}$$

ただし $c = \cos\theta$, $s = \sin\theta$ とおいた．この計算はベクトルの長さを n とすると，積が $4n$ 回，和が $2n$ 回の合わせて $6n$ 演算となっている．

FPR は，ベクトルとスケーリング成分を分離し，各個に更新すること，平面回転を対角 1 となるようスケーリングすることによって，ベクトル更新における演算量を削減する．いまある 0 でないスカラー d_p, d_q によって $d_p y_p = x_p$, $d_q y_q = x_q$ とおく．そして式 (4) を変形し次を得る:

$$\begin{aligned} \begin{bmatrix} x'_p & x'_q \end{bmatrix} &= \begin{bmatrix} y_p & y_q \end{bmatrix} \begin{bmatrix} d_p c & d_p s \\ -d_q s & d_q c \end{bmatrix} \\ &= \begin{bmatrix} y_p & y_q \end{bmatrix} \begin{bmatrix} 1 & \frac{d_p s}{d_q c} \\ -\frac{d_q s}{d_p c} & 1 \end{bmatrix} \begin{bmatrix} d_p c & \\ & d_q c \end{bmatrix}. \end{aligned} \tag{5}$$

これは y_p, y_q 自体の更新とスカラーの更新が分離した式になっている．より簡単に，$\alpha = \frac{d_p s}{d_q c}$, $\beta = -\frac{d_q s}{d_p c}$ とおき，更新後の値を次のように定義する：

$$\begin{aligned} y'_p &= y_p + \beta y_q, & y'_q &= \alpha y_p + y_q, \\ d'_p &= d_p c, & d'_q &= d_q c. \end{aligned} \tag{6}$$

このとき $x'_p = y'_p d'_p$, $x'_q = y'_q d'_q$ が成り立つため，これはスケーリングした形における更新式である．また式 (6) は xAXPY 計算の形であるため積和演算を利用でき，演算量は $4n$ 回へと削減されている．

FPR の問題点はベクトルの拡大である．計算後，$\begin{bmatrix} y'_p & y'_q \end{bmatrix}$ のスペクトルノルムは $|\sec\theta|$ 倍される．ヤコビ法では多数の回転を行うため，拡大が集積し，やがてオーバーフローする虞がある．しかしながら巡回ヤコビ法の場合には，これは悲観的な想定である．なぜならば，$|\theta| \le \frac{\pi}{4}$ の条件があるため，$|\sec\theta| \le \sqrt{2}$ という小さな値となり，また実際の計算において $\theta^{(k)}$ は急速に 0 へ収束するためである．

一方，アルゴリズムとして完成させるためには楽観的でい続けることはできない．Anda & Park の Self-scaling Rotation (SSR) [1] は FPR の演算量削減効果を維持しつつ，オーバーフローの可能性を減らす手法であり，LAPACK の片側ヤコビ SVD 手法にも実装されている．SSR は FPR に加えて，2 つの新たな更新式を用いる：

$$\begin{aligned} &\begin{bmatrix} d_p & \\ & d_q \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \\ &= \begin{bmatrix} 1 & \\ -\frac{d_q s}{d_p c} & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{d_p sc}{d_q} \\ & 1 \end{bmatrix} \begin{bmatrix} d_p c & \\ & d_q/c \end{bmatrix} \qquad (7) \\ &= \begin{bmatrix} 1 & \frac{d_p s}{d_q c} \\ & 1 \end{bmatrix} \begin{bmatrix} 1 & \\ -\frac{d_q sc}{d_p} & 1 \end{bmatrix} \begin{bmatrix} d_p/c & \\ & d_q c \end{bmatrix}. \qquad (8) \end{aligned}$$

これらは計算手順は異なるがどれも xAXPY を用いて実装できる形であり，演算量も同一である．重要な点はスケーリング項の形であり，式 (7,8) では，片方が拡大，片方が縮小となっている．そこで 3 つの回転を使い分けることで，スケーリング項の制御が可能となる．

SSR では次の選択基準によって 3 つの回転を使い分ける:

- $d_p \ge 1 \cap d_q \ge 1$ のとき，式 (5),
- $d_p < 1 \cup d_q < 1$ かつ $d_p > d_q$ のとき，式 (7),
- 以上のどれでもない場合，式 (8).

この基準の結果として，d_p, d_q は 1 のあたりを上下するような動作をするため，オーバーフローに至る可能性を減らせる．しかし頻繁に動作が切り替わることは高速化の上では問題である．分岐にも計算コストが必要であり，またループ最適化手法の適用を困難にする．そこで SSR の安全性と分岐の排除の両立ができる手法が求められる．

ヤコビ法において FPR や SSR を用いる場合は，行列とスケーリング成分を管理する．具体的には $B^{(0)} = A^{(0)}$, $D^{(0)} = I$, $U^{(0)} = V^{(0)}$ とおき，$D^{(k)} B^{(k)} D^{(k)} = A^{(k)}$, $U^{(k)} D^{(k)} = V^{(k)}$ が成り立つよう，$D^{(k)}$ の対角要素，$B^{(k)}$ の列・行，$U^{(k)}$ の列を上式を用いて更新する．

3. スケーリングによる Self-scaling Rotation の分岐排除手法

我々は SSR の問題である分岐をほぼ排除する手法を提案する．我々のアイデアは，楽観的に FPR を用い，オーバーフローが発生しそうな状況でのみ SSR を用いることである．ただし複雑な更新式や選択基準には手を入れず，行

列のスケーリングと初期値の設定のみでこれを実現する．

具体的には 2 つの正定数 w_0, w_1 を用いて，$B^{(0)}, D^{(0)}$ の値を前節のものの代わりに次を用いる：

$$\bar{B}^{(0)} = w_0 A, \qquad \bar{D}^{(0)} = w_1 I. \tag{9}$$

このとき $\bar{A} := \bar{D}^{(0)}\bar{B}^{(0)}\bar{D}^{(0)} = w_0 w_1^2 A$ であるため，定数倍を除いて一致する．そこで A の代わりに $\bar{A}$ に対して SSR を用いたヤコビ法を行うことで容易に A の固有対を得る．一方，この変形によって，$\bar{D}^{(0)}$ の初期値を w_1 によって任意に設定可能となる．そこで w_1 を大きな値に設定すれば，$\bar{D}^{(k)}$ の対角要素が縮小し 1 に達するまでの間は，SSR の選択基準によって式 (5) のみが選ばれる．

この手法において w_0 と w_1 の設定基準が重要である．$\bar{D}^{(k)}$ の対角要素ははじめ縮小し，1 以下となると SSR の選択基準によって 1 あたりを上下する．そこで $\bar{D}^{(k)}$ の対角要素を $O(1) \approx 1$ で下から押さえられると仮定すると，

$$\begin{aligned} O(1)\left\|\bar{B}^{(k)}\right\|_{\max} &\le \left\|\bar{D}^{(k)}\bar{B}^{(k)}\bar{D}^{(k)}\right\|_{\max} \\ &\le n\left\|\bar{D}^{(0)}\bar{B}^{(0)}\bar{D}^{(0)}\right\|_{\max} \\ &\le n w_0 w_1^2 \|A\|_{\max}. \end{aligned} \tag{10}$$

よって，最後の項がオーバーフローしないように w_0 と w_1 を決定することを目標とする．我々は次の値を用いている：

$$w_1 = \sqrt{\frac{c_{\max}}{n w_0 \|A\|_{\max}}}. \tag{11}$$

ただし $c_{\max}$ を正規化数の最大値とする．

次に w_0 の設定基準が問題である．w_0 が小さいほど，w_1 は大きくなる．しかし行列のダウンスケーリングは精度悪化の可能性がある．例えば，行列が非正規数を含むようになれば精度低下の虞が生じる．また，ヤコビ法は相対精度で固有値を計算可能な場合があるため，極めて微小な固有値を含む場合，微小なスケーリングであってもアンダーフローの可能性がある．そこで $w_0 = 1$ が最も安全である．微小な固有値は無視してもよい場合 (倍精度の場合 10^{-300} 程度など)，次の値が有用である：

$$w_0 = \min\left(1, \|A\|_{\max}^{-1}\right). \tag{12}$$

さらに，行列が特別な構造を持っていないなど，相対精度で固有値を求める必要がない場合，次の値が有用である：

$$w_0 = c_{\text{eps}}^{-1} c_{\text{sfmin}} \|A\|_{\max}^{-1}. \tag{13}$$

ただし c_{eps} は計算機イプシロン，c_{sfmin} は，0 の次に小さな正の正規化数 $c_{\min}$ に対して安全係数をかけた値であり，$c_{\text{sfmin}} := c_{\text{eps}}^{-1} c_{\min}$ と定義する．つまりこの値は安全係数を二重にかけているが，これは非正規化数から十分以上に距離を置きたいという意図である．

我々の手法の利点は，SSR 手法自体には一切手を加えず，事前のスケーリングと初期値設定のみで実装できる点である．分岐排除ができればカーネルチューニングが容易となり，分岐のコストも削減できる．また我々の手法は，良い状況では高速動作し，危険な状況では安全を重視する，という合理的な仕組みとなっている点も好ましい．ただし，稀ではあるが SSR で追加された更新式を使う可能性があることに注意が必要である．そこでプログラムには SSR に必要なものをすべて実装しておかなければならない．

```
1  void driver(
2  char transA,char transB,int nk,int b,
3  double*A,int lda,double*B,int ldb,double*cs){
4    double pA[BJ][BK];
5    double pB[BI][BK];
6    for(int ii=0; ii<b; ii+=BI){
7    for(int jj=0; jj<b; jj+=BJJ, cs+=2*BJJ*BI){
8    for(int kk=0; kk<nk; kk+=BK){
9      packb(transB, kk, ii, B, ldb, pB);
10     for(int j=jj; j<jj+BJJ; j+=BJ){
11       packa(transA, kk, j, A, lda, pA);
12       ukernel(pA, pB, cs+2*(j-jj)*BI);
13       unpacka(transA, kk, j, pA, A, lda);
14     }
15     unpackb(transB, kk, ii, pB, B, ldb);
16 }}}}
```

図 5　ドライバーの簡略化したソースコード

4. ヤコビ回転カーネルの構成

非対角ブロック更新の計算は図 4 のように，単純な三重ループ構造（二重ループ+列ベクトル計算）となっており，行列積の三重ループ構造と類似している．そこで我々は Level-3 BLAS で用いられる階層的なループブロック化手法 [4] を適用することで，ヤコビ回転の高性能化を試みた．

4.1 構成の概要

我々のヤコビ回転計算ルーチンははじめに回転成分を調べ，$\theta^{(k)} = 0$ の回転の割合や SSR の追加更新式が含まれているかを確認する．前者があまりに多い場合は回転をスキップすることで演算量を削減できる．また，後者の回転は 1 つでもあれば以下のカーネルは利用できないため，確認が必要となる．以上に当てはまらない場合，次に述べるドライバーが呼び出される．ドライバーは高度に最適化される部品であるマイクロカーネルを動作させる部品である．

図 5 にドライバーの具体的なコードを示す．ドライバーは図 4 の三重ループをブロック化することで，マイクロカーネルが担当するループを固定サイズにし，キャッシュ再利用性を向上させる．固定サイズのループは SIMD 並列化やループアンローリング化の対象となり，図 5 における定数 BJ や BK がそれに相当する．また，BI と BJJ はキャッシュ再利用性向上のためのブロックサイズである．このようなループブロック化が可能かは，行列積と比べて明白ではないが，巡回順序の同値性から言える．また単にループ

```
1 void ukernel(double A[BJ][BK],double B[BI][BK],
      double*cs){
2   for(int i=0; i<BI; ++i){
3     for(int j=0; j<BJ; ++j, cs+=2){
4       for(int k=0; k<BK; ++k){
5         double t = A[i][k];
6         A[i][k] += cs[1] * B[j][k];
7         B[i][k] += cs[0] * t;
8 }}}}
```

図 6　ナイーブなマイクロカーネル実装のソースコード

依存性解析を行ってもよい．

ドライバーはまた，再利用されるデータを並び替えることでデータアクセス性能を向上させる．図 4 の pA と pB が並び替えたデータを格納するための配列であり，packa, unpacka, packb, unpackb の 4 つが並び替えのためのルーチンである．並び替えルーチンによって，lda や ldb によるストライドアクセス，また transA や transB による転置などを隠蔽し，マイクロカーネル内部に適したデータ順序へ変換できる．

我々の実装ではさらに次のような改変を行っている．

- 端数ループのためのコードの追加
- kk ループのスレッド並列化
- packa, unpacka とマイクロカーネルの結合
  パックしたデータをそのまま SIMD レジスター上で用いるため，直接アクセス用・転置アクセス用のマイクロカーネルをそれぞれ作成する．
- unpackb と packb の結合
  unpackb と，次のループに呼び出される packb とを結合したルーチンを作り，アクセス局所性を高める．
- パック済みデータ用ドライバー
  事前に行列の全要素を並び替えておくことで，ドライバー内部での並び替えを省く．MKL の xGEMM も同様の機能を持つ．ただし V には適用可能だが，両側回転である A には適用できない．

4.2　マイクロカーネルの構成

図 6 にマイクロカーネルのナイーブな実装を示す．ここでは高速平面回転を用いているため，再内側が積和演算となっていることに注意されたい．このようにマイクロカーネルは単純な構造を持っており，コンパイラーによる最適化に適している．またこのように性能面で重要な部分を小さなコード片として切り出すことで，手動最適化を行った場合でも移植性の問題を小さく抑えられる．

我々は図 6 に相当するコードに最適化指示詞などを加えたソースコードも用意しており，未知の環境や，優れたコンパイラーがある場合などではそれを用いるよう設計している．しかし我々は現時点においてコンパイラーのみによって理想的なオブジェクトコードを出力させるに至っていない．そこで我々は手動による最適化を行ったマイクロカーネルを多数実装しており，次に詳細を示す．

表 1　ヤコビ回転カーネルの設定

	SDB	HSW	KBL	SKX	KNL	SP8	SP11
BK	8	8	8	16	16	16	16
BJ	4	4	4	8	8	8	8
BI	144	160	144	96	96	208	208
BJJ	320	320	320	480	200	360	360
i-unroll	2	2	2	2	2	2	2
i-sched				✓	✓	✓	✓
i-preload						✓	✓
prefetch	cs	cs	cs	cs	cs	cs	cs
trans	✓	✓	✓	✓	✓		
impl	intrin	intrin	intrin	intrin	intrin	asm	asm

4.3　マイクロカーネル実装の詳細

我々はいくつかの CPU 環境に対してマイクロカーネルの高性能化を試みた．対象とした CPU は次のものである: Intel Sndybridge-E (SDB), Haswell (HSW), Kabylake (KBL), Skylake-SP (SKX), Knights Landing (KNL), 富士通 SPARC64 VIIIfx (SP8), XIfx (SP11)．ここではブロック幅のような単なるパラメーターチューニングだけではなく，SIMD 並列化やループアンローリング・スケジューリングのような大規模なコード改変が必要なもの，命令選択などの大局的な知識が必要となるものも行っている．そこで現状では実装作業のほぼすべてが自動化できていない．

我々の基本的な戦術は次である．1) A[BJ][BK] の全要素をレジスター上に配置，2) k ループをフルアンロール& SIMD 並列化，3) j ループをフルアンロール，4) i ループをアンロール，また必要に応じてプリフェッチとプリローディング．とくに重要なものは戦術 1 であり，データ再利用性を高めるだけでなく，ロード・ストア命令の数自体を減らすことで，命令単位での演算密度を向上させる．一方で，レジスターの容量は小さく，さらに他のデータの保存にも使われるため，おのずと BJ や BK の上限ができる．そこで戦術 4 との関係や詳細な命令スケジューリング，命令選択などと同時に考えることで，“よさそうなもの” を選ぶ．

命令スケジューリング・選択では次のことを考慮した．レジスタ数が少ない Intel 社の CPU では，Out-of-Order 機能にスケジューリングを任せた．富士通社の CPU はレジスタ数が非常に多いため，積極的にスケジューリングを行った．Intel 社の SIMD 演算命令は 3 オペランドであるため，積和演算は入力のうち 1 つを破壊する．これは図 6 のような，書き換え前の値を計算に用いるときに問題となり得る．実際に，KNL では命令デコーダーが狭帯域であり，非演算命令（レジスター間コピー命令など）の比率が実行性能を制約するため，KNL での実行効率は必ず 66% より小さくなることが判明した．富士通社の CPU は 4 オペランドの積和演算であるためこの問題はない．

表 1 に現状の実装におけるパラメーターと適用した最適化手法の一覧を示す．表のうち，i-unroll は i ループのア

ンロール幅，i-sched はアンロールした命令の命令レベル並列性を基に並び替えたか，i-preload はロード命令を前の回転に行うか，prefetch はどの変数についてのプリフェッチを行ったか，trans は転置アクセス用マイクロカーネルが実装されているか，impl はマイクロカーネルの実装手法であり，intrin は C 言語と SIMD-intrinsic 関数の組み合わせ，asm はアセンブリ言語を用いて実装したことを表す．

5. 実験結果

ここではヤコビ回転カーネルの性能測定結果を示す．実験は，パラメーターを変化させながらドライバーを複数回呼び出すことで行う．このため，データ並び替えのコストやメモリーアクセスのコストも含む．kk ループをスレッド並列化した場合と，シングルスレッドで実行した場合の両方で実験を行い，行列サイズはマルチスレッド時は $b = 16$–2048, シングルスレッド時は $b = 16$–1024 とし，また，$nk = 2b$ と設定した．このときの演算量は $8b^3$ であり，大きさ 2b の正方行列積の半分となっている．transA と transB の設定は，両方が非転置 (N)，両方が転置 (T)，両方がパック済み (P)，の 3 種類とした．実験は同じパラメーターごとに 11 回行い，最初の 1 回を除いた上で，最小，最大，平均値をグラフに示す．各実験の間には大量のメモリー書き込みを行うことでキャッシュ上にデータが残らないようにしている．また，Intel の CPU は SIMD 演算パイプラインをスリープ状態から復帰させるために，大きなサイズの DGEMM を実行した．

性能測定に利用した計算機を表 **2** に示す．Intel の CPU は動的な周波数制御機能 (Turbo Boost) を備えたものがあるが，ピーク性能の予測が難しくなるため，無効にしている．ただし，SKX については単純な動作で周波数を変化させるため，有効にしている．表にはシングルスレッド時/マルチスレッド時の周波数を記載している．また SP11 については 1NUMA ノードのみを使っている．

まず図 **7** にシングルスレッド時のピーク性能比を示す．性能は transA/B によって大きく変化するが，多くの条件で P が最大性能となっている．そのときのピーク性能比は，KNL を除くすべての環境で 80%に到達している．N や T を用いると性能が低下するものが多いが，その場合であっても，SND, HSW, KBL, SP8 の環境ではピーク性能比 80%に近い値となっている．KNL の性能は他と比べて劣っており，P を用いても 60%に達しない．また KNL は性能のブレが他と比べて大きいことも確認できる．次に図 **8** にマルチスレッド時の性能を示す．こちらのグラフは，シングルスレッドのときのグラフと比べてなだらかなカーブとなっており，コア数の多い KNL で顕著である．ピーク性能比自体もシングルスレッド時よりも低下しているが，P の場合，KNL を除くすべての CPU においてピーク性能比 80%に達している．

6. まとめ

我々は Block-Oriented 巡回ヤコビ法の非対角ブロック更新の高速化に取り組んだ．第一に，ヤコビ回転の演算量を低減する手法である FPR と SSR の問題点の解決に取り組み，楽観的に FPR を用い，危険な状況では SSR に切り替える手法を考案した．また第二に，非対角ブロック更新の計算パターンを解析し，Level-3 BLAS で用いられる階層的なループブロック化を適用することで，計算カーネルを構成した．我々のヤコビ回転カーネルは，Level-3 BLAS 構成に倣い，データ並び替えによるデータ順序抽象化機能や，マイクロカーネル構成による高い性能移植性を持つ．この構成を基にした我々の実装は，7 つの異なる CPU 環境上で高い性能を示している．

我々は今後，本手法で作成した計算カーネルをヤコビ法プログラムに組み込むことを計画している．本稿で取り上げた非対角ブロック更新は計算の主要な部分であるため，計算性能の向上が期待される．また我々は，他のヤコビ法計算部分の性能向上や，非対角ブロック更新の並列化と並列ヤコビ法との組み合わせなどにより，さらなる改良を試みたいと考えている．

謝辞

本研究は筆者らと電気通信大学 山本有作教授との共同研究の一環として行われたものである．

本研究には理化学研究所の共同利用計算機システム (Hokusai) を利用した．

本論文の結果（の一部）は，理化学研究所のスーパーコンピュータ「京」を利用して得られたものです．（課題番号:ra000005）

参考文献

[1] Anda, A. A. and Park, H.: Fast Plane Rotations with Dynamic Scaling, *SIAM J. Matrix Anal. Appl.*, Vol. 15, No. 1, pp. 162–174 (online), DOI: 10.1137/S0895479890193017 (1994).

[2] Demmel, J. W. and Veselić, K.: Jacobi's Method is More Accurate than QR, *SIAM J. Matrix Anal. Appl.*, Vol. 13, No. 4, pp. 1204–1245 (online), DOI: 10.1137/0613074 (1992).

[3] Forsythe, G. E. and Henrici, P.: The cyclic Jacobi method for computing the principal values of a complex matrix, *Trans. Am. Math. Soc.*, Vol. 94, No. 1, pp. 1–23 (online), DOI: 10.1090/S0002-9947-1960-0109825-2 (1960).

[4] Goto, K. and Geijn, R. A. V. D.: High-performance implementation of the level-3 BLAS, *ACM Trans. Math. Softw.*, Vol. 35, No. 1, pp. 1–14 (online), DOI: 10.1145/1377603.1377607 (2008).

[5] Jacobi, C. G. J.: Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen., *J. für die reine und Angew. Math.*, Vol. 30, pp. 51–94 (1846).

[6] Nazareth, L.: On the convergence of the cyclic Jacobi

表 2　計算機諸元

	SDB	HSW	KBL	SKX	KNL	SP8	SP11
model	Xeon E5-2660	Core i7-4790	Core i5-7300HQ	Xeon Gold 6126	Xeon Phi 7210	SPARC64 VIIIfx	SPARC64 XIfx
frequency	2.2GHz	3.6GHz	2.5GHz	3.5/2.3GHz	1.1GHz	2.0GHz	1.975GHz
SIMD width	256bits	256bits	256bits	512bits	512bits	128bits	256bits
# of SIMD regs.	16	16	16	32	32	128	128
# of cores	8	4	4	12	64	8	16
L1D cache size	256KiB	128KiB	128KiB	384KiB	4MiB	256KiB	1MiB
L2D cache size	2MiB	1MiB	1MiB	12MiB	32MiB	6MiB	12MiB
LL cache size	20MiB	8MiB	6MiB	19.75MiB			
peak perf. (single)	17.6GFlop/s	57.6GFlop/s	40GFlop/s	112GFlop/s	35.2GFlop/s	16GFlop/s	31.6GFlop/s
peak perf. (multi)	140.8GFlop/s	230.4GFlop/s	160GFlop/s	883.2GFlop/s	2252.8GFlop/s	128GFlop/s	505.6GFLop/s
compiler ver.	icc 18.0.1		icc 19.0.1.144			fcc 2.0.0 20180712	fcc K-1.2.0-25

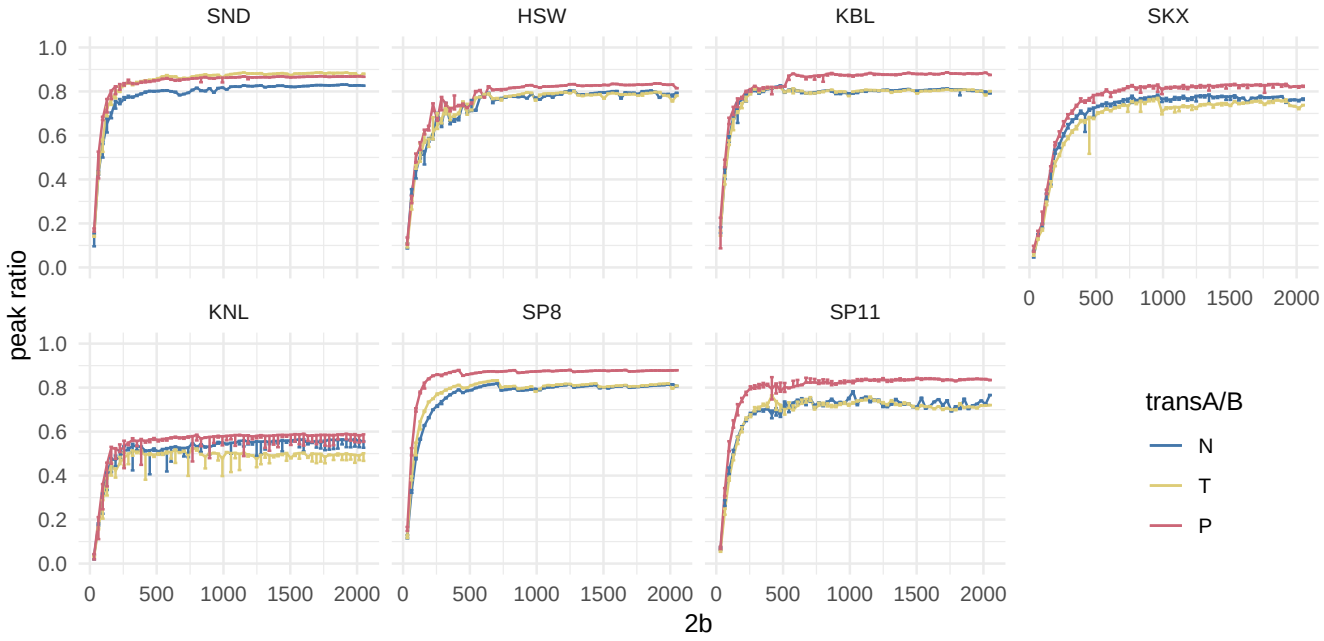

図 7　ヤコビ回転カーネルの性能（シングルスレッド）

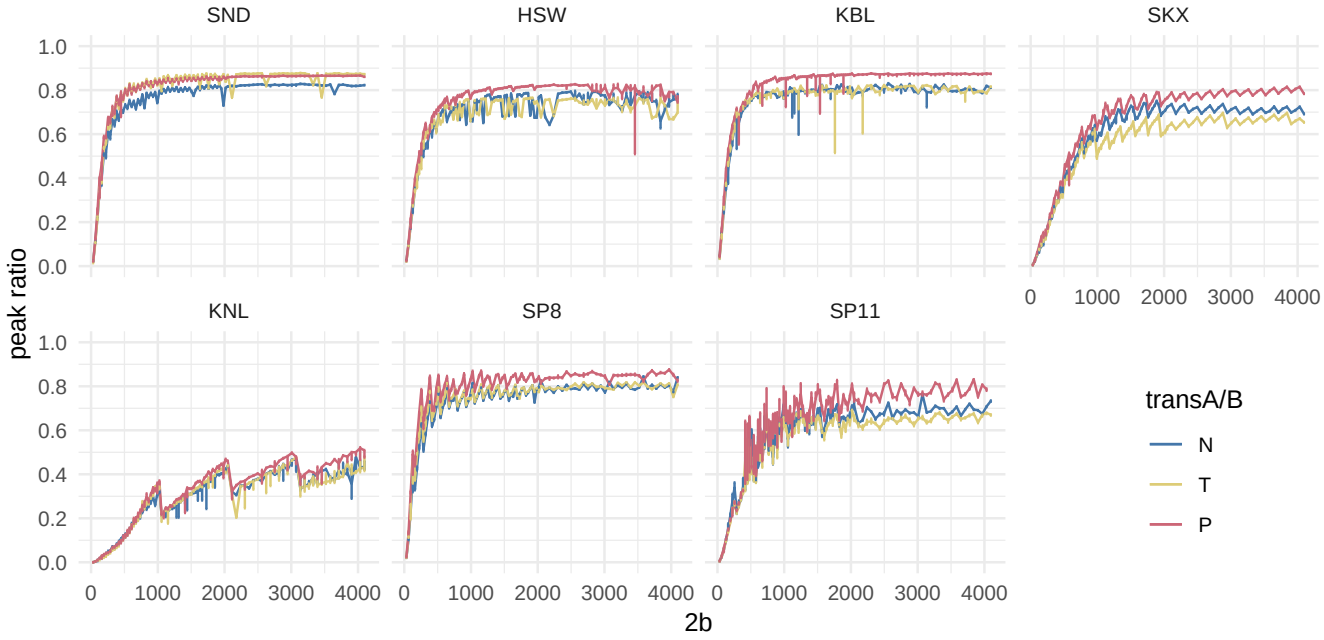

図 8　ヤコビ回転カーネルの性能（マルチスレッド）

method, *Linear Algebra Appl.*, Vol. 12, No. 2, pp. 151–164 (online), DOI: 10.1016/0024-3795(75)90063-4 (1975).

[7] Van Zee, F. G., van de Geijn, R. A. and Quintana-Ortí, G.: Restructuring the Tridiagonal and Bidiagonal QR Algorithms for Performance, *ACM Trans. Math. Softw.*, Vol. 40, No. 3, pp. 1–34 (online), DOI: 10.1145/2535371 (2014).