高性能計算機に適したヤコビ SVD 手法の実装技術と性能解析

工藤 周平

電気通信大学 大学院情報理工学研究科

博士（工学）の学位申請論文

２０１８年３月

高性能計算機に適したヤコビ SVD 手法の実装技術と性能解析

博士論文審査委員

主査　　山本　有作　教授

委員　　山本　野人　教授

委員　　緒方　秀教　教授

委員　　成見　哲　　教授

委員　　山﨑　匡　准教授

委員　　横川三津夫　教授

Analysis and implementation techniques of the Jacobi SVD algorithm for high performance computers

Shuhei Kudo

Abstract

The massive-parallelism of recent high-performance computers (HPC) requires all algorithms running on them, including the singular value decomposition (SVD) algorithm, to have high-level of scalability. To achieve this goal, we focus on the one-sided Jacobi SVD (OSJ) algorithm. The OSJ with extension techniques like blocking and parallelization has a potential for high parallel efficiency due to its large-grained parallelism, but some of its theoretical properties are unknown. Furthermore, implementation techniques of the OSJ for HPC have not been studied well.

This thesis aims to analyze the blocking and parallelization techniques of OSJ both theoretically and experimentally and provides new parallelization techniques for HPC. It consists of seven chapters.

In the first chapter, the motivation and contribution of the thesis are described.

In the second and third chapters, as the background, the current trend of HPC and applications of SVD are described. The idea and detailed algorithm of the OSJ and the existing extension techniques are also described.

In the fourth chapter, a new bound on the orthogonality error of Hari's V2 method is provided. The V2 method is a blocking method for the OSJ suitable for HPC. The bound is tighter than the existing one due to Drmač, thanks to the exploitation of the diagonally scaled structure of the matrix.

In the fifth chapter, a new implementation method for HPC based on 2D blocked data distribution and all-reduce type communication is described. Theoretical analysis of the method shows the number of communication is reduced compared with the existing data distributions. Experimental results on highly parallel machines support the theoretical prediction and show good strong-scalability of the method. Bečka's dynamic ordering method, which can reduce the number of iteration of OSJ, is also analyzed and a new bound on the global convergence rate of the method is provided.

In the sixth chapter, implementation techniques of DSYRK for a many-core CPU, a new architecture of CPU which is used in HPC, is considered. DSYRK is a variation of matrix-matrix multiplication used in the V2 method. The new parallelization technique of DSYRK which utilizes all the three dimensions for parallelism in the matrix-matrix multiplication accelerates the performance of DSYRK on Xeon Phi, an Intel's many-core CPU, up to 75% of the theoretical peak performance.

The last chapter concludes the thesis and provides the future work of this study.

高性能計算機に適したヤコビ SVD 手法の実装技術と性能解析

工藤　周平

概要

現代の高性能計算機は高い並列性をもっており,特異値分解（SVD）のような科学技術計算に用いられる基本的な行列計算についてもこのような高い並列性に対応することが求められている．片側ヤコビ法は広く用いられている行列計算ライブラリ LAPACK に実装された SVD 計算手法の一つであり，高い計算精度と実用的な速度を併せ持つ．また，ブロック化や並列化などの拡張手法と組み合わせることで，高い計算効率と粗粒度な並列性を持たせることができる．そのため片側ヤコビ法は高性能計算機に適すると考えられているが，ブロック化や並列化については理論的・実験的検証が不十分であり，また現代の高性能計算機に向けた実装の研究も進んでいなかった．

本論文は，片側ヤコビ法の拡張手法について理論的・実験的解析を行うとともに高い並列性能を持つ実装手法の開発とその解析を行うものであり，7 つの章で構成される．

第 1 章では現状の片側ヤコビ法が持つ精度面での利点と性能面での問題点について実例を用いて示すことで本研究の目的を説明し，また本論文の成果の概略と構成を述べる．

第 2 章では本研究の背景となる事柄についてまとめる．第一に現代の高性能計算機が持つ特性についてまとめ，高度な並列性に対応することが SVD 計算アルゴリズムに不可欠であることを示す．第二に，SVD とその科学技術計算における応用を示し，また近年明らかにされた SVD の誤差の性質についてまとめる．そして最後に SVD 計算アルゴリズムについて概要を示し，ヤコビ法やその SVD への応用手法である片側ヤコビ法の基礎となるアイデアについて説明する．

第 3 章では片側ヤコビ法とそれに対する既存の拡張手法について詳細に解説する．ここではじめに片側ヤコビ法の詳細な手順を示し，その問題点を指摘することで拡張手法の必要性を説明する．そして，第一に並列化と密接な関係があるヤコビ法の巡回順序について，第二に片側ヤコビ法の高性能化に必要なブロック化手法について，第三に近年開発されヤコビ法の劇的な高性能化につながった前処理手法について説明する．

第 4 章では本論文の成果の 1 つである，片側ヤコビ法のブロック化したときにおける誤

差解析の結果を示す．ブロック化手法はいくつか考えられるが本論文ではとくに高性能計算に適した Hari の V2 手法を対象にし，片側ヤコビ法の収束性に影響を与える直交性に対する誤差について調べている．本研究における解析は Drmač による先行研究から発展させたものであり，行列の対角スケーリングされた構造を利用することでより良い上限が得られている．また多数の行列を用いた実験による検証により，上限に出現する行列に依存した係数が実用上小さいことを示している．

第 5 章では第一に，本論文の 2 つ目の成果である二次元ブロック分割によるデータ配置と AllReduce 型の計算を用いた分散メモリ向け並列化手法について論じている．現代の高性能計算機のほぼすべてが分散メモリ型並列計算機であるが，このような計算機ではデータ配置が計算や通信の特性を決定づける．本論文ではまず V2 手法の計算パターンを利用して，二次元ブロック分割と AllReduce 型計算の組み合わせた新規の並列化手法を示している．そして計算量や通信量を理論的に解析し，この手法が従来のデータ分散手法と比較してオーダーの意味で通信回数を削減することを示している．また，高性能計算機による実験的検証によってこの手法の性能が理論から得られた予測に従うことを確かめている．さらに京コンピュータを用いた大規模並列計算機上での性能検証を行い，1 万次元の行列の SVD を計算するときに約 2 万 5 千コアまで性能が向上するという高い強スケーラビリティを確認している．第二に，本論文の 3 つ目の成果である，Bečka らの動的順序と呼ばれる並列化手法について理論的検証を行い，一次収束性に対する新しい上界を求めている．この上界は従来のものとは異なり並列数と反比例する形となっており，動的順序の並列計算に適した特性を示している．

第 6 章では本論文の 4 つ目の成果である，DSYRK の高性能実装手法について論じている．DSYRK は BLAS において定義された対称な形を持つ行列積であり，V2 手法における主要な計算の 1 つである．本研究ではメニーコア CPU という将来使用されると目されるアーキテクチャを持つ，Xeon Phi を用いたときにおける問題点を検証する．DSYRK の既存実装では DSYRK の持つ対称な構造のため Xeon Phi が持つ高い性能を活かせていなかったが，本研究ではメニーコア CPU の持つ高い並列性に対処するため，行列積の持つ 3 次元的な並列化軸をすべて利用する新規並列化アルゴリズムを示している．また Xeon Phi 上で性能検証結果では，理論ピーク性能の約 76%に達する高い性能を得ている．

第 7 章では本論文をまとめ，今後の展望について述べている．

目次

図目次

表目次

記号の定義

$\mathbb{R}, \mathbb{C}$	それぞれ，実数，複素数の集合.
$\mathbb{R}^n$	n 次元の実ベクトルの集合．特に指定がなければ，列ベクトル.
$\mathbb{R}^{m\times n}, \mathbb{C}^{m\times n}$	大きさ $m\times n$ の行列．前者は実行列，後者は複素行列．$n=1, m=1$ の場合はそれぞれ列ベクトル，行ベクトルと一致するものとして扱う.
$\alpha, \beta, \ldots$	スカラー．特に指定がなければ実数.
$a, b, \ldots$	スカラーまたはベクトル．ベクトルは特に指定がなければ列ベクトル．特に指定がなければ実数，または実ベクトル.
$A, B, \ldots$	行列．特に指定がなければ実行列.
$\left[\cdot\right]$	行列，またはベクトルの要素ごとの表現．要素はスカラー，ベクトルまたは行列. 空白部分は 0 が埋まっているものとして扱う.
$\mathrm{diag}(a, b, \cdots)$	$a, b, \cdots$ を対角に並べた対角行列．$a, b, \cdots$ の代わりに行列 $A, B, \cdots$ を並べた場合は，ブロック対角行列.
$\mathbb{0}_{m,n}$	$m\times n$ の零行列．$n=1$ の場合は零ベクトル，$m=1$ の場合は行ベクトルの零ベクトル，$m=n=1$ のときは 0 と一致するものとして扱う．$m=n$ の場合は $\mathbb{0}_n$ と省略し，また誤解がない場合は完全に添え字を省略し $\mathbb{0}$ と書く.
$\mathbb{1}_{m,n}$	$m\times n$ の行列であり，すべての要素が 1 であるもの．零行列と同様に，$n=1$ の場合は要素がすべて 1 の列ベクトル，$m=1$ の場合は要素がすべて 1 の行ベクトル，$m=n=1$ の場合は 1 と一致するものとして扱う．$m=n$ の場合は $\mathbb{1}_n$ と省略して書き，また誤解がない場合は完全に添え字を省略し $\mathbb{1}$ と書く.
$I_{m,n}$	$m=n$ の場合は単位行列 $I_{n,n}=\mathrm{diag}(1,1,\ldots)$．$m>n$ の場合は $I_{m,n}=\begin{bmatrix} I_{n,n} \\ \mathbb{0}_{m-n,n} \end{bmatrix}$ のように，$m<n$ の場合は $I_{m,n}=\begin{bmatrix} I_{m,m} & \mathbb{0}_{m,n-m} \end{bmatrix}$ のように零行列で拡張したもの．$m=n$ のときはとくに I_n と省略し，誤解がない場合は完全に添え字を省略し I と書く.
$A^\top$	任意の行列 A の転置行列.
A^*	任意の行列 A の随伴行列.
$\{\cdot\}$	集合.
$\mathrm{rank}\, A$	任意の行列 A の階数.
$\|A\|_2$	任意の行列 A の 2 ノルム.
$\|A\|_\mathrm{F}$	任意の行列 A のフロベニウスノルム.
$\|A\|_\mathrm{max}$	任意の行列 A の最大値ノルム．A の要素の絶対値の最大値に等しい.

$\lvert A \rvert$	任意の行列 A の絶対値行列．すべての要素の絶対値をとって並べた行列．
$\mathbf{u}$	浮動小数点数の丸めの単位．特に指定がなければ，倍精度のものを想定し，$\mathbf{u} \approx 1.1 \times 10^{-16}$．単精度浮動小数点数の場合は $\mathbf{u_s} \approx 6.0 \times 10^{-8}$．
$O_{x,y,\dots}(f(x,y,\dots))$	ランダウの記号．$x, y, \dots$ は実変数であり，$f(x,y,\dots)$ は実多変数関数．ある定数 α, β が存在し，$\forall x, y \geq \alpha \Rightarrow \left\lvert O_{x,y,\dots}(f(x,y,\dots)) \right\rvert \leq \beta \left\lvert f(x,y,\dots) \right\rvert$ が成り立つことを表す．変数が明白の場合は添え字を省略し $O(f(x,y,\dots))$ と書く．

第 1 章

緒論

1.1 目的

特異値分解 (Singular Value Decomposition, SVD) は基本的な行列演算の 1 つであり，信号処理や電子状態計算などの科学技術計算に出現する．片側ヤコビ法は特異値分解を行うための手法の 1 つであり，ある種の行列に対してすべての特異値を高い相対精度で計算できるという優れた特徴を持つが [1]，計算速度の遅さが問題であった．Drmač と Veselić は QR 前処理と呼ばれる前処理手法やその他の反復回数削減手法を組み合わせることによって，片側ヤコビ法の高精度性を保ちつつ，計算速度を大幅に改善した手法を開発した [2, 3]．この結果，片側ヤコビ法は他手法と比べて単に精度がよいだけでなく，速度の面でも他手法に対抗し得る SVD 計算手法として注目を集めるようになった．この成果は SIAM/LA Prize として表彰され [4]，また，標準的な行列計算ライブラリである LAPACK 3.2.1 [5] の一部として公開され，広く用いられるようになっている．

図 1.1に LAPACK に実装されている 4 つの特異値分解手法の精度を示す．この実験では倍精度の片側ヤコビ法（DGEJSV）で計算した特異値 $\check{\sigma}_i$ を基準として，単精度の各手法の特異値 $\hat{\sigma}_i$ との相対誤差の最大値 $\max_i \frac{|\hat{\sigma}_i - \check{\sigma}_i|}{\check{\sigma}_i}$ を調べている．実験で用いた手法は次の 4 つである：

- 片側ヤコビ法 SGEJSV (MKL 11.3)
- 二重対角化＋ QR 法 SGESVD (MKL 11.3)
- 二重対角化＋ DC 法 SGESDD (MKL 11.3)
- 二重対角化＋二分法 SGESVDX (LAPACK 3.7.1)

このうち二分法のみ MKL11.3 において実装されていないため，LAPACK 3.7.1 のものを利用しているが，内部の処理の大部分は MKL において実装された SGEBRD と SSTEVX が使われている．図 1.1から，Drmač らの片側ヤコビ法では対角行列 D_1, D_2 によって $A = D_1 B D_2$ のように行列が両側からスケーリング（Grade）された場合においても，特異値の相対誤差は $\kappa_2(D_1)$ や $\kappa_2(D_2)$ に依存せず，$\kappa_2(B)$ に比例したものとなる．また，ほぼ同様の結果が Drmač らの論文にも示されている [3, Figs. 3.1, 3.3].

一方，図 1.2は倍精度で実行した場合の各手法 (DGEJSV, DGESVD, DGESDD, DGESVDX) の実行時間を示している．図より，Drmač らによる改良版の片側ヤコビ法であっても，他手法とはまだ計算速度に大きな開きがあることが分かる．片側ヤコビ法は他の 3 手法と比べて単に計算速度が遅いだけでなく強スケーラビリティーにおいても劣っており，最も高速な DGESDD と比べて実行時間の比は，1 コアの場合でも約 7 倍，10 コアをすべてを使った場合には約 35 倍にも達する．

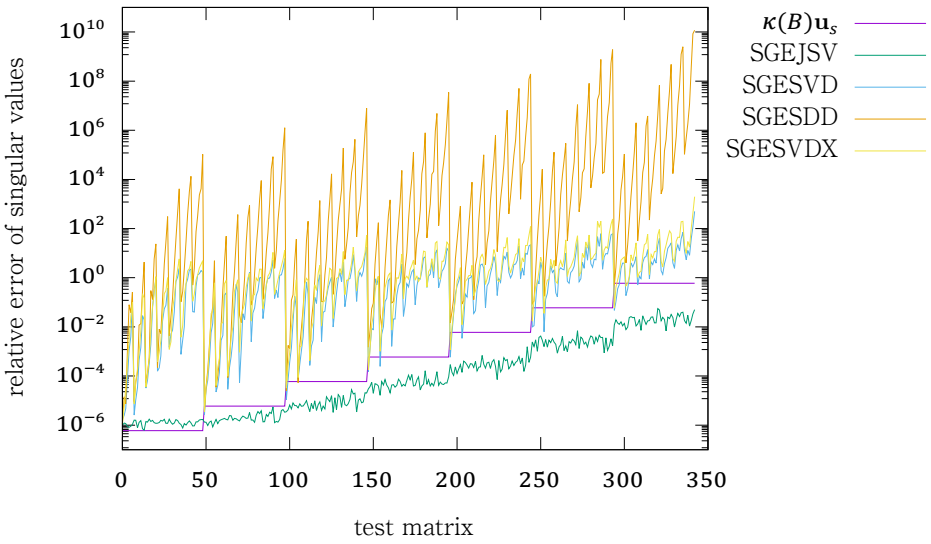

図1.1　各 SVD 手法の特異値の相対誤差．テスト行列は大きさが 100×100 であり，対角行列 D_1, D_2 によって Graded された $A = D_1BD_2$ の形をしており，$\kappa_2(B) \approx 10^i$, $\kappa_2(D_1) \approx 10^j$,$\kappa_2(D_2) \approx 10^k$ は特異値の対数が一様分布する乱数で定められている $(1 \le i, j, k \le 7)$．横軸は次で定義する行列の番号：$49(i-1)+7(j-1)+(k-1)$, 縦軸は単精度計算を用いた各手法で求めた特異値 $\hat{\sigma}_i$ と倍精度の片側ヤコビ法で計算したより高精度な結果 $\check{\sigma}_i$ との相対誤差の最大値 $\max_i \frac{|\hat{\sigma}_i-\check{\sigma}_i|}{\check{\sigma}_i}$．$\mathbf{u}_s \approx 5.96\times10^{-8}$ は単精度浮動小数点数の丸めの単位．

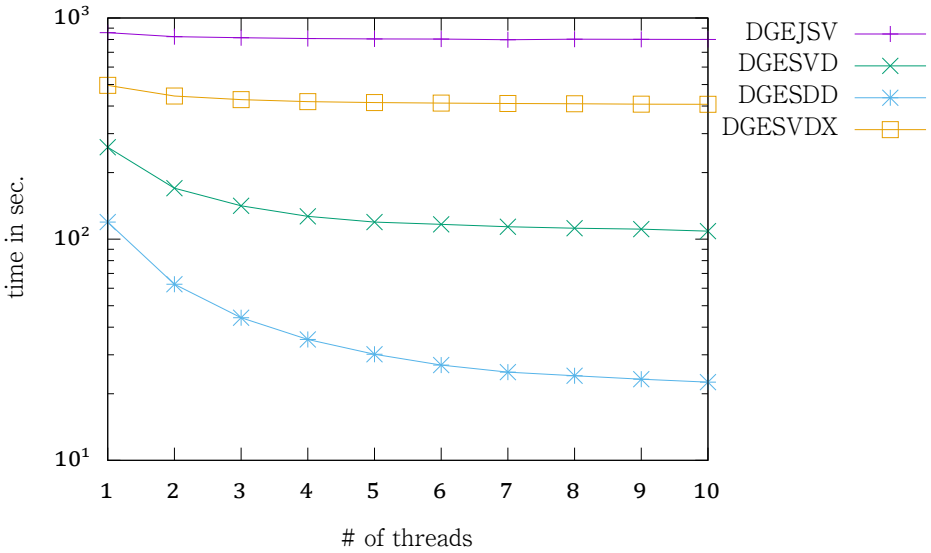

図1.2　各 SVD 手法の実行時間．行列は 5000×5000 次元であり，対角行列 D_1, D_2 によって $A = D_1BD_2$ の形を持ち，$\kappa_2(D_1) \approx \kappa_2(B) \approx \kappa_2(D_2) \approx 10$ は特異値の対数が一様分布する乱数で定められている．実験は `Xeon E5-2660 v2` の最大 10 コアを用いて行い，縦軸は実行時間，横軸はスレッド数．

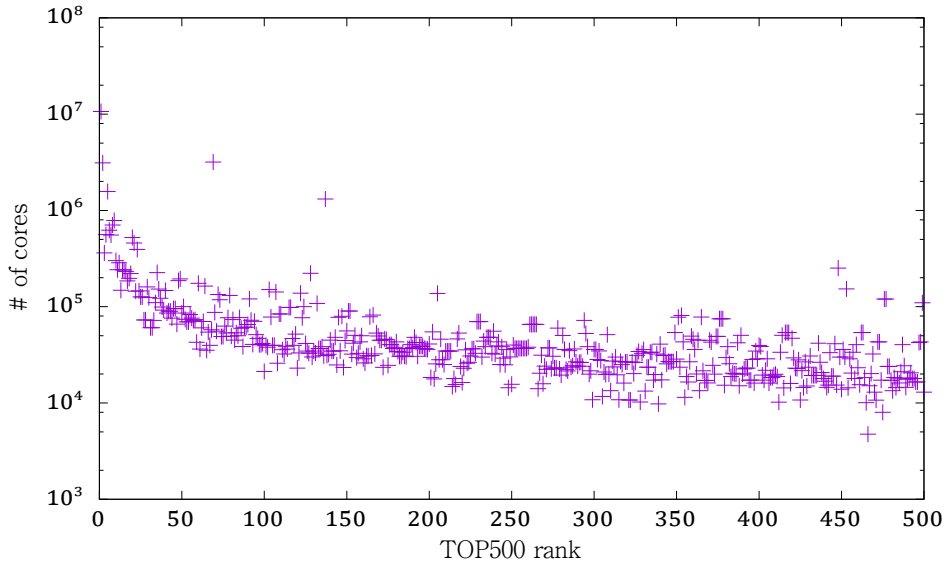

図1.3　2017 年 6 月時点での Top500 構成マシンにおけるコア数．横軸は Top500 の順位．

このように片側ヤコビ法は特異値分解の誤差については他手法と比して優れているが，現状において計算速度の面では Drmač らが言うようには優れていない．この理由は，彼らの実装にはブロック化や並列化といった最近の計算機アーキテクチャに合わせた拡張手法が取り入れられていないためである．これには次のような理由がある．片側ヤコビ法のブロック化や並列化は，他の行列分解で行うような，数式上同一な計算手順の入れ替えではなく，反復の収束性といった数理的性質自体を変化させてしまうものであり，収束の解析や誤差解析などの議論が大きく異なるものとなってしまうためである．そのため，片側ヤコビ法は他手法と比べて計算速度面で後れを取ってしまっており，これらの拡張手法の理論的に不明な点を補完することが現代的な計算機に適応させるために必要となっている．

一方，現代の計算機環境を取り巻く環境は大きく変化してきている．図 1.3はスーパーコンピューターの計算速度ランキングである Top500 [6] における 2017 年 6 月時点での登録マシンにおけるコア数を示したものである．現代においては Top500 の中でも最も低位にある計算機であっても 1 万並列に到達しており，最上位のものでは 1,000 万並列にもなっている．このように科学技術計算に用いられる計算機の並列性は極端なまでに増大しているため，この上でプログラムが効率的に動作するためには高い並列性を有しなくてはならない．

そこで本論文では，並列計算機に適した高い計算速度を持つ（＝高性能な）SVD 手法を実用化するため，片側ヤコビ法の拡張手法について理論的に不明な点を明らかにすることで，実用的な手法にすることを第一の目標とし，その上で，現代のスーパーコンピュータのような極限的な並列性を持つ計算機に適合する実装手法を開発することを第二の目標として，次の各項目をテーマに研究を行った．

- ブロック化による誤差や収束性への影響の解析
- 並列化による計算速度への影響の理論的解析
- 超並列計算機に適した実装技術の開発
- 将来の計算機への適用手法の検討

1.2 成果

本研究の第一の成果は，Hari の V2 手法 [7] と呼ばれる片側ヤコビ法のブロック化手法に対して理論的な誤差解析を行うことで，計算結果の直交性に対する誤差上界を求めたことである．Hari の V2 手法はブロック化した場合の計算性能の良い計算手法であるが，Cholesky QR 法という精度の悪い手法を内部的に利用しており，収束性や誤差が不明であった．本研究の結果は，片側ヤコビ法が用いられる状況であれば，Hari の V2 手法による直交性の誤差は十分小さくなることを示しており，高性能な片側ヤコビ法実装に対して理論的保証を付け加えることになる．

第二の成果は，Bečka らの並列化手法 (動的順序) [8] を用いたときの，一次収束性に対するより新しい上限を求めたことである．Bečka らの動的順序は貪欲法の考えで複数ある部分問題の選択肢から収束を速めると推測されるものを優先的に選ぶ．そのため片側ヤコビ法の収束性を実用的には高めるが，動的な選択順序をとるため，収束性の解析が難しく，並列化した場合でも収束速度が変わらないという緩い上限しか得られていなかった．本論文で示す一次収束性の上限は並列数に対して反比例する形となっているため，並列化によって片側ヤコビ法の性能が向上することを保証する．とくに大規模な計算機においては，計算時間が金銭・環境負荷のコストに直結するため，スケーラビリティーに関する一定の保証は利用者にとって大きな意味がある．

第三の成果は，大規模並列計算機に適した実装の開発と性能評価である．これまで片側ヤコビ法や関連する手法の並列化や，並列計算機上での性能評価は複数行われてきているが，1,000 並列を超えるような大規模並列環境下での実験は数少なく，高橋 [9] や筆者 [10] による固有値分解のためのヤコビ法に対するものだけであり，SVD に対するものはなかった．本研究では Hari の V2 手法が持つ計算パターンに適した AllReduce 型の並列化手法を開発し，またこの並列化手法に適した二次元ブロック分割を用いたときの，演算量や通信量，通信回数を調べている．これらの結果から，AllReduce 型の並列化手法は演算量にオーダーの意味で小さなオーバーヘッドはあるが，その代わりに通信量・通信回数をオーダーの意味で減少させられることがわかる．この特性は並列度の高い昨今の高性能計算機に適しているものと考えられる．また，京コンピュータを用いた性能評価においては，3,072 ノード，計 24,576 コアを用い，1 万次元の行列を用いて計算するという，行列の次元とコア数が大きいという高い並列性が求められる状況においても性能がスケールするという結果となっている．また，Bečka らの動的順序を適用した場合に追加される計算の高性能実装手法や，新たな通信パターンに対するマッチングを用いた通信削減手法を示している．

第四の成果は，ブロック化した片側ヤコビ法に用いられる計算部品の性能評価と，将来の計算機に向けた実装技術の検討である．ここでは，ブロック化した片側ヤコビ法に現れる行列積の 1 種である，DSYRK（行列とその転置の積）について議論する．将来の計算機に利用されると目されているメニーコア CPU のような環境では，行列積のような単純な計算ルーチンにおいても低性能になることがあることを示し，また，高い並列性に適した DSYRK の並列化手法について検討する．

1.3 論文構成

最初に研究の目的と課題，そして成果について述べた．

2 章の背景では第一に，現在，そして将来の科学技術計算をとりまく環境について述べ，SVD 計算

において必要となる事柄をまとめる．次に現状の SVD 計算手法とその特性について示す．

3 章では，本研究で取り扱う片側ヤコビ法のアルゴリズムについて示し，誤差解析や拡張手法（前処理，ブロック化，並列化）などの既存研究を紹介する．．

4 章では，片側ヤコビ法に対するブロック化による，直交性に対する誤差上界について述べる．この章の内容は関連論文 [a] を基に証明を詳細化したものである．

5 章では分散メモリ並列環境向けの高性能実装手法と，Bečka の並列化手法の新たな一次収束の上限について述べる．5 章の内容は関連論文 [b,c] を基に発展させたものである．

6 章では，片側ヤコビ法に登場する計算ルーチンについて，将来の計算機アーキテクチャへの適用について考察する．この章の内容は関連論文 [d] を基にしている．

7 章では本論文をまとめる。

第 2 章

背景

2.1　高性能計算とそのトレンド

計算機の誕生とその演算性能の急速な拡大によって，科学技術計算によって解くことが可能な問題も増大を続けてきた．その中でも，スーパーコンピュータと呼ばれる高性能計算機は個人で容易に所有可能な計算機 (PC) と比べると数桁以上性能が上回るため，スーパーコンピュータが科学技術計算を先導してきた．例えば，現在，PC の演算性能は約百 GFlop/s，一次記憶容量は 10GByte 前後となっている一方で，スーパーコンピュータのベンチマークランキングである Top500 の 2017 年 6 月における第一位にある計算機である National Supercomputing Center in Wuxi の Sunway TaihuLight は約 125PFlop/s の演算性能を持ち，約 1.3PByte の一次記憶容量を持つ．すなわち，演算性能によって単純に見積もれば，Sunway TaihuLight によって 1 時間で計算可能な問題であっても，PC では 100 年程度の時間がかかることになり，また仮想記憶を利用したとしても 1PByte のストレージを持つ PC は一般的ではないため，そもそもプログラムを実行可能ではない可能性がある．そこでスーパーコンピュータを効率的に利用する技術が重要となっている．

現状では，スーパーコンピュータの演算性能は拡大の一途をたどっているが，それはクロック周波数の向上のような純粋な性能のスケーリングではなく，システムの様々な特性の変化を伴うものである．特性の変化の中でも最も顕著なものが並列度の増加であり，また，相対的な通信性能（レイテンシ，バンド幅）やメモリ性能の低下なども発生している．

図 2.1は Top500 に登録されたシステムの最も高速なもの，中間（250 位）のもの，最も遅いものにおける，Linpack 性能（左）とコア数（右）の推移を示している．図 2.1にあるように，Top500 に登録されたシステムの演算性能は増加の一途をたどっているが，コア数についても同じ傾向がみられる．演算性能の向上が 100 万倍程度であるのに対して，コア数の増大が 1 万倍程度となっており，性能向上の大きな部分が並列度の増大によってもたらされたことがわかる．

McCaplin による SC16 の招待講演 [11, 12] では，Top500 のトレンドについて，特に性能バランスの変化についての観測と，ハードウェア技術のトレンドについて述べられている．そこでは，5 つの性能特性についての傾向がまとめられている．5 つの特性はそれぞれ，CPU の理論ピーク性能，メモリバンド幅，メモリレイテンシ，通信バンド幅，通信レイテンシであり，CPU の理論ピーク性能は年毎に 50% から 60% の増加，メモリバンド幅は 23% 以下程度の増加，メモリレイテンシは 4% 以下程度の増加，通信バンド幅は 20% 以下程度の増加，通信レイテンシは 20% 以下程度の低下としている．よってメモリバンド幅や通信バンド幅が演算性能に対して小さな増加傾向となっているため，相対的には性

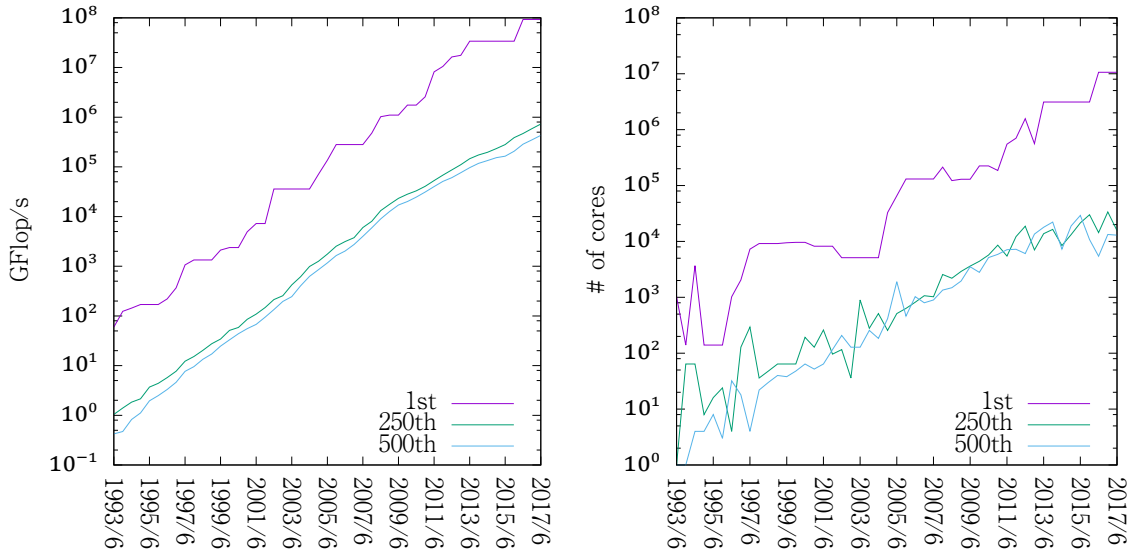

図2.1　Top500 に登録されたシステムの最も高速なもの，中間（250 位）のもの，最も遅いものにおける，Linpack 性能とコア数の推移．左図が Linpack 性能，右図がコア数．

能低下となっている．また通信レイテンシは低下（改善）の傾向となっているが，メモリレイテンシは悪化の傾向にある．McCaplin はさらに，待ち行列理論の Little の法則 [13] について言及している．Little の法則は，後入れ先出しの待ち行列（queue）について，ある仮定の下で，待ち行列の中の平均人数 L と，待ち行列内での平均待ち時間 W，待ち行列へやってくる単位時間あたりの平均人数 λ の関係を表したものであり，

$$L = \lambda W \tag{2.1}$$

が成り立つ．この理論はそのままメモリや通信における並列性に応用できる．いま W をメモリ・通信のレイテンシ，λ をメモリ・通信バンド幅とおけば，L は同時にアクセスしなければならないバイト数となる．すなわち，高いバンド幅を達成するためにはメモリアクセス・通信の並列性が必要となることを示しており，レイテンシの減少に対してバンド幅の増大が上回っている場合，並列度も増大することになる．

また McCaplin は CPU の演算性能向上について，Pentium4 から Skylake までの Intel の CPU の理論ピーク演算性能向上要因について示している．彼は，CPU の理論ピーク演算性能を次の 3 つの値の積に分解した．1 つは CPU の動作クロック周波数，2 つ目の値はコア数，3 つ目の値は 1 クロック当たりの演算量である．そこで理論ピーク性能の対数をとれば，各項目が性能に寄与する割合を求めることができる．図 2.2に彼のグラフを再現したものを示す．このように，Intel の CPU のクロック周波数はむしろ下降傾向にあり，性能向上の要因は，コア数と 1 クロック当たりの演算量の増大によるものとなっている．すなわち，CPU 内部の並列性も増加の一途をたどっていることがわかる．

一方，将来のスーパーコンピュータ開発プロジェクトとして，各国，各団体が 1EFlop/s (10^{18}Flop/s) の演算性能を目指すエクサスケールコンピューティングに取り組んでいる．我が国においては文部科学省による「フラッグシップ 2020 プロジェクト」が 2020 年頃からの運用開始を目指し，京コンピュータの 100 倍の性能（約 1EFlop/s に相当）を持つシステムの構築を行う [14]．米国においては

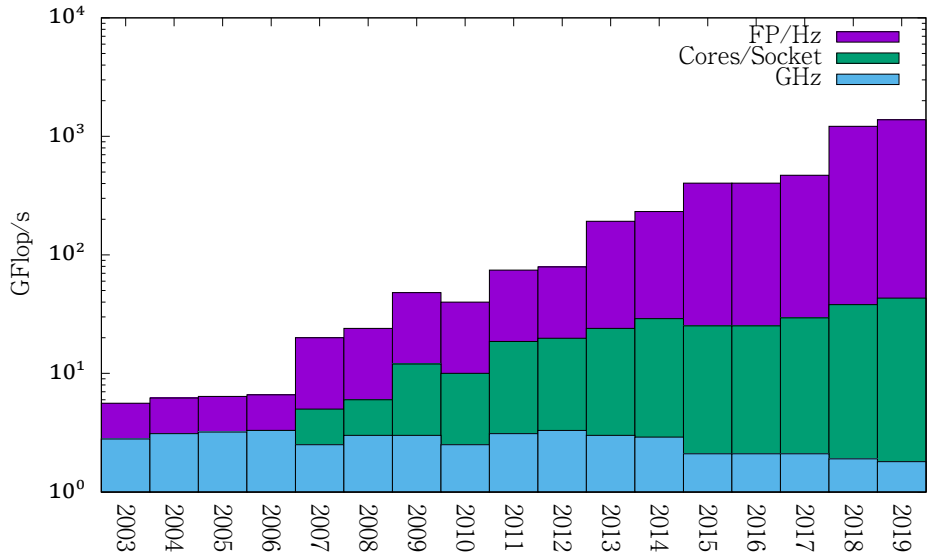

図2.2　Intel の CPU における性能向上要因を示したグラフ．McCaplin のスライド [12] を再現したものである．横軸 2003 年から 2019 年の開始時点における Intel の CPU の性能を示したもの．2018 年と 2019 年は McCaplin による予測値．

Department of Energy (DOE) による 2 つのプロジェクト (Aurora, ECP) が 2021 年を目標に 1EFlop/s の性能を持つ計算機の開発を発表している [15, 16]．また中国や欧州においてもエクサスケールコンピュータの計画がなされている [17, 18, 19]．具体的にこれらのプロジェクトでどのようなシステムが作られるか予測することは難しい．実際に，2017 年 10 月には ECP の計画が見直されるなど，いまだに流動的である．しかし，いくつかの予測から，エクサスケールコンピュータは次のようなものとなるだろう．まず，電力はコスト面の制約から 10MW 前後が限度になる．一方，現在の半導体進化のトレンドから集積度は向上するがクロックスピードは同程度のままとなるため，性能向上はそのまま並列度の上昇となる．そこでエクサスケールコンピュータでは，Sunway TaihuLight の約 10 倍となる，1 億並列となる予測がある．

このように，現状においても，また将来的にも高性能計算機は演算性能の向上を続けているが，それには計算機の特性の大きな変化を伴っており，質的な変化と呼ぶべきほどである．そこでこのような計算機を科学技術のために有効活用するためには，様々な数値計算のアルゴリズムにおいても，大きな変化が必要とされている．

2.1.1　現代の高性能計算機の構成

Top500 に登場するシステムにも複数種類のものがあるが，現代のシステムに共通する特徴としては，どれも通信網で結ばれた，分散メモリ型の並列計算機となっていることである．実際に，2017 年 6 月の Top500 において，すべてのシステムが何らかの通信網を備えており，その中でも Ethernet が最も多く 208 システム，Infiniband が 177，Omnipath が 38，その他が 77 となっている．その他には，Cray 社の Gemini, Aries interconnect [20] や IBM Blue Gene/Q の通信網 [21, 22]，富士通の Tofu netwrok [23], Tianhe [24, 25] や, Sunway TaihuLight の通信網 [26] が含まれている．これらの中でどの

ようなソフトウェア環境が使われているのか詳細には不明であるが，500 システムのうち 498 システムがことごとく Linux を OS として用いており，残り 2 つも IBM の AIX である．また，通信網も 423 システムが Ethernet や Infiniband, Omnipath といった標準的なシステムであるため，Message Passing Interface (MPI) [27] がサポートされているものと考えられる．残りの多くを占める Cray Aries や IBM Blue Gene/Q，他に富士通 FX や Tianhe, Sunway TaihuLight では MPI が使用可能であることが確認できるため，MPI が高性能計算機における標準的な通信インターフェースだと言えるだろう．

また，通信網で結ばれたそれぞれの計算ノードのほとんどは Intel のサーバー向け CPU が用いられ，451 システムにも及ぶ．次に多いのが IBM の Power であり 19 システム，次に Intel Xeon Phi の 16 システムが続く．これらの CPU について，1 ソケット当たりのコア数によって分類すると，10 コア以上のものが 408 システム，20 コア以上のものが 37 システム，30 コア以上のものが 19 システムとなっている．500 システムのうちアクセラレータを持つシステムが 90 あり，そのうち NVIDIA の GPU を持つものが 74，アクセラレータ向けの Intel Xeon Phi を持つものが 16 あるが，その中に 2 つの混合構成のものが 3 つある．このように，10 コア以上 20 コア未満のようなマルチコアとメニーコアの境界にある CPU が数多く，また，アクセラレータを持つシステムが顕著な割合で存在することがわかる．20 コア以上の CPU にアクセラレータとしてメニーコア CPU を組み合わせるものは存在しないが，代わりに NVIDIA の GPU を組み合わせるものは存在する．アクセラレータとしてメニーコア CPU を持つシステムと 20 コア以上の CPU を持つシステムとの合計は 55 となり，GPU を持つシステムの 75 と同じ程度に存在することがわかる．

そこで，本論文では，現代と近未来を代表する高性能計算機のモデルとして，次のような 2 つの種類のシステムを考えることにする．どちらのシステムも，MPI が利用可能であるような通信網で接続された多数のノードで構成される分散メモリ型のクラスターマシンであり，各ノードについて，片方は 10 コア程度から数百コアで構成される大規模マルチコア，またはメニーコア CPU を用いるもの，もう一方は，通常の CPU にメニーコア CPU や GPU のようなアクセラレータを組み合わせたものである．例えば，2017 年 6 月の Top500 における上位 10 システムでは，Sunway TaihuLight, Sequoia, Cori, Oakleaf-PACS, K computer, Mira, Trinity が前者に該当し，Tianhe-2, Piz Daint, Titan が後者に分類される．

2.1.2 通信回避型アルゴリズム

このような，急速な並列度の増大に対するアルゴリズム面での解決策の 1 つが，通信回避型アルゴリズム (Communication avoiding algorithm, CAA) と呼ばれる手法である．ここでいう「通信」とは，いわゆる通信網を通じたノード間通信だけでなく，コアとメモリ間のデータ移動や，コア間のデータ移動なども含まれている．従来，このような「通信」の問題はハードウェアに合わせたチューニングやスケジューリングの変更によって解決されてきたが，CAA では，通信量や通信回数の削減のため，数理的アルゴリズムや問題の数学的定式化を直接変更することによって，より大きな改善を目指すものである．CAA の研究では具体的なアルゴリズムの提案の他に，通信量・通信回数の理論的下界の算出のような計算理論的研究もおこなわれている．そこで，CAA の中でも理論的下界を達成したものを通信最適アルゴリズムと呼ぶ．

現在，CAA として多くのアルゴリズムが研究・開発されているが，その中でも行列分解を対象としたものが多数あり，部分ピボット付き LU 分解 [28]，Cholesky 分解 [29]，非正定値対称分解 [30]，QR

分解 [31]，部分ピボット付き QR 分解 [32, 33] などがある．この中には，通信が削減される代わりに演算量が定数倍されるものも含まれるが，高度な並列性が要求される環境では有用であると考えられている．

本研究で用いる片側ヤコビ法の拡張手法（並列化，ブロック化）は，従来使われてきた二重対角化を用いる手法と比較すると，演算量がオーダーの意味で増大する一方で通信回数がオーダーの意味で削減されるため，CAA の一種であるということができる（詳細は次章を参照のこと）．ただし，前述の行列分解のための CAA が基となるアルゴリズムの拡張手法になっているのに対して，片側ヤコビ法と二重対角化を用いる手法とでは基となるアルゴリズムが異なる点では違いがある．

2.2 SVD とその応用

特異値分解 (Singular Value Decomposition, SVD) は行列分解の 1 種である．一般に，任意の複素行列 $A \in \mathbb{C}^{m\times n}$ に対して 3 つの行列 U, Σ, V が存在し，

$$A = U\Sigma V^{*} \tag{2.2}$$

を満たす．これを複素行列に対する Full-SVD と呼ぶ．ただし $U \in \mathbb{C}^{m\times m}$ と $V \in \mathbb{C}^{n\times n}$ はユニタリ行列であり，$\Sigma \in \mathbb{C}^{m\times n}$ は $m \geq n$ の場合，ある対角行列 $\check{\Sigma} \in \mathbb{R}^{n\times n} = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ によって，$\Sigma = \begin{bmatrix} \check{\Sigma} \\ \mathbb{0}_{m-n,n} \end{bmatrix}$ と書かれる．（$m < n$ の場合，零行列 $\mathbb{0}_{n,n-m}$ が $\check{\Sigma}$ の右側に挿入される．）

本論文ではより簡単な形である，$m \geq n$ のときの実行列 $A \in \mathbb{R}^{m\times n}$ の薄型特異値分解 Thin-SVD に限定する．これは A を次の 3 つの行列に分解する：

$$A = U\Sigma V^{\mathsf{T}}. \tag{2.3}$$

ここで $U \in \mathbb{R}^{m\times n}$ は列直交行列，$V \in \mathbb{R}^{n\times n}$ は直交行列，$\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ は非負要素を持つ対角行列である．以下では，U と V の i 番目の列ベクトルをそれぞれ u_i, v_i と表記する．

式 (2.3) は実行列に対して Full-SVD(2.2) を計算し，U の一部を切り落としてコンパクトに表現したものとなっている．Thin-SVD から Full-SVD を構成することは可能であり，Thin-SVD で求めた $m \times n$ の U へ右側に $m - n$ 本の直交基底を追加し，また，Σ の下側に零行列を追加して $m \times n$ になるよう拡張すればよい．そこで本論文では Thin-SVD のみを扱う．また，$m < n$ の場合は A の転置 A^{T} の SVD を計算し，U と V を置き換えれば A の SVD が得られるため，$m \geq n$ の場合についてのみ考える．また複素行列の SVD と実行列の SVD では演算量や基本となる演算などが変わってくるが，実行列向けのアルゴリズムのアナロジーとして複素行列向けのものを作ることができるため，ここでは実行列のもののみを取り扱う．そこで本論文では以降から SVD と書いたときに，$m \geq n$ の実行列に対する Thin-SVD を意味するものとする．

SVD は固有値分解 (Eigenvalue Decomposition,EVD) と密接な関係がある．いま $C = A^{\mathsf{T}}A$ とおいたとき，直交行列 X と対角行列 Λ が存在し，$CX = X\Lambda$ を満たす．すなわち

$$X^{\mathsf{T}}CX = (AX)^{\mathsf{T}}(AX) = \Lambda \tag{2.4}$$

である．つまり，AX は列が直交した行列であり，列直交行列 U と $\Sigma = \sqrt{\Lambda}$ によって $AX = U\Sigma$ が成り立つようにとることができる．（一般には U は一意に決まらないが，上式を満たすものを 1 つ選ぶもの

とする.）すなわち，C の EVD を計算すれば A の SVD が得られる．また，$A = U\Sigma X^{\mathsf{T}}$ は A の SVD であるため，A の SVD を計算すれば，C の EVD も同時に得られる．通常の場合，SVD よりも EVD は高速に計算可能であるため，EVD によって SVD を代用可能である場合がある．しかし，現実の計算においては計算誤差の影響によって，SVD と比べて EVD によって SVD を代用した場合の誤差が増大することがあるため，このように代用可能な場合は行列 A の性質が良い場合，すなわち，最大特異値と零でない最小特異値の比が小さい場合 (良条件) に限られる．逆説的な結果として，SVD が利用されるのは悪条件な場合が主となるため，計算誤差に対して特別な配慮が必要となることが多い．

他に，$A_{\mathrm{aug}} = \begin{bmatrix} \mathbb{O} & A \\ A^{\mathsf{T}} & \mathbb{O} \end{bmatrix}$ は対称行列であり，固有値 $\pm\sigma_1, \pm\sigma_2, \dots, \pm\sigma_n$ を持ち，それぞれに対応する固有ベクトルとして

$$q_i = \frac{1}{\sqrt{2}} \begin{bmatrix} u_i \\ \pm v_i \end{bmatrix}, \quad i = 1, \dots, n \tag{2.5}$$

を持つため，A_{aug} の EVD によって A の SVD を計算することができる．この手法でも SVD を計算することができるが，通常の EVD 手法で計算する場合，A_{aug} の非零構造を破壊しながら計算が進行するため，データ量が 2 倍に増大すること，行列サイズが増大することにより演算量が増大するため，SVD の標準的な手法（後述の Golub-Kahan や Lawson-Hanson-Chan）と比べて，演算量が大きくなること，また，標準的な EVD 手法は q_i 同士の直交性を高いレベルで実現するが，それは u_i や v_i の直交性を意味しないことから，A_{aug} の EVD から SVD を計算することは行われないが，A が構造を持つ場合 (Golub-Kahan や Lawson-Hanson-Chan によって二重対角化されている場合など) においては議論されており [34, 35]，実際に LAPACK の xBDSVDX は A_{aug} を並び替えて三重対角行列としたものの EVD を計算する．

2.2.1 SVD の応用

SVD の単純な応用に，行列近似がある．実行列 A の SVD を $A = U\Sigma V^{\mathsf{T}} = \sum_{i=1}^{n} \sigma_i u_i v_i^{\mathsf{T}}$ と表したとき，総和を最初の r 個のみをで打ち切って得られる A_r を考える：

$$A_r = \sum_{i=1}^{r} \sigma_i u_i v_i^{\mathsf{T}}. \tag{2.6}$$

このとき，

$$A - A_r = \sum_{i=r+1}^{n} \sigma_i u_i v_i^{\mathsf{T}} \tag{2.7}$$

であるから，σ_i が非昇順で並べられていることより，$\|A - A_r\|_2 = \sigma_{r+1}$ が成り立つ．一方，$\|A\|_2 = \sigma_1$ であるから，A_r は A の近似となっており，誤差行列の相対誤差 $\frac{\|A-A_r\|_2}{\|A\|_2} = \frac{\sigma_{r+1}}{\sigma_1}$ となる．そこで $\sigma_1 \gg \sigma_{r+1}$ を満たすような十分小さな r があれば，A_r は A の良い低ランク近似となる．より強力に $\sigma_r \gg \sigma_{r+1}$ として，行列 A の numerical rank を正確に求めることも行われている [36]．行列の低ランク近似は，行列を格納するためのデータ量を削減できる利点があり，また，行列積や行列和などの基本的な演算の演算量も小さくできるため，巨大な密行列に対する行列演算手法の中で用いられる．また，データの中から誤差などの相対的に小さな成分を取り除き，重要な傾向を得ることを目的として，主成分分析 (Principle Component Analysis, PCA) などに用いられる．この場合，相対的に大きな特異値とそれに

対する特異ベクトルを求めることが目的であるため，EVD を用いた SVD によっても十分な精度で計算できる場合があるが，高精度な解が必要な場合は SVD を用いる．

もう 1 つの SVD の応用が最小二乗近似 [37, Chapter 6] である．$m > n$ のとき与えられた行列 A とベクトル b に対して $Ax = b$ を満たすベクトル x は一般には存在しないため，最も誤差が小さくなる

$$\arg\min_{x} \|Ax - b\|_2 \tag{2.8}$$

を求めることに意味がある．この計算手法には主に次の 4 つがある：

- 正規方程式 $A^\top Ax = A^\top b$ を解く方法
- A の QR 分解 $A = QR$ を計算し，$x = R^{-1}Q^\top b$ を計算する方法
- A の SVD $A = U\Sigma V^\top$ を計算し，$x = V\Sigma^{-1}U^\top b$ を計算する方法
- 拡大行列に対する方程式 $\begin{bmatrix} I & A \\ A^\top & \mathbb{0} \end{bmatrix}\begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ \mathbb{0} \end{bmatrix}$ を解く方法

正規方程式を解く手法は，A の条件数を極度に悪化させ精度に影響するため，A が事前に良条件であるとわかっている場合にしか用いられない．A の拡大行列を用いる手法は，拡大行列の LU 分解や，非零構造を使うならば，A の QR 分解や SVD を計算することになるため，演算量の面でのメリットはあまりないが，行列が iterative に拡大されていく場合などで用いられる．A の QR 分解や SVD を用いる手法は A が悪条件の場合に正規方程式を使う手法よりも高精度に計算できるが，A がランク落ちしている場合，QR 分解を用いるものよりも SVD を用いる手法がより信頼性の高い結果を得られる．

一方，行列 A に誤差が入っている場合，真の解や真の最小二乗解を厳密に求めることには意味がなく，$Ax = b$ に近いものの中でより性質の良い x を選びたい，という場合がある．例えば，ある定数 α について

$$\arg\min_{x} \|Ax - b\|_2^2 + \alpha \|x\|_2^2 \tag{2.9}$$

を最小化することが考えられる．これは行列に対する Tiknov's regularization と呼ばれる．この解は $x = \left(A^\top A + \alpha I\right)^{-1} A^\top b$ と書けるが，A が良条件であると事前にわからない場合は，A の SVD によって計算する．

また，与えられた行列 C に対する EVD を計算したい場合，C が正定値対称行列 (Symmetric Positive Definite Matrix, SPD) であるならば，C のコレスキー分解 $C = LL^\top$ を計算しておき，$A = L^\top$ とおいて A の SVD を計算すれば，C の EVD が得られる．この手法は通常，EVD に対して SVD の計算が遅いため，用いられることはないが，例えば，EVD と比べてコレスキー分解が正確に求められる行列であれば，行列の条件数を低減できるため，より高精度に EVD が計算可能となる可能性がある．また，本論文で扱う SVD 向けのヤコビ法は，EVD 向けのヤコビ法と同程度の速度となるため，ヤコビ法を用いるならばこの手法を用いることに問題はない．EVD には，電子状態計算のような応用があるため，そこにヤコビ SVD を応用できる．

2.3 SVD の誤差

SVD はこのように誤差が影響しやすい悪条件の行列に対して適用される．計算機上で扱う行列は，何らかの方法で得た観測値や，そのような値をもとに生成したもの，あるいは何らかの法則で生成したものであり，特殊な行列，例えばすべての要素が整数で表されるものなどを除けば，計算機に格納

する以前に誤差を持っている．また，計算機上に格納するためには二進数で表現する必要があるため，二進化するときに丸め誤差が発生する．浮動小数点数は計算機上で標準的に用いられる数値表現であり，IEEE [38] によって標準化されており，また，多くの計算機上で専用設計の回路が搭載されており，四則演算などが高速かつ正確に実行できる．しかし，浮動小数点数は一定の範囲の数値に対して一定の相対精度を持つような数値表現となっており，例えば倍精度浮動小数点数ならば十進数で 16 桁程度しか正確に保持できない．多倍長整数による分数表現や任意精度演算のようなものを用いればこの丸め誤差は任意に小さくできるが，そもそもの数値が持つ誤差よりも小さくする必要はなく，また，分数表現や任意精度演算はデータサイズや，四則演算などのコストが大きい．そこで，行列には要素単位で誤差が入っているものと考え，最良の場合でもその大きさは浮動小数点数の丸め誤差と同程度であると考えることは妥当である．

いま真に計算したい値を並べた行列を A として，それを浮動小数点数で丸めた行列を A_f と表記する．このとき丸め誤差を並べた行列を E とおいたときに，

$$A_f = A + E \tag{2.10}$$

が成り立つものとする．この誤差 E によって A の特異値や特異ベクトルにどれほどの誤差が入るだろうか．いま浮動小数点数の丸め誤差の単位 $\mathbf{u}$ を定義する．これは倍精度の場合 $\mathbf{u} \approx 1.1 \times 10^{-16}$ となる．A の各要素に対して $\mathbf{u}$ に比例する相対誤差が入っていると想定するが，任意の A に対しては要素の値の分布を想定することができないため，行列ノルムによって誤差の上限を評価する．すなわち，次の不等式を用いる：

$$|E| \leq \mathbf{u}\,|A|\,, \tag{2.11}$$

$$\|E\|_2 \leq \mathbf{u}\sqrt{\operatorname{rank} A}\,\|A\|_2\,. \tag{2.12}$$

ただし行列に対する不等式は要素ごとに定義し，式 (2.11) から式 (2.12) は Higham [39, Lemma 6.6] による．ここではより一般化して，ある定数 ϵ によって

$$\|E\|_2 \leq \epsilon\,\|A\|_2 \tag{2.13}$$

が成り立つものとして誤差を ϵ によって表す．式 (2.12) では $\epsilon \leq \mathbf{u}\sqrt{\operatorname{rank} A}$ を想定している．

E によって A の特異値・特異ベクトルはどの程度動くのか，Drmač [2, Section 2] や Demmel [40, Section 1] などにまとめられている結果を示す．Weyl の定理を用いれば，A の特異値 σ_i に対して A_f の特異値 $\hat{\sigma}_i$ について

$$|\sigma_i - \hat{\sigma}_i| \leq \epsilon\,\|A\|_2 = \epsilon\sigma_1 \tag{2.14}$$

が成り立つ．さらに，A と A_f の左特異ベクトルをそれぞれ u_i, $\hat{u}_i$, 右特異ベクトルをそれぞれ v_i, $\hat{v}_i$ とおけば，u_i と $\hat{u}_i$, v_i と $\hat{v}_i$ 同士の正弦は次のように押さえられる：

$$\sin\theta \leq \frac{\epsilon}{\operatorname{abs\_gap}(i, A, A_f)} \tag{2.15}$$

ただし，abs_gap は次のように定義される：

$$\operatorname{abs\_gap}(i, A, A_f) \equiv \min_{j \neq i} |\sigma_i - \hat{\sigma}_j|/\sigma_1. \tag{2.16}$$

ただし，$m > n$ の場合は次で定義する：

$$\operatorname{abs\_gap}(i, A, A_f) \equiv \min\left(\min_{j \neq i} |\sigma_i - \hat{\sigma}_j|/\sigma_1,\ \ \sigma_i/\sigma_1\right). \tag{2.17}$$

式 (2.14)，特異値の絶対誤差は最大特異値に比例する；つまり，相対的に小さな特異値は大きな誤差を持つことになる．言い換えれば，特異値の相対誤差は A の条件数に比例する．また，abs_gap は特異値の絶対差に比例する．そのため，相対的には密集していない特異値であった場合でも，正弦が大きくなることがある．また $m > n$ の場合には，相対的に小さな特異値に対応する特異ベクトルは真の特異ベクトルと大きく傾きが異なる．

これは数値表現に内在する誤差である．つまり行列の各要素を浮動小数点数として表した時点でこのような誤差が生じてしまうため，どのような SVD 手法を用いたとしても，計算する以前からこのような誤差が入っている．当然，丸めを行った場合にも浮動小数点数の行列として正確に表現できる行列も有限個だけ存在するが，ごく一部である．そこで，浮動小数点数であらわされた行列は少なくともこれと同程度の誤差を持つと仮定することは妥当である．また，通常の SVD 手法では，数値表現によって発生する誤差と同程度の誤差の解を計算することを目標にする．すなわち，真の行列 A に対して，ある $\epsilon = f(m,n)\mathbf{u}$ に対して $\|E\|_2 \leq \epsilon \|A\|_2$ となるような誤差 E を持つ行列 $\bar{A} = A + E$ の特異値・特異ベクトルを計算する．ここで f は m や n に対して緩やかに増加する関数とする．

しかしながら，ここまでの誤差の解析はあくまで上限である．本来，丸め誤差 E は要素ごとに入るものであるが，評価を行うために行列ノルムによって代表値としている．そこで誤差 E に何らかの構造がある場合にはより良い上限を評価できる．また，SVD 計算においてもその構造を破壊しないように進行できれば，その上限を保持できる可能性がある．

ここでは第一に正則な対角行列 $D = \mathrm{diag}(d_1, d_2, \ldots, d_m)$ によって片側スケーリングされた行列 $A = BD$ について考える．A と B の第 i 列ベクトルをそれぞれ a_i，b_i とおく．丸め誤差の入った行列 $\bar{A} = A + E$ を考えると式 (2.11) が成り立つため，E の第 i 列ベクトルを e_i に対しても

$$|e_i| \leq \mathbf{u}|d_i||b_i|. \tag{2.18}$$

すなわちノルムを評価すると

$$\|e_i\|_2 \leq \mathbf{u}|d_i| \, \|b_i\|_2 . \tag{2.19}$$

よって $F = ED^{-1}$ とおくと

$$\bar{A} = A + E = (B + F)D \tag{2.20}$$

であり，

$$\|F\|_2 \leq \mathbf{u}\sqrt{\mathrm{rank}\, B} \, \|B\|_2 \tag{2.21}$$

が成り立つ．すなわち誤差に対して片側スケーリングの構造を見出すことができる．このとき A が正則かつ $\|F\|_2 < \sigma_{\min}(B)$ ならば，次が成り立つ [1, Theorem 2.17]：

$$\frac{|\sigma_i - \hat{\sigma}_i|}{\sigma_i} \leq \frac{\epsilon}{\sigma_{\min}(B)} \leq \kappa_2(B)\epsilon. \tag{2.22}$$

つまり，特異値の相対誤差は A ではなく B の条件数に比例する．また Demmel は特異ベクトルの誤差が A に依存せず B に依存する結果を示している [1, Corollary 2.23]．この議論において D は正則であれば任意の行列を用いることができるため，例えば $\kappa_2(B)$ が最小となるような D を計算することができれば，特異値の相対誤差における良い上限が計算できる．このように誤差が構造を持つ場合には一般の誤差の場合と比べて特異値や特異ベクトルにおける誤差の上限が変わり，高精度な解を計算できる可能性を持つため，SVD の計算においては誤差の持つ構造を破壊しないことに注意しなければならない．

高精度な特異値を保持する，より広い誤差の構造を考えるうえで，Rank-Revealing Decomposition (RRD) による表現が提案されている [41]．RRD は次のような形を持つ行列分解である：

$$A = XDY^{\mathsf{T}}. \tag{2.23}$$

ただし，D は対角行列であり，X と Y は良条件な行列である．

定理 2.1 (Demmel [41], Theorem 2.1). **$A = XDY^{\mathsf{T}}$ が RRD であり，$A = U\Sigma V^{\mathsf{T}}$ と SVD できるものとする．このとき誤差を持った行列 $\hat{A} = \hat{X}\hat{D}\hat{Y}^{\mathsf{T}}$ とその SVD $\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^{\mathsf{T}}$ について，ある $0 \le \epsilon < 1$ が存在して，次が成り立つとする：**

$$\hat{X} = X + \delta X, \quad \text{where } \frac{\|\delta X\|_2}{\|X\|_2} \le \epsilon \tag{2.24}$$

$$\hat{D} = D + \delta D, \quad \text{where } D \text{ and } \delta D \text{ are diagonal and } |\delta D| \le \epsilon |D| \tag{2.25}$$

$$\hat{Y} = Y + \delta Y. \quad \text{where } \frac{\|\delta Y\|_2}{\|Y\|_2} \le \epsilon \tag{2.26}$$

また $\eta = \epsilon(2+\epsilon)\max(\kappa_2(X), \kappa_2(Y))$ と $\eta' = 2\eta + \eta^2$ を定義する．このとき A と $\hat{A}$ の特異値間の誤差を次のように抑えられる：

$$\frac{|\sigma_i - \hat{\sigma}_i|}{\sigma_i} \le \eta' \tag{2.27}$$

さらに，特異値間の正弦が次のように押さえられる：

$$\sin\theta \le \sqrt{2}\left(\frac{1+\eta'}{1-\eta'} \cdot \frac{\eta'}{\mathrm{rel\_gap}(i, A) - \eta'} + \eta\right). \tag{2.28}$$

ただし，rel_gap は次のように定義され：

$$\mathrm{rel\_gap}(i, A) \equiv \min\left(\min_{j \ne i} \frac{|\sigma_i - \hat{\sigma}_i|}{\sigma_i}, 2\right) \tag{2.29}$$

すくなくとも η' より大きいものとする．

すなわち，X や Y が十分良条件であれば，X や Y に対するノルムの意味で相対的に微小な誤差や，D の対角要素に対する相対的に微小な誤差によっては，特異値や特異ベクトルは大きく変化しないことがわかる．

ここで次のような積型の誤差を持つ行列を考える．つまりある微小な誤差行列 E と F によって $\hat{A} = (I+E)A(I+F)$ と書かれる場合を考える．このとき，A の RRD を考えると

$$\hat{A} = (I+E)A(I+F) \tag{2.30}$$

$$= (X + EX)D(Y^{\mathsf{T}} + Y^{\mathsf{T}}F) \tag{2.31}$$

であるから，$\|E\|_2$ や $\|F\|_2$ が十分小さければ，特異値・特異ベクトルに入る誤差も小さいことが言える．そのため，SVD の計算においてはこの積型の誤差を小さく保つことが重要である．

例えば，二重対角行列における非零要素に対する要素ごとの相対誤差はこのような積型の誤差として書ける．そこで，二重対角行列は丸め誤差が入ったとしても元の行列の特異値・特異ベクトルを高精度に計算でき，また，そのための計算手法がいくつか知られている（例えば，Fernando と Parlett

による Differential QD 法 [42] など）．ただし，一般の行列を二重対角行列に変換する二重対角化は積型の誤差とはならないことが知られており，二重対角化をした時点で，特異値の相対精度は失われてしまう．

Demmel は片側ヤコビ法を基に，積型の誤差を小さく保つ SVD 計算手法を開発した [41, Algorithms 3.1, 3.2]．この手法では，事前に具体的に A の RRD が計算できていることが必要である．一般の行列から積型の誤差を小さく保つように RRD を計算することは困難であるが，いくつかの種類の行列に対してその手法が提案されており，Cauchy 行列や Vandermonde 行列，Totally Positive 行列，Totally Unimodular 行列などの構造を持った行列に対する方法が示されている．

また Higham による Rank-Revealing QR 分解を用いた手法は，対角行列 D_1, D_2 によって両側スケーリングされた行列 $A = D_1 B D_2$ に対して，実用上小さな誤差で RRD を計算することができる．LAPACK による片側ヤコビ法の実装では Higham の RRD を用いており，実際に前章の図1.1における結果では両側スケーリングに依存しない高い精度で特異値を計算できていることがわかる．

2.4 SVD 計算アルゴリズム

2.4.1 二重対角化を用いた SVD アルゴリズム

行列 A の特異値を計算することは対称非負定値行列 $A^\top A$ の固有値を計算することと数学的には同等であるが，行列の固有値は変数によってシフトした行列の行列式の根を求めることと同等である．すなわち，多項式の根を計算する必要があるが，一般に 5 次以上の多項式の根を代数的に有限回の演算で求めることは不可能である．そこで，SVD の計算は何らかの反復解法によって十分高精度な解が得られるまで計算を繰り返す必要がある．そのときに密行列のまま計算を進行すると，データ量が多いため結果として演算量も増大してしまうため，事前に代数的な操作で零の多い構造を持った行列に変換しておくことで演算量を削減できる．また，零構造をうまく定めることによって，その構造に適した特異値計算アルゴリズムを構築できる．この戦略に基づく手法が二重対角化を用いた SVD アルゴリズムである．

このアルゴリズムは 2 つのステップ，特異ベクトルを求める場合には 1 つ追加され 3 つのステップで構成されている．第一に，行列を直交変換によって二重対角行列へと変換するステップ，第二に，二重対角行列の特異値・特異ベクトルを計算するステップ，第三に，第一のステップの逆変換を行うことで，二重対角行列の特異ベクトルを元の行列の特異ベクトルに戻すステップである．第一のステップで直交変換を用いることは重要である．直交変換は行列の誤差をノルムの意味で増大させないため，精度の面で優れており，また第三のステップで用いる逆変換を容易に計算可能である．

このアルゴリズムは標準的に用いられている手法であり，LAPACK や，分散メモリ向けの行列計算ライブラリである ScaLAPACK [43] において実装されている．

このような直交変換による中間行列を用いる手法は当初 EVD 向けに開発された．Golub [44] では，Givens や Householder の開発した（彼らの名前が冠された）Givens 変換や Householder 変換による三重対角化・Hessenberg 化手法が挙げられている．三重対角行列の固有値計算手法としては Strum 列を用いた二分法が用いられた．

その後，1961 年に Francis [45, 46] と Kublanovskaya [47] によって独立に発表された QR 法は，20 世紀を代表する 10 のアルゴリズムの 1 つとして Computing in Science & Engineering 誌において表彰

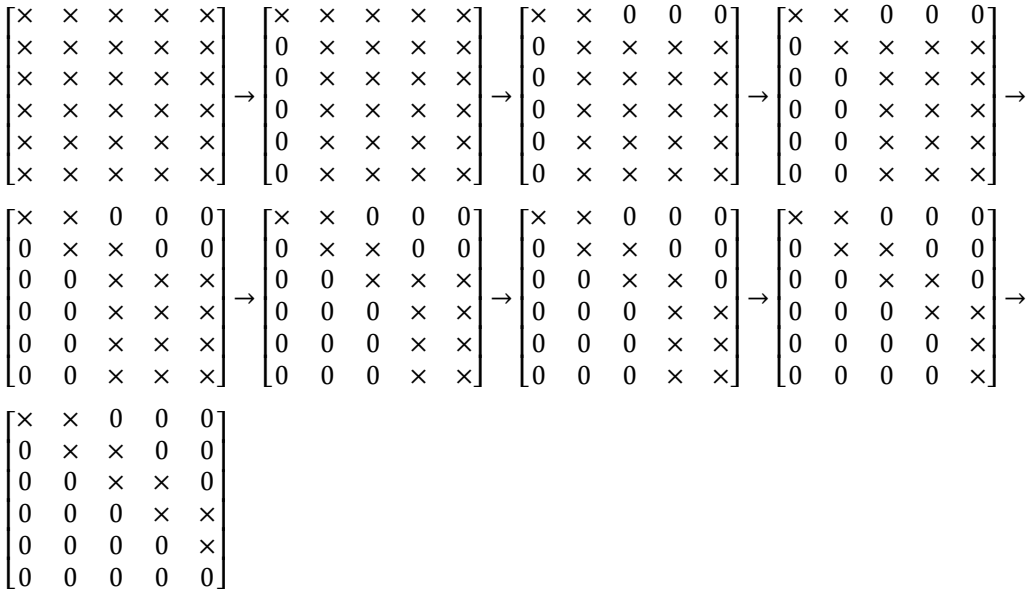

図2.3　GK 法の進行手順．最も左の列，最も上の行から始めて列と行へ交互に 0 を挿入していく．

されたものであり，Simplex 法や Quick Sort，FFT などと並べられている [48]．Francis は非対称行列に絞って議論しており，実際に現在でも非対称向け EVD 手法として第一選択肢であるが，対称行列に対しても効率的に実行できる手法である．QR 法の歴史については，Golub と Uhlig による記事 [49] が興味深い．

SVD に対する二重対角化を用いた手法はこれらの EVD に対する手法を拡張したものだと言える．

二重対角化の方法

二重対角化の手法として標準的なものは，Golub と Kahan の手法（GK 法） [50, 51] やそれを縦長行列向けに改良した Lawson, Hanson と Chan [52] の手法（LHC 法）である．

GK 法のアルゴリズムについては Golub と Reinsch による解説 [51] がわかりやすい．この手法は図2.3のように，行列 A へ Householder 変換を両側から交互に作用させていくことで，行列を二重対角の形にする．彼らの表記に倣い，左側から作用させる Householder 変換を $P^{(k)}$, 右側から作用させるものを $Q^{(k)}$ と書き，最終的に得られる上二重対角行列を J とおく．$P^{(k)}$, $Q^{(k)}$ はそれぞれを構成する単位長さのベクトル $x^{(k)}$, $y^{(k)}$ によって

$$P^{(k)} = I - 2x^{(k)}x^{(k)\top} \quad (k = 1, 2, \dots, n), \tag{2.32}$$

$$Q^{(k)} = I - 2y^{(k)}y^{(k)\top} \quad (k = 1, 2, \dots, n-2) \tag{2.33}$$

と書かれる．このとき最終的に

$$P^{(n)}P^{(n-1)} \cdots P^{(1)}AQ^{(1)}Q^{(2)} \cdots Q^{(n-2)} = J \tag{2.34}$$

が成り立つようにしたい．

補題 2.2. 与えられた相異なるベクトル x と y が同じ大きさを持っている場合，すなわち $x \neq y$ かつ $\|x\|_2 = \|y\|_2$ である場合，

$$z = \frac{x - y}{\|x - y\|_2} \tag{2.35}$$

によって構成される Householder 変換 $H = I - 2zz^\top$ は次を満たす：

$$y = Hx, \tag{2.36}$$
$$x = Hy. \tag{2.37}$$

証明. 例えば標準的な教科書 [53, Section 3.2] などにあるように

$$Hx = x - \frac{2(x - y)\,(x - y)^\top x}{\|x - y\|_2^2} \tag{2.38}$$
$$= x - \frac{2(x - y)(x^\top x - y^\top x)}{x^\top x - y^\top x - x^\top y + y^\top y} \tag{2.39}$$
$$= x - \frac{2(x^\top x - y^\top x)}{2(x^\top x - y^\top x)}(x - y) = y. \tag{2.40}$$

また，$H^\top = I - 2zz^\top = H$ であり，$z^\top z = 1$ を用いると

$$H^\top H = I - 4zz^\top + 4zz^\top zz^\top = I \tag{2.41}$$

であるため H は直交行列．よって $H^{-1} = H^\top = H$ であるため，$x = Hy$ が成り立つ． □

補題 2.3. ベクトル x, y, z の第 i 成分を x_i, y_i, z_i と書く．z が単位長さを持ち，第 j 成分が 0 であるとき，$y = Hx$ の第 j 成分を考えると

$$y_j = x_j - 2z_j z^\top x = x_j. \tag{2.42}$$

よって，z の第 j 成分が 0 であれば，H は第 j 成分を不変に保つ．

補題 2.4. $z^\top x = 0$ の場合，

$$Hx = x - 2zz^\top x = x. \tag{2.43}$$

以上より，Householder 変換を用いることで，一部の値を不変に保ちながら，あるベクトルを同じ大きさを持つ別のベクトルに変換することができることがわかる．そこで $P^{(1)}$ は A の第 1 列を，第 1 要素を除いてすべて 0 とするような変換にすることで図2.3での第一ステップのような形にすることができる．また $Q^{(1)}$ は $y^{(1)}$ の第一要素を 0 とすることで $P^{(1)}A$ の第 1 列の構造を破壊せずに，$P^{(1)}A$ の第 1 行の第 3 要素以降を 0 にすることができる．このように，すでに作った構造を破壊しないように $x^{(k)}$ や $y^{(k)}$ の先頭要素を 0 に設定しながら，変換を行っていくことで，行列を二重対角の形にすることができる．

GK 法は，前のステップで変換した行列の値を用いて今のステップの変換を計算するため，逐次的な計算となる．また行列の Householder 変換は行列ベクトル積や行列の rank-1 更新が主な計算であり，データ再利用性が悪い．Dongarra ら [54] は GK 法の手順を並び替えて，Householder 変換のうち rank-1 更新を複数回分溜めておき，まとめて行うブロック化手法を開発した．これによって，演算のうち半分が Level-3 BLAS に相当する演算となるため，この手法は Level-2.5 手法などとも呼ばれ，元の手順と比べて性能を改善できるが，行列ベクトル積の部分については改善できない．そこで，計算の全体をブロック化可能である QR 分解などの他の行列分解アルゴリズムと比べて低性能であるという問題があった．

Lawson と Hanson は行列 A が縦長行列 ($m \gg n$) である場合に，A の QR 分解 $A = QR$ を先に求めておき，R の二重対角化を GK 法で行う手順を考案した．Chan はこの手法について詳しく解析している．この手法は QR 分解のコストが追加される一方で，二重対角化における演算量が減少する．Golub ら [55, Section 5.4.4] によれば，GK 法の演算量は $4mn^2 - 4n^3/3$ である一方，QR 分解の演算量は $2mn^2 + 2n^3$ である．すなわち，おおよそ $m \geq 3n$ が成り立つ場合，LHC 法が演算量の面で有利となる．

GK 法では行列の両側から逐次的に変換を行っていたがこれらの手法の他に，片側変換を用いる手法がある．この手法のオリジナルの考え方は Ralha [56] によるものがあり，その安定性を改良した Barlow [57] の手法がある．これらの手法は，前節で示したような，片側スケーリングされた行列に対して，高精度な特異値を計算できるが，改良手法においても悪条件の行列においては左特異ベクトルの直交性が悪くなることが知られており，実用の域に達していない．また，Barlow は演算量面での GK 法に対する優位性を示しているが，計算パターンは GK 法と同じく，逐次的な Householder 変換の適用となっているため，演算効率の面では GK 法と同程度になると予測される．

二重対角行列の特異値分解

GK 法などで求めた二重対角行列 J の特異値・特異ベクトルを計算する手法はいくつか存在する。Golub, Kahan らによる二重対角化手法の最初の論文 [50] では，Francis の QR 法を SVD 向けに拡張した手法も示されている．この手法は単に QR 法と呼ばれる．QR 法は Demmel らによってその高精度性が証明され [58], Fernando [42] に性能面，精度面で完成されたが，二重対角化の時点で特異値・特異ベクトルに大きな誤差が入ることに注意が必要である．この手法は ScaLAPACK に実装された唯一の手法である．

SVD を計算するための分割統治法 [59] は，並列計算可能な部分が多く，また計算の大部分を行列積として書けるため，Level-3 BLAS を活用でき，高性能であるが，QR 法と比べると手順が煩雑であり，いまだに ScaLAPACK において実装は存在せず，また，分散並列向け public な実装も存在しない．

二分法・逆反復法や MRRR 法などの EVD 向け手法に対しても，SVD への拡張がある．二分法については LAPACK の xBDSVDX において，J に対する拡張行列 A_{aug} を用いて，EVD 向けルーチンを使って計算する．しかし，同等の手法は ScaLAPACK には移植されていない．MRRR 法の SVD への適用については Williams らによる議論 [34] などがあるが，EVD 向けの MRRR 法の時点で複雑なアルゴリズムであり，SVD 向けにさらに複雑となることが予想され，LAPACK 向け実装もまだであり，分散並列向け実装も public には存在を確認できない．

2.4.2 ヤコビ法

ヤコビ法はもともと Jacobi [60] によって同名の定常反復法の前処理手法として開発されたものであり，実用的な計算機の存在しなかった当時は多くの興味をひかなかったが，1949 年に Goldstine ら [61] によって対称行列向けの EVD 手法として再発見されて以降，多くの理論的解析や拡張手法の提案がなされている [44, Section 5].

ヤコビ法の基礎となるアイデアはごく単純である．簡単に計算できる直交変換によって行列を少しずつ対角行列に近づけていくことである．このように基となるアイデアが単純であるので，多くの派生手法が開発されたが，現状において最も成功している子孫が，片側ヤコビ法である．

そこでここでは，ヤコビ法と，その SVD 向け派生手法について解説する．

ヤコビ法

ヤコビ法は対称行列 C に対して，繰り返し単純な直交変換 $G^{(k)}$ を作用させていくことで，徐々に対角行列に近づけていく．いま初期行列 $C^{(1)} = C$ とおくと，

$$C^{(k+1)} = {G^{(k)}}^{\mathsf{T}} C^{(k)} G^{(k)} \tag{2.44}$$

のように進行していく．このとき $C^{(k)}$ が対角行列 Λ に収束すれば，

$$\Lambda = {G^{(\infty)}}^{\mathsf{T}} \cdots {G^{(2)}}^{\mathsf{T}} {G^{(1)}}^{\mathsf{T}} C G^{(1)} G^{(2)} \cdots G^{(\infty)} \tag{2.45}$$

となり，直交行列 $G^{(k)}$ の積が固有ベクトル，Λ の対角要素が固有値となる．実際の計算においては $C^{(k)}$ が十分対角行列に近づいたあるステップ M において計算を打ち切り，その対角成分を計算で求めた固有値，それを並べた行列を $\hat{\Lambda}$ とおき，M ステップまでの直交行列 $G^{(k)}$ の積を計算で求めた固有ベクトルとする．

直交行列 $G^{(k)}$ は Givens 回転とする．すなわちある $p \neq q$ と θ について

$$G^{(k)} = \begin{array}{c} \\ p \\ q \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} p & & q & \end{array} \\ \begin{bmatrix} I & & & \\ & c & & s \\ & & I & \\ & -s & & c \\ & & & & I \end{bmatrix} \end{array} \tag{2.46}$$

という形を持つ．ただし $c = \cos\theta$, $s = \sin\theta$ とする．この $G^{(k)}$ を両側から作用させることによって，$C^{(k)}$ の 2 つの列，2 つの行の値が変化するが，それらは次のようにまとめられる．$C^{(k)}$ の第 i,j 成分を $c_{i,j}$，$C^{(k+1)}$ の第 i,j 成分を $b_{i,j}$ と書くと

$$\begin{cases} b_{i,j} = c_{i,j}, & i,j \neq p,q \\ b_{p,k} = b_{k,p} = c_{p,k}\cos\theta - c_{q,k}\sin\theta, & k \neq p,q \\ b_{q,k} = b_{k,q} = c_{p,k}\sin\theta + c_{q,k}\cos\theta, & k \neq p,q \\ b_{p,p} = \frac{c_{p,p}+c_{q,q}}{2} + \frac{c_{p,p}-c_{q,q}}{2}\cos 2\theta - c_{p,q}\sin 2\theta & \\ b_{q,q} = \frac{c_{p,p}+c_{q,q}}{2} - \frac{c_{p,p}-c_{q,q}}{2}\cos 2\theta + c_{p,q}\sin 2\theta & \\ b_{p,q} = b_{q,p} = \frac{c_{p,p}-c_{q,q}}{2}\sin 2\theta + c_{p,q}\cos 2\theta & \end{cases} \tag{2.47}$$

が得られる [53, eq. 3.1.5].

ここで θ の設定が問題であるが，ヤコビ法では変換後の非対角要素 $b_{p,q} = b_{q,p} = 0$ となるように

$$\tan 2\theta = \frac{-2c_{p,q}}{c_{p,p} - c_{q,q}} \tag{2.48}$$

から設定する．これを満たす θ は無数に存在するが，$G^{(k)}$ を単位行列に近づけるため $|\theta| \leq \frac{\pi}{2}$ を満たすものを選ぶ．

最終的に $C^{(k)}$ が収束するためにはどの非対角要素を消去するのか，(p, q) の設定が重要である．Jacobi によるもともとの戦略では，絶対値の最も大きい非対角要素を消去するように選択する．そのためこの手法を古典的ヤコビ法と呼ぶ．これは，行列を対角行列に近づけたいという要求からすれば自然な選択だが，絶対値最大の要素を探索するために余計な計算量がかかってしまう．そこで，あらかじめ定めた順序に従って要素を選択する手法があり，巡回ヤコビ法と呼ばれている．このような消去順序については 3 章で議論する．

ヤコビ法を理解するうえで重要なことは，非対角要素を消去することが結果として，2×2 の部分行列に対する固有値問題を解いていることと同等だということである．すなわち，$C^{(k)}$ の (p, p), (p, q), (q, p), (q, q) と，$C^{(k+1)}$, $G^{(k)}$ の同じ位置にある要素を取り出すと

$$\begin{bmatrix} b_{p,p} & b_{p,q} \\ b_{q,p} & b_{q,q} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} c_{p,p} & c_{p,q} \\ c_{q,p} & c_{q,q} \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{2.49}$$

が成り立つが，$b_{p,q} = b_{q,p} = 0$ となるように θ を設定するため，これは 2×2 の対称行列に対する固有値分解となっている．すなわち，ヤコビ法は，大きな行列の固有値分解を解くために，小さな部分行列の固有値分解を繰り返す手法だと言える．

Kogbetliantz の手法

このヤコビ法を SVD に拡張した手法の 1 つが Kogbetliantz の手法である．Matejaš らの論文 [62] における引用によると，これは Kogbetliantz が 1954 年に開発した手法であり，次に述べる片側ヤコビ法よりも数年早い．この手法は，行列 A の SVD を計算するときに，ヤコビ法と同じく，2×2 の部分行列の特異値分解を繰り返す．

Kogbetliantz の手法は正方行列 A に対して両側から Ginves 変換をかけることによって進行する．すなわち，まず $A^{(0)} = A$ とおき，

$$A^{(k+1)} = {W^{(k)}}^{\mathsf{T}} A^{(k)} G^{(k)} \tag{2.50}$$

を繰り返す．このとき，$A^{(k)}$ が対角行列 Σ に収束すれば，$W^{(k)}$ の積は左特異ベクトルの行列 U となり，$G^{(k)}$ の積は右特異ベクトルの行列 V となる．

$W^{(k)}$ や $G^{(k)}$ は，同じ要素 (p,q) に関する，角度の異なる Givens 回転とする．すなわち

$$W^{(k)} = \begin{array}{c} \\ p \\ q \end{array} \begin{array}{c} \begin{array}{ccccc} p & & & & q \end{array} \\ \begin{bmatrix} I & & & & \\ & \cos\phi & & \sin\phi & \\ & & I & & \\ & -\sin\phi & & \cos\phi & \\ & & & & I \end{bmatrix} \end{array}, \tag{2.51}$$

$$G^{(k)} = \begin{array}{c} \\ p \\ q \end{array} \begin{array}{c} \begin{array}{ccccc} p & & & & q \end{array} \\ \begin{bmatrix} I & & & & \\ & \cos\theta & & \sin\theta & \\ & & I & & \\ & -\sin\theta & & \cos\theta & \\ & & & & I \end{bmatrix} \end{array}. \tag{2.52}$$

このとき，θ, ϕ は計算結果の行列 $A^{(k+1)}$ の (p,q) 要素と (q,p) 要素が 0 となるように定める．これは結局ヤコビ法のときのアナロジーとなっており，2×2 の部分行列の EVD を計算する代わりに，$W^{(k)}$ や $G^{(k)}$ の 4 つの要素が $A^{(k)}$ の 2×2 部分行列の左特異ベクトル，右特異ベクトルとなっており，結果として，2×2 部分行列の SVD を計算していることになる．

Kogbetliantz 法は A が正方行列であることが必要だが，LHC 法のように行列を事前に QR 分解をしておけば，自然にこの条件が満たされる．また，Kogbetliantz 法を三角行列に対して行った場合，消去順序を工夫すると，三角構造を保存したまま進行することができる．より正確に述べれば，上三角，下三角の構造を交互に繰り返しながら進行する．この場合，演算量を削減できる利点があり，また，収束性の証明 [63] や誤差解析 [62] では三角構造を用いることを前提としているものがある．

Matejaš [62] による実験においては，注意深く実装した Kogbetliantz 法は片側ヤコビ法よりも多くの行列で高精度に SVD を計算できる．また，ブロック化をする場合には Kogbetliantz 法の方が片側ヤコビ法よりもよく用いられてきた．この理由は，ブロック化した片側ヤコビ法の誤差が不明であったためであり，実際に，注意した手順を用いなければブロック化した片側ヤコビ法は収束しなくなる．この点については 4 章で詳細を示す．本研究において Kogbetliantz 法ではなく片側ヤコビ法を選んだ理由は，第一に誤差や収束性の解析が片側ヤコビ法の方が容易であること，第二に両側からの変換となる Kogbetliantz 法は片側からの変換となる片側ヤコビ法と比べて，メモリアクセスパターンや通信パターンが悪くなることである．ただし，5 章後半にある Bečka の動的順序の実装手法や大域収束性の理論的解析は，片側ヤコビ法だけでなく Kogbetliantz 法にも適用できる．

Hestenes の手法 (片側ヤコビ法)

片側ヤコビ法は Hestenes [64] による別のヤコビ法の SVD 向け拡張である．名前の由来は，この手法が前記の 2 手法と異なり，行列の一方向から変換を行うという特徴からである．Chartres [65] は後年，独立に同様の手法を開発している．ただし，Chartres は EVD 向けにも同様に片側から変換を行う手法も示している．Kaiser による JK Method [66] も同様に後年になってから独立に開発された片側ヤコビ法の一種であるが，Kaiser はそれが SVD の計算をしていることには気づかず，また，対称行列の SVD が EVD に一致するものと誤解していた[*1]．Nash [67] がそのことを指摘しているが，さらに JK Method を正しく動作させるための修正法を提案している．

[*1] 対称かつ非負正定値行列であればこれは正しいが，一般の対称行列に対してはこれは成り立たない．

片側ヤコビ法の基礎となるのは，行列 A の SVD を計算することが $C = A^{\mathsf{T}}A$ の EVD を計算することと同じであるという事実である．片側ヤコビ法は本質的に C に対するヤコビ法と同等である．ただし，精度の悪化を避けるため具体的に $C = A^{\mathsf{T}}A$ を計算することは避け，必要に応じて C の一部の要素を上式によって計算することで計算を進行させる．

いま $A^{(1)} = A$ とおいたとき，片側ヤコビ法は次のように進行する：

$$A^{(k+1)} = A^{(k)}G^{(k)}. \tag{2.53}$$

ただし $G^{(k)}$ はヤコビ法で用いられたものと同じ直交変換であり，ヤコビ法の反復の行列 $C^{(k)}$ との間に常に次の関係が成り立つ：

$$C^{(k)} = {A^{(k)}}^{\mathsf{T}}A^{(k)}. \tag{2.54}$$

この意味で，片側ヤコビ法はヤコビ法と同等である．

このプロセスを繰り返した結果，$C^{(k)}$ が対角行列 Λ に収束したとすると，そのとき $A^{(k)}$ は列直交行列 U に対角行列 $\Sigma = \sqrt{\Lambda}$ を右側から掛けた行列に収束する．また Givens 変換 $G^{(k)}$ の積が右特異ベクトル V となる．

ここで問題となるのは，$G^{(k)}$ の計算方法である．$G^{(k)}$ の要素を計算するためには $C^{(k)}$ の 4 つの要素が必要となる．これらは次のように計算される．いま $A^{(k)}$ の第 i 番目の列ベクトルを a_i と書く．このとき $C^{(k)}$ の第 i,j 要素 c_{ij} は

$$c_{i,j} = a_i^{\mathsf{T}}a_j \tag{2.55}$$

によって計算される．よって $G^{(k)}$ の計算に必要な 4 つの要素は $A^{(k)}$ の 2 つの列ベクトルから再構築できる．

片側ヤコビ法が優れている点は，本質的にヤコビ法と同等であることである．このため，ヤコビ法の収束性などの理論的振る舞いに関する研究の成果をそのまま片側ヤコビ法に用いることが可能である．一方で，誤差に関しては，計算手順が異なるためヤコビ法と異なってしまうが，片側変換はむしろ誤差に対してよい方向に作用する．片側ヤコビ法では Givens 回転によって同じ行にあるデータが“混ざって”しまうが，異なる行のデータが混ざることはない．これによって，行ごとに独立した誤差の上限を出すことが可能となり，QR 分解における列ごとの後退安定性と同じように，行ごとの後退安定性を導くことができる．この結果，行を片側スケーリングされた行列に対して良い誤差上界を得られる (転置した行列に対して式 (2.22) と同じ結果が得られる)．また，片側変換は計算パターンの面でも優れており，行列の更新や $C^{(k)}$ のデータの再構築の演算などの主要な演算が列ベクトル単位の演算となるため，列ベクトル方向にデータが連続となる Fortran 配列では，データアクセスが連続となる．また，通信なども列ベクトル単位で行えるようになる．

片側ヤコビ法における問題点は，具体的に $C^{(k)}$ の要素を保持しないことである．そのため，収束判定がより難しくなる．また，ヤコビ法で行われたような，$C^{(k)}$ の絶対値最大の非対角要素を選択する消去順序を用いることも困難になる．収束判定方法については，消去のための計算に組み込んでしまう手法があり，3 章で詳しく述べる．消去順序については，巡回ヤコビ法のように，あらかじめ決められた順序を用いることが行われるが，非対角要素を計算量の小さい近似手法によって計算する手法もあり，5 章において詳しく紹介する．

第３章

片側ヤコビ法とその拡張

この章ではまず片側ヤコビ法のアルゴリズムの説明を完結させるため，アルゴリズム全体と，消去順序について既存の研究について述べる．また，前処理手法やブロック化手法，並列化手法について既存の結果について述べる．

3.1　片側ヤコビ法

3.1.1　計算手順

ヤコビ法自体の計算手順については，Rutishauser によるもの [68] が，早い時代にもかかわらずほぼ完成された形となっている．ただし，反復の終了条件などについては後の時代に出てきた知識を基に改善する必要がある．片側ヤコビ法については，Hestenes の論文 [64] における説明は 2 章にあるようなごく簡単なものとなっており，ヤコビ法との関係について記されているのみである．アルゴリズムの詳細については，ヤコビ法や片側ヤコビ法の誤差について述べている Demmel の論文 [1, Algorithm 4.1] が簡潔でわかりやすく，また，ヤコビ法の高精度性の知識を利用した反復の終了条件を使っており，参考になる．Drmač らによる LAPACK 実装について解説された論文の第 2 編 [3] では，行列の非零構造や値の構造，数値計算におけるオーバーフローなどを考慮するアルゴリズムなど，詳細について議論されているが，あまりに煩雑となるため，ここでは Demmel の論文のものをベースに擬似プログラムを記述する．

図3.1は片側ヤコビ法の擬似プログラムである．このプログラムは入力として $m \times n$ 行列 A と収束判定の閾値 `tol` をとり，結果として U, Σ, V, そして，A に V をかけた AV を返す．プログラムの主要部分は二重ループとなっており，内側ループはある方法で作られた添え字のペアを 1 巡回するループであり，外側ループはそれを収束するまで繰り返すものとなっている．添え字のペアの列 `pairs` はサブルーチン `jacobi_pairs` によって作られる．具体的なその中身については後述するが，このサブルーチンは 1 から n までの整数のペアをすべて少なくとも 1 つずつ含むものを生成するため，`pairs` は少なくとも $\frac{n(n-1)}{2}$ 個のペアを含む．

内側ループの中身は 5 つのパートに分けられる．1 つは $A^{\mathsf{T}}A$ の部分行列を再構築するパートであり，3 つの内積で構成されている．この計算によって部分行列 $C_{i,j} = \begin{bmatrix} x & z \\ z & y \end{bmatrix}$ が得られる．次のパートは a_i, a_j の直交性を調べるものであり，$\mathtt{t} = \frac{\left|a_i^{\mathsf{T}} a_j\right|}{\|a_i\|_2 \|a_j\|_2}$ を計算している．この値が十分小さければその列

```
subroutine OSJ(A = [a_1  a_2  ⋯  a_n], tol):
        V  =  [v_1  v_2  ⋯  v_n]  =  I_{n,n}
        do
                maxt = 0
                pairs = jacobi_pairs()
                for (i, j) in pairs
                        // compute (i, j) submatrix of A^⊤ A
                        x  =  a_i ^⊤ a_i
                        y  =  a_j ^⊤ a_j
                        z  =  a_i ^⊤ a_j
                        // check threshould
                        t = |z| / √(xy)
                        maxt = max(maxt, t)
                        if t > tol:
                                // compute rotation
                                ζ  =  (y − x)/(2z)
                                τ  =  sign(ζ)/(|ζ| + √(1 + ζ^2))
                                c  =  1/√(1 + τ^2)
                                s  =  cτ
                                // update A
                                q  =  a_i
                                a_i  =  cq  −  sa_j
                                a_j  =  sq  +  ca_j
                                // update V
                                q  =  v_i
                                v_i  =  cq  −  sv_j
                                v_j  =  sq  +  cv_j
                        end
                end
        while maxt > tol
        for i = 1, n
                σ_i  =  ‖a_i‖_2
        end
        Σ  =  diag(σ_1, σ_2, …, σ_n)
        U  =  AΣ^{−1}
        return (U, Σ, V, A)
```

図3.1　片側ヤコビ法の擬似プログラム

のペアをさらに直交化する意味がないため，次の `if` 文で以降の処理がスキップされる．`if` 文の中では，最初に Givens 回転の要素を計算する．この計算は処理が重くまた精度も悪い三角関数の計算を省くため，三角関数の関係式から得られたものであり，Rutishauser [68, p.6 c] においてもほぼ同じものが用いられている．4 つ目と 5 つ目のパートは行列の更新であり，A と V に対してそれぞれ行う．

内側ループで登場する t がすべて $t \leq$ `tol` を満たすときに限り，`maxc` $\leq$ `tol` が満たされる．すなわち，内側ループで一度も行列の更新を行わなかったときに限り，外側ループを脱出できるため，結局，ある巡回のはじめの行列に対してすべての非対角要素を再構築し，それが直交性の条件を満たしているかどうかを確かめることと同じことになる．このように，一部の処理のスキップのための判定と収束判定を同時に計算することができ，収束判定のための処理を別に用意する必要がない点で無駄が少ない．ただし，最後の 1 巡回は行列積 $A^{\mathsf{T}}A$ を非常に非効率的な方法で計算することと等しいため，最後の 1 巡回分においては無駄が残っている．

また `tol` はユーザーが与えるものとしているが，計算誤差のため，あまりに小さな値にしてしまうと巡回が終了しなくなる．z が内積で計算されるため，丸め誤差の単位を $\mathbf{u}$ としたときの，内積の誤差の上限である $\gamma_m = \frac{m\mathbf{u}}{1-m\mathbf{u}}$ 程度の大きさにすることが考えられる．しかし実際には内積の平均的な誤差である $\sqrt{m}\mathbf{u}$ に設定しても収束することが多いため，可能な限り A を直交化するため $\texttt{tol} = \sqrt{m}\mathbf{u}$ と設定する．この設定は Drmač らによる LAPACK における片側ヤコビ法の実装である xGESVJ でも用いられている．

外側ループから脱出できた後は，Σ と U を計算する．収束状態の A を直接利用する場合にはこの処

理を省くことができる．また，結果として V の値も返すが，V が必要ない場合には内側ループの内，V の更新の計算を省くことができるため，より効率的となる．

3.1.2 演算量削減手法

この擬似プログラムは，次の点を改良することができる．第一に，部分行列の復元のところでは 3 つの内積を行っているが，このうち $x = a_i^\top a_i$ と $y = a_j^\top a_j$ の値を記録しておくことで，この 2 つの内積は削減できる．x や y の値は，内側ループの 4 つ目のパートである，A の更新によって変化してしまうが，Givens 回転が $C_{i,j}$ を対角化するものであるため，A の更新後の x や y の値は，$C_{i,j}$ の固有値と等しい．そこで Givens 回転の計算と同時に $C_{i,j}$ の固有値も計算し記録しておけば，次のタイミングで a_i や a_j が使われるときに記録した値をそのまま使うことができる．

また，A や V の更新の計算も高速化できる．Anda と Park による Fast Plane Rotation [69] は，Givens 回転に対して右側から対角スケーリングを施すことで，A, V の更新における計算量を $\frac{2}{3}$ に削減し，また，変更前の値を保存するベクトル q を使わずに済む手法を提案している．これには条件に合わせていくつかのパターンがあるが，そのうち 1 つを示すと，いま更新前のベクトルを a_i, a_j とする．a_i の更新では $cq - sa_j = ca_i - sa_j$ を計算するため，これを $1/c$ 倍したものを a_i' とする．

$$a_i' = a_i - (s/c)a_j. \tag{3.1}$$

また，a_j の計算では $sq + ca_j = sa_i + ca_j = sa_i' + (c + s^2/c)a_j$ であるため，これを $1/(c + s^2/c) = c/(c^2+s^2) = c$ 倍したものを a_j' とする（ただし $c = \cos\theta, s = \sin\theta$ より，三角関数の法則から $c^2+s^2 = 1$ を用いた）．

$$a_j' = (cs)\, a_i' + a_j. \tag{3.2}$$

どちらの計算も Level-1 BLAS の xAXPY の形となっており，BLAS の実装を用いることができるようになる．このとき a_i' は求めたかったベクトルの $1/c$ 倍のものとなっており，a_j' は c 倍されているため，その情報を記録しておく．結果として列ベクトルが定数倍だけされた状態で保存されることになるので，その値を d_i とおけば，次のタイミングで計算される x, y, z の計算結果をそれぞれ $d_i^2, d_j^2, d_i d_j$ で割る必要がある．また，上の更新式も，a_i や a_j がそれぞれすでに d_i, d_j 倍されていると想定して書き換える必要がある．ここでは Fast Plane Rotation の 1 つの形を示したが，a_i' ではなく a_j' を先に計算するものなどいくつか種類があり，d_i や d_j の値が過剰に小さくなったり大きくなったりしないよう，どれかを選択する．

このように，$A^\top A$ の対角要素の記録と，Fast Plane Rotatoin を用いることで，演算量を削減でき，更新のスキップが行われなかった場合の演算量は，$10m + O(1)$ となる．また，V の計算を行わない場合には $6m + O(1)$ となり，演算量を削減できる．また，計算に利用するデータ量が削減されるため，キャッシュヒット率が上がるなどの利点もある．ここで 1 巡回におけるペアの個数を $\frac{n(n-1)}{2}$ とおけば，1 巡回あたりの演算量は $5mn^2 + O(n^2 + mn)$ または $3mn^2 + O(n^2 + mn)$ となる．ただし計算のほとんどは Level-1 BLAS の内積 xDOT と積和 xAXPY であり，演算性能は低い．また計算は逐次的であり，このままの形で計算するならば並列化可能な部分は Level-1 BLAS の中だけである．さらに，行列によっては巡回回数が過大となることが考えられる．このとき，巡回回数を s とすればそのまま演算量が s 倍となるため，実行時間もそれに比例して増大する．

そこで，性能を改善するための方策として，第一に，並列化可能な部分を見出すことが考えられる．

これについては次節で取り扱う．また第二に，Level-3 BLAS を利用できるようブロック化を行うことが考えられる．これについてはその次の節で取り扱う．最後に，巡回回数を減らす方法が必要である．これについてはこの章の最後の節で取り扱う．

3.2 巡回順序と収束性

ヤコビ法における巡回順序は，計算手順を決定づける．したがって巡回順序はヤコビ法の演算性能や，並列化，通信パターンなどに大きな影響を与えるため，ヤコビ法の性能面において重要なものである．同時に，計算手順の違いはヤコビ法の理論的な収束性にも影響を与えるため，収束性に対する議論も不可欠である．ヤコビ法の収束性に対する議論においては，その理論的なふるまいを調べることが目標であるため，誤差のない場合における解析が多い．そこでそのような解析においてはヤコビ法に対する議論がヤコビ法と数式上同一である片側ヤコビ法にそのまま適用できる．

この節では，第一に，ヤコビ法の巡回順序と収束性について既存研究をまとめる．次に，巡回順序を工夫することによる高性能化手法について既存の研究結果を述べる．

3.2.1 古典的巡回順序と収束性

ここではまず，収束性の基本となる，ヤコビ法の非対角要素の二乗和の一次収束性について述べる．ヤコビ法の一次収束性はごく基本的であるため，Goldstine ら [61] のごく初期の論文においても触れられており，森 [53, 3.1 節] や杉原ら [70, 8.3 節] の教科書においても言及されている．

対称行列 C に対して Givens 回転 G によるヤコビ法の 1 ステップを施すことで (p,q) 要素を消去し，$\tilde{C}$ が得られたとする．すなわち

$$\tilde{C} = G^{\top} C G. \tag{3.3}$$

フロベニウスノルムは直交変換によって不変であるため

$$\left\|\tilde{C}\right\|_{\mathrm{F}}^2 = \|C\|_{\mathrm{F}}^2. \tag{3.4}$$

いま C と $\tilde{C}$ の第 i,j 要素をそれぞれ $\tilde{c}_{i,j}$, $c_{i,j}$ と書くとき，以上から

$$\sum_{i,j} \tilde{c}_{i,j}^2 = \sum_{i,j} c_{i,j}^2. \tag{3.5}$$

ここで式 (2.47) の第 2, 3 式について，ある k に対して両辺を自乗したものを足し合わせると，

$$\begin{aligned} \tilde{c}_{pk}^2 + \tilde{c}_{qk}^2 &= c_{p,k}^2 \cos^2\theta + c_{q,k}^2 \sin^2\theta - 2c_{p,k}c_{q,k}\cos\theta\sin\theta + \\ &\quad c_{p,k}^2 \sin^2\theta + c_{q,k}^2 \cos^2\theta + 2c_{p,k}c_{q,k}\cos\theta\sin\theta \\ &= c_{p,k}^2 + c_{q,k}^2. \end{aligned} \tag{3.6}$$

よって変更があるものは式 (2.47) の内，最後の 3 つの式のみである．よって式 (3.5) より，

$$\tilde{c}_{p,p}^2 + \tilde{c}_{q,q}^2 + 2\tilde{c}_{p,q}^2 = c_{p,p}^2 + c_{q,q}^2 + 2c_{p,q}^2. \tag{3.7}$$

いま G は $\tilde{C}$ の p,q 要素が 0 となるように定めたため，

$$\tilde{c}_{p,p}^2 + \tilde{c}_{q,q}^2 = c_{p,p}^2 + c_{q,q}^2 + 2c_{p,q}^2. \tag{3.8}$$

これは，$\tilde{c}_{p,p}^2 + \tilde{c}_{q,q}^2$ が $c_{p,p}^2 + c_{q,q}^2$ よりも大きいことを言っているが，他の対角要素は不変であるため，結局，$\tilde{C}$ の対角要素の二乗和は C のものよりも大きくなる．そこでいま行列の非対角要素の二乗和を計算する $\mathrm{offd}(C) = \sum_{i \neq j} c_{i,j}^2$ を定義すると，

$$\mathrm{offd}(\tilde{C}) = \mathrm{offd}(C) - 2c_{p,q}^2 \leq \mathrm{offd}(C). \tag{3.9}$$

すなわち，ヤコビ法では非対角要素の二乗和は非増加である．

式 (3.7) は，非対角要素を 0 にする Givens 回転が局所的に最も非対角要素の二乗和を減少させることを示している．また，式 (3.9) から，最も絶対値の大きな非対角要素を消去すれば最も大きな減少幅になることを示している．そこでこれは貪欲解法の一種だと言える．このように，絶対値の最も大きな非対角要素を選んでそれを消去する手法を古典的ヤコビ法と呼ぶ．当然これは 1 ステップの計算において最も減少幅が大きくなるようにしているだけであり，複数ステップを同時に考えたときに最も減少幅が大きくなるかどうかの保証はない．また，最も絶対値が大きな非対角要素を選ぶには探索を行う必要があり，これには 1 ステップごとに $O(n^2)$ の計算量がかかる．これは要素の更新にかかる演算量 $O(n)$ よりも大きく，非常に遅くなってしまう．Goldstine は絶対値最大の要素ではなく，以下の証明で利用するように，自乗の平均値よりも大きいものを選ぶことにも言及しているが，最悪計算量としては変化がない．杉原は，ヤコビ法における要素の変更が部分的であることを利用して探索にかかる計算量を $O(n)$ 程度にする手法に言及しているが，しかし，それでも演算性能を大きく減少させることになる．そこで古典的ヤコビ法が用いられることはあまりなく，特定の順序によって非対角要素を選択する巡回ヤコビ法が主に用いられる．

古典的ヤコビ法を用いたときの一次収束性の証明は上式から自然に導き出すことができる．数列の最大値は平均値と同じかより大きいため，

$$2c_{p,q}^2 \geq \frac{2}{n(n-1)} \mathrm{offd}(C). \tag{3.10}$$

そこで式 (3.9) より，

$$\mathrm{offd}(\tilde{C}) \leq \left(1 - \frac{2}{n(n-1)}\right) \mathrm{offd}(C). \tag{3.11}$$

すなわち，非対角要素の二乗和は一定の比率で減少する．この比率は n が大きくなると 1 に近づいていき，1 ステップあたりの収束性が悪化する．しかし同様に非対角要素の数も増えていくため，要素ごとの消去回数の平均，つまり，消去した要素の数を非対角要素で割った値がちょうど 1 となる $\frac{n(n-1)}{2}$ ステップを古典的ヤコビ法の 1 巡回と数えると，1 巡回当たりの収束率は

$$\left(1 - \frac{2}{n(n-1)}\right)^{\frac{n(n-1)}{2}} \tag{3.12}$$

となる．これも $n > 1$ に対して増加関数であるが $n \to \infty$ の極限で e^{-1} に収束する．

一方，局所二次収束性については議論が複雑になり，ヤコビ法を紹介する教科書，例えば森 [53] や木村 [71] においても議論を省くものが多い．ごく簡単に説明するならば，次のようになる．ヤコビ法が進行し，非対角要素の絶対値が十分小さく $\epsilon \ll 1$ 以下となったとき，ある条件を満たせば，Givens 回転の回転角 θ や正弦も同様に絶対値が小さくなり $|\theta|, |\sin\theta| \ll 1$，逆に余弦は 1 に非常に近くなる．そこで式 (2.47) より，消去する要素を除くすべての要素は ϵ^2 程度しか変化せず，消去する要素のみが 0 となる．通常の状態では，一度消去した要素であっても後の反復で式 (2.47) により，なんらかの値が

入ってしまうことが繰り返されるが，この状態では，一度消去した要素には ϵ^2 程度の値しか入らない．そこで，0 を ϵ^2 程度に保存したままプロセスを進行することができ，すべての非対角要素を ϵ^2 程度まで消去できる．この結果，ϵ 程度であった非対角要素が ϵ^2 程度に減少するため，二次収束となる．杉原らの教科書における証明 [70, 定理 8.6] はおおむねこの説明に従っている．また，議論がより複雑となる重複固有値が存在する場合における証明も含んでいるが，ここでは重複固有値がない場合における定理を紹介する．

定理 3.1 (杉原，室田 [70] 定理 8.6). **行列 C の固有値を $\lambda_1 > \lambda_2 > \cdots > \lambda_n$ とし，$\Delta = \min_{i \neq j} |\lambda_i - \lambda_j|$ を定義する．いま $F = \mathrm{offd}(C)$ と定義し，**

$$F < \left(\frac{2\Delta}{6 + \sqrt{n-2}} \right)^2 \tag{3.13}$$

を満たすとき，C に対して $N = \frac{n(n-1)}{2}$ ステップの古典的ヤコビ法を繰り返した後の行列 C_N と $F_N = \mathrm{offd}(C_N)$ は次を満たす：

$$F_N \leq \frac{n-2}{2} \left(\frac{F}{\Delta - 2\sqrt{F}} \right)^2 . \tag{3.14}$$

定理において固有値の最小差 Δ が出現するのは，式 (2.48) が対角要素の差を分母に持つためである．古典的ヤコビ法はどのような状態からでも一次収束をするが，二次収束はこのように条件付きである．通常，二次収束に入れば数巡回で収束条件を満たすようになるため，そこで，ヤコビ法の反復回数は，いかに速く非対角要素を小さくすることで二次収束を開始させるかがカギとなる．しかし，二次収束の開始条件に固有値差があるため，例えば，密集固有値がある場合にはヤコビ法の二次収束がなかなか始まらず，収束性が悪化することが考えられる．また，大きな固有値と，相対的に小さな密集固有値がある場合，Δ の値は小さくなるが，初期の非対角要素の二乗和は大きな固有値に依存して大きくなるため，収束性がさらに悪化するものと考えられる．そのような行列は結果的に大きな条件数を持つことになる．

3.2.2 巡回ヤコビ法

消去する非対角要素を探索ではなく固定の順序で選択する，すなわち巡回ヤコビ法は，Henrici [72] によれば，Gregory [73] によって開発された．Gregory の論文は最初期の計算機の 1 つである ILLIAC を用いており，最大でも $n = 20$ の行列でのテストを含むものであるが，実際に複数の行列での性能テストを行った結果を示しており興味深い．彼の実験においては，巡回ヤコビ法を用いたプログラムは古典的ヤコビ法を用いたプログラムと比べて，“良い” 非対角要素を選択できないため，反復回数は 2 倍程度多くなってしまうが，探索が必要ない分，実行時間については 3 倍程度高速となる結果となっている．これは，初期の，演算が相対的に遅かった計算機において，しかも n が小さく，n^2 のコストと n のコストとの差が小さい条件であることを考えると，興味深い．ただし反復回数が増えた分，残差が増大してしまっていることも確認できる．

Gregory の論文では，巡回ヤコビ法のアイデアは，上記の二次収束性の簡単な説明から着想を得ている．つまり，二次収束が始まるような状況では，すべての ϵ 程度の非対角要素を消去することが重

要であり，消去の間における要素の変化は ϵ^2 程度であるため，よほど要素間で絶対値の大きさが偏っていなければ，どの順番で消去しても同程度の結果が得られるだろう，ということである．

巡回ヤコビ法ではどのような順序でペアを選択するかが重要である．このようなペアの選択順序を“巡回順序”と呼ぶ．Gregory は“sequence along successive rows”となる順序を用いており，これは後年，行巡回順序 (Row Cyclic Ordering) と呼ばれるようになる順序である．具体的には次のような順序となる．

$$\mathfrak{J}_{\mathrm{RC}} = \begin{matrix} (1,2) & (1,3) & (1,4) & \cdots & (1,n) \\ & (2,3) & (2,4) & \cdots & (2,n) \\ & & \ddots & \cdots & \vdots \\ & & & (n-2,n-1) & (n-2,n) \\ & & & & (n-1,n) \end{matrix} \tag{3.15}$$

表記をわかりやすくするため二次元的に配置しているが，これは左上から右下へと行ごとに並べている．この順序はごく単純であり，実装が容易なだけでなく，理論的にも解析がしやすい．また逆に列ごとに進行する順序，列巡回順序 (Column Cyclic Ordering) も考えることができ，具体的には次のような順序である．

$$\mathfrak{J}_{\mathrm{CC}} = (1,2)(1,3)(2,3)(1,4)(2,4)(3,4)(1,5)\cdots(n-2,n-1)(1,n)(2,n)\cdots(n-1,n). \tag{3.16}$$

巡回順序には次のような 3 つの同値関係を考えることができる．それらは順に，順序の回転，並び替え，番号の付け替え，である．第一に，巡回ヤコビ法は何度も同じ巡回を繰り返すため，順序のどの位置から始めても同じことになる．つまり，$N = n(n-1)/2$ とおき，2 つの巡回順序 $\mathfrak{J}_1 = \mathfrak{p}_1\mathfrak{p}_2\cdots\mathfrak{p}_N$ と $\mathfrak{J}_2 = \mathfrak{p}_i\mathfrak{p}_{i+1}\cdots\mathfrak{p}_N\mathfrak{p}_1\mathfrak{p}_2\cdots\mathfrak{p}_{i-1}$ があったとき，

$$\mathfrak{J}_1 =_{\mathrm{shift}} \mathfrak{J}_2 \tag{3.17}$$

が成り立つと書く．この $=_{\mathrm{shift}}$ により N 個の異なる順序を同値だと考える。第二に、連続する 2 つのペアに含まれる数字がすべて異なるとき，それを並び替えることができる．例えば，行巡回順序に含まれる $(1,2)$ と $(1,3)$ は連続しているが 1 を共有しているため並び替えることができず，$(1,n)$ と $(2,3)$ は $n > 3$ ならば共通の数字は存在しないため，並び替えることができる．この並び替えが可能である理由は式 (2.47) から理解できる．例えば，$(1,2)$ と $(3,4)$ のペアについて考えると，前者は第 $1,2$ 行と第 $1,2$ 列の各 2 つの行と列を変更するが，$(3,4)$ 要素の消去に使用する $c_{3,3}, c_{3,4}, c_{4,3}, c_{4,4}$ の 4 つの要素は変更しない．その逆もしかりであるため，これらの順序を並び替えても得られる 2 つの Givens 回転は変化しない．また，Givens 回転 $G(1,2,\theta_{1,2})$ と $G(3,4,\theta_{3,4})$ の積は，順番を並び替えても変化しない．そこで数字の異なるペアを並び替えたときに得られる行列は元の順序で得られるものと同一になる．いま 2 つのペア $\mathfrak{p}_1 = (p,q)$ と $\mathfrak{p}_2 = (r,s)$ が異なる数字のみを持っていることを

$$\mathfrak{p}_1 \neq \mathfrak{p}_2 \Leftrightarrow p \neq r \wedge p \neq s \wedge q \neq r \wedge q \neq s \tag{3.18}$$

と定義する[*1]．このとき，2 つの巡回順序 $\mathfrak{J}_1 = \mathfrak{p}_1\mathfrak{p}_2\cdots\mathfrak{p}_i\mathfrak{p}_{i+1}\cdots\mathfrak{p}_N$ と $\mathfrak{J}_2 = \mathfrak{p}_1\mathfrak{p}_2\cdots\mathfrak{p}_{i+1}\mathfrak{p}_i\cdots\mathfrak{p}_N$ があったとき，$=_{\mathrm{swap}}$ を

$$\mathfrak{p}_1 \neq \mathfrak{p}_2 \Rightarrow \mathfrak{J}_1 =_{\mathrm{swap}} \mathfrak{J}_2 \tag{3.19}$$

と定義する．この同値関係は強力であり，非常に多くの順序が同値であることを表せる．第三に，任意の行列に対する収束性について考えるならば，行列が置き換え行列によって行・列が入れ替えられ

[*1] $\mathfrak{p}_1$ と $\mathfrak{p}_2$ は巡回順序におけるペアであるから暗黙に $p \neq q$ と $r \neq s$ が成り立っていると考えている．

たときにおいても収束しなければならない．巡回順序における数字を置き換えて得られる順序は元の順序と同値であると考える。ただし 3 つ目の同値性は使用しないものもあるため、$=_{\mathrm{shift}}$ と $=_{\mathrm{swap}}$ の 2 つの内どちらかが成り立つことを $=_{\mathrm{J}}$ と書くことにする．このような同値類を考えることによって，1 つの順序に対して得られた結果を他の順序に適用することが可能となる．例えば，Hansen [74] は行巡回順序と列巡回順序が $=_{\mathrm{J}}$ において（より正確には $=_{\mathrm{swap}}$ において）同値であることを示している．これは直感的に理解できる．行巡回順序のあるペアは，その直上（1 つ上の行の同じ列にある）ペアの直後にまで移動できるからである．つまり，あるペア $\mathfrak{p}_1 = (p, q)$ はその直上にあるペア $\mathfrak{p}_2 = (p-1, q)$ とは並び替えられないが，その右にあるすべてのペア $\mathfrak{p}_k = (p-1, q+k)$ とは $k \geq 1$ ならば並び替えられる．よって，どちらかに対して収束性の証明を行えばもう一方に適用できるようになる．

巡回ヤコビ法の収束性の証明は古典的ヤコビ法のようには直感的ではない．行巡回順序と同値な順序についての，大域収束性についての証明が Forsythe [75] によって行われており，また，Henrici [72] は同じ順序を用いたとき，重複固有値がない場合における二次収束性を示しており，Kempen [76] はそれを重複固有値に拡張した．近年にも研究は続いており Hari [77] は定数項を考えたよりシャープな結果となっている．

3.2.3 巡回順序と並列性

巡回ヤコビ法が重要な理由は，杉原ら [70, 8.3.3 節] の教科書に簡単に触れられており，Golub ら [55, Section 8.4.6] で 1 例が示されている通り，巡回順序を工夫することによって並列性を持たせることができる点である．このように並列性を持った巡回順序を用いたヤコビ法を並列巡回ヤコビ法と呼ぶ．

並列巡回ヤコビ法では，複数の消去を同時に行うことで並列化を行う．これが可能な理由は，順序の同値性の議論と同じく，隣り合う 2 つのペアが数字を共有しない場合には入れ替えても同じ結果が得られるという事実である．すなわち，それらの 2 つの消去を同時に行っても，同じ Givens 回転が得られる．ただし，ヤコビ法では Givens 回転を両側からかけなくてはならず，完全なデータ並列ではない点に注意が必要である．つまり右側からの Givens 回転を先に並列で行った後，左側からの回転を行う必要がある．片側ヤコビ法は完全なデータ並列となるためこのような工夫は必要なく，順番を並び替えることができる隣り合うペアが存在する場合には，完全に独立に列ベクトルペアの直交化を行うことができる．

このように数字を共有しない隣り合うペアの処理を並列に行うことができ，その隣にまた数字を共有しないものがあれば，それも同時に行うことができるため，1 から n までの数字を用いて，1 つも数字が重ならずに作ることのできる最大のペアの個数が，最大の並列数となる．すなわち最大 $\frac{n}{2}$ 個のペアを並列に処理することができる．このように並列に処理可能なペアの列を並列ペアを呼ぶことにする．また並列ペアを並列に消去する手順を並列ステップと呼ぶ．

式 (3.15) からわかるように，行巡回順序では，1 つ目から順に行うことを考えると並列性を見出せない．しかし処理を進めていくと $(1, n)$ と $(2, 3)$ は並列に処理が可能である．また，同値関係を使って並び替えを行っていけば，並列に処理するできるものを増やすこともできる．しかしそのような探索は手間がかかるため，あらかじめ並列に消去可能なものを隣り合うように並べた順序を用いることが普通である．このように並列化を考えたうえで作られた順序がいくつか提案されており，そのうち代表的なものが Round Robin 順序である．

Round Robin 順序は，例えば野球やサッカーなどの対戦ゲームにおいて多数のチームが存在すると

きに，総当たり戦を行う場合などに用いられる．対戦期間を削減するためには，複数の試合を同時に行うことが方法であるが，1 つのチームは複数の試合には同時に参加できないため，組み合わせを考える必要がある．Round Robin 順序は次のような表を用いると理解しやすい．いま 8 つのチームが存在するとして，最初のゲームは $(1,2)(3,4)(5,6)(7,8)$ のように隣り合うものをペアにする．これを次のような表とする．

1	3	5	7
2	4	6	8

この表において，縦に並ぶ数字をペアとする．新しいペアを作成するには，左上の数字だけを固定し，残りの数字を “回転” させる．つまり，

1	5	7	8
3	2	4	6

を得る．また次のペアは

1	7	8	6
5	3	2	4

である．この回転を $n-1$ 回行えば最初の状態に戻ることがわかる．その間に相異なる $n/2$ 個の並列ペアを生成し続けるため，$\frac{n(n-1)}{2}$ 個のすべてのペアを生成できる．表からわかるとおり，n が偶数でない場合はこのようなペアを生成できないが，その場合は値 0 を追加し，0 とのペアに対する処理はスキップすることで対処する．Round Robin 順序については Golub らによって，ここで用いた表を使って説明されている．Round Robin 順序を用いた最初期のものは Brent ら [78] である．

それ以前の並列順序生成手法としては，最も早いものとして Sameh [79] のものがあるが，Round Robin 順序のようには洗練されたものではない．Luk らによる論文 [80] は Round Robin 順序のような “回転” によって生成される多くの順序がまとめられており，また，それらの同値関係についても述べている．Stewart [81] による Odd Even 順序は，行巡回順序と同値であるという点で強力である．ただしこの順序は n が偶数である場合でも最大の並列数 $n/2$ を実現できないステップができてしまい，合計 n の並列ステップが生成されてしまう．

3.2.4 Modulus 順序

反対角順序は，行巡回順序を並び替えた順序の 1 つである．反対角順序は次のように書くことができる：

$$\begin{aligned}\mathfrak{J}_{\mathrm{AD}} = {} & (1,2)(1,3)(1,4)(2,3)(1,5)(2,4)(1,6)(2,5)(3,4)\cdots \\ & (1,n)(2,n-1)\cdots(\lfloor n/2\rfloor,\lceil n/2\rceil+1) \\ & (2,n)(3,n-1)\cdots(\lfloor n/2\rfloor,\lfloor n/2\rfloor+2) \\ & \vdots \\ & (1-n,n).\end{aligned} \tag{3.20}$$

ここで，$(1,1)$ 要素を通る反対角線を第 1 反対角線，$(1,2)$ 要素と $(2,1)$ 要素を通るものを第 2 反対角線, 一般に $i \le n$ のとき $(1,i)$ 要素を通るもの，$i > n$ のときは $(i-n+1,n)$ を通るものを第 i 反対角線と呼ぶことにし，とくに $(1,n)$ 要素を通るものを主反対角線と呼ぶことにする．反対角順序は，第 2 反対角線から順に反対角線上の右上から左下にペア（ただし上三角部分に属するもの）を選ぶものである．言い換えれば，あるペアは右上（1 つ上の行にある 1 つ右の列の）ペアの後に並べられてい

る．反対角順序が行巡回順序と $=_{\mathrm{J}}$ において（厳密には $=_{\mathrm{swap}}$ において）同値であることは直感的に理解できる．行巡回順序のあるペアはその直上にあるペアよりも前に移動させることはできないが，直上のペアの後ろまでは移動できる．そこで当然，右上のペアの後ろまで移動することもできる．反対角順序では同じ反対角線上にあるペアを同時に消去できる．つまりあるペア $\mathfrak{p}_1 = (p, q)$ と同じ反対角線上にあるペアは $\mathfrak{p}_k = (p + k, q - k)$ と表されるため，$k \neq 0$ であれば，$\mathfrak{p}_1 \neq \mathfrak{p}_k$ が言えるためである．ただし主反対角上の並列ペアが $\lfloor n/2 \rfloor$ 個あるのに対し，他の反対角の並列ペアはそれよりも少なくなってしまうという問題がある．そこで順序を回転すること，つまり $=_{\mathrm{shift}}$ の同値性を用いることで並列に処理できるペアを増やしたものが Modulus 順序である．Modulus 順序の行巡回順序との同値性は Luk らによって行われた [82]．Modulus 順序は近年には Hari や Singer らのグループによって積極的に利用されており，Singer [83] のように実装手法の詳細が解説されているため，この研究においては Modulus 順序を基にした手法を用いており，5 章の実験などで利用している．

Modulus 順序では回転によって，まず主反対角線上のペアを先頭に移動する．次に第 $n + 1$ 反対角線上のペアを選ぶが，n が偶数の場合は並列ペアの数が 1 つ少なくなる．そして第 $n + 2$ 反対角線上のペアを選ぶとき，同時に第 2 反対角線上のペアを選択し，以降は第 $n + i$ 反対角線上のペアを選ぶとき，同時に第 i 反対角線上のペアを選択する．すなわち第 i 反対角線上のペアを第 $n + i$ 反対角線上のペアの後ろになるような並び替えを行っている．これが可能であることは次のように理解できる．$i < n$ のとき，第 i 反対角線上のペアを構成する数字はすべて i 以下であるが，一方で，ある k に対して第 $n + k$ 反対角線上のペアを構成する数字はすべて k よりも大きい．そこで第 i 反対角線上のペアは $k > i$ のすべての第 $n + k$ 反対角線上のペアの前にまで移動することができる．また同じ理由で，第 i 反対角線上のペアと第 $n + i$ 反対角線上のペアを同時に消去できることがわかる．

以上から，Modulus 順序もまた odd-even 順序と同様に，n が偶数の場合においても $n/2$ 個の並列ペアを生成しない並列ステップが存在し，1 巡回の順序を生成するために n 並列ステップが必要となる．そこで並列に処理する場合には仕事をせず休んでしまうプロセッサが出現してしまうため，その場合において，使用していない数字をペアにして並列ペアに加えることが考えられる．Singer らの Modified Modulus 順序 (MMO) はそのように生成された順序であり，1 巡回に $\frac{n(n-1)}{2}$ ではなく $\frac{n^2}{2}$ 個のペアを生成するものとなっている．このように 1 巡回に 2 度以上生成されるペアが存在する場合にはその順序をとくに擬似巡回順序と呼ぶ．そのため MMO は擬似巡回順序の 1 つである．また 5 章における分散並列向けの実装では MMO を用いている．

MMO の生成順序は図を使うとわかりやすい．図3.2は $n = 8$ のときにおける MMO の生成手順を示したものである．MMO は主反対角線上のペアから始め，第 $n + i$ 反対角線上のペアと第 i 反対角線上のペアを選択していく．“あまりのペア” は必ず偶数番目の並列ステップで 1 つ発生し，$n/2$ 副対角成分を左上から右下に順に選択していることがわかる．図 3.3の MMO の生成プログラムにおいてもこの手順が反映されている．このプログラムは Singer らの実装を簡略化したものである．このプログラムは，並列数の 2 倍 w (ここでは $w = n$ とする)，スレッド IDi，並列ステップ数 r を受け取り，消去する要素の添え字 (p, q) とそのステップが “あまりのペア” かどうかを知らせるフラグ b の三つ組みを返す．

MMO は通信においても優れた特性を示す．この順序が最初に生成するペアは $(1, n)(2, n - 1)(3, n - 2)\cdots$ であるが，次に生成するものは $(2, n)(3, n - 1)(4, n - 2)\cdots$ のように，ペアのうち 1 つの数字を 1 だけ変更したものである．例えば片側ヤコビ法を並列化したときに，1 つのノードが 1 つのペアを保持するようなデータ分散を考える．このとき，最初の並列ペアでは，1 つ目のノードが第 1 番目と第 n 番目の列ベクトル，2 つ目のノードが第 2 番目と第 $n - 1$ 番目の列ベクトルを保持する．次の並列ペア

図3.2　$n = 8$ のときにおける Modified Modulus 順序のペア生成順序．表の (i, j) 位置にある数字は，(i, j) ペアが生成される並列ステップの番号を表しており，例えば $(2, 4)$ ペアは 6 つ目の並列ステップで消去される．ペアの中に 2 つの数字がある場合にはその両方のステップで処理される．

```
1  subroutine modulus-pair(w, i, r):
2          if r < w - 2*i:
3                  return (i + r, w - i - 1, false)
4          else if r == w - 2*i:
5                  return (i + r - w/2, i + r, true)
6          else:
7                  return (w/2 - i, i + r - w/2, false)
```

図3.3　Modified Modulus 順序の生成プログラム

では 2 と n をペアにするため，1 つ目のノードは 2 つ目のノードから 2 番目の列ベクトルを受け取り，2 つ目のプロセッサは 3 つ目のノードから 3 番目のものを受け取るなど，隣り合う番号のノードから 1 本の列ベクトルを受け取ることになる．そこで，全結合のような通信網でなくとも，隣り合うノード同士が結合する 1 次元形状の通信網で効率的に通信が行える．このような，通信網におけるデータ移動も考慮した並列巡回順序の生成手法についても Luk らによって言及されており [80]，同じように 1 次元形状の通信形状を考慮している Eberlein らのもの [84] や，Mantharam ら [85] によるより複雑な通信網を考えたものなどがある．

3.2.5　Block Oriented 順序

巡回順序において，隣り合うペアに同じ数字のものが存在すると，データ再利用性が生まれる．例えば，片側ヤコビ法を行巡回順序を用いて計算する場合，$(1, 2)$ の次に $(1, 3)$ を計算するため，1 つ目の列ベクトルを再利用することになる．そこで 1 本分の列ベクトルのデータをキャッシュ上に載せることができれば，メモリから CPU へのデータ転送が列ベクトル 1 つ分だけ削減することができる．さらに，キャッシュサイズが十分大きければ，複数本の列ベクトルをキャッシュ上に格納することがで

きる．そこで，行巡回順序を例えば次のように同値変形によって並び替えることを考える：

$$\begin{aligned}\mathfrak{I}_{\mathrm{BC}} = {} & (1,2)(1,3)(2,3)(1,4)(2,4)(3,4)(1,5)(2,5)(3,5)(1,6)(2,6)(3,6)\cdots(1,n)(2,n)(3,n)\\ & (4,5)(4,6)(5,6)(4,7)(5,7)(6,7)(4,8)(5,8)(6,8)(4,9)(5,9)(5,9)\cdots(4,n)(5,n)(6,n)\\ & \cdots\\ & (3i+1,3i+2)(3i+1,3i+3)(3i+2,3i+3)\\ & (3i+1,3i+4)(3i+2,3i+4)(3i+3,3i+4)\\ & (3i+1,3i+5)(3i+2,3i+5)(3i+3,3i+5)\cdots\\ & (3i+1,n)(3i+2,n)(3i+3,n)\\ & \cdots.\end{aligned}$$

この順序は，行巡回順序において，3 行ずつに分けて考え，同値変形によって $3i+1$ 行目に対して $3i+2$ 行目のペアを最も先頭に近づくように並び替え，さらに $3i+3$ 行目のペアについても同様の操作を行ったものである．この結果，例えば，$(2,3)$ のペアは $(1,n)$ との置き換え，$(1,n-1)$ との置き換え，などと繰り返し，置き換え不能となる $(1,3)$ の直後に移動し，同様に，$(2,4)$ は $(1,4)$ の直後，$(3,4)$ は $(2,4)$ の直後まで移動する，といったように並び替えられる．この結果，得られた順序は $(1,4)(2,4)(3,4)(1,5)(2,5)(3,5)\cdots$ のように，ペアの右要素を固定し，左要素を 1 から 3 まで変化させ，右要素を 1 つ動かし，また左要素を 1 から 3 まで変化させる，といったことを繰り返す．そこで 4 本の列ベクトルのデータをすべてキャッシュ上に載せることができれば，ペアの右要素に対応する列ベクトルについては 3 回，左要素に対応する列ベクトルに対しては平均的に約 $n/2$ 回，再利用されることになる．また，並び替えを行う行の数は任意に設定できるため，キャッシュに載せたい分だけ並び替える本数 n_{bo} 本を設定すればよい．

この変換は，行巡回順序から列順序への置き換えを部分的に行ったものとしてみることができる．そこで行巡回順序を n_{bo} 行ごとだけ置き換えた順序をブロック化列巡回順序と呼ぶ．また，$n_{\mathrm{bo}}=n$ と設定し，行巡回順序において，すべての行のペアを同値変形によって最も若い順序となるように置き換えたときに得られる順序が列巡回順序となる．

ブロック化列巡回順序を次のように $n_{\mathrm{bo}}\times n_{\mathrm{bo}}$ のまとまりごとに分割する：

$$\begin{aligned}\mathfrak{I}_{\mathrm{BR}} = {} & \{(1,2)(1,3)(2,3)\}\ \{(1,4)(2,4)(3,4)(1,5)(2,5)(3,5)(1,6)(2,6)(3,6)\}\cdots\\ & \{(1,n-2)(2,n-2)(3,n-2)(1,n-1)(2,n-1)(3,n-1)(1,n)(2,n)(3,n)\}\\ & \{(4,5)(4,6)(5,6)\}\ \{(4,7)(5,7)(6,7)(4,8)(5,8)(6,8)(4,9)(5,9)(5,9)\}\cdots\\ & \{(4,n-2)(5,n-2)(6,n-2)(4,n-1)(5,n-1)(6,n-1)(4,n)(5,n)(6,n)\}\\ & \cdots\\ & \{(3i+1,3i+2)(3i+1,3i+3)(3i+2,3i+3)\}\cdots.\end{aligned}$$

この順序をみると，$n_{\mathrm{bo}}\times n_{\mathrm{bo}}$ の対角ブロック（$\{(1,2)(1,3)(2,3)\}$ や $\{(4,5)(4,6)(5,6)\}$ など）の後に，$n_{\mathrm{bo}}\times n_{\mathrm{bo}}$ の非対角ブロック（$\{(1,4)(2,4)(3,4)(1,5)(2,5)(3,5)(1,6)(2,6)(3,6)\}$ など）を行ごとに並べた構造となっている．そこで，ブロック化列巡回順序は，$n_{\mathrm{bo}}\times n_{\mathrm{bo}}$ ブロック分割した行列に対してブロック要素単位で行巡回順序を用い，選択したブロック要素に対して列巡回順序を適用したものとしてみることができる．このようにブロック分割した行列のブロック要素単位ですべてのブロックが選択されるよう巡回順序を設定し，さらにブロック要素内部においても全要素が選択されるよう巡回順序を設定する，再帰的な構造を持った順序を Block Oriented 順序と呼ぶ．

このような再帰的な順序の組み合わせにおいて次の点に注意を要する．第一に，ブロック要素単位の順序は対角ブロックも選択する必要があることである．なぜならば，対角ブロックの中にも非対角要素が含まれるからである．例えば $(1,1)$ ブロック要素の中には $\{(1,2)(1,3)(2,3)\}$ が含まれているため，これらも消去しなければならない．そこで，ブロック化列巡回順序は，ブロック要素単位の順序をみると対角ブロックを含むような行巡回順序となっている．このように対角ブロックを含むような行巡回順序を Block Oriented 行巡回順序 (BORCO) と呼ぶ．第二に，ブロック内部の消去順序を考える必要があり，とくに非対角ブロック内部の消去順序は長方領域に対するペアを生成しなければならないことである．例えば，$(1,2)$ ブロック要素の中にある要素が $\{(1,4)(2,4)(3,4)(1,5)(2,5)(3,5)(1,6)(2,6)(3,6)\}$ となっており，3×3 の正方領域（ブロック幅が均一でない場合は一般に長方領域）となっている．上のブロック化列巡回順序においては，対角ブロックに対しては通常の列巡回順序，非対角ブロックに対しては，列巡回順序を拡張したものを使用していると考えることができる．

再帰的な順序の組み合わせを行ったときの巡回順序の同値性も定義できる．まずブロック要素内部の順序を 1 つに固定した場合を考える．いま対角ブロックも含めた上三角ブロックの総数を N_{B} とおく．またブロック要素単位の順序を $\mathfrak{J}_{\mathrm{B1}}=\mathfrak{b}_1\mathfrak{b}_2\cdots\mathfrak{b}_{N_{\mathrm{B}}}$ とおく．各ブロック要素単位のペア $\mathfrak{b}_i=\langle p,q\rangle$ はブロック要素内部のペアの列と対応しているため，あるブロック要素単位の順序 $\mathfrak{J}_{\mathrm{B1}}$ と $\mathfrak{J}_{\mathrm{B2}}$ が同値であるということは，それぞれのブロック要素単位のペアを対応するブロック要素内部のペアの列に展開したときに両者が $=_{\mathrm{shift}}$ や $=_{\mathrm{swap}}$ において同値であることとすることが自然な定義である．このとき，ブロック要素単位の “回転” した $\mathfrak{J}_{\mathrm{B2}}=\mathfrak{b}_i\mathfrak{b}_{i+1}\cdots\mathfrak{b}_{N_{\mathrm{B}}}\mathfrak{b}_1\mathfrak{b}_2\cdots\mathfrak{b}_{i-1}$ は，展開したときに元の順序 $\mathfrak{J}_{\mathrm{B1}}$ を回転したものとなるため，ブロック要素単位の回転は $=_{\mathrm{shift}}$ において同値であると言える．また，あるブロック要素単位のペア $\mathfrak{b}_i=\langle p,q\rangle$ と $\mathfrak{b}_{i+1}\langle r,s\rangle$ の数字が重ならないならば，つまり次の 4 つの条件，$p\neq r, p\neq s, q\neq r$ そして $q\neq s$ が満たされるならば（ただしここでは $p=q$ や $r=s$ が成り立ってもよいとしている），$\mathfrak{b}_i$ のブロック要素内部の任意のペア $\mathfrak{p}_1$ と $\mathfrak{b}_{i+1}$ のブロック内部の任意のペア $\mathfrak{p}_2$ について $\mathfrak{p}_1\neq\mathfrak{p}_2$ が成り立つ．そこでブロック要素単位のペアの不等号を

$$\mathfrak{b}_i=\langle p,q\rangle\neq\mathfrak{b}_{i+1}\langle r,s\rangle\Leftrightarrow p\neq r\wedge p\neq s\wedge q\neq r\wedge q\neq s \tag{3.21}$$

と定義すれば，$\mathfrak{b}_i\neq\mathfrak{b}_{i+1}$ が成り立つとき，2 つのブロック要素単位の順序 $\mathfrak{J}_{\mathrm{B1}}=\mathfrak{b}_1\mathfrak{b}_2\cdots\mathfrak{b}_i\mathfrak{b}_{i+1}\cdots\mathfrak{b}_{N_{\mathrm{B}}}$ と $\mathfrak{J}_{\mathrm{B2}}=\mathfrak{b}_1\mathfrak{b}_2\cdots\mathfrak{b}_{i+1}\mathfrak{b}_i\cdots\mathfrak{b}_{N_{\mathrm{B}}}$ を展開したとき，$\mathfrak{b}_{i+1}$ 内部のペアはすべて $\mathfrak{b}_i$ のペアの前に移動することができるため，両者が $=_{\mathrm{swap}}$ において同値であることがわかる．以上より，ブロック要素単位の順序に対する回転，並び替えは，展開したときにおけるペアに対する回転，並び替えに相当するため，$=_{\mathrm{J}}$ における同値変形であると言える．

またここまではブロック要素内部の順序が同じである場合を考えたが，逆に，ブロック要素単位の順序が同じだがブロック要素内部の順序が異なる場合においても，展開したときに同値となる場合は同値だとしたい．つまりそれらの 2 つの順序において，対応する位置のブロック要素単位のペアに所属するペアの集合が同じでかつ，それらのペアの順序が $=_{\mathrm{swap}}$ において同値である場合は，2 つの順序は同値であると考える．例えば，ブロック要素内部の行巡回順序，列巡回順序，ブロック化列巡回順序はすべて $=_{\mathrm{swap}}$ において同値であり，結果として，ブロック要素単位の順序が同じでかつブロック要素内部の順序が 3 つの内の異なるものを用いている順序は同値である．

以上の同値性を用いて，BORCO と同値でかつ並列化に適した順序，Block Oriented Modulus 順序 (BOMO) を作ることができる．BOMO は BORCO と同様に，対角ブロックを含むように拡張された Modulus 順序である．ただし拡張するときに BORCO との同値性を保つようにしなければならない．

そのためにまず，Block Oriented 反対角順序 (BOADO) を考える．この順序は第 1 反対角線から順に反対角線のペアを選択するものであり，上三角だけでなく対角のペアも選ぶようにしたものである．これは BORCO と同値であることは直感的に理解できる．つまり，対角のペアを含めたすべてのペアは，その直上のペアの後ろにまで移動できる．つまり右上のペアの後ろにも移動できるためである．そして BOADO を回転することで反対角線上のペアを最初に選ぶように変更し，$i \geq 1$ のとき，第 $n+i$ 反対角線上のペアと第 i 反対角線上のペアを同時に消去するものが BOMO である．これが可能であるのは，第 i 反対角線上のペアは対角のペアを含めても i 以下の数字のみを用いており，一方，第 $n+k$ 反対角線上のペアは対角のペアを含めても，k よりも大きい数字を用いているためである．よって BOMO は BORCO とブロック要素単位での回転と並び替えによって一致するため，ブロック要素内部の順序が同じであれば両者は同値である．また，ブロック要素内部の順序を列巡回順序にすれば BORCO は行巡回順序と同値となるため，BOMO においてもブロック要素内部の順序を列巡回順序と同値ものにすれば，行巡回順序と同値となる．すなわち，既存の収束性の証明を利用することができる．BOMO は 5 章で示す，1 ノード上での片側ヤコビ法の実装に用いている．

Block Oriented 順序はあくまでブロック要素単位でペアを並べた順序であり，Block Oriented 順序を用いたヤコビ法・片側ヤコビ法は，消去順序が特別である点を除けば，ただのヤコビ法・片側ヤコビ法である．この点，次節で説明するブロックヤコビ法とは異なる．ブロックヤコビ法は非対角ブロック要素を消去する，ヤコビ法のアナロジーとなっているため，計算手順などで似てはいるが，ヤコビ法とは数式上異なるものである．

3.3 ブロックヤコビ法

ブロックヤコビ法は同時に複数の要素を消去することで，収束性を高めるとともに，演算効率を高める手法である．ブロックヤコビ法は，行列をブロック分割することを考えて，ブロック要素をブロック直交変換によって消去することを繰り返す．その点においてブロックヤコビ法はヤコビ法のブロック要素単位のアナロジーであると言える．このブロック化によってブロックヤコビ法は行列積が演算の中心となり，性能が向上する．しかし，同時に複数の要素を消去するという観点から別のブロック化手法も提案されている．La Budde [86] や Ruhe [87] の手法は，対角要素を除いた列あるいは行ベクトルの要素を同時に消去する手法であり，ヤコビ法とブロックヤコビ法の中間に位置する Level-2 手法だと言える．実際に計算の大部分を Level-2 BLAS の行列積によって計算できる．

この節では，まず La Budde や Ruhe の手法を簡単に紹介した後に，ブロックヤコビ法についてその計算手順と，片側ヤコビ法のブロック化について，また，ブロック化した場合の収束性について既存の議論を示す．

3.3.1 ヤコビ法の Level-2 化手法

La Budde の手法のアイデアは，Householder 変換によってあるベクトルの要素の二乗和を 1 か所に集めておき，それを Givens 回転によって消去することである．Householder 変換は容易に計算でき，また Householder 変換による要素の更新は Level-1 BLAS が主体となる Givens 回転と違い，Level-2 BLAS である行列ベクトル積 xSYMV や行列の対称階数 2 更新である xSYR2 によって計算できるため，演算効率が向上する．

いま行列 C の第 i,j 要素を $c_{i,j}$，第 i 列を c_i と書き，C の第 1 列を対象に計算を行うことを考える．La Budde の手法では第一に，左側から Householder 変換をかけることによって C の第 1 行を不変に保ちつつ，C の第 1 列を 1 つ目の要素と 2 つ目の要素を残してすべて 0 にする．Householder 変換を $H = I - 2zz^{\mathsf{T}}/\|z\|_2^2$ と書けば，補題2.3の通り，C の第 1 行を不変に保つためには z の第 1 要素が 0 でなくてはならない．また，補題2.2の通り，c_i の 3 要素目以降を 0 にするためには，z の 3 要素目以降は c_i の 3 要素目以降と同じ値を持たなくてはならない．また，c_i の 2 要素目以降のノルムと z の 2 要素目以降のノルムは同じでなくてはならない．以上から，

$$\alpha = \sqrt{\sum_{i=2}^{n} c_{i,1}^2} \tag{3.22}$$

とおくとき，

$$z = \begin{bmatrix} 0 & \alpha & c_{3,1} & c_{4,1} & \cdots & c_{n,1} \end{bmatrix}^{\mathsf{T}} \tag{3.23}$$

が要件を満たす．この Householder 変換を C の両側からかけることで $\bar{C} = HCH$ を得るとき，左側からの変換では第 1 行が保たれ，また第 1 列が所要の形となり，その後右側からの変換を行えば，第 1 列がその形のまま保たれ，対称性より，第 1 行が第 1 列と同じ形となる．そこで

$$\bar{C} = HCH = \begin{bmatrix} c_{1,1} & \alpha & 0 & \cdots & 0 \\ \alpha & \bar{c}_{2,2} & \bar{c}_{2,3} & \cdots & \bar{c}_{2,n} \\ 0 & \bar{c}_{3,2} & \bar{c}_{3,3} & \cdots & \bar{c}_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \bar{c}_{n,2} & \bar{c}_{n,3} & \cdots & \bar{c}_{n,n} \end{bmatrix}. \tag{3.24}$$

このとき，α は第 1 列の 2 要素目以降の二乗和の平方根であるから，他の要素よりも大きな値を持っていると推測される．そこで Givens 回転 $G(1,2,\theta)$ をかけることで α を消去することを行うのが，La Budde の手法である．Givens 回転によって非零構造が破壊されるため，繰り返し同じ列に対して処理を行い，十分要素が収束すれば他の列に移行していく．La Budde の手法はこのように Householder 変換と Givens 回転を組み合わせ，非対角要素の二乗和の減少させるヤコビ法の基本的なアイデアを効率的に実現するものであるが，ヤコビ法とは全く異なった原理によって動作するため，解析は難しい．

Ruhe のアルゴリズムも Householder 変換を用いるものであるが，基となるアイデアが異なる．Ruhe は，収束につれて非対角要素が小さくなったときに，ヤコビ法の回転角 θ も小さくなることに着目し，そのような状況においては Givens 回転の積を Householder 変換によって近似できると考えた．すなわち，非対角要素が十分小さく，$|\theta| \ll 1$ のとき，ヤコビ法で用いる Givens 回転の正弦と余弦は

$$\begin{cases} \sin\theta &= \dfrac{c_{p,q}(1+O(\theta^2))}{c_{p,p}-c_{q,q}} \\ \cos\theta &= 1+O(\theta^2) \end{cases} \tag{3.25}$$

と表される [87, Eq. 2.1]．そこで例えば簡単のため $n = 3$ とした場合，$\frac{n(n-1)}{2}$ 個の Givens 回転の積 $G = G(1,2,\theta_1)G(1,3,\theta_2)G(2,3,\theta_3)$ を計算したとき，

$$G = \begin{bmatrix} 1 & c_{1,2}/(c_{1,1}-c_{2,2}) & c_{1,3}/(c_{1,1}-c_{3,3}) \\ c_{1,2}/(c_{2,2}-c_{1,1}) & 1 & c_{2,3}/(c_{2,2}-c_{3,3}) \\ c_{1,3}/(c_{3,3}-c_{1,1}) & c_{2,3}/(c_{3,3}-c_{2,2}) & 1 \end{bmatrix} (1+O(\theta^2)). \tag{3.26}$$

を得る．Ruhe の手法ではこの構造を用いて，$\frac{n(n-1)}{2}$ 個の Givens 回転の積を $n-1$ 本の Householder 変換の積で近似するものである．Ruhe はこの手法と Rayleigh 商反復との関係を説明しており，この手法は収束付近においてはヤコビ法と同じ動作をするが，全体としては異なる原理によって動作するものである．

このように La Budde の手法は，非対角要素の二乗和の減少というヤコビ法の基本的なアイデアを拝借しており，また，Ruhe の手法はヤコビ法の収束間際の動作を近似している．しかし，どちらもヤコビ法が持つごく単純な構造を引き継いでおらず，解析が難しい．また，Level-2 BLAS は当時主流であったベクトル計算機に対しては効率的であったが，より演算性能とメモリバンド幅の比が大きくなった現代の計算機においては十分なデータ再利用性がなく，計算効率は低い．

3.3.2 ブロックヤコビ法

van Loan によれば [88]，最初にブロックヤコビ法を解析したのは Hansen であり，1960 年の博士論文中においてである．van Loan の論文は，Kogbetliantz 法のブロック化が述べられており，また，Round Robin 順序を用いた並列化との組み合わせも検討されている．Foulser [89] は Kogbetliantz 法ではなく通常のヤコビ法のブロック化について述べており，並列化についても言及している．

ブロックヤコビ法では行列 C をブロック分割し，

$$C = \begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,w} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,w} \\ \vdots & \vdots & \ddots & \vdots \\ C_{w,1} & C_{w,2} & \cdots & C_{w,w} \end{bmatrix} \tag{3.27}$$

とおく．ここで w は一辺のブロック数であり，C は対称行列のため行も列も同じ数で分割する．また $C_{1,i}$ の列数を n_i とおくとき，

$$n = \sum_{i=1}^{w} n_i \tag{3.28}$$

が成り立つ．また，C は対称行列であるため，任意の i,j について $C_{i,j}$ の行数は n_i と等しいとする．分割数 w は n 以下の任意の数にすることができるが，並列化のときに簡単になるように w を偶数とする．また，以下の説明では簡単のため，すべてのブロック幅が等しく $n_i = n_\mathrm{b}$ とおく．

ブロックヤコビ法は C に対して両側からブロック回転行列 $G(p,q,\bar{V})$ をかけることで進行する．$\bar{V}$ は $2n_\mathrm{b} \times 2n_\mathrm{b}$ の直交行列であり，

$$\bar{V} = \begin{bmatrix} \bar{V}_{1,1} & \bar{V}_{1,2} \\ \bar{V}_{2,1} & \bar{V}_{2,2} \end{bmatrix} \tag{3.29}$$

とブロック分割される．ブロック回転行列 $G(p,q,\bar{V})$ は Givens 回転行列の 4 つの三角関数要素を $\bar{V}$ の各ブロック要素で置き換えたものであり，

$$G(p,q,\bar{V}) = \begin{array}{c} \\ p \\ q \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} p & & q & \end{array} \\ \begin{bmatrix} I & & & \\ & \bar{V}_{1,1} & & \bar{V}_{1,2} \\ & & I & \\ & \bar{V}_{2,1} & & \bar{V}_{2,2} \\ & & & & I \end{bmatrix} \end{array} \tag{3.30}$$

と書ける．ブロック幅 $n_\mathrm{b}=1$ の場合には $\bar{V}$ は 2×2 の回転行列にすることができ，またブロック回転行列は Givens 回転行列にできる．

ブロック回転行列によって C を両側変換したとき，$\bar{C}$ が得られたとする．すなわち

$$\bar{C}=G(p,q,\bar{V})^{\mathsf{T}}CG(p,q,\bar{V}). \tag{3.31}$$

このとき，$\bar{C}$ の各ブロック要素について次が成り立つ．

$$\begin{cases} \bar{C}_{p,q}=C_{p,q}, \quad i,j,\neq p,q \\ \bar{C}_{k,p}=\bar{C}_{p,k}^{\mathsf{T}}=C_{k,p}\bar{V}_{1,1}+C_{k,q}\bar{V}_{2,1}, \quad k\neq p,q \\ \bar{C}_{k,q}=\bar{C}_{q,k}^{\mathsf{T}}=C_{k,p}\bar{V}_{1,2}+C_{k,q}\bar{V}_{2,2}, \quad k\neq p,q \end{cases} \tag{3.32}$$

これは式 (2.47) の 1 つ目の式から 3 つ目の式をブロック要素に置きなおしたものである．ただし，4 つ目と 5 つ目の式，つまり (p,p), (p,q), (q,p), (q,q) の 4 つのブロック要素については次の式で表される．

$$\begin{bmatrix} \bar{C}_{p,p} & \bar{C}_{p,q} \\ \bar{C}_{q,p} & \bar{C}_{q,q} \end{bmatrix}=\begin{bmatrix} \bar{V}_{1,1} & \bar{V}_{1,2} \\ \bar{V}_{2,1} & \bar{V}_{2,2} \end{bmatrix}^{\mathsf{T}}\begin{bmatrix} C_{p,p} & C_{p,q} \\ C_{q,p} & C_{q,q} \end{bmatrix}\begin{bmatrix} \bar{V}_{1,1} & \bar{V}_{1,2} \\ \bar{V}_{2,1} & \bar{V}_{2,2} \end{bmatrix}. \tag{3.33}$$

つまりこれは式 (2.49) をブロック要素に置きなおしたものとみることができ，ブロック要素の更新などの計算がちょうどヤコビ法のアナロジーになっていることがわかる．

$\bar{V}$ はヤコビ法のアナロジーとして非対角ブロックを零行列になるようにしたい．これはつまり，式 (3.33) から考えると，直交変換による両側変換を用いた対称行列のブロック対角化である．ブロック対角化は中途半端に対角化した状態に見えるため，一見，対角化よりも高速に計算できそうに思われるが，実際には高速かつ高精度にブロック対角化を行う手法はあまり知られていない．中務らによる Spectral Divide & Conquer 法 [90] は，筆者の知る限り最も有望な手法であるが，利用者が設定したブロックサイズにブロック対角化するための手法ではなく，また演算量を考えると，実際にはブロック対角化ではなく対角化を行う方が高速であり，ヤコビ法をはじめとするすでによく解析された，実装も広まっているアルゴリズムを使うことも可能であり，厳密な意味でブロック対角化をする理由はない．そこで，$2n_\mathrm{b}\times 2n_\mathrm{b}$ の小行列の対角化を行うことがブロックヤコビ法の中核の部分となる．

2 章において，ヤコビ法は全体の固有値分解を行うために 2×2 の固有値分解を繰り返す手法であると説明したが，同じように解釈すれば，ブロックヤコビ法は全体の固有値分解を行うために，$2n_\mathrm{b}\times 2n_\mathrm{b}$ の小行列の固有値分解を繰り返す手法だと言える．

ブロックヤコビ法についてもヤコビ法と同じく一次収束性の議論ができる．いま，非対角ブロックのフロベニウスノルムの自乗和を表す $\mathrm{Offd}(C)\equiv\sum_{i\neq j}\left\|C_{i,j}\right\|_\mathrm{F}^2$ を定義する．直交変換によってフロベニウスノルムは不変だから

$$\left\|\bar{C}\right\|_\mathrm{F}^2=\|C\|_\mathrm{F}^2. \tag{3.34}$$

ここで式 (3.32) を考えると，このうち 1 つ目の式は値が不変なので関係がなく，第 2 式，第 3 式について考えると，ある $k\neq p,q$ について

$$\begin{bmatrix} \bar{C}_{k,p} & \bar{C}_{k,q} \end{bmatrix}=\begin{bmatrix} C_{k,p} & C_{k,q} \end{bmatrix}\begin{bmatrix} \bar{V}_{1,1} & \bar{V}_{1,2} \\ \bar{V}_{2,1} & \bar{V}_{2,2} \end{bmatrix} \tag{3.35}$$

と書けるため，これは直交変換である．すなわちこのブロックのフロベニウスノルムは不変である．そこで非対角ブロックのフロベニウスノルムの自乗和の変化に関与するものは式 (3.33) で表される，両

側変換が行われる部分だけであり，$\bar{V}$ によってこの $2n_\mathrm{b} \times 2n_\mathrm{b}$ が対角化されるとすれば，その分だけ，非対角ブロックのフロベニウスノルムの自乗和が減少する．つまり，

$$\mathrm{Offd}(\bar{C}) = \mathrm{Offd}(C) - 2\left\|C_{p,q}\right\|_\mathrm{F}^2. \tag{3.36}$$

この式から直接，古典的ヤコビ法をブロックヤコビ法に拡張した古典的ブロックヤコビ法を考えることができる．すなわち，非対角ブロックの中からフロベニウスノルムが最大のブロックを選び，そのブロック要素を消去するというものである．古典的ブロックヤコビ法の一次収束性も簡単に示すことができる．すなわち，最大値は平均値と等しいかより大きいため，フロベニウスノルム最大の非対角ブロック $C_{p,q}$ は

$$\left\|C_{p,q}\right\|_\mathrm{F}^2 \geq \frac{2}{w(w-1)}\,\mathrm{Offd}(C). \tag{3.37}$$

よって式 (3.36) に代入して上限をとれば，

$$\mathrm{Offd}(\bar{C}) \leq \left(1 - \frac{2}{w(w-1)}\right)\mathrm{Offd}(C). \tag{3.38}$$

これは，式 (3.11) の行列サイズ n をブロック数 w に置き換えたものである．$w \leq n$ であるため，収束率は改善しているが，一方で一回当たりの演算量も増えたため簡単には比較できない．ヤコビ法の 1 ステップ当たりの演算量は 3 つの演算を行う要素の更新を 4 つのベクトルに対して行うため，$12n + O(1)$ であるが，これは Fast Plane Rotation のテクニックを用いると $\frac{2}{3}$ 倍され，$8n + O(1)$ となる．さらに行列の対称性を利用すると演算量を半減でき，$4n + O(1)$ となる．ブロックヤコビ法は，$n \times 2n_\mathrm{b}$ の行列と $2n_\mathrm{b} \times 2n_\mathrm{b}$ の行列の積を 2 回行うため，$16nn_\mathrm{b}^2 + O(nn_\mathrm{b} + n_\mathrm{b}^3)$ となるが行列の対称性を利用すると $8nn_\mathrm{b}^2 + O(nn_\mathrm{b} + n_\mathrm{b}^3)$ となる．$O(n_\mathrm{b}^3)$ の項は，小行列の対角化の部分であるが，$n_\mathrm{b} \ll n$ だと仮定し，この項が小さいものと考える．また，後述する CS 分解を用いた手法を用いれば，演算量を半減でき，$4nn_\mathrm{b}^2 + O(nn_\mathrm{b} + n_\mathrm{b}^3)$ となる．結果として，ヤコビ法の 1 巡回の演算量の主要項は $4n^3$ となり，ブロックヤコビ法の 1 巡回の演算量の主要項 $4n^3$ となって一致する．そこでヤコビ法のときと同じように 1 巡回に相当する $\frac{w(w-1)}{2}$ ステップの計算を行ったときの収束率を考えると，これは式 (3.12) の n を w で置き換えた

$$\left(1 - \frac{2}{w(w-1)}\right)^{\frac{w(w-1)}{2}} \tag{3.39}$$

となり，こちらもやはり w が n よりも小さい分，改善している．

このようにブロックヤコビ法は 1 巡回当たりの演算量は主要項を見れば変化がないが，ブロック化ができ行列積を多く利用できるため，より高性能であり，また，古典的ブロックヤコビ法の場合，一次収束の収束率はブロック幅の分だけ行列サイズが縮小したのと同じような影響をもたらし，改善する．二次収束性についてはどうなるか．これについては山本ら [91] において証明があり，固有値差が十分大きい場合に二次収束することを示している．山本らは，さらに後に議論する並列化した古典的ブロックヤコビ法についても議論しており，並列化した場合でも二次収束性や一次収束性が古典的ブロックヤコビ法と変わらないことを示している．これは，並列化が悪影響を及ぼさないという意味では重要であるが，並列化に効果があるかどうかがわからない，という意味では弱い結果となっている．この点について，一次収束性への並列化の効果に関しては 5 章で議論し，並列化に一定の効果があることを示す．

ブロックヤコビ法は La Budde や Ruhe の手法とは異なり，このようにヤコビ法のブロック要素単位のアナロジーとなっているため，ヤコビ法に適用できる，巡回順序や，並列化手法が適用できる．消去するための数字のペアを生成するという意味では両者に違いがないためである．Drmač による巡回ブロックヤコビ法の大域収束性の証明 [92] は近年の成果であるが，いくつかの条件を満たす場合における収束性を示している．この条件のうち重要なものが，$\bar{V}$ に対する条件であり，$\bar{V}$ が UBC 条件と呼ばれる条件を満たす必要がある．UBC 条件も，ヤコビ法にある条件のアナロジーとして理解できる．ヤコビ法においては回転角 θ の絶対値が $\pi/4$ より小さくなる必要があったが，これは，回転によって要素があまりに大きく変わってしまうことを防ぐためである．同様に $\bar{V}$ に対しても，同じようなものが考えられる．いま，$\bar{V}$ の Cosine-Sine 分解 (CS 分解)

$$\bar{V} = \begin{bmatrix} Z_1 & \mathbb{0} \\ \mathbb{0} & Z_2 \end{bmatrix} \begin{bmatrix} C & S \\ -S & C \end{bmatrix} \begin{bmatrix} W_1 & \mathbb{0} \\ \mathbb{0} & W_2 \end{bmatrix} \tag{3.40}$$

を考える．ここで Z_1, Z_2, W_1, W_2 は直交行列であり，ブロック要素内での回転を表す．C と S は対角行列であり，$C = \mathrm{diag}(\cos\phi_1, \cos\phi_2, \dots, \cos\phi_{n_\mathrm{b}})$, $S = \mathrm{diag}(\sin\phi_1, \sin\phi_2, \dots, \sin\phi_{n_\mathrm{b}})$ とする．このときに $\cos\phi_i > \mathrm{const}$ が UBC 条件である．$\mathrm{const} > 0$ は行列や行列サイズに依存せず，ブロックサイズに依存する定数であり，すなわち，ϕ の絶対値が上から抑えられなければならないことを意味する．Drmač は実際にこの条件を満たすための手順も示しているが，これは $\bar{V}$ のピボット付き QR 分解を行うものであり，高コストであるため，4 章や 5 章の実験では用いていない．また，このような条件がなくても巡回ブロックヤコビ法は収束することが多く，実験においては収束の停滞は一度も見られなかった．

3.3.3 片側ブロックヤコビ法

片側ヤコビ法に対してもブロック化を考えることができる．片側ヤコビ法は $C = A^\mathsf{T}A$ へのヤコビ法であるため，片側ブロックヤコビ法もこのアイデアから作ることが適切だと考えられる．すなわち $A = \begin{bmatrix} A_1 & A_2 & \cdots & A_w \end{bmatrix}$ と列ブロックに分割する．このとき C の第 i,j ブロック要素について $C_{i,j} = A_i^\mathsf{T} A_j$ が成り立つ．そこでブロックヤコビ法のときと同じように $C_{p,q}$ を消去するブロック回転行列 $G(p,q,\bar{V})$ を考えると，このときの $\bar{V}$ は C の 4 つの要素を用いた部分行列に対する固有値分解を行ったときの固有ベクトルを並べた行列であり，式 (3.33) を満たすものである．これを用いて A を片側から更新して $\bar{A}$ を得たとき，

$$\bar{A} = AG(p,q,\bar{V}) \tag{3.41}$$

であり，$\bar{A}$ の各列ブロックに対して

$$\bar{A}_i = A_i, \quad i \neq p,q \tag{3.42}$$
$$\bar{A}_p = A_p \bar{V}_{1,1} + A_q \bar{V}_{2,1}, \tag{3.43}$$
$$\bar{A}_q = A_p \bar{V}_{1,2} + A_q \bar{V}_{2,2}, \tag{3.44}$$
$$\tag{3.45}$$

が成り立つ．また，$\bar{V}$ は部分行列を対角化するものであるから，ある $2n_\mathrm{b} \times 2n_\mathrm{b}$ の対角行列 Λ が存在し，

$$\begin{bmatrix} \bar{A}_p \bar{A}_q \end{bmatrix}^\mathsf{T} \begin{bmatrix} \bar{A}_p \bar{A}_q \end{bmatrix} = \bar{V}^\mathsf{T} \begin{bmatrix} A_p A_q \end{bmatrix}^\mathsf{T} \begin{bmatrix} A_p A_q \end{bmatrix} \bar{V} = \bar{V}^\mathsf{T} \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} \bar{V} = \Lambda \tag{3.46}$$

が成り立たなければならない．そこでこれは $X = \begin{bmatrix} A_p & A_q \end{bmatrix}$ に対する SVD であると考えることができ，$\bar{V}$ はこの行列の右特異ベクトルを並べたものである．また，$\bar{U}$ を X の左特異ベクトルを並べた行列，$\bar{\Sigma}$ を X の特異値を並べた対角行列だとすると $X\bar{V} = \bar{U}\bar{\Sigma} = \begin{bmatrix} \bar{A}_p & \bar{A}_q \end{bmatrix}$ を計算することになる．

実際に上の手順で SVD を計算する場合には，$\begin{bmatrix} \bar{A}_p & \bar{A}_q \end{bmatrix}$ に入る誤差に注意しなければならない．Bischof [93] は小さな特異値がある場合にこの手順によって $\begin{bmatrix} \bar{A}_p & \bar{A}_q \end{bmatrix}$ が十分に直交化できないことを述べている．実際に $\bar{V}$ の計算手法によっては収束しなくなることがあるが，4 章で対象とした手法を用いれば直交性に対する誤差が小さく，しかも高速に計算できる．この列ブロックペアの直交化手法については 4 章で詳しく述べる．

片側ブロックヤコビ法においても片側ヤコビ法のときと同じように演算量削減手法が Hari によって提案されている [94]．これは 2 つのテクニックが使われており，1 つは，列ブロックの直交化を行った後には $\bar{C}_{p,p}$ や $\bar{C}_{q,q}$ が対角行列になっていることを用いるものであり，片側ヤコビ法のときと同じようにこの対角行列の対角要素を記録しておくことで，以降のステップで $C_{p,p}$ や $C_{q,q}$ の値が必要になった時に再構築をスキップできるというものである．よってブロック要素の再構築は $C_{p,q}$ のものだけを行えばよく，$C_{p,p}$ や $C_{q,q}$ が対称行列であることを考慮すれば，再構築のための演算量を約半分にできる．またもう 1 つの手法は CS 分解のブロック対角構造を用いるものであり，$\bar{V}$ を CS 分解することで，列ブロックの更新のおける演算量を約半分にするものである．どちらも数式上正しく動作するが，誤差がある環境においてどうなるかはあまり多くの検証が行われておらず，理論的にも不明であり，また列ブロック幅が大きい場合は CS 分解の演算量が大きくなる．そこで本論文ではこの手法は用いていない．

3.4 前処理

この章の最後に，ヤコビ法に対する前処理手法について紹介する．ヤコビ法は反復回数が行列に依存するため，行列によっては非常に計算が遅くなってしまう．そこで何らかの前処理によって行列を反復回数が小さいものに変換し，計算時間を削減することが考えられる．しかし，ヤコビ法は反復解法であるため，ヤコビ法がそれ自身の前処理になっているという言い方ができる．それにもかかわらず，なぜ他の異なった前処理が必要であるのか．Fernando と Parlett [95, Section 12] は implicit Cholesky 法と呼ばれる SVD の反復解法をヤコビ法の前処理に利用できる可能性について言及しているが，ヤコビ法が，ヤコビの定常反復法の前処理として開発された経緯に触れて，“前処理” の前処理を作ろうとしている点を指摘している．そしてその節の最後をこう締めくくっている．

> Why not use the implicit Cholesky algorithm with shifts as a preconditioner? There is no loss of accuracy. The next question is: if the shifts are well chosen why switch to Jacobi? Time will tell.

たしかに，いくら機能を追加して，新しい組み合わせを考えて実際に性能向上したとしても，複雑性の増大を許すだけのものであるのかどうかは疑問である．後年，これに対して Drmač はこう答えている [2, Remark 3.5]:

> One of the messages of this paper is that switching to the Jacobi SVD algorithm can be a good idea, provided the algorithm is modified to fully exploit the work that can be done by various

preconditioners.

つまり 2 つの手法の良いとこどりをしたい，ということである．本研究の目的から答えるとするならば，それは性能におけるトレードオフである．あくまで並列化が容易であり高性能計算機に適しているヤコビ法を使いたいが，反復回数の増大と比較して許容できる範囲だけ並列計算の難しい別の手法を前処理として用いている．これは Drmač の答えを並列計算機における性能面でより具体化したものである．

QR 前処理の前身は，正定値対称行列に対する Cholesky 前処理であった．実際には $C = A^{\mathsf{T}}A$ に対する Cholesky 前処理は A に対する QR 前処理と同じであり，自然な SVD への拡張となっている．

3.4.1 Cholesky 前処理

正定値対称行列 C の EVD を計算したい場合，Cholesky 分解 $C = LL^{\mathsf{T}}$ を計算しておき，$A \leftarrow L^{\mathsf{T}}$ をおけばこの問題は A に対する SVD に帰着する．一方，$\hat{A} \leftarrow L$ とおき，$\hat{A}$ の SVD を片側ヤコビ法によって計算したとする．すなわち

$$L = U\Sigma V^{\mathsf{T}} \tag{3.47}$$

を計算すると，これは $L^{\mathsf{T}}L$ の EVD を計算したことに相当するが，

$$LL^{\mathsf{T}} = U\Sigma V^{\mathsf{T}}V\Sigma U^{\mathsf{T}} = U\Sigma^2 U^{\mathsf{T}} \tag{3.48}$$

であるため，L に対する片側ヤコビ法で得られた左特異ベクトルが C の固有ベクトルとなっており，固有値は Σ^2 により求めることができる．

Veselić と Hari [96] の手法はこの手順に従っており，ただし，Cholesky 分解ではなく，対角ピボット付き Cholesky 分解を用いている．すなわち，ある置き換え行列 P によって

$$P^{\mathsf{T}}CP = LL^{\mathsf{T}} \tag{3.49}$$

を計算し，L の特異値分解 $L = U\Sigma V^{\mathsf{T}}$ を計算する．このとき C の固有値分解 $C = (PU)\Sigma^2 (PU)^{\mathsf{T}}$ が得られる．Veselić らはこの手順におけるメモリ使用量の削減効果や実行時間の削減のほかに，$\hat{C}$ がより対角行列に近づくことによって反復回数が削減することを指摘している．

なぜこの手法がうまくいくのか．ピボットがない場合について考えると，$C = LL^{\mathsf{T}}$ に対して $\hat{C} = L^{\mathsf{T}}L$ を計算することは

$$\hat{C} = L^{\mathsf{T}}L = \left(L^{-1}L\right)\left(L^{\mathsf{T}}L\right) = L^{-1}CL \tag{3.50}$$

となり，L による相似変換となっている．すなわち C の固有値は保存される．この手順はこの後 $C \leftarrow \hat{C}$ とおきなおして繰り返し実行することで固有値解法の一種である Cholesky LR 法と呼ばれる手法になる．Cholesky LR 法は Rutishauser の LR 法を対称行列に適用したものである．LR 法はごく初期の固有値計算手法として用いられたものであり，Golub [97] によれば，1958 年に発表されたものである．LR 法はより安定に計算できる Francis の QR 法に取って代わられたが，QR 法をはじめとするその後の多くの固有値計算手法の基礎の 1 つとなっている．LR 法やその関連手法についての歴史的経緯については杉原ら [70, 第 7 章ノート] が簡潔にまとめており，また Golub [44, Section 6] がより詳しい．

LR 法や Cholesky LR 法は，用いるならば，行列を事前に三重対角や Hessenberg などの形に変換しておくことで反復における計算量を減らすが，Cholesky 前処理では密行列に対してそのまま適用する．

LR 法の収束性については同じく杉原らの教科書 [70, 定理 7.8] がわかりやすいが，そのままの形であっても一次収束となり，対称行列の場合には対角行列へと収束する．実際には固有値シフトなどのテクニックによって収束を速めることもできる．

そこで Veselič らの手法はたった 1 ステップであるが Cholesky LR 法を適用することで行列を対角行列に近づける処理であると理解できる．

3.4.2 QR 前処理

Cholesky LR 法では Cholesky 分解と並び替えの積を繰り返すが，このうち行列積と Cholesky 分解は 1 度の QR 分解に置き換えることができる．つまり $L = QR$ を計算すれば，$L^\mathsf{T}L = R^\mathsf{T}Q^\mathsf{T}QR = R^\mathsf{T}R$ は $L^\mathsf{T}L$ の Cholesky 分解である．そこで $L \leftarrow R^\mathsf{T}$ とおきなおし，QR 分解することを繰り返すことは Cholesky LR 法と数式上同一のプロセスとなる．この手順は Fernando ら [95] によれば Faddeev らによって 1968 年に認識されていた．この手順は陽に $\hat{C}$ を計算しないため，implicit Cholesky 法と呼ばれる．そこで Cholesky 前処理の応用として implicit Cholesky 法を用いた前処理を考えることができる．すなわち C の EVD を計算するときに先に Cholesky 分解 $C = LL^\mathsf{T}$ を求めておき，以降は必要な回数だけ implicit Cholesky 法を繰り返すものである．

ここで C ではなく，$C = A^\mathsf{T}A$ となる A が与えられている場合，最初の Cholesky 分解も不要となり，A の QR 分解をすぐにはじめることができる．そこで A の特異値分解を求める場合の前処理としては QR 前処理がより適切となる．また，A が縦長行列である場合は A の QR 分解を行うことで R はサイズの縮小された行列となる．これは 2 章で示した LHC 法と同じように，SVD における演算量を減らす効果がある．このように A の QR 分解を行い R^T に対して片側ヤコビ法を適用することで，より収束に近い状態から片側ヤコビ法を始めることができ，また，演算量も減らせる可能性がある．

Drmač らが用いている前処理は，QR 分解の代わりに列ピボット付き QR 分解を用いている．Drmač らは列ピボットの効果を多くの観点から説明している [2, Section 3] が，重要な観点は，列ピボット付き QR 分解が，対角スケーリングを施したときの条件数を削減する，ということである．行列 A の列ピボット付き QR 分解 $AP = QR$ を Busiger と Golub のピボット戦略 [98] で計算したとき，次が満たされる [2, Eq. 3.2].

$$R_{i,i}^2 \geq \sum_{k=i}^{j} R_{k,j}^2. \quad 1 \leq i < j \leq n \tag{3.51}$$

このとき，$D = \mathrm{diag}(R_{1,1}, R_{2,2}, \ldots, R_{n,n})$ を考えると，$D^{-1}R$ は対角が 1 で非対角要素の絶対値が 1 で抑えられる三角行列であるので，$\kappa_2(D^{-1}R) \leq 2^n$ で抑えられる [99, p. 155] [2, Remark 3.2]．これは Busiger と Golub のピボット戦略が貪欲法となっているためもたらされるものであり，例えば Gu と Eisenstat [100] のものではより良い上限となっている．また，ほとんどの行列に対してはこれほどまでに増大することはなく，おおむね $O(n)$ 程度の値となることが知られている．

これは結果として R^T の列スケーリングした条件数が小さいということである．片側ヤコビ法は行列を列スケーリングした列直交行列 $U\Sigma$ へと変換すること，つまり列スケーリングした条件数を $\mathbf{1}$ とすることが目的であるため，ピボット付き QR 分解を行った行列は，目標の状態に近い状態から処理が始まっていると言える．この意味で，ピボット化は収束性を向上するものと考えられる．

列ピボット付き QR 分解を使うことは，また別の利点がある．列ピボット付き QR 分解は行列の階数を計算するために用いられており，高精度に Null 空間を分離できる．そこで，R^T に対する片側ヤコ

ビ法ではランク落ちを無視して計算をすることができる．例えば図3.1において σ_i が 0 か極めて 0 に近い値となってしまった場合，U の値を計算することが不可能となるため，特別な処理をしなくてはならないが，QR 前処理を行った場合は，事前にこの状況を排除できる．

3.4.3 Drmač と Veselić の手法

Drmač らによって実装された LAPACK における片側ヤコビ法の前処理 [2, Section 5] は QR 前処理を基にしているが，計算したい値（左特異ベクトル・右特異ベクトルを計算するかどうか）や，QR 前処理がどれほどうまくいったか（行スケーリングした条件数が十分小さくなったかどうか）に応じて処理を切り替える．具体的には，implicit Cholesky 法の反復回数を変更し，ごくまれに 1 回だけのもの，通常は 2 回，そして最大 3 回まで行う．また，その結果行列の条件数がよくなっていることを用いて，本来計算が必要な値を行列方程式から算出する．そこでここでは，左と右の両方の特異ベクトルを計算する場合に選択される 2 段階の前処理と 3 段階の前処理についてのみ説明する．

2 段階の前処理

2 段階の前処理でははじめ，QR 前処理の手順通り

$$AP = Q_1 R \tag{3.52}$$

を計算する．このとき R の行スケーリングした条件数が十分良ければ，次の反復ではピボッティングを行わず，$R^\mathsf{T} = Q_2 R_2$ を計算し，R^T に対して片側ヤコビ法を適用する．ただし，QR 分解と転置をまとめて LQ 分解

$$R = LQ_2 \tag{3.53}$$

を計算する．

この後，$L = U\Sigma V^\mathsf{T}$ を片側ヤコビ法によって計算すれば，A の SVD は

$$A = Q_1 R P^\mathsf{T} = (Q_1 U)\,\Sigma\left(V^\mathsf{T} Q_2 P^\mathsf{T}\right) \tag{3.54}$$

を得る．これには 2 回の行列積（正確には Householder 変換の列として Q_1 や Q_2 が保存されるため，Householder 変換の適用）が含まれており，演算量が多い．そこで式 (3.53) より

$$R = U\Sigma V^\mathsf{T} Q_2 \tag{3.55}$$

を式変形し，

$$Q_2^\mathsf{T} V = R^{-1} U\Sigma \tag{3.56}$$

から $Q_2^\mathsf{T} V$ を計算できる．これは Householder 変換の適用の演算量の主要項 $2n^3$ に対して主要項として n^2 の演算量で計算できるため，より高速である．また，Householder 変換の適用は複雑な計算となるため，Level-3 BLAS で定義された演算である，三角行列による行列方程式の計算 xTRSM はより高性能であると期待される．さらに片側ヤコビ法の中での V の更新を省くことができる．これは片側ヤコビ法の演算量を $\frac{3}{5}$ にする効果があり，高速化につながる．

この前処理は反復回数の削減のためには十分強力であり，また，次に示す 3 段階の前処理と比べて演算量が少ないため，5 章で示す高性能計算向けの片側ブロックヤコビ法実装においてはこちらのもののみを実装し，利用した．ただし，4 章の実験においては，2 段階の前処理と 3 段階の前処理を自動

で切り替える LAPACK のルーチンを利用しており，2 段階だけのものを実装した高性能計算機向け実装との比較をしている．

3 段階の前処理

1 度目の前処理

$$AP_1 = Q_1 R_1 \tag{3.57}$$

によって十分 R_1 の行スケーリングした条件数が小さくならなければ，さらに 2 回の implicit Cholesky 法の反復を繰り返す．つまり，

$$R_1^\top P_2 = Q_2 R_2, \tag{3.58}$$
$$R_2 = L Q_3. \tag{3.59}$$

この後，$L = U\Sigma V^\top$ を計算する．そこで

$$A = Q_1 R_1 P_1^\top = Q_1 P_2^\top R_2^\top Q_2^\top P_1 = Q_1 P_2^\top Q_3^\top L^\top Q_2^\top P_1 = (Q_1 P_2^\top Q_3^\top V)\Sigma(U^\top Q_2^\top P_1) \tag{3.60}$$

によって A の SVD が計算できる．

ここでも 2 段階の前処理と同様に V の計算を省くことができる．すなわち式 (3.58) より

$$R_2 = U\Sigma V^\top Q_3, \tag{3.61}$$

すなわち

$$Q_3 V = R_2^{-1} U\Sigma. \tag{3.62}$$

この後，$Q_1 P_2 (Q_3^\top V)$ と $P_1^\top Q_2 U$ を計算すればよいため，合計で 2 回の Householder 変換の適用と 1 回の行列方程式の計算によって計算できる．

第 4 章

列ブロックペアの直交化手法の誤差解析

片側ブロックヤコビ法の主な計算は，あるステップにおける行列の列ブロックペア $X = \begin{bmatrix} A_i & A_j \end{bmatrix}$ の SVD を計算し，列直交化することである．いま X の SVD を $X = U_X \Sigma_X V_X^{\mathsf{T}}$ と書けば，V_X を右側からかけることで $Y \equiv XV_X = U_X \Sigma_X$ となるため，列直交化できる．この計算の特徴は 2 つあり，第一に X が縦長行列であることで，簡単のためすべての列ブロックが同じ幅を持っていると仮定すると，縦の長さ対横の長さの比が $w : 2$ となる．第二に，X の列スケーリングした条件数が多くの場合，小さくなることである．これは，もともとの行列 A が QR 前処理された行列であることが主な理由である．場合によっては A の列スケーリングした条件数が反復の間に上昇することがあるが，Demmel らの実験 [1] によれば，この上昇幅は小さく，また反復に従って急速に 1 に収束していくことが示されている．X はある反復における A の列ブロックペアであるから，X の列スケーリングした条件数はその反復における A のものよりも小さくなる．そこで，X の列スケーリングした条件数が小さいものと想定する．

X の右特異ベクトル V_X によって列直交化することは単なる SVD と行列積の計算であるが，有限精度の演算によって行う場合には問題があることが Drmač [92, Eq. 40] によって示されている．Drmač が解析している誤差は列直交化で生じる誤差の一部に過ぎないが，彼の結果は誤差が直交性に及ぼす影響について重要なことを示唆する．いま計算によって求めた X の右特異ベクトル $\hat{V}_X$ が真の右特異ベクトル V_X から誤差 δV_X だけずれており，$\hat{V}_X = V_X + \delta V_X$ と書けるとする．またある正則な対角行列 D によって $X = X'D$ と書き，X' はすべての列ベクトルがノルム 1 になっているとする．このとき $X\hat{V}_X$ について次の方程式が成り立つ：

$$X\hat{V}_X = XV_X + X\delta V_X = (U_X + X'\delta F)\Sigma_X, \tag{4.1}$$

$$\delta F = D\delta V_X \Sigma_X^{-1}. \tag{4.2}$$

これは，δV_X のノルムが小さい場合でも，直交行列 U_X からのずれ $X'\delta F$ は大きくなりうることを示している．これは X を十分に列直交化できないことにつながり，片側ブロックヤコビ法全体の収束が滞ることにつながる．これは，Drmač [92, Eq. 41] によって後退誤差が小さくなることが示されていることとは対照的である．

しかし，上記は粗い解析であり，行列 X の構造を考慮してより詳しい解析を行うことで，直交性の誤差に対してより小さな上界を求められる可能性がある．そこで，本研究では列ブロックペアの直交化手法の 1 つである Hari らの V2 手法 [101, Section 4.1]（以降から単に V2 手法と呼ぶ）について，理論的な誤差解析を行い，いくつかの条件が成り立つならば，直交性の誤差が小さいことを示す．V2 手法は，Cholesky QR 法という誤差の大きな手法を内部的に利用するにも関わらず，実験においては高

精度に計算できている．そこでこの章では，まず V2 手法についてその手順を示した後に，理論的誤差解析の結果を示す．また，Hari の V2 手法を用いて片側ブロックヤコビ法を実装したときの，実際の Y の直交性や，SVD 全体の誤差を実験的に調べる．この章の内容は関連論文 [a] において簡略に示した証明を詳細化し，非並列向けプログラムの実験結果を示したものである．

4.1 列ブロックの直交化手法

いま $X = \begin{bmatrix} A_i & A_j \end{bmatrix}$ が $n \times l$ 行列であるとする．l は列幅であり，A_i と A_j それぞれの列幅 n_i, n_j の和 $l = n_i + n_j$ である．また A は QR 前処理をした後の行列を想定しているため，A_i, A_j の行数は n とする．列ブロックの直交化は本質的には SVD であるため，片側ヤコビ法を直接 X に対して行うことも可能である．しかし片側ヤコビ法は性能が低いため，その代わりに，行列を $l \times l$ 行列へと縮小してからその行列に対して片側ヤコビ法を適用する手法がある．その手法の 1 つが $C = X^{\mathsf{T}}X$ を計算し，C の固有値分解 $C = V_X D_X V_X^{\mathsf{T}}$ から V_X を計算する手法であり，これは Bečka や Okša [102] によって用いられている．もう 1 つの手法が X の QR 分解 $X = QR$ を利用する方法である．R の SVD $R = U_R \Sigma_R V_R^{\mathsf{T}}$ を計算すれば，$X = (QU_R)\Sigma_R V_R^{\mathsf{T}}$ は X の SVD となるため，$V_X = V_R$ とおいてよい．これは 2 章で示した LHC 法であるが，R の SVD を標準的な二重対角化を用いる手法ではなく，片側ヤコビ法によって計算する点が異なる．V2 手法はこのうち後者を用いるものである．

ここで QR 分解を行う手法が問題であるが，V2 手法では Cholesky QR 法を用いている．Cholesky QR 法とは，まず Gram 行列 $C = X^{\mathsf{T}}X$ を計算し，この行列の Cholesky 分解 $C = R^{\mathsf{T}}R$ を計算して，$Q = XR^{-1}$ とおく手法である．このとき，$X = QR$ が成り立ち，また，$Q^{\mathsf{T}}Q = \left(XR^{-1}\right)^{\mathsf{T}}\left(XR^{-1}\right) = R^{-\mathsf{T}}CR^{-1} = I$ も成り立つため，Q，R はそれぞれ X の Q 因子，R 因子となる．Cholesky QR 法は例えば Golub [55, Theorem 5.2.2] の証明の中で用いられるなど，理論的な存在は知られており，また，行列が縦長である場合に他の QR 分解手法と比べて高速に計算できる [103]．これは，計算の主要部が行列積 $C = X^{\mathsf{T}}X$（Level-3 BLAS の xSYRK）と三角行列の求解 $Q = XR^{-1}$（Level-3 BLAS の xTRSM）から成っており，どちらも一種の行列積として効率的に計算できるためである．しかし，C の条件数が X の条件数の自乗となるため，計算誤差の大きなアルゴリズムであることも知られている．なお，深谷の論文 [103] は演算量を 2 倍にする代わりに計算誤差を小さくする Cholesky QR2 法と呼ばれる改良手法について解説している．

Cholesky QR 法によって X の QR 分解を計算した後，片側ヤコビ法により R の右特異ベクトル行列 V_X を求めることになるが，Hari はそのために V1，V2，V3 の 3 つの手法を提案している [101][*1]．

V1 手法は，R に対する片側ヤコビ法において R の右側からかけられた Givens 回転 $G^{(i)}$ の積を計算するものであり，片側ヤコビ法の反復回数を M とすると，$V_X = \prod_i^M G^{(i)}$ である．この手法は，Givens 回転の積の計算が Level-1 BLAS に相当する計算となっているため低性能であるが，Hari らは実験において他の 2 手法よりも高精度になるとしている．V2 手法は，$R = U_R \Sigma_R V_R^{\mathsf{T}}$ と分解したとき，行列方程式から $V_X = V_R = R^{-1}U_R\Sigma_R$ を計算するものである．数式上は V1 手法と同じ結果が得られるはずであるが，数値計算上では誤差が異なる．V3 手法は同様に行列方程式から $V_R^{\mathsf{T}} = \Sigma_R^{-1}U_R^{\mathsf{T}}R$ が成り立つため，$V_X = V_R = R^{\mathsf{T}}U_R\Sigma_R^{-1}$ を計算する方法である．Hari らは実験において，V3 手法を用いたときヤコビ

[*1] これらの手法は，片側ブロックヤコビ法とよく似た手法である，ブロック化した J-Jacobi 法 [101] に対して提案されたものであるが，全く同じ手順を片側ブロックヤコビ法に適用できる．

```
subroutine V1(X):
  C = X^T X
  R = chol(C) // C = RR^T
  [U_X, Σ, V_X] = JacobiSVD(R)

  return XV_X
```

```
subroutine V2(X):
  C = X^T X
  R = chol(C) // C = RR^T
  [U_X, Σ] = JacobiSVD(R)
  V_X = R^{-1} U_X Σ
  return XV_X
```

```
subroutine V3(X):
  C = X^T X
  R = chol(C) // C = RR^T
  [U_X, Σ] = JacobiSVD(R)
  V_X = R^T U_X Σ^{-1}
  return XV_X
```

図4.1　V1, V2, V3 手法の擬似プログラム

法が収束しなくなる場合があることを示している．これらの手順をまとめたものが図 4.1である．

そこで実験において収束した V1,V2 手法について理論的解析を行いたいが，ここでは，Cholesky QR 法に対する既存の誤差解析 [104] に用いられているテクニックを基に，V2 手法が持つ行列単位の構造を利用することで V2 手法について誤差解析が得られた．そこで以降ではこの結果を示す．

4.2　Hari の V2 手法に対する誤差解析

ここでは列ブロックペアの直交化に伴う 2 種類の誤差を調べる．1 つ目の誤差は，結果の直交性に関する誤差であり，2 つ目の誤差は後退誤差である．いま V2 手法によって有限精度によって計算した V_X を $\hat{V}_X$，同様に有限精度で計算した $Y = X\hat{V}_X$ を $\hat{Y} = fl(X\hat{V}_X)$ と書く．1 つ目の誤差 δU は，ある直交行列 $\bar{U}$ と対角行列 $\bar{\Sigma}$ によって，

$$\hat{Y} = \left(\bar{U} + \delta U\right)\bar{\Sigma} \tag{4.3}$$

と書けた場合の δU と定義する．すなわち，結果の行列 $\hat{Y}$ に対して列スケーリングを行って得られる行列の，直交行列からのずれである．通常，δU が大きければ $\hat{Y}$ の直交性が悪くなるため，片側ブロックヤコビ法が収束しなくなる．また 2 つ目の誤差 δX はある直交行列 $\bar{V}$ によって，

$$\hat{Y} = (X + \delta X)\,\bar{V} \tag{4.4}$$

と書けた場合の δX と定義する．どちらの式においてもそれを満たす行列と誤差の組は複数存在し得るが，誤差の上限を求めることが目標であるため，なにか 1 つこのような行列の組が見つかればよい．

δX が大きい場合，元の行列と大きく異なる行列 $X + \delta X$ の SVD を計算することになるため，片側ブロックヤコビ法における特異値や特異ベクトルの誤差が大きくなる．そこでこの節ではまず Cholesky QR 法（正確には，それを改良した Cholesky QR2 法）に対する誤差解析を行っている山本ら [105] の証明を参考に，1 つ目の直交性誤差の解析を行う．また，2 つ目の後退誤差については簡単に，Drmač [92, Eq. 41] の式を V2 手法へ詳細化する．

ここでは証明のために次のような表記を行う．ある行列 A に対して，それを有限精度演算によって計算した近似値を $\hat{A}$，対角行列によって列スケーリングしたものを A' とする．また近似値を列スケーリングしたものを $\hat{A}'$ と書く．また，行列 A の第 i,j 要素を $A_{i,j}$ と書く．行列 A に対して，その要素の絶対値 $A_{i,j}$ を要素とする行列を $|A|$ と書く．行列に対する不等式は要素ごとの定義とし，丸め誤差の単位を $\mathbf{u}$ とする．以下に登場する任意の m に対して $\gamma_m \equiv \frac{m\mathbf{u}}{1-m\mathbf{u}} < 1.01m\mathbf{u} = O(m\mathbf{u})$ が満たされるものとする．また，$n \geq l$ かつ，$n^2\mathbf{u} \ll 1$ が成り立つとする．

また証明のための道具として次の結果を用いる．Higham の教科書 [106] によれば，$m \times n$ 行列 A, B と実数 $\alpha \geq 0$ について次が成り立つ：

$$|A| \leq \alpha \Rightarrow \|A\|_2 \leq \sqrt{mn}\alpha, \tag{4.5}$$

$$|A| \leq |B| \Rightarrow \|A\|_2 \leq \sqrt{\operatorname{rank} B}\,\|B\|_2, \tag{4.6}$$

$$\|A\|_2 \leq \||A|\|_2 \leq \sqrt{\operatorname{rank} A}\,\|A\|_2 . \tag{4.7}$$

また，$n \times k$ 行列 B_2 について，行列積 AB_2 を有限精度で計算した場合，その誤差 E は次のように書ける：

$$fl(AB_2) = AB_2 + E, \tag{4.8}$$
$$|E| \leq \gamma_n |A|\,|B_2| . \tag{4.9}$$
$$\tag{4.10}$$

ただし，$A = B_2^{\mathsf{T}}$ の場合，$AB_2 = (AB_2)^{\mathsf{T}}$ が成り立つが，誤差についても $E = E^{\mathsf{T}}$ が成り立つものとする．これは自然な制約である；結果が対称行列となる行列積では上・下三角部分の片方のみ計算し，もう一方の値と一致させることで演算量を減らすため，誤差も対称となるためである．また与えられた正則な $n \times n$ の三角行列 T と与えられたベクトル b について，方程式 $Tx = b$ を数値的に計算し $\hat{x}$ を得たとき，ある δT が存在し，

$$(T + \delta T)\hat{x} = b, \tag{4.11}$$
$$|\delta T| \leq \gamma_n |T| . \tag{4.12}$$

よってノルムで評価すれば $\|\delta T\|_2 \leq O\left(n^{\frac{3}{2}}\mathbf{u}\right)\|T\|_2$.

また，Gram 行列の間に成り立つ次の性質を用いる．

補題 4.1. 正則な正方行列 A, B について

$$A^{\mathsf{T}}A = B^{\mathsf{T}}B \tag{4.13}$$

が成り立つとき，

$$AB^{-1} = A^{-\mathsf{T}}B^{\mathsf{T}} = \left(AB^{-1}\right)^{-\mathsf{T}} \tag{4.14}$$

すなわち，転置が逆行列に等しいため，AB^{-1} は直交行列．したがって，ある直交行列 W が存在して $A = WB$ と書ける．

4.2.1 V2 手法の直交性誤差

本小節では，V2 手法の手順に従って，4 つのステップに分けて直交性誤差の評価を行う．第 1 のステップは Cholesky QR 法による X の QR 分解，第 2 のステップは片側ヤコビ法による R の特異値分解，第 3 ステップは V_X の計算，第 4 ステップは行列積 XV_X である．

Cholesky QR 法の誤差

最初に，直交化したい行列 X をある対角行列 D によって，列ベクトルが単位長さを持つように $X = X'D$ と変換する．すなわち，$D = \mathrm{diag}(d_1, d_2, \ldots, d_l)$ とし，X の i 番目の列ベクトルを x_i とおくとき，$d_i = \|x_i\|_2$ である．V2 手法ではまず X に対して Gram 行列 $C = X^{\mathsf{T}}X$ を計算し，その後，Cholesky 分解 $C = RR^{\mathsf{T}}$ を計算する．このとき有限精度計算で求めた C と R をそれぞれ $\hat{C}, \hat{R}$ とおくとき，次が成り立つ．

補題 4.2. Gram 行列の計算および Cholesky 分解における誤差をぞれぞれ E_1，E_2 とする．すなわち，

$$\hat{C} = C + E_1 = X^\top X + E_1, \tag{4.15}$$
$$\hat{R}^\top \hat{R} = \hat{C} + E_2 = C + E_1 + E_2. \tag{4.16}$$

このとき

$$|E_1|_{i,j} \le \gamma_n d_i d_j \tag{4.17}$$
$$|E_2|_{i,j} \le \gamma_{l+1}\sqrt{\hat{C}_{i,i}\hat{C}_{j,j}} \le \gamma_{l+1}(1+\gamma_n) d_i d_j = O(l\mathbf{u}) d_i d_j. \tag{4.18}$$

証明. 行列積の誤差評価から $|E_1| \le \gamma_n |X|^\top |X|$. よって，

$$\begin{aligned} |E_1|_{i,j} &\le \gamma_n \sum_{k=1}^{n} |X|_{k,i}|X|_{k,j} \\ &\le \gamma_n \left\| |x_i| \right\|_2 \left\| |x_j| \right\|_2 \\ &= \gamma_n d_i d_j. \end{aligned} \tag{4.19}$$

E_2 についての最初の不等式は Demmel[107, Lemma 2.1] による．2 つ目の不等式については，式 (4.17) より $\hat{C}_{i,i} \le (1+\gamma_m) d_i^2$ を代入し，

$$\gamma_{l+1}(1+\gamma_n) = O(l\mathbf{u})(1+O(n\mathbf{u})) = O(l\mathbf{u}). \tag{4.20}$$

□

以上より，$\hat{R}' = \hat{R}D^{-1}$ と定義すれば，$\hat{R}'^\top \hat{R}' = X'^\top X' + E_3$. ただし，

$$E_3 = D^{-1}(E_1 + E_2)D^{-1}, \tag{4.21}$$
$$|E_3| \le O(n\mathbf{u}) \ll 1. \tag{4.22}$$

ここで，式 (4.5) を用いると，

$$\|E_3\|_2 \le O(nl\mathbf{u}) \ll 1 \tag{4.23}$$

が得られる．

以下では，

$$O(nl\mathbf{u})\kappa_2^2(X') \ll 1 \tag{4.24}$$

が成り立つことを仮定する．前処理を行った片側ブロックヤコビ法では，列スケーリングを行った X の条件数が小さくなることから，これは妥当な仮定であると考えられる．いま，行列 $X'^\top X'$ の最小固有値・最大固有値をそれぞれ $\lambda_{\min}(X'^\top X')$，$\lambda_{\max}(X'^\top X')$ と書くと，X' の列ベクトルの長さがすべて 1 であることから，$\lambda_{\max}(X'^\top X') \ge 1$ となる．したがって，式 (4.24) より，

$$\lambda_{\min}(X'^\top X') \gg \lambda_{\max}(X'^\top X') O(nl\mathbf{u}) \ge O(nl\mathbf{u}). \tag{4.25}$$

また，$\hat{R}'^\top \hat{R}' = X'^\top X' + E_3$ であるから，式 (4.22)，(4.5) と Weyl の定理より，

$$\lambda_{\min}(\hat{R}'^\top \hat{R}') \ge \lambda_{\min}(X'^\top X') - \|E_3\|_2 \ge \lambda_{\min}(X'^\top X') - O(nl\mathbf{u}) \ge O(nl\mathbf{u}) \tag{4.26}$$

が成り立つ．これは，

$$\left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \ll 1 \tag{4.27}$$

とも書き換えられる．

ここで $\hat{R}$ が R とどれほど近いかを評価する．具体的には後に出現する形のために，$\hat{R}^{-1}$ を R^{-1} によって表すことにする．

補題 4.3. 式 (4.24) の仮定の下で，$\left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \ll 1$ が成り立つとき，ある直交行列 W_1 と誤差 E_4 が存在し，

$$\hat{R}^{-1} = R^{-1}(W_1 + E_4), \tag{4.28}$$

$$\|E_4\|_2 \leq \left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \tag{4.29}$$

が成り立つ．

証明.

$$\begin{aligned}\left(R\hat{R}^{-1}\right)^\top \left(R\hat{R}^{-1}\right) &= \hat{R}^{-\top} C \hat{R}^{-1} \\ &= \hat{R}^{-\top}(\hat{R}^\top \hat{R} - DE_3D)\hat{R}^{-1} \\ &= I - \hat{R}'^{-\top} E_3 \hat{R}'^{-1} \equiv I + E_5.\end{aligned} \tag{4.30}$$

両辺を見比べると E_5 は対称行列となるため，E_5 の EVD $E_5 = W_2\Gamma_1 W_2^\top$ を考えると，

$$\begin{aligned}\|E_5\|_2 &= \|\Gamma_1\|_2 \\ &\leq \left\|\hat{R}'^{-1}\right\|_2^2 \|E_3\|_2 \\ &\leq \left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \ll 1.\end{aligned} \tag{4.31}$$

ただし最後から 2 番目の不等号では式 (4.23)，最後の不等号では式 (4.27) を用いた．また，式 (4.30) において $I + E_5 = W_2(I + \Gamma_1)W_2^\top = \left((I + \Gamma_1)^{\frac{1}{2}} W_2^\top\right)^\top \left((I + \Gamma_1)^{\frac{1}{2}} W_2^\top\right)$ と書き，補題4.1において，$A = R\hat{R}^{-1}$，$B = \left((I + \Gamma_1)^{\frac{1}{2}} W_2^\top\right)$ とおけば，ある直交行列 $W_3 = AB^{-1}$ が存在し，

$$R\hat{R}^{-1} = W_3 (I + \Gamma_1)^{\frac{1}{2}} W_2^\top. \tag{4.32}$$

よって，$(I + \Gamma_1)^{\frac{1}{2}} = I + \Gamma_2$ とおけば，$W_1 = W_3 W_2^\top$, $E_4 = W_3\Gamma_2 W_2^\top$ とおくことで，

$$\hat{R}^{-1} = R^{-1}(W_1 + E_4) \tag{4.33}$$

と書ける．ここで Γ_1, Γ_2 は対角行列であり，$\|\Gamma_1\|_2 \ll 1$ であるため，$(I + \Gamma_1)^{\frac{1}{2}}$ のテイラー展開の 1 次の項までを考えることで，

$$\|E_4\|_2 = \|\Gamma_2\|_2 \simeq \frac{1}{2}\|\Gamma_1\|_2 \leq \frac{1}{2}\left\|\hat{R}'^{-1}\right\|_2^2 O(nl\mathbf{u}) \tag{4.34}$$

となる．ただし，最後の不等号では式 (4.31) を用いた．よって，補題が成り立つ． □

次に，$\hat{R}'$ の条件数を調べる．$\hat{R}'^{\mathsf{T}}\hat{R}' = R'^{\mathsf{T}}R' + E_3$ が成り立つため，

$$\left\|\hat{R}'\right\|_2^2 \leq \|R'\|_2^2 + \|E_3\|_2 \leq (1 + \|E_3\|_2)\|R'\|_2^2. \tag{4.35}$$

よって平方根をとれば，$\|E_3\|_2 \ll 1$ であるため，

$$\left\|\hat{R}'\right\|_2 \leq \left(1 + \frac{1}{2}\|E_3\|_2\right)\|R'\|_2 \leq O(1)\|R'\|_2 = O(1)\|X'\|_2. \tag{4.36}$$

一方，$\hat{R}'^{\mathsf{T}}\hat{R}' = X'^{\mathsf{T}}X' + E_3$ と式 (4.25)，(4.23) より，

$$\left\|\hat{R}'^{-1}\right\|_2 \leq O(1)\left\|X'^{-1}\right\|_2. \tag{4.37}$$

以上より，次の補題を得る．

補題 4.4. 式 (4.24) の仮定の下で，

$$\kappa_2(\hat{R}') = O(1)\kappa_2(R') = O(1)\kappa_2(X'). \tag{4.38}$$

片側ヤコビ法によって得られた左特異ベクトル行列 $\hat{U}$ の誤差

次に $\hat{R}$ に対する片側ヤコビ法が正常に終了し，$\hat{R}$ の左特異ベクトル行列と特異値を要素とする対角行列の積の近似値 $\hat{T} = \hat{U}\hat{\Sigma}$ を得たとする．片側ヤコビ法が計算する値は $\hat{U}$ や $\hat{\Sigma}$ ではなく $\hat{T}$ であるため，以下では $\hat{T}$ を直接利用する．$\hat{U}$ や $\hat{\Sigma}$ は $\hat{T}$ から数式上で導き出されたものであり，$\hat{\Sigma}$ は正確に $\hat{T}$ の列ベクトルのノルムを対角に持ち，$\hat{U}$ の列ベクトルはすべて単位長さであるとする．また $\hat{T}$ と $\hat{U}$ の各列ベクトルをそれぞれ $\hat{t}_i, \hat{u}_i$ と書き，$\hat{\Sigma}$ の対角要素を $\hat{\sigma}_i$ と書く．片側ヤコビ法はすべての列ベクトルのペアが十分直交したときに終了するため，誤差を含めたときの終了条件は次のように書ける：$i \neq j$ について，

$$fl(t_i^{\mathsf{T}}t_j) \leq fl\left(\mathrm{tol}\sqrt{t_i^{\mathsf{T}}t_i}\sqrt{t_j^{\mathsf{T}}t_j}\right). \tag{4.39}$$

ただし tol は 3 章で示した片側ヤコビ法の実装に合わせて $\mathrm{tol} = \sqrt{l}\mathbf{u}$ とおく．

補題 4.5. $i \neq j$ について終了条件 (4.39) が成り立つとき，次が成り立つ：

$$\left|\hat{u}_i^{\mathsf{T}}\hat{u}_j\right| \leq O(l\mathbf{u}). \tag{4.40}$$

証明. まず式 (4.39) の右辺について，

$$\begin{aligned} fl\left(\mathrm{tol}\sqrt{t_i^{\mathsf{T}}t_i}\sqrt{t_j^{\mathsf{T}}t_j}\right) &\leq (1+\mathbf{u})^4\,\mathrm{tol}\sqrt{fl(t_i^{\mathsf{T}}t_i)}\sqrt{fl(t_j^{\mathsf{T}}t_j)} \\ &\leq (1+\mathbf{u})^4\left(1+\frac{1}{2}\gamma_l\right)^2\mathrm{tol}\,\|t_i\|_2\left\|t_j\right\|_2 \\ &\leq O(1)\mathrm{tol}\,\|t_i\|_2\left\|t_j\right\|_2. \end{aligned} \tag{4.41}$$

ただし，第 1 の不等号での $(1+\mathbf{u})^4$ は，2 回の平方根とその積，および tol との積で生じる誤差による．また，第 2 の不等号では $fl(t_i^\top t_i) \le \|t_i\|_2^2 (1+\gamma_l)$ を用いた．次に式 (4.39) の左辺を評価する．一般に，長さ l の 0 でないベクトル x, y と定数 $\alpha \ge 0$ について

$$\left|fl(x^\top y)\right| \le \alpha \tag{4.42}$$

$fl(x^\top y) = x^\top y + e$ と書くとき，内積に対する誤差解析の結果 [106] から

$$|e| \le \gamma_l \ |x|^\top |y| \le \gamma_l \|x\|_2 \|y\|_2 . \tag{4.43}$$

よって

$$\begin{aligned} \left|x^\top y\right| &\le \left|fl(x^\top y)\right| + |e| \\ &\le \alpha + \gamma_l \|x\|_2 \|y\|_2 . \end{aligned} \tag{4.44}$$

そこで $x = t_i$, $y = t_j$, $\alpha = fl\left(\mathrm{tol}\sqrt{t_i^\top t_i}\sqrt{t_j^\top t_j}\right)$ とおけば，

$$\begin{aligned} \left|t_i^\top t_j\right| &\le (O(1)\mathrm{tol} + \gamma_l) \|t_i\|_2 \left\|t_j\right\|_2 \\ &= O(l\mathbf{u}) \|t_i\|_2 \left\|t_j\right\|_2 . \end{aligned} \tag{4.45}$$

よって両辺を $\|t_i\|_2 \left\|t_j\right\|_2$ で割れば補題が成り立つ． □

補題 4.6. $\hat{U}$ に対して，ある直交行列 $\bar{U}$ と

$$\left\|\delta\hat{U}\right\|_2 \le O(l^2\mathbf{u}) \ll 1. \tag{4.46}$$

を満たす誤差行列 $\delta\hat{U}$ が存在して，

$$\hat{U} = \bar{U} + \delta\hat{U} \tag{4.47}$$

と書ける．

証明. 終了条件より $\left|\hat{U}^\top\hat{U} - I\right| \le O(l\mathbf{u})$ が成り立つ．そこで

$$\left\|\hat{U}^\top\hat{U} - I\right\|_2 \le O(l^2\mathbf{u}). \tag{4.48}$$

つまり，1 だけシフトした行列のスペクトル半径が $O(l^2\mathbf{u})$ になるということである．そこで対称行列 $\hat{U}^\top\hat{U}$ の固有値分解を $\hat{U}^\top\hat{U} = Z(I+\Gamma_3)Z^\top$ とする．このとき Γ_3 は対角行列であり，$|\Gamma_3| \le O(l^2\mathbf{u}) \ll 1$. そこで $(I+\Gamma_3)^{\frac{1}{2}} = (I+\Gamma_4)$ とおき，$Z(I+\Gamma_3)Z^\top = \left((I+\Gamma_4)Z^\top\right)^\top \left((I+\Gamma_4)Z^\top\right)$ と書くとき，補題4.1より，ある直交行列 W_4 が存在して，次の式が成り立つ．

$$\hat{U} = W_4(I+\Gamma_4)Z^\top = W_4Z^\top + W\Gamma_4Z^\top. \tag{4.49}$$

よって，それぞれ $\bar{U} = W_4Z^\top$, $\delta\hat{U} = W_4\Gamma_rZ^\top$ とおくと，$\bar{U}$ は直交行列で，$\delta\hat{U}$ は

$$\left\|\delta\hat{U}\right\|_2 = \|\Gamma_4\|_2 \le O(l^2\mathbf{u}) \tag{4.50}$$

を満たす． □

V_X の計算

次に V_X を $V_X = R^{-1}U\Sigma$ によって計算するが，実際には誤差を持った行列 $\hat{V}_X = fl(\hat{R}^{-1}\hat{T})$ を計算する．このとき $\hat{V}_X$ の各列ベクトルを $\hat{v}_i$ と書くと，三角行列に対する線型方程式の求解における誤差 $\delta\hat{R}_i$ によって

$$\begin{aligned}\hat{v}_i = fl(\hat{R}^{-1}\hat{t}_i) &= (\hat{R} + \delta\hat{R}_i)^{-1}\hat{t}_i \\ &= \left(\left(I + \delta\hat{R}_i\hat{R}^{-1}\right)\hat{R}\right)^{-1}\hat{t}_i \\ &= \hat{R}^{-1}\left(I + \delta\hat{R}_i\hat{R}^{-1}\right)^{-1}\hat{t}_i \end{aligned} \tag{4.51}$$

と書ける．ただし，は式 (4.11) で定義される，三角行列を係数とする方程式の求解における誤差である．ここで上で定義した対角行列 D によって $\hat{R}' = \hat{R}D^{-1}$ と $\delta\hat{R}_i' = \delta\hat{R}_i D^{-1}$ を定義する．

補題 4.7. いま式 (4.24) が成り立つと仮定する．このときある F_i が，

$$I + F_i = \left(I + \delta\hat{R}_i\hat{R}^{-1}\right)^{-1} \tag{4.52}$$

を満たし，

$$\|F_i\|_2 \le O(l^{\frac{3}{2}}\mathbf{u}\kappa_2(\hat{R}')). \tag{4.53}$$

証明. 三角行列を係数とする方程式の求解における誤差は $\left|\delta\hat{R}_i\right| \le \gamma_l\left|\hat{R}\right|$ を満たす．この両辺は非負行列であるから，左から非負の対角行列 D^{-1} をかけると，これを絶対値の中に入れることができて，

$$\left|\delta\hat{R}_i'\right| \le \gamma_l\left|\hat{R}'\right|. \tag{4.54}$$

よって $\left\|\delta\hat{R}_i'\right\|_2 \le O\left(l^{\frac{3}{2}}\mathbf{u}\right)\left\|\hat{R}'\right\|_2$. そこで

$$\begin{aligned}\left\|\delta\hat{R}_i\hat{R}^{-1}\right\|_2 &= \left\|\delta\hat{R}_i'\hat{R}'^{-1}\right\|_2 \\ &\le \left\|\delta\hat{R}_i'\right\|_2\left\|\hat{R}'^{-1}\right\|_2 \\ &\le O\left(l^{\frac{3}{2}}\mathbf{u}\right)\left\|\hat{R}'\right\|_2\left\|\hat{R}'^{-1}\right\|_2 = O\left(l^{\frac{3}{2}}\mathbf{u}\right)\kappa_2(\hat{R}') \ll 1. \end{aligned} \tag{4.55}$$

ここで，式 (4.24) が成り立つとき，補題4.4より $O\left(l^{\frac{3}{2}}\mathbf{u}\right)\kappa_2(\hat{R}') \le O\left(nl\mathbf{u}\right)\kappa_2(\hat{R}') = O\left(nl\mathbf{u}\right)\kappa_2(X') \ll 1$ であることを用いた．よって $I + F_i = \left(I + \delta\hat{R}_i\hat{R}^{-1}\right)^{-1}$ のノイマン級数が収束し，

$$I + F_i = \sum_{k=0}^{\infty}\left(-\delta\hat{R}_i\hat{R}^{-1}\right)^k. \tag{4.56}$$

また，

$$\|F_i\|_2 = \frac{\left\|\delta\hat{R}_i\hat{R}^{-1}\right\|_2}{1 - \left\|\delta\hat{R}_i\hat{R}^{-1}\right\|_2} \le O\left(l^{\frac{3}{2}}\mathbf{u}\right)\kappa_2(\hat{R}'). \tag{4.57}$$

□

上記の証明中で用いたように，仮定 (4.24) の下では $O\left(l^{\frac{3}{2}}\mathbf{u}\right)\kappa_2(\hat{R}') \ll 1$ が成り立つ．このとき，

$$\begin{aligned}\hat{v}_i &= \hat{R}^{-1}(I+F_i)\hat{t}_i = \hat{R}^{-1}(\hat{u}_i + F_i\hat{u}_i)\hat{\sigma}_i \\ &= \hat{R}^{-1}(\hat{u}_i + \delta\hat{u}_i)\hat{\sigma}_i. \end{aligned} \tag{4.58}$$

ただし $\delta\hat{u}_i = F_i\hat{u}_i$ と定義した．そこで $\delta\hat{u}_i$ を並べた行列 $\delta\hat{U} = [\delta\hat{u}_1\delta\hat{u}_2\cdots\delta\hat{u}_l]$ を定義すると，

$$\begin{aligned}\hat{V}_X = \hat{R}^{-1}(\hat{U}+\delta\hat{U})\hat{\Sigma} &= \hat{R}^{-1}(I+\delta\hat{U}\hat{U}^{-1})\hat{U}\hat{\Sigma} \\ &= \hat{R}^{-1}(I+E_6)\hat{U}\hat{\Sigma}\end{aligned} \tag{4.59}$$

ただし $E_6 = \delta\hat{U}\hat{U}^{-1}$. このとき $\|E_6\|_2$ を評価したい．

補題 4.8. 式 (4.24) の仮定の下で，，

$$\|E_6\|_2 \le \left\|\delta\hat{U}\right\|_2 \left\|\hat{U}^{-1}\right\|_2 \le O(l^2\mathbf{u})\kappa_2(\hat{R}'). \tag{4.60}$$

証明. まず $\hat{U}$ は式 (4.49) のように，特異値分解によって $\hat{U} = W(I+\Gamma_2)Z^{\mathrm{T}}$ と表され，$|\Gamma_2| \le O(l^2\mathbf{u})$ であるから，$\hat{U}$ の最小特異値は $1 - O(l^2\mathbf{u})$ 以上である．そこで

$$\left\|\hat{U}^{-1}\right\|_2 \le 1 + O(l^2\mathbf{u}). \tag{4.61}$$

また，

$$\left\|\delta\hat{U}\right\|_2 \le \left\|\delta\hat{U}\right\|_{\mathrm{F}} \le \sqrt{\sum_{i=1}^{l}\|\delta\hat{u}_i\|_2^2} \le O(l^2\mathbf{u})\kappa_2(\hat{R}'). \tag{4.62}$$

よって両者の積を計算すれば，$\|E_6\|_2 \le O(l^2\mathbf{u})\kappa_2(\hat{R}')$. □

行列積の計算

最後に行列積 XV_X における誤差を調べる．$\hat{Y} = fl(X\hat{V}_X)$ とおくとき，

$$\hat{Y} = X\hat{V}_X + E_{MM}, \tag{4.63}$$

$$|E_{MM}| \le \gamma_l |X| \left|\hat{V}_X\right|. \tag{4.64}$$

ここで $\hat{V}_X$ 自体が誤差を持つため，$X\hat{V}_X$ と E_{MM} の両方を評価しなければならない．まず前者について，式 (4.59) と式 (4.28) を代入し，

$$\begin{aligned}X\hat{V}_X &= X\hat{R}^{-1}(I+E_6)\hat{U}\hat{\Sigma} \\ &= XR^{-1}(W_1+E_4)(I+E_6)\hat{U}\hat{\Sigma}.\end{aligned} \tag{4.65}$$

ここで X の QR 分解 $X = QR$ を考え，式 (4.47) を代入すると

$$\begin{aligned}X\hat{V}_X &= Q(W_1+E_4)(I+E_6)\hat{U}\hat{\Sigma} \\ &= Q(W_1+E_4+W_1E_6+E_4E_6)(\bar{U}+\delta\hat{U})\hat{\Sigma} \qquad (4.66) \\ &= \left(QW_1\bar{U}+E_7\right)\hat{\Sigma}. \qquad (4.67)\end{aligned}$$

ただし $E_7 = Q(E_4 + W_1E_6 + E_4E_6)(\bar{U} + \delta\hat{U}) + QW_1\delta\hat{U}$ であり，

$$\begin{aligned}\|E_7\|_2 &\leq \|E_4 + W_1E_6 + E_4E_6\|_2 \left\|\bar{U} + \delta\hat{U}\right\|_2 + \left\|\delta\hat{U}\right\|_2 \\ &\leq O(l^2\mathbf{u})\kappa_2(\hat{R}') + O(nl\mathbf{u})\left\|\hat{R}'^{-1}\right\|_2^2 + O(l^2\mathbf{u}) \\ &\leq O(nl\mathbf{u})\kappa_2^2\left(\hat{R}'\right). \end{aligned} \tag{4.68}$$

次に行列積の誤差 E_{MM} について評価すると，$|X|\left|\hat{V}_X\right| = |X'|DD^{-1}\left|R'^{-1}(W_1 + E_4)(I + E_6)\hat{U}\right|\hat{\Sigma}$ であるから，

$$\begin{aligned}\left\|E_{MM}\hat{\Sigma}^{-1}\right\|_2 &\leq O(l^2\mathbf{u})\|X'\|_2\left\|R'^{-1}\right\|_2\|W_1 + E_4\|_2\|I + E_6\|_2\left\|\hat{U}\right\|_2 \\ &\leq O(l^2\mathbf{u})\kappa_2(X'). \end{aligned} \tag{4.69}$$

最後に式 (4.68) と式 (4.69) の評価を使って式 (4.63) を書き直し，また，補題4.4によって条件数を $\kappa_2(X')$ に統一すると，次を得る．

定理 4.9. $X \in \mathrm{R}^{n\times l}$ は $n \geq l$ である列フルランクな行列とする．また，X の各列のノルムを並べた対角行列 D によって $X = X'D$ と書いたとき，$O(nl\mathbf{u})\kappa_2^2(X') \ll 1$ を満たすとする．X に対する V2 手法が失敗せず終了し，$\hat{Y}$ が得られたとき，ある列直交行列 $\bar{U}$ と対角行列 $\hat{\Sigma}$, 誤差 δU が存在し，

$$\hat{Y} = (\bar{U} + \delta U)\hat{\Sigma}, \tag{4.70}$$

$$\|\delta U\|_2 \leq O(nl\mathbf{u})\kappa_2^2(X'). \tag{4.71}$$

前処理を行った片側ブロックヤコビ法においては列スケーリングした X の条件数が小さいと想定しているため，定理4.9の結果は非常に望ましい．

注意 4.10. 定理4.9では簡単のため，X' の各列ベクトルが単位長さとなるように D を決めたが，その代わりに，任意の正則でかつ対角要素が正の対角行列 D を用いることができる．その場合，証明において X' の列ベクトルの長さを用いている部分が変化し，具体的には E_1 と E_2 が変化する．またそれらに依存する誤差も変化し，E_3, E_4, E_5, E_7 そして $\kappa_2(\hat{R}')$ を再評価しなければならない．

いま，X が列フルランクであり，D によって列スケーリングした行列 X' について，$O(nl\mathbf{u})\kappa_2^2(X') \ll 1$ が成り立つと仮定する．ただし，D の定数倍の変化は $O(nl\mathbf{u})\kappa_2^2(X') \ll 1$ に影響を与えないことに注意し，D は $\|X'\|_2 = 1$ となるように選ぶとする．このとき，

$$O(nl\mathbf{u})\left\|X'^{-1}\right\|_2^2 = O(nl\mathbf{u}) \cdot \frac{\kappa_2^2(X')}{\|X'\|_2^2} = O(nl\mathbf{u})\kappa_2^2(X') \ll 1 \tag{4.72}$$

も成り立つ．このとき，式 (4.17) と式 (4.18) は

$$|E_1| \leq \gamma_n |X|^{\mathsf{T}}|X|, \tag{4.73}$$

$$|E_2| \leq \gamma_{l+1}\left|\hat{R}\right|^{\mathsf{T}}\left|\hat{R}\right|. \tag{4.74}$$

ただし 2 つ目の式は Higham [39, Theorem 10.3] による．そこで $E_3 = D^{-1}(E_1 + E_2)D^{-1}$ も変化し，

$$\|E_3\|_2 \leq \gamma_n l\|X'\|_2^2 + \gamma_{l+1}l\left\|\hat{R}'\right\|_2^2. \tag{4.75}$$

ただし定理4.9と同じ仮定を用いると，

$$\left\|\hat{R}'\right\|_2^2 \le \|X'\|_2^2 + \gamma_n l \|X'\|_2^2 + \gamma_{l+1} l \left\|\hat{R}'\right\|_2^2, \tag{4.76}$$

よって，

$$\left\|\hat{R}'\right\|_2^2 \le \frac{\|X'\|_2^2 + \gamma_n l \|X'\|_2^2}{1 - \gamma_{l+1} l} \le O(1) \|X'\|_2^2 = O(1). \tag{4.77}$$

そこで式 (4.75) は簡略化できる：

$$\|E_3\|_2 \le O(nl\mathbf{u}) \|X'\|_2^2 = O(nl\mathbf{u}) \ll 1. \tag{4.78}$$

次に $\hat{R}'$ の条件数について考えると，Weyl の定理から，$R'^{\top}R'$ と $\hat{R}'^{\top}\hat{R}'$ の最小固有値をそれぞれ $\lambda_{\min}$, $\lambda'_{\min}$ とおけば，

$$\lambda'_{\min} \ge \lambda_{\min} - \|E_3\|_2 \ge \lambda_{\min} - O(nl\mathbf{u}). \tag{4.79}$$

ここで $\lambda_{\min} = \frac{1}{\left\|R'^{-1}\right\|_2^2} = \frac{1}{\left\|X'^{-1}\right\|_2^2} \gg O(nl\mathbf{u})$ であるから，結局

$$\left\|\hat{R}'^{-1}\right\|_2 \le O(1) \left\|R'^{-1}\right\|_2. \tag{4.80}$$

そこで

$$\kappa_2(\hat{R}') = O(1)\kappa_2(R'). \tag{4.81}$$

最後に $\|E_4\|_2 = \frac{1}{2}\|E_5\|_2$ を評価すると，式 (4.31) の第二式における E_3 の評価を置き換えるため，

$$\|E_4\|_2 = \frac{1}{2}\|E_5\|_2 \le \left\|\hat{R}'^{-1}\right\|_2^2 \|E_3\|_2 \le O(nl\mathbf{u})\kappa_2^2(X'). \tag{4.82}$$

よってこれを式 (4.68) の第一式に代入すれば，

$$\|E_7\|_2 \le O(nl\mathbf{u})\kappa_2(X') + O(l^2\mathbf{u})\kappa_2(\hat{R}') + O(l^2\mathbf{u}) = O(nl\mathbf{u})\kappa_2(X'). \tag{4.83}$$

結局，E_7 の評価は同じ形となった．ただし，ここでの X' は上記の仮定が成り立つ任意の D によって対角スケーリングされたものであるから，そのようなもののなかで最も良いものを使うことができる．その結果，X の各列ベクトルを長さ 1 にスケーリングする D_{col} を使った場合よりも，直交性の誤差の限界を小さくできる可能性がある．Sluis [108] はこのような対角スケーリングによる条件数の変化について調べており，次が成り立つことを示している：

$$\kappa_2(XD_{\mathrm{col}}^{-1}) \le \sqrt{l} \min_{D \in \mathcal{D}} \kappa_2(XD). \tag{4.84}$$

ただし $\mathcal{D}$ は正則な対角行列の集合である．すなわち，D_{col} はほぼ最適値に近いスケーリングとなっており，最適な D を選んできたとしても大きな改善はされない．

注意 4.11. 定理4.9では $\hat{R}$ の左特異ベクトル行列 $\hat{U}$ を計算するために片側ヤコビ法を用いることを前提としているが，証明の続きで用いている条件は式 (4.40) のみである．そこで直交性の条件，式 (4.40) を満たす任意の SVD 手法によって左特異ベクトル行列を計算しても同じ直交性の誤差の上界を得ることが可能である．さらに極端な例として，$\hat{U} = I$ としてもこの誤差の上界は変化しない．

実際には，片側ヤコビ法以外の手法を用いた場合，次の小節で示す，後退誤差が問題となる．

4.2.2 V2 手法の後退誤差

直交性に対する誤差においては $\hat{V}_X$ がどれだけ直交行列に近いかを使わなかった．しかし，片側ブロックヤコビ法は片側からの直交変換を繰り返すことが基本であるため，$\hat{V}_X$ が直交行列に十分近いことは必要なことである．この点に関して，すでに知られた事実があるため，それを紹介する．ある三角行列 $\hat{R}$ に対して片側ヤコビ法を適用し，左特異ベクトルと対角行列の積 $\hat{T}$ を得たとする．そして，$\hat{V}_X = fl(\hat{R}^{-1}T)$ によって $\hat{V}_X$ を計算したものとする．このとき，Drmač によれば，次の定理が成り立つ．

定理 4.12 (Drmač[2], Eqs. (5.3), (5.7), (5.8)). **$\hat{V}_X$ に対して，直交行列 $\bar{V}_X$ と誤差 $\delta\hat{V}_X$ が存在し，**

$$\bar{V}_X = \hat{V}_X + \delta\hat{V}_X, \tag{4.85}$$

$$\left\|\delta\hat{V}_X\right\|_2 \leq \kappa_{\mathrm{R}}(\hat{R}) \cdot O(sl^2\mathbf{u}). \tag{4.86}$$

ただし s は片側ヤコビ法における収束までにかかった巡回回数であり，$\kappa_{\mathrm{R}}(\hat{R})$ は，$D_{\mathrm{row}}\hat{R}$ の各行が単位長さを持つように対角行列 D_{row} によって行スケーリングしたときの条件数 $\kappa_{\mathrm{R}}(\hat{R}) = \kappa_2(D_{\mathrm{row}}\hat{R})$ である．

注意 4.13. この定理は二重対角化を用いた特異値計算手法，GK 法では成り立たない．そこで高い直交性を持った V_X を得るためには，$\hat{R}$ から左特異ベクトル行列を得るときに，片側ヤコビ法を用いる必要がある．

いま $\tilde{x}_j$ を X の第 j 番目の行ベクトルとおき，$\hat{Y}$ の第 j 番目の行ベクトルを $\tilde{y}_j = fl(\tilde{x}_j\hat{V}_X)$ と書く．このとき，ある行ベクトル δx が存在して，

$$\tilde{y}_j = (\tilde{x}_j + \delta x)\bar{V}_X, \tag{4.87}$$

$$\|\delta x\|_2 \leq O(l^{\frac{3}{2}}\mathbf{u})\left\|\tilde{x}_j\right\|_2\left\|\hat{V}_X\right\|_2 + \left\||\tilde{x}_j|\right\|_2\|\delta V_X\|_2 \tag{4.88}$$

が成り立つ．そのため，行ごとの後退誤差 δx の大きさの上限は $\kappa_{\mathrm{R}}(\hat{R})$ に比例する．

Drmač は $\kappa_2(\hat{R}')$ が非常に 1 に近い場合に近い場合に $\kappa_{\mathrm{R}}(\hat{R})$ も同様に小さくなることを示している [2, Proposition 3.1]. そこで，片側ブロックヤコビ法の反復が進行していくと $\kappa_{\mathrm{R}}(\hat{R})$ も小さくなると期待される．実際に後述の実験の中では，$\kappa_{\mathrm{R}}(\hat{R})$ は $\kappa_2(\hat{R}')$ と比べて大きすぎる値になることは観測されず，むしろ頻繁により小さな値となっていた．

しかし，$\kappa_{\mathrm{R}}(\hat{R})$ が小さいことについての理論的な保証は存在しない．そこで，実装においては $\kappa_{\mathrm{R}}(\hat{R})$ が大きすぎる場合には V2 手法を使わずに，$\hat{V}_X$ が直交行列に近いことが保証されている V1 手法に切り替える．この切り替え手法では，どのように条件数を計算し，どのような条件で切り替えを行うかが重要である．条件数の計算手法は実装の容易さと計算の高速性を考えて，LAPACK の条件数推定ルーチンである xTRCON を用いることにする．これは 1 ノルムに対する条件数 κ_1 または ∞ ノルムに対する条件数 κ_∞ を推定するものであるが，ここでは $\kappa_{\mathrm{R}}(\hat{R}) = \kappa_2(D_{\mathrm{row}}\hat{R}) \approx \kappa_1(D_{\mathrm{row}}\hat{R})$ によって代用することにする．また，切り替えの判定基準は

$$\kappa_1(D_{\mathrm{row}}\hat{R}) \leq \sqrt{l} \tag{4.89}$$

とし，xTRCON に推定した値がこれを満たせば V2 手法，満たさなければ V1 手法を用いるようにした．実際には以下の実験においてはこの切り替えは頻繁には発生しない．

4.3 実験による解析

以上のように理論的な誤差解析はできたが，そこには行列に依存する定数項が含まれ，また，あくまで上限であるため，実際の傾向とは異なることが予測される．そこで，多くのテスト行列に対して誤差の定数項や，片側ブロックヤコビ法全体の誤差を調べることで，実際の行列における誤差の傾向を調査する．

このテストでは 2 つのプログラムを用いた．1 つは LAPACK の片側ヤコビ法実装をブロック化することを想定し，LAPACK の前処理と，独自に作成した片側ブロック化実装を組み合わせたプログラム，もう 1 つは，5 章で詳細を説明する分散並列向けプログラムである．ただしテストにおいては多くの種類の行列を用いることを優先したため，行列サイズや並列数が 5 章で用いるものよりもずっと小さなものとなっているが，このテスト結果はその延長線上に結果を予測するための重要な情報を与えるものと考えられる．これらの 2 つのプログラム最も大きな相違点が前処理であり，前者のプログラムでは，LAPACK に実装された Drmač らによる複雑な選択基準を用いた前処理の自動切換えを行うものが使われているのに対して，後者のプログラムでは，単純かつ高速に計算できる 2 段階の前処理のみを使っていることである．また，前者では非並列向けの巡回順序である Row Cyclic 順序を用いているが，後者では並列向けである Modified Modulus Ordering を用いている．さらに，Hari の V1 手法の実装と（切り替え手法を用いた）V2 手法との誤差の比較も行った．

テスト行列は，特異値分布 (mode)，条件数 (κ)，行列サイズ ($m = n$)，列ブロック数 (w) の各項目が異なる乱数行列を用いた．条件数 κ は $10^5, 10^{10}, 10^{15}$ の 3 つのものを用いた．行列サイズは $m = n = 200,\ 400,\ 800,\ 1600$ の 4 種類を用いた．列ブロック数は $w = 10,\ 20,\ 40$ の 3 つを用いた．また，特異値分布は条件数 κ と行列サイズ $m = n$ が与えられたときに，行列の特異値 σ_i をどのように設定するかであり，LAPACK のテスト行列生成ルーチン DLATM1 によって生成される．各 mode の値によって設定される特異値は次のようになっている．

mode=1 1 つだけ大きな特異値を持ち，残りが小さいもの: $\sigma_1 = 1, \sigma_2 = \cdots \sigma_n = 1/\kappa$.
mode=2 1 つだけ小さな特異値を持ち，残りが大きいもの: $\sigma_n = 1/\kappa, \sigma_1 = \cdots \sigma_{n-1} = 1$.
mode=3 特異値が等比数列となるもの: $\sigma_i = \kappa^{(i-1)/(n-1)}$.
mode=4 特異値が等差数列となるもの: $\sigma_i = 1 - \frac{1-\kappa}{n-1}(i-1)$.
mode=5 自然対数をとったときに一様分布となる乱数: $\sigma_i = \mathrm{e}^{-r \log \kappa}$, ただし r は $(0,1)$ 区間に一様分布する乱数.

このとき，$\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ とし，テスト行列 A は乱数で生成された直交行列 Q_1, Q_2 によって次のように生成する：

$$A = Q_1 \Sigma Q_2. \tag{4.90}$$

また各項目の組み合わせに対して，乱数シードの異なる 5 つの行列を作成した．これらの組み合わせの総数は $3 \times 4 \times 3 \times 5 \times 5 = 900$ となり，非常に多数であるため，以下の結果では，mode ごとの最大値を使ったり，または mode ごとにわけて，さらに次のようなインデックスごとに乱数シードだけを

表4.1　非並列向けプログラムにおける DTRCON によって推定した $\kappa_1(\hat{R}')$ の mode ごとの最大値．N/A はその巡回で列ブロックペアの直交化がすべてスキップされたか，片側ブロックヤコビ法がすでに収束したことを表す．

巡回回数	mode = 1	mode = 2	mode = 3	mode = 4	mode = 5
1	22.687	1.000	34.289	31.498	35.216
2	1.156	N/A	1.894	3.733	1.841
3	1.011	N/A	1.019	1.329	1.034
4	1.002	N/A	1.000	1.019	1.000

表4.2　非並列向けプログラムにおける DTRCON によって推定した $\kappa_1(D_{\mathrm{row}}\hat{R})$ の mode ごとの最大値．N/A の意味は表4.1と同じ．

巡回回数	mode = 1	mode = 2	mode = 3	mode = 4	mode = 5
1	12.230	1.000	19.244	22.664	18.190
2	1.123	N/A	1.789	3.417	1.685
3	1.011	N/A	1.018	1.308	1.031
4	1.002	N/A	1.000	1.019	1.000

変更したときの最大値を計算したりした:

$$\mathrm{index} = 9\log_2(n/200) + 3\log_2(w/10) + \log_{10}(\kappa)/5 - 1. \tag{4.91}$$

例えば，$m = n = 400$, $\kappa = 10^{10}$, $w = 10$ の組み合わせのインデックスは index $= 9 + 0 + 2 - 1 = 10$ となる．

実験では次の 4 つの値を測定した．1 つ目は，誤差の上限に出現する定数である，$\kappa_2(X')$ と $\kappa_{\mathrm{R}}(R)$ である．実際には数値計算によって求めた値に対して条件数推定ルーチン DTRCON を使って 1 ノルムに対する条件数を計算するため，その値は $\kappa_1(\hat{R}')$ と $\kappa_1(D_{\mathrm{row}}\hat{R})$ の推定値である．ここで κ_1 は 1 ノルムの条件数であり，D_{row} は $D_{\mathrm{row}}\hat{R}$ の行のノルムを 1 にする対角行列である．それぞれの値について，1 巡回毎に 4 巡回まで，mode ごとの最大値を測定した．2 つ目の値は計算によって求めた $\hat{Y}$ の直交性であり，$\left\|\hat{\Sigma}^{-1}\hat{Y}^{\mathsf{T}}\hat{Y}\hat{\Sigma}^{-1} - I\right\|_{\max}$ の値の内，すべての巡回における最大値を index ごとに示す．3 つ目の値は片側ブロックヤコビ法全体の誤差であり，計算によって求められた行列 A の左特異ベクトル $\hat{U}$ と右特異ベクトル $\hat{V}$, 特異値 $\hat{\Sigma}$ とおくとき，$\left\|A - \hat{U}\hat{\Sigma}\hat{V}^{\mathsf{T}}\right\|_{\max}$ の値を index ごとに示す．また，4 つ目の値は $\hat{V}$ の直交性であり，$\left\|\hat{V}^{\mathsf{T}}\hat{V} - I\right\|_{\max}$ の値を index ごとに示す．$\hat{U}$ の直交性については，片側ブロックヤコビ法が正常に終了した場合には小さな値となるため，省略した．実際に，実験においてはすべての組み合わせにおいて失敗は発生しなかった．

はじめに，非並列向けプログラムにおける $\kappa_1(\hat{R}')$ と $\kappa_1(D_{\mathrm{row}}\hat{R})$ の推定値をそれぞれ表4.1, 表4.2に示す．また同様に分散並列向けプログラムにおける $\kappa_1(\hat{R}')$ と $\kappa_1(D_{\mathrm{row}}\hat{R})$ の推定値をそれぞれ表4.3と表4.4に示す．テスト行列の中には条件数がとても大きなものも含まれているが，すべての表の中での最大値は 107.607 となっており，片側ブロックヤコビ法の中で出現する行・列スケーリングした条件数がとても小さな値となっていることがわかる．また，列スケーリングした条件数は行スケーリングした条件数と比べて 2 倍程度大きな値となっていることがわかる．どの mode を用いた場合でも条件

表4.3　分散並列向けプログラムにおける DTRCON によって推定した $\kappa_1(\hat{R}')$ の mode ごとの最大値．N/A の意味は表4.1と同じ．

巡回回数	mode = 1	mode = 2	mode = 3	mode = 4	mode = 5
1	6.345	1.001	107.607	67.372	105.948
2	1.344	N/A	7.758	10.983	8.417
3	1.027	N/A	1.424	2.091	1.580
4	1.003	N/A	1.003	1.118	1.003

表4.4　分散並列向けプログラムにおける DTRCON によって推定した $\kappa_1(D_{\mathrm{row}}\hat{R})$ の mode ごとの最大値．N/A の意味は表4.1と同じ．

巡回回数	mode = 1	mode = 2	mode = 3	mode = 4	mode = 5
1	5.579	1.000	40.841	40.343	41.980
2	1.291	N/A	5.043	8.268	5.189
3	1.025	N/A	1.208	1.781	1.223
4	1.003	N/A	1.001	1.115	1.003

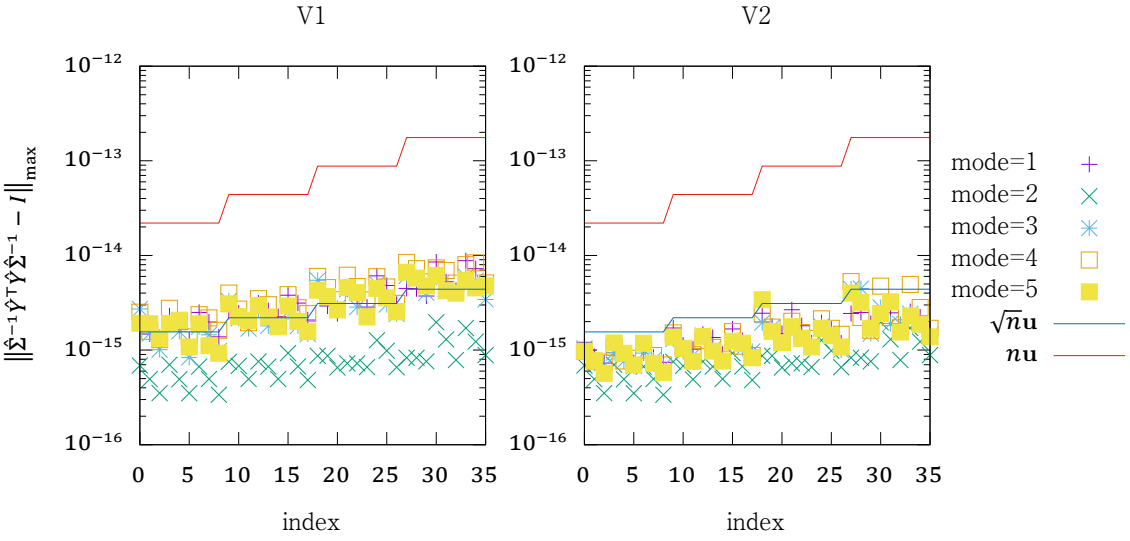

図4.2　非並列向けプログラムにおける $\hat{Y}$ の直交性．参考のため，内積の誤差の上限とほぼ等しい $n\mathbf{u}$ と内積の誤差の平均的な値となる $\sqrt{n}\mathbf{u}$ も示す．左図が V1 手法，右図が V2 手法．

数は巡回に従って急速に 1 に近づいており，2 巡回目で最大 10 程度，3 巡回目で最大 2 程度となっている．特異値分布によっても条件数は大きく変わっており，mode = 1 や mode = 2 のときに小さな値となる傾向があり，とくに mode = 2 は 2 巡回目にはすべての列ブロックの直交化をスキップしている．非並列向けプログラムにおける条件数は分散並列向けプログラムにおけるものよりも総じて小さい傾向があるが，前処理の違いの結果が表れているものと推察される．

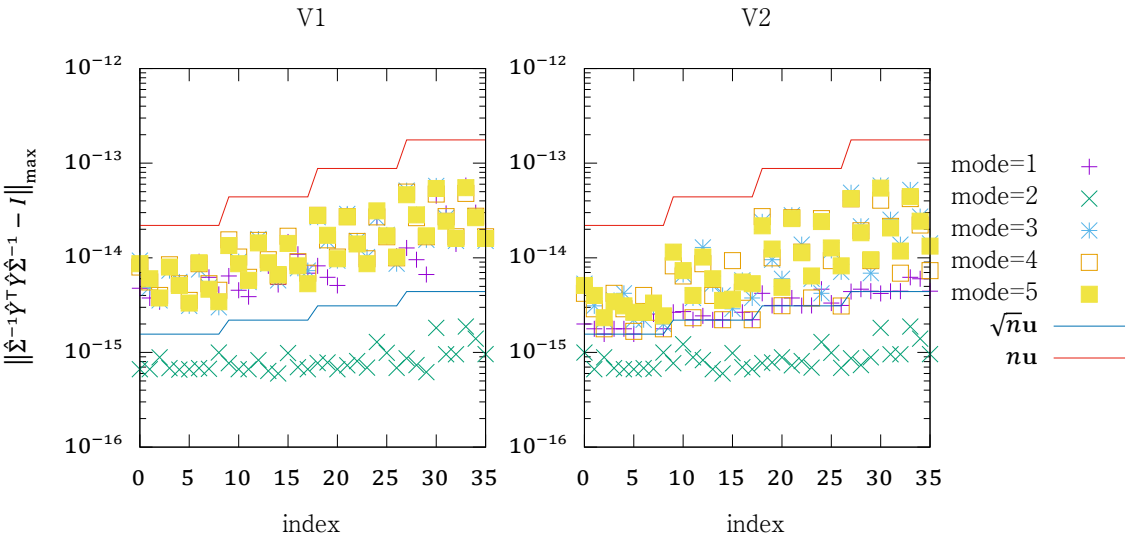

図4.3　分散並列向けプログラムにおける $\hat{Y}$ の直交性．左図が V1 手法，右図が V2 手法．

次に，$\hat{Y}$ の直交性を示す．ここでは，非並列向け，分散並列向けそれぞれのプログラムにおいて，Hari の V1 手法と V2 手法とで比較を行う．図4.2は非並列向けプログラム，図4.3は分散並列向けプログラムにおける $\hat{Y}$ の直交性である．非並列向けにおける $\hat{Y}$ の直交性を見ると，mode = 2 を除き，V2 のものが総じて V1 のものよりも小さな誤差となっている．しかしどちらも $\sqrt{n}\mathbf{u}$ と同程度の小さな値となっているため，高精度に直交化できていると考えられる．また，V2 における $\hat{Y}$ の直交性の上限は $\kappa_2^2(X')$ に比例する値となっていたが，実際の計算ではその影響がほとんど見られない．一方，分散並列向けの実装では V1 と V2 とでほぼ同程度の誤差となっていることがわかる．例外的に mode = 1 の場合は V2 の方が誤差が小さい．また，とくに mode = 5 のときがわかりやすいが，誤差が周期的な変化をしていることがわかる．変化の周期を考えると，誤差が κ に依存して大きく変化していることがわかる．

次に，片側ブロックヤコビ法の残差を示す．図4.4と図4.5 はそれぞれ，非並列向けプログラムと分散並列向けプログラムにおける残差である．残差については V1, V2 手法でほとんど違いがみられない．ただし非並列向けに対する mode = 4 のものだけ V1 よりも V2 で大きな値となっている．非並列向けと分散並列向けを比べると，後者の方が誤差が大きいものが多い．しかし，$n\mathbf{u}$ と同程度，または小さい程度であるので，十分高精度であると言える．

次に，片側ブロックヤコビ法の $\hat{V}$ の直交性を示す．図4.4と図4.5 はそれぞれ，非並列向けプログラムと分散並列向けプログラムにおける $\hat{V}$ の直交性である．この誤差についても，V1 と V2, 非並列向けと分散並列向けとで値が大きく異なり，V1 よりも V2, 非並列向けよりも分散並列向けの方が大きなものとなっている．しかし，もっとも誤差が大きいものが多い，分散並列向けの V2 手法においても誤差が $n\mathbf{u}$ 程度となっており，十分高精度に計算できていることがわかる．

最後に，V2 から V1 への切り替えの頻度についての述べる．非並列向けプログラムにおいては V2 から V1 への切り替えは最大でも 3 回であり，3 回の切り替えが発生したものは 900 の行列の内 1 つだけで，2 回の切り替えが発生したものは 6 つ，まったく切り替えが発生しなかったものは 868 個であった．また，分散並列向けプログラムにおいては，473 個の行列で切り替えがまったく発生しなか

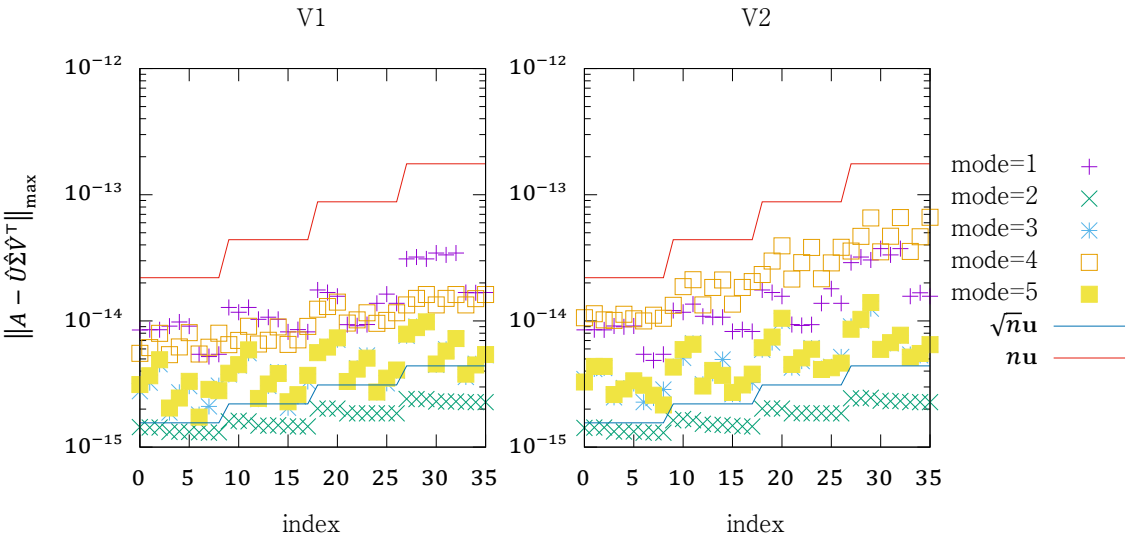

図4.4　非並列向けプログラムにおける片側ブロックヤコビ法の残差．左図が V1 手法，右図が V2 手法．

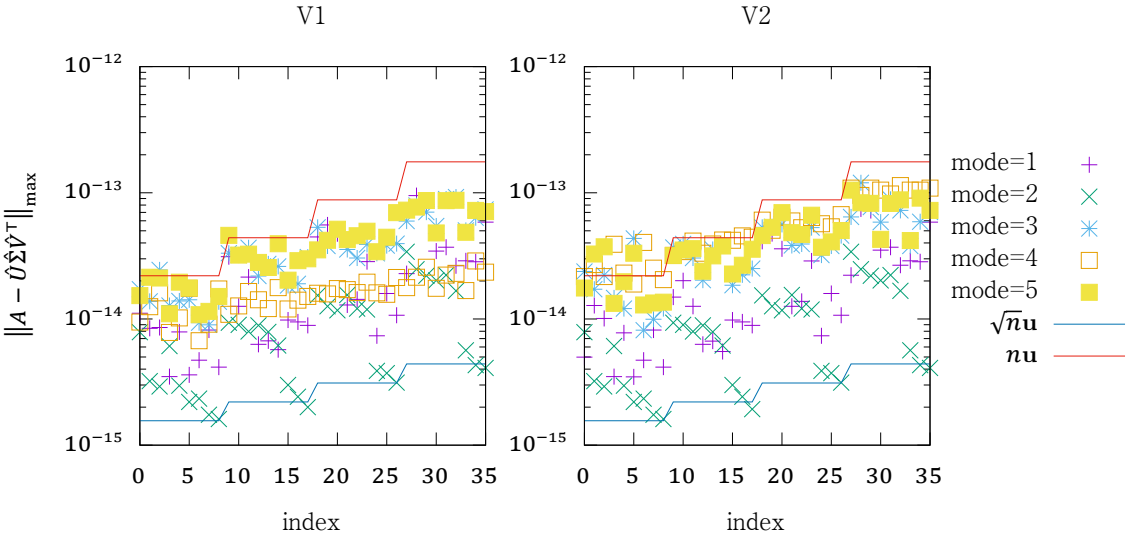

図4.5　分散並列向けプログラムにおける片側ブロックヤコビ法の残差．左図が V1 手法，右図が V2 手法．

った．すべての並列ステップの内，1 つでも切り替えを含む並列ステップの割合が 10% 以下のものが 892 個であり，この割合の最も大きなものであっても 12.5% であった．そこで，切り替えによる性能への影響は小さいだろうと推測されるが，分散並列向けにおいては切り替えが少ないながらも多くの行列で発生するため，切り替えを行うべきであると考えられる．そこで 5 章における実装においては，V2 から V1 への切り替えを行っている．

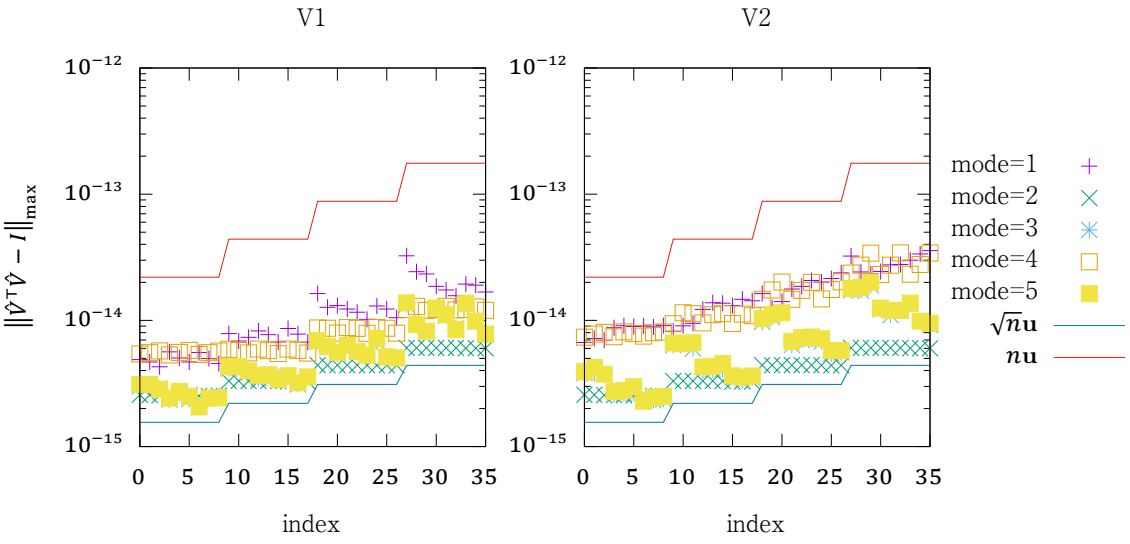

図4.6　非並列向けプログラムにおける片側ブロックヤコビ法の $\hat{V}$ の直交性．左図が V1 手法，右図が V2 手法．

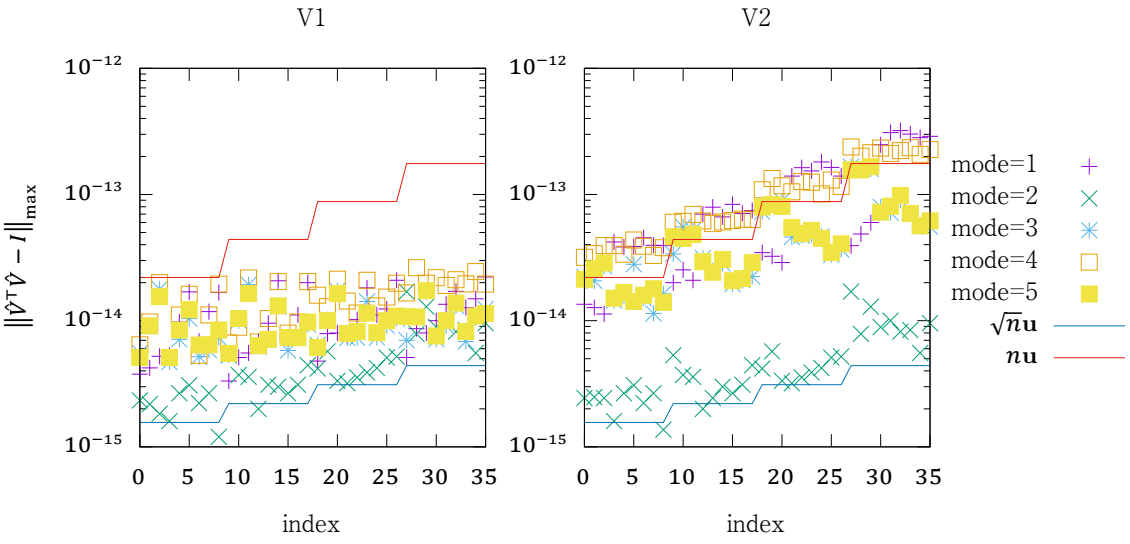

図4.7　分散並列向けプログラムにおける片側ブロックヤコビ法の $\hat{V}$ の直交性．左図が V1 手法，右図が V2 手法．

第 5 章

高性能実装手法

この章では片側ヤコビ法の高性能計算機に適した実装技術について述べる．これには主に 3 つの内容が含まれている．第一に，1 ノード内での片側ヤコビ法の実装技術についてであり，多数のコアを持つ CPU に向けて共有メモリ並列化手法について検討する．ここで用いた手法はいくつかの既存研究のものを組み合わせたものであるが，実装自体については独自のものである．またこの実装は，分散メモリ並列化したプログラム内部で呼び出されるものである．第二に，分散データ並列化においてとくに重要となるデータ分散手法についてであり，3 つのデータ分散を比較し，とくに並列度が高い場合に優れていると考えられる二次元ブロック分割について，その実装手法を議論する．この手法は関連論文 [b,c] で用いた Bečka らの Variable Blocking について，4 章において示した Hari の V2 手法に適するように拡張したものである．第三に，通信回数，演算量をともに削減する可能性を持つ Bečka の動的順序について，最近行った理論検証の結果を示し，また，実装時の問題とその解決手法について述べる．ここでの内容には関連論文 [b,c] において示した演算性能向上手法とともに，Bečka の動的順序における一次収束性のより良い上限や，二次元ブロック分割を用いたときの性能検証が含まれる．

5.1　1 ノード内での片側ヤコビ法の実装技術

分散並列向けの片側ブロックヤコビ法の実装の内部で，1 ノード上において小行列の SVD を計算するために片側ヤコビ法の実装が必要である．そこで，小行列の SVD も無視できない程度の実行時間となるため，高速化の必要がある．小行列の SVD に対しても片側ブロックヤコビ法を使うことが 1 つの手ではあるが，4 章の誤差解析ではこのような再帰的な利用を想定していないため，ここでは片側ヤコビ法の既存の実装をマルチコア CPU 向けに対応させることで高速化する．

片側ヤコビ法の実装は前出の LAPACK における Drmač らの実装があり，前処理付きの片側ヤコビ法を行うものが xGEJSV，その中で呼ばれる片側ヤコビ法単体のものが xGESVJ となっている．高速な前処理実装にも意味があるが，ここで考えるものは，すでに前処理済みの行列の部分行列に対する片側ヤコビ法であり，十分反復回数が少ないと考えられるため，前処理を使わない片側ヤコビ法単体のもの，xGESVJ を取り扱う．

xGESVJ は LAPACK に採用された，信頼のおける実装であるため，xGESVJ の共有メモリ並列化においては，並列化のためどうしても実装不可能なものを除いてはそこに用いられているテクニックと同等のものを実現し，性能のみ改善することを目標とした．本研究での実装は，xGESVJ の機能の内，次のものは同等に実現した:

- Fast Plane Rotation を使った演算量削減
- 三角構造を用いた演算量削減
- Block Oriented 順序によるキャッシュ再利用性の向上

次のものについては似たような効果を持つ別のものに置き換えた:

- Rijk の消去順序

一方，次のものについては並列化のため実現できていない：

- 対角に近い要素を優先的に消去する動的順序

その上で，ブロック要素単位に BOMO を適用することで，スレッド並列化を実現し，高速化を実現した．この実装は Modulus 順序を並列化のために用い，さらにブロック内部の順序をブロック化列巡回順序とすることでキャッシュ再利用性を向上させるという点においては Singer ら [109] のものに近い．しかし本研究の実装は，xGESVJ との置き換えを狙い，xGESVJ の多くの機能について同等のものか並列化に適した代替手法を用意したという点で，完成度の高いものとなっている．そこでこの節では，本研究における手法を説明し，また，実験によって xGESVJ と反復回数や計算速度を比較する．

5.1.1 xGESVJ の構成

xGESVJ のプログラムの構成自体は十分に練られたものにはなっていない．xGESVJ は内部的に 2 つの大きなサブルーチン xGSVJ0 と xGSVJ1 を呼び出し，この 3 つのサブルーチンの合計で 3,476 行 (LAPACK 3.7.1) となっている．以下は xGESVJ の一部 (875 行目から 890 行目まで) を切り出してきたものである：

```
875         IF( AAQQ.GE.ONE ) THEN
876            ROTOK = ( SMALL*AAPP ).LE.AAQQ
877            IF( AAPP.LT.( BIG / AAQQ ) ) THEN
878               AAPQ = ( DDOT( M, A( 1, p ), 1, A( 1,
879     $                                 q ), 1 )*WORK( p )*WORK( q ) /
880     $                                 AAQQ ) / AAPP
881            ELSE
882               CALL DCOPY( M, A( 1, p ), 1,
883     $                                    WORK( N+1 ), 1 )
884               CALL DLASCL( 'G', 0, 0, AAPP,
885     $                                     WORK( p ), M, 1,
886     $                                     WORK( N+1 ), LDA, IERR )
887               AAPQ = DDOT( M, WORK( N+1 ), 1,
888     $                                 A( 1, q ), 1 )*WORK( q ) / AAQQ
889            END IF
890         ELSE
```

これは 2 つの列ベクトルの内積を計算する部分であり，オーバーフローやアンダーフローに対処するためスケーリングを行うか否かで処理が分かれる．さらに`AAQQ.LT.ONE` の場合のコードもほぼ似たようなものとなり，この 1891 行目以降に続いている．また以下は同じ xGESVJ の別の部分 (1178 行目から 1197 行目まで) を取り出してきたものである：

```
1178      IF( AAQQ.GE.ONE ) THEN
1179         IF( AAPP.GE.AAQQ ) THEN
1180            ROTOK = ( SMALL*AAPP ).LE.AAQQ
1181         ELSE
1182            ROTOK = ( SMALL*AAQQ ).LE.AAPP
1183         END IF
```

```
1184          IF( AAPP.LT.( BIG / AAQQ ) ) THEN
1185             AAPQ = ( DDOT( M, A( 1, p ), 1, A( 1,
1186     $                                  q ), 1 )*WORK( p )*WORK( q ) /
1187     $                                  AAQQ ) / AAPP
1188          ELSE
1189             CALL DCOPY( M, A( 1, p ), 1,
1190     $                                     WORK( N+1 ), 1 )
1191             CALL DLASCL( 'G', 0, 0, AAPP,
1192     $                                      WORK( p ), M, 1,
1193     $                                      WORK( N+1 ), LDA, IERR )
1194             AAPQ = DDOT( M, WORK( N+1 ), 1,
1195     $                                  A( 1, q ), 1 )*WORK( q ) / AAQQ
1196          END IF
1197       ELSE
```

このコードは部分的な違いはあるがほぼ前出のものと同じものとなっている．さらに同様のコードが xGSVJ0 の中に 2 回 (429 行目から 439 行目までと 722 行目から 739 行目まで)，xGSVJ1 の中に 1 回 (421 行目から 437 行目まで)，出現する．このように，冗長な部分が多いコードとなっている．

xGESVJ と xGSVJ0，xGSVJ1 はそれぞれ，下記の三角構造利用と Block Oriented 順序のため，ペアの生成順序が異なる片側ヤコビ法の実装を持っており，5 つのペア生成ループが存在することになる．そこでループごとに列ベクトルの直交化を行うコードを記述している．列ベクトルの直交化の計算がごく単純なものであるならばこのような構成も間違いではないが，3 章の通り，xGESVJ は Fast Plane Rotation を用いた演算量の削減や，さらにオーバーフロー，アンダーフローを防ぐためのスケーリング処理を行っているため，約 230 行にも及び，単純ではない．この結果として片側ヤコビ法の実装だけで 3,000 行を超えるものとなってしまっている．これは片側ヤコビ法の高性能化において大きな障害となる．

当然，列ベクトルの直交化は列ベクトルの直交化として 1 つのサブルーチンとして取り出すべきである．すなわち，xGESVJ が 5 回繰り返して記述している部分，3 章の図 3.1のプログラムにおいては，内側ループの内部の処理に相当する部分を 1 つのサブルーチンとして切り出す．本研究におけるサブルーチンの実装は，C 言語を用いてテストコードも含めて 287 行となり，Drmač らの実装より行数は多くなったが，再利用性が高く，Drmač らの実装で 5 回登場するものが 1 つにまとめることができるため，実質的にコード量が 5 分の 1 となっている．このサブルーチンの名前を`orthcols`とし，以下の擬似プログラムでも利用する．

5.1.2 消去順序

Rijk の順序と列ソーティング

Rijk の消去順序 [110, Section 2.4] は部分的に動的なペアの選択を行うことで収束を高めることを目的としている．Rijk の消去順序は行巡回順序を基にしている．行巡回順序は次のように考えることができる．始めに，行を選択し，その行の要素を選択する．例えば，第一に 1 行目を選択し，その中の要素 $(1,2)$ $(1,3)$ $(1,4)$ … を選択していく．次に 2 行目を選択し，その中の要素 $(2,3)$ $(2,4)$ $(2,5)$ … を選択していく．ヤコビ法における “消去する要素” は片側ヤコビ法においては “直交化する列ベクトルのペア” となっている．そこで，行巡回順序は，1 つ列ベクトル，ピボット列を選択し，もう一方の列ベクトルを次々の選択していく順序だと言える．Rijk の消去順序では，ピボット列の選択時にノルム最大の列の置き換えを行う．つまり，行巡回順序で 1 列目を選択するときに，$1,\ldots,n$ 列目の中から最もノルムが大きいものを選び，1 列目とノルム最大の列を置き換えて，残りは同じように進行する．2 列目を選択するときは同様に $2,3,\ldots,n$ 列目の中からノルム最大のものと置き換えて，残りは同じよう

に進行する．これは選択ソートを消去順序に組み込んだものだと言える．選択ソートははじめに，1 から n 要素目までの要素の中らか最大のものを探索し，第 1 要素と置き換え，次は 2 から n 要素目までの要素の中から最大のものを探索し，第 2 要素と置き換える，といった動作を繰り返す．そこで Rijk の消去順序は，Givens 回転の角度 θ が常に 0 であれば，列ベクトルをノルム非昇順に並び替えることができる．θ が 0 でなければ列のノルムが変化しながら進行するため，完全にソートすることは不可能だが，反復が進行し，列のノルムの変化が無視できる程度になれば，選択ソートと同等のことが発生する．

Rijk は彼の消去順序の利点を，古典的ヤコビ法との関係で解説している．つまり，ヤコビ法における非対角要素は $c_{i,j} = a_i^\top a_j = \beta_{i,j} \|a_i\|_2 \|a_j\|_2$ と書ける．ただし $\beta_{i,j}$ は a_i と a_j の余弦角度だとする．そこで a_i がノルム最大の列ベクトルだとすると，非対角要素の値も大きい傾向にあり，a_i とのペアから順に消去していくことは，絶対値の大きな非対角要素を優先的に消去することにつながる．Mascarenhas [111] は 2.41 次収束のための消去順序で有名であるが，ソーティング自体が収束に与える影響についても調べており，事前に行列のソートを行うだけでも収束性を改善する結果を示している．

Rijk の消去順序は動的な選択が行われるためそのままの形では並列化が難しい．そこで，共有メモリ並列化においては，巡回毎に消去を始める前に列ベクトルをノルム非昇順に並び替える方法を採用する．これは Rijk の消去順序において消去の中にソーティングが組み込まれているという巧妙さと比べると，ソーティングを消去では別に用意しなければならず，冗長となるが，並列化には適している．ヤコビ法の収束を考えれば，巡回が進行すると列のノルムの変化が小さくなり，列の入れ替えは少なくて済むだろうと考えられる．そこで，ほとんど整列済みの列に対して効率的なソーティングアルゴリズムを用いることが適切である．本研究では，Python や Java のソートアルゴリズムとして使われている TimSort を用いている．TimSort については Auger らの論文 [112] が詳しいが，ほぼ整列した数列に対して改良された Merge Sort の一種である．また，ノルムのソーティングと列ベクトルの並び替えを別に行うことで，列ベクトルの移動を最小化することが適切だと考えられる．

並列化とキャッシュ利用

xGESVJ においては Rijk の順序に Block Oriented 順序（ブロック要素単位の行巡回順序とブロック内部の行巡回順序の組み合わせ）を用いることでキャッシュ再利用性を高めている [3, Algorithm 1]. 本研究の実装では Rijk の動的選択の効果をソーティングによって実現できるものとし，並列化のため BOMO を用い，キャッシュ再利用性のため，ブロック内部ではブロック化列巡回順序を用いる．そこで 2 種類のブロック幅が必要となる．1 つは，BOMO 用のブロック幅であり，これは BOMO の並列数がスレッド数と一致するように設定する．また，もう 1 つのブロック幅はブロック内部のブロック化列巡回順序のブロック幅であり，行列サイズとキャッシュサイズに依存して決定する．

三角構造の利用

xGESVJ は入力行列が三角行列である場合，ブロック三角要素単位の非零構造を利用して演算量を削減する．また，非零構造のため演算量が少ない部分行列に対して複数回，消去を行うことで，演算量を抑えつつ収束をより進める．ここでは Hari の V2 手法を用いたときに出現する上三角行列の場合における xGESVJ の動きを説明する．

xGESVJ ではまず入力行列 A を 4 つの列ブロックに分割する．A は上三角であるため，

$$A = \left[\begin{array}{c|c|c|c} A_1 & A_2 & A_3 & A_4 \\ \hline \mathbb{0} & \begin{array}{c|c} \multicolumn{2}{c}{} \\ \hline \mathbb{0} & \end{array} & \begin{array}{c|c} \multicolumn{2}{c}{} \\ \hline \mathbb{0} & \end{array} & \end{array}\right] \tag{5.1}$$

という形を持つ．すべての列ブロックが同じ幅を持つならば，A_1 は A の 4 分の 1 の行数となるため，A_1 に対する片側ヤコビ法は他の部分行列よりも小さなコストで行うことができる．そこで，A_1 に対しては 2 巡回の片側ヤコビ法を適用し，次に A_2，そして A_1 と A_2 のペア，最後に A_3 に対して片側ヤコビ法を適用する．これを消去順序として書くと $(1,1)(1,1)(2,2)(1,2)(3,3)$ となる．これは Block Oriented 順序のブロック要素単位の順序であるため，対角ブロックを選択するものも含まれる．また，すべてのブロック要素を選択するものではないため巡回順序でも擬似巡回順序でもない．これはあくまですぐに破壊される三角構造を利用して小さな演算量で消去を行うことを目的としているためであり，収束するまでの反復は後の巡回に任せるためである．

三角構造の利用における並列化は，それぞれの A_1, A_2, A_3 に対する片側ヤコビ法を BOMO を用いて行うことで実現できる．A_1, A_2 のペアに対する並列化では非対角ブロックの消去であるため，長方領域が出現するが，Modulus 順序から A_1 のみや A_2 のみに対するペアを除いた順序を用いて並列化する．また，全体の並列順序と同じように，再帰的にブロック化列巡回順序と組み合わせることでキャッシュ再利用性を高める．

プログラムの概要

以上の機能を組み込んだ擬似プログラムが図 5.1である．この擬似プログラムでは簡単のため，擬似プログラム中に現れる割り算がすべて割り切れるものとしている．また，冗長となるため，三角構造を利用する部分については省いているが，この擬似プログラムを再帰的に呼び出すことで実現可能である．擬似プログラム中の `nb1` は内側のブロック化列巡回順序のためのブロック幅であり，定数 `CASHSIZE` によって決める．最外側の `do` ループは巡回毎のループであり，収束判定と A の列ベクトルのノルム非昇順によるソートを行う．ただし列ベクトルのソートでは A と V を同じように並び替えなければ A と V が対応しなくなるため，`sort-columns` は A と V の両方を引数とし，A の並び替え順序によって V を並び替える．`CASHSIZE` は CPU のコアごとのキャッシュサイズを基に設定する値であるが，実装においては手作業で探索した値を用いており，実装環境において行列サイズ $m = n = 3,000$ 付近で最も高速となった `CASHSIZE` $= 24,000$ に設定している．この擬似プログラムは OpenMP によって並列化することを前提とし，`omp-parallel-do` によって，CPU のコア数と同じ数だけスレッドを立ち上げる．そして `omp_num_threads` と `omp_thread_num` によってスレッド数とスレッド番号を取得する．その内側の `r` に関するループは並列ステップ数であり，`modulus-pair` によって対応するペアを生成する．また `modulus-pair` が“あまりのペア”を生成した場合は `flag` が `true` を返し，そうでない場合は `false` を返すため，`flag` の値によって，対角ブロックを選ぶか非対角ブロックを選ぶかを選択する．`if` 文の内側は 3 重ループとなっており，プログラムの最初に決定した `nb1` の幅のブロック化列巡回順序を行うようになっている．

```
subroutine MTOSJ(A = [a_1  a_2  ⋯  a_n], tol):
        nb1 = CASHSIZE / n
        V  =  [v_1  v_2  ⋯  v_n]  =  I_{n,n}
        do
                maxt = 0
                sort-columns(A, V)
                omp-parellel-do
                        id = omp_thread_num()
                        nt = omp_num_threads()
                        nb = n/nt
                        for r=1:2*nt
                                (pp,qq,flag) = modulus-pair(2*nt, id, r)
                                if flag == false:
                                        for p2=nb*(pp-1)+1:nb*pp:nb1
                                                for q=nb*(qq-1):nb*qq:
                                                        for p=p2:p2+nb1-1
                                                                t = orthcols(a_p, a_q, v_p, v_q)
                                                                maxt = max(maxt, t)
                                                        end
                                                end
                                        end
                                else if flag == true:
                                        for p2=nb*(pp-1)+1:nb*pp:nb1
                                                for q=p2+1:nb*pp:
                                                        for p=p2:q
                                                                t = orthcols(a_p, a_q, v_p, v_q)
                                                                maxt = max(maxt, t)
                                                        end
                                                end
                                        end
                                        for p2=nb*(qq-1)+1:nb*qq:nb1
                                                for q=p2+1:nb*qq:
                                                        for p=p2:q
                                                                t = orthcols(a_p, a_q, v_p, v_q)
                                                                maxt = max(maxt, t)
                                                        end
                                                end
                                        end
                        end
                end
        while maxt > tol
        for i = 1, n
                σ_i  =  ‖a_i‖_2
        end
        Σ  =  diag(σ_1, σ_2, …, σ_n)
        U  =  AΣ^{-1}
        return (U, Σ, V, A)
```

図5.1　共有メモリ並列用の片側ヤコビ法の擬似プログラム

5.1.3　マルチコア CPU を用いた性能検証

この節の最後に，xGESVJ と本研究での実装 MTOSJ の性能を比較する．ここでは 2 つの異なるテスト結果を示す．1 つは並列化のために変更した消去順序や列ソーティングによる収束性への影響を調べるため，4 章と同様に多くのテスト行列に対して，巡回回数や演算量を調べるテストである．ここでは多くの行列に対して調査するため比較的小さな行列を対象とする．もう 1 つは実行時間自体の測定であり，限られた種類ではあるが 1 つ目のテストと比べて大きな行列を用いたテストを行う．

1 つ目のテストで用いた行列は 4 章と同じく，行列サイズ $m = n = 200,\ 400,\ 800,\ 1,600$ と並列化のための列ブロック数 $w = 2\mathrm{nt} = 10,\ 20,\ 40$[*1], 条件数 $\kappa = 10^5,\ 10^{10},\ 10^{15}$, 特異値分布 mode$= 1,\ 2, \ldots, 5$ のすべての組み合わせごとに異なる乱数シードを用いて 5 つの異なるものを作った．また結果は，

[*1] $w = 2\mathrm{nt}$ はスレッド数の 2 倍の値であり，並列化を行っていない xGESVJ にはまったく影響しない

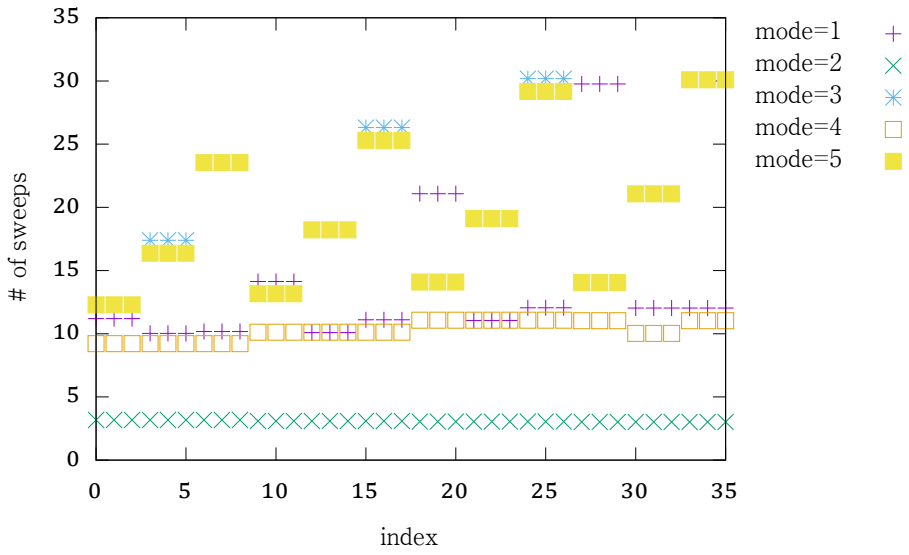

図5.2　xGESVJ の巡回回数．縦軸が巡回回数．横軸が行列番号．

mode ごとに式 (4.91) によって番号付けて整理し，異なる乱数シードを持つものの中で最大の値を持つものを示している．また，巡回回数は選択したペアの個数を $\frac{n(n-1)}{2}$ で割ったものと定義する．このように定義する理由は，xGESVJ が対角に近い要素を複数回消去するため 1 巡回におけるペアの個数が $\frac{n(n-1)}{2}$ の倍数とならないためである．また，規格化演算量 c_{sweep} は次の形で定義する：

$$c_{\mathrm{sweep}} = 3s - 2r_{\mathrm{skip}}s. \tag{5.2}$$

ただしここで s は上で定義した巡回回数であり，r_{skip} は選択したペアの内，列ベクトル同士が十分直交していたため A の更新をスキップしたものの割合である．このとき $n^3 c_{\mathrm{sweep}}$ が V の計算を行わない片側ヤコビ法における演算量の主要項となるため，c_{sweep} はそこから巡回回数とスキップの割合によって決まる係数を抜き出したものとみることができる．

図5.2と図5.3はそれぞれ xGESVJ と MTOSJ の巡回回数である．この図から，巡回回数が特異値分布や条件数に大きく依存していることがわかる．とくに mode = 3 や mode = 5 は相対的に小さな密集特異値を持つような特異値分布であるため，収束が遅く，条件数が大きいときには巡回回数が 30 を超えるものもある[*2]．一方で，mode = 2 の場合の巡回回数はほぼ定数となっている．xGESVJ と MTOSJ を比較すると，MTOSJ の方に小さなばらつきがみられるが，ほぼ同じ巡回回数となっていることがわかる．そこで，このテストにおいて，巡回回数の面では，MTOSJ は xGESVJ の性能を損なわずに並列化できていることがわかる．

図5.4と図5.5はそれぞれ，xGESVJ と MTOSJ の規格化演算量である．規格化演算量は巡回回数に比例した項を持つため，巡回回数のグラフと同様の傾向がみられ，特異値分布や条件数によって大きく値が異なる．しかし規格化演算量は巡回回数とは異なり xGESVJ と MTOSJ とで値が大きく異なるものが多く存在し，最大で約 15，割合にして約 38% の増大となるものがある．これは逆算すると

[*2] 前処理を行うことでこのような行列に対しても小さな巡回回数で収束させることが可能であるが，このテストでは xGESVJ と MTOSJ の比較を目的としているため，前処理の効果については考慮していない．

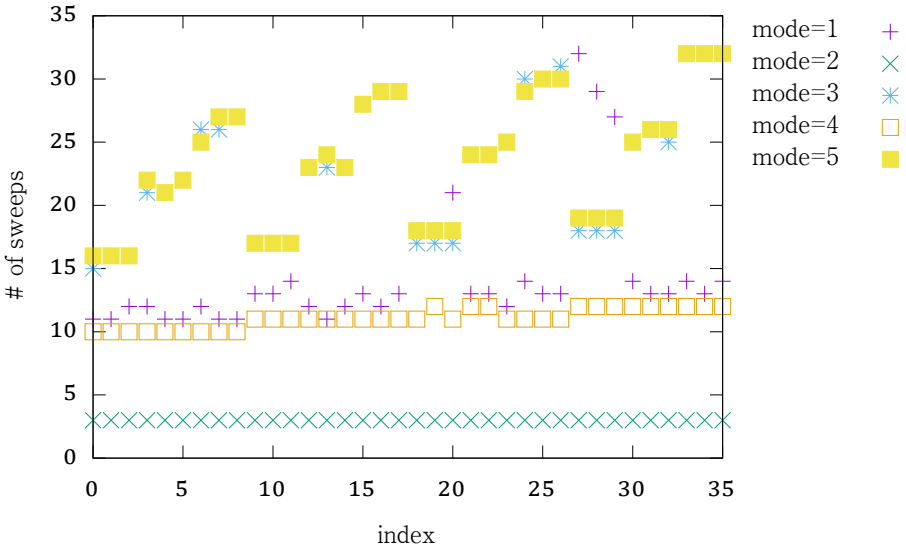

図5.3　MTOSJ の巡回回数．縦軸が巡回回数．横軸が行列番号．

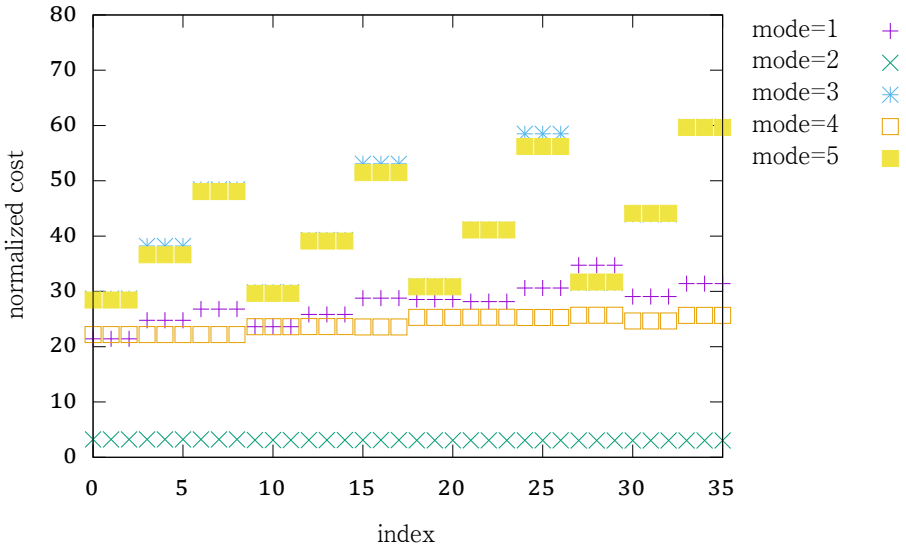

図5.4　xGESVJ の規格化演算量．縦軸が規格化演算量．横軸が行列番号．

MTOSJ の方が $r_{\mathtt{skip}}$ が小さいことを示しており，すなわち，無駄な計算が行われていることを示している．しかしこのオーバーヘッドは 10 割に満たないため，単純計算では 2 並列以上を用いれば単体計算時よりも高速化することが期待される．そこで，MTOSJ は xGESVJ と比べて並列化によって計算の無駄は生まれるが，並列化の加速によって実行時間自体は高速化することが期待される．

2 つ目のテストでは条件数 $\kappa = 10^{10}$ と並列化のための列ブロック数 $w = 2\mathtt{nt} = 20$，そして特異値分布 mode $= 5$ を固定し，行列サイズ $m = n = 512,\ 768,\ 1,152, \ldots,\ 8,760$ と約 1.5 倍ずつ変化させたときの実行時間を示す．測定は異なる 5 つの乱数シードを用いて行い，その平均を示す．テストは 10 コ

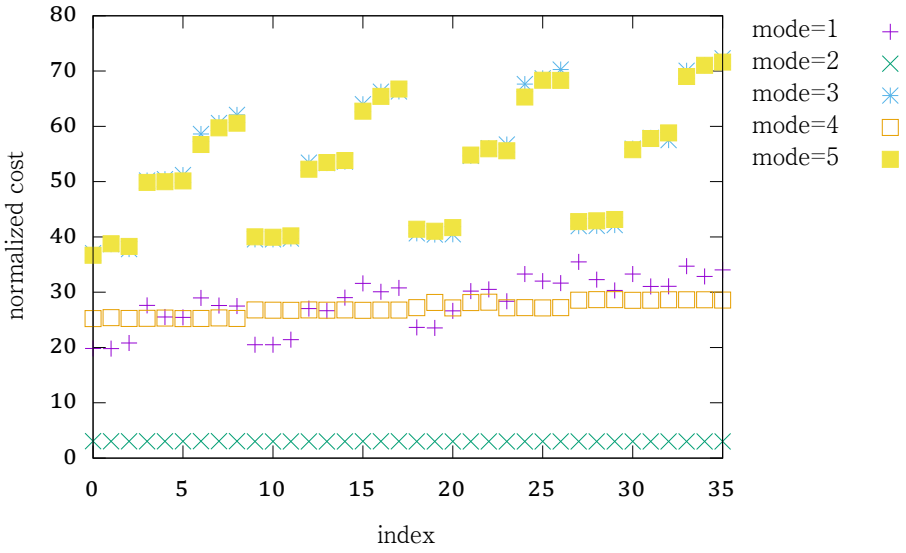

図5.5　MTOSJ の規格化演算量．縦軸が規格化演算量．横軸が行列番号．

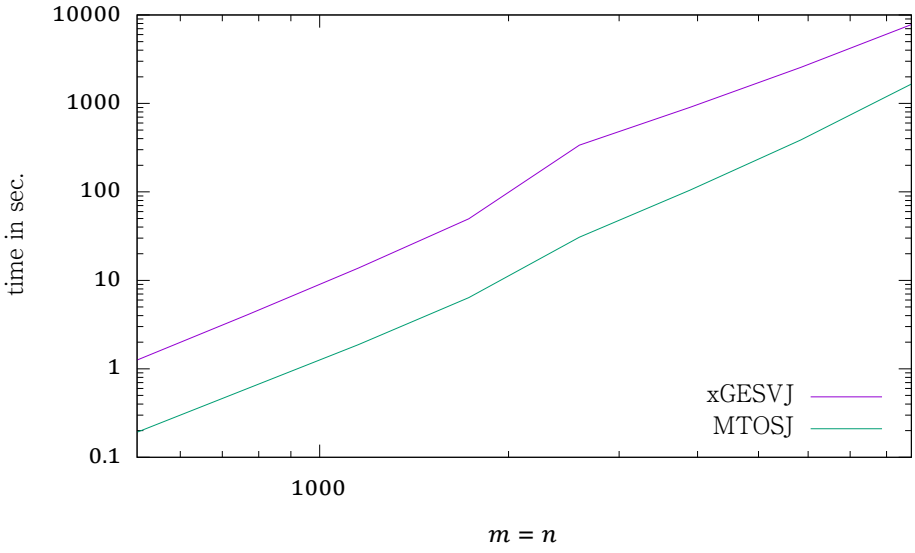

図5.6　xGESVJ と MTOSJ の実行時間の比較

アの CPU Xeon E5-2660 v2 上で行ったため，ちょうどすべてのコアを用いる設定となっている．この CPU は 25MB のキャッシュを持っているため，行列サイズが約 $1,800$ を超えると行列のデータがキャッシュに収まらないようになるため，メモリアクセスの影響が表れる可能性がある．

図 5.6は xGESVJ と MTOSJ の実行時間の結果を示している．MTOSJ は並列化の効果によって xGESVJ よりも高速となっており，行列サイズが大きいところでは約 4.7 倍，行列サイズが小さいところでは約 6.6 倍となっている．その中間のサイズでは性能差はより大きくなっており，$m = n = 2,592$ のときが最大で約 11 倍となっている．これはコア数よりも大きな加速効果となっているが，MTOSJ

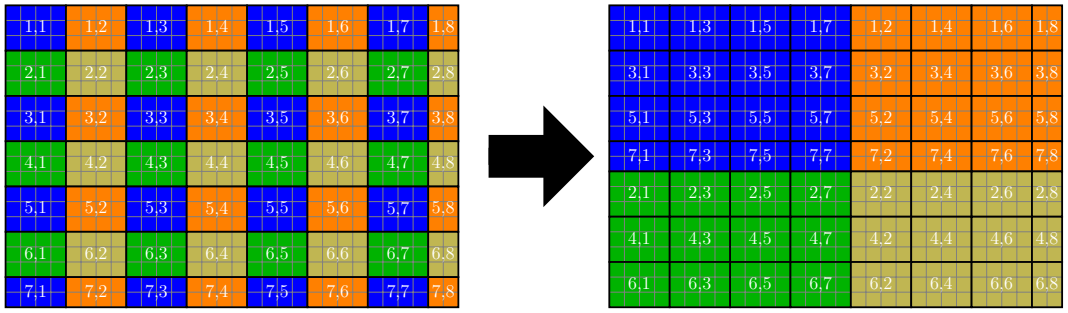

図5.7　二次元ブロックサイクリック分割の例．20×30 行列を 3×4 のブロックで分割し，2×2 ノードへ配置した．左図は元の行列上にそのブロックを配置するノードによって色を付けたもの，右図はノードごとにどのブロックを持つのか番号を振り分けたものである．

において CASHSIZE をこの行列サイズ近辺で最適となるようにチューニングした結果が現れていると考えられる．

以上のように，xGESVJ の並列化を行い，共有メモリ上で高速に動作する片側ヤコビ法の実装を得た．この実装は以下の分散メモリ並列化実装で用いる．

5.2　分散メモリ並列化とデータ分散

現代の高性能計算機のほとんどが分散メモリ並列を行っているため，これに対応することは不可欠であるが，分散メモリ並列においては，データ分散が，通信量，通信回数の両方に大きな影響を与える．行列計算におけるデータ分散手法は，ScaLAPACK [43, 113, 114] における図 5.7のような二次元ブロックサイクリック分割が標準的である．二次元ブロックサイクリック分割は，LU 分解のようなブロック化されたアルゴリズムに適しており，また単純な二次元ブロック分割と比べて，再帰的に行列サイズを縮小していくことで進行するアルゴリズム（LU 分解を含む）に適用した場合において，負荷の不均衡が緩和されるという特徴を持つ．

ScaLAPACK はこのデータ分散手法に対応した行列基本演算ライブラリ PBLAS 上に構築されており，行列に対する代数的な操作を抽象化したインターフェース上で行うことができる点で優れている．また，SVD や QR 分解を含む多くの行列分解を実装しており，機能的である．そこで，QR 前処理・後処理における計算は ScaLAPACK のデータ分散と実装を用いるのが適切であると考えられる．しかし，片側ヤコビ法に対して適切なデータ分散は必ずしも他の行列分解のデータ分散とは一致しない．片側ヤコビ法においてはその内在する並列性を利用するため，データ移動やデータ分散を直に扱う方が容易であり，抽象化によってデータ移動が陽にわからなくなったり，データ構造が固定されて自由に書き換えにくくなったりすると，片側ヤコビ法に適した並列化が不可能になってしまう．

そこで，前処理においては ScaLAPACK のデータ分散と実装を用い，片側ヤコビ法を行う前と行った後にデータ再配置を行うことで，片側ヤコビ法においては別の適切なデータ分散を行うアプローチが Bečka らによって提案されている [115]．ここでは，このアプローチに従い，片側ヤコビ法におけるデータ分散手法のみについて議論する．

この節では，まず基本となる一次元ブロック分割と，別の分割手法である Variable Blocking について述べる．そして，二次元ブロック分割を片側ブロックヤコビ法に適用してかつ，Communication

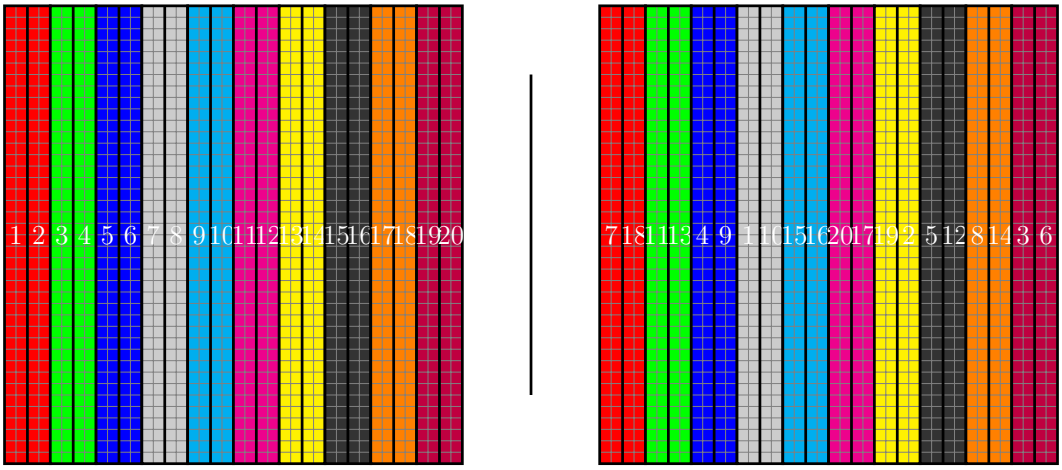

図5.8　一次元ブロック分割の例．ここでは 40×40 行列を列ブロック数 $w = 20$ で分割し，10 ノードへ配置している．色の塗り分けはノードごとに行っている．左図は単純なペア $\{(1,2),(3,4),\ldots\}$ を用いた場合，右図はより一般の複雑な順序のペア $\{(7,8),(11,13),\ldots\}$ を用いた場合である．割り当てが変化した場合，データ移動を行うことでペアを割り当てるノードを変化させる．

Avoiding の観点から議論を発展させる．

5.2.1　一次元ブロック分割

片側ブロックヤコビ法は列ブロックペア単位で計算を行うため，列ブロックペアを 1 つのノードに格納する，1 次元ブロック分割が自然なデータ分散となっている．図 5.8にデータ配置を図示する．

1 次元ブロック分割では，列ブロックペアに対してノードを割り当てる．次の並列ペアを処理する場合には，新たにペアに対するノードの割り当てを計算して，通信によってペアのデータをそのノードに移動する．そこで，通信は 1 ノード当たり 1 度の並列ステップに 1 度ずつ発生するが，列ブロックペアの直交化はノードごとに独立に計算できる．

この分割手法の最大の問題点は列ブロック幅がノード数 p に反比例することであり，列ブロック幅の平均値 $l_{\mathrm{abs}} = \frac{n}{2p}$ となる．1 回の列ブロックペアの直交化の演算量は 1 ノード当たり $O\left(nl_{\mathrm{abs}}^2\right) = O\left(\frac{n^3}{p^2}\right)$ であり，データ量は $O\left(nl_{\mathrm{abs}}\right)$ となるため，$O\left(l_{\mathrm{abs}}\right) = O\left(\frac{n}{p}\right)$ の演算当たり 1 データの移動が発生することになるが，現代における通信の帯域幅は演算性能と比べてずっと狭いため，l_{abs} は十分大きくなければ通信が律速となる．また 1 度の通信当たりの演算量が小さいことは，通信レイテンシーの影響が大きく出るため，好ましくない．

そこで，一次元ブロック分割を適用するためには，ノード数 p を n に対して十分小さくするか，逆に n を p に対して十分大きくしなければならない．前者の場合，高性能計算機の高い性能を活かせず，後者の場合，演算量は $O(n^3)$ で増大するため，p の増加に対して計算時間が急速に増大してしまい，現実的ではない．

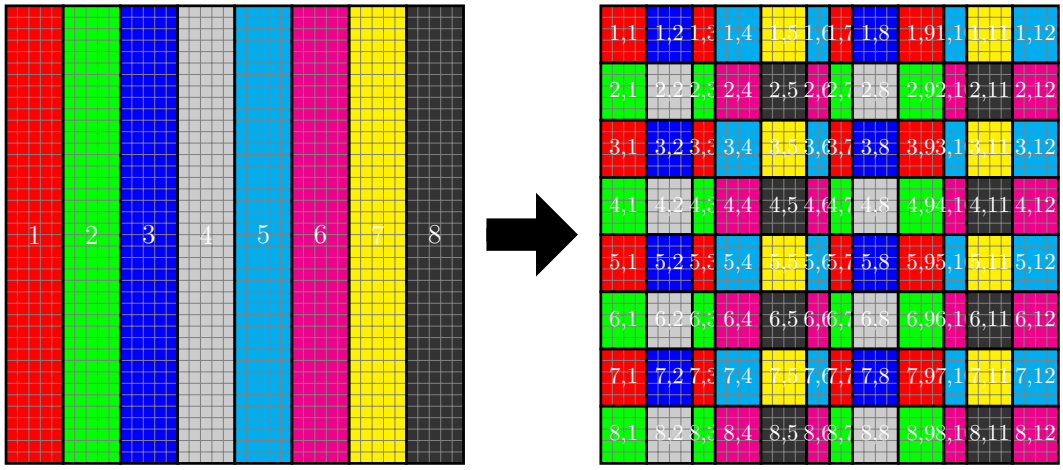

図5.9　Variable Blocking のデータ分散の例．左図は 40×40 行列を $p = 8$ プロセッサで分割したもの．右図は $w = 4, k = 4$ としたとき，並列ペア $\{(1,3)(2,4)\}$ が与えられ，それぞれの列ブロックペア内で 5×4 のブロックで二次元ブロックサイクリック分割したもの．図内の番号付けはブロック分割した部分行列の番号，色の塗分けはそのブロックを保持するノードごとに行っている．

5.2.2　Variable blocking

一次元データ分割の問題点を解決するには，列ブロック数とノード数を一致させないようにしなければならない．すなわち，列ブロックペアの直交化の計算を複数ノードで行えばよい．1 つの列ブロックペアの直交化を行うノード数を k とすると，列ブロック数 $w = \frac{2p}{k}$，平均列ブロック幅 $l_{\mathrm{abs}} = \frac{nk}{2p}$ となるため，1 次元分割のときと同様に計算すると，$O\left(\frac{nk}{p}\right)$ の演算当たり 1 データ移動が発生することになる．しかし，ここでの問題は列ブロックペアの直交化の並列化手法とそのためのデータ分散手法である．単純なアイデアは，列ブロックペアの直交化を ScaLAPACK で実装することであり，ScaLAPACK の二次元ブロックサイクリック分割を用いることである．

Bečka らの Variable Blocking [116, 117] は，ScaLAPACK の二次元ブロックサイクリックデータ分散を利用するために考案されたデータ分散であり，一次元ブロック分割と二次元ブロックサイクリックデータ分割を混合したものとなっている．もともと Variable Blocking はブロック化された Kogbetliantz 法向けに開発されたものであるが，その後に片側ヤコビ法に拡張されている [117]．Variable Blocking では，データ移動を行う単位として一次元ブロック分割を用い，データ移動を行った後，計算を行うときに二次元ブロックサイクリック分割に変換する．しかしながら，彼らの実装では陽的なデータ移動は行わず，仮想的なノードグループを考え，ノードグループに属するノードの置き換えによって実現する．

いま，$w = 4, p = 8$ の場合を考える．すなわち $k = 4$ であり，1 つの列ブロックペアの直交化計算に 4 つのノードを用いる．行列のデータは p ノードに一次元的に分割されており，$A = [\check{A}_1 \check{A}_2 \cdots \check{A}_p]$ について，$\check{A}_j$ を第 j ノードが保持する．これは片側ヤコビ法の列ブロック A_i よりも細かい分割となっており，$A_i = [\check{A}_{ki/2} \check{A}_{ki/2+1} \cdots \check{A}_{k(i+1)/2-1}]$ のように，1 つの列ブロック A_i に対して $k/2$ ノードが持つデータを割り当てる．いま，並列ペアとして $\{(1,3),(2,4)\}$ が選ばれたとき，2 つの列ブロックペア $X_1 = [A_1 A_3], X_2 = [A_2 A_4]$ について列ブロックペアの直交化を行うが，A_1 と A_3 は第 $1, 2, 5, 6$ ノードが

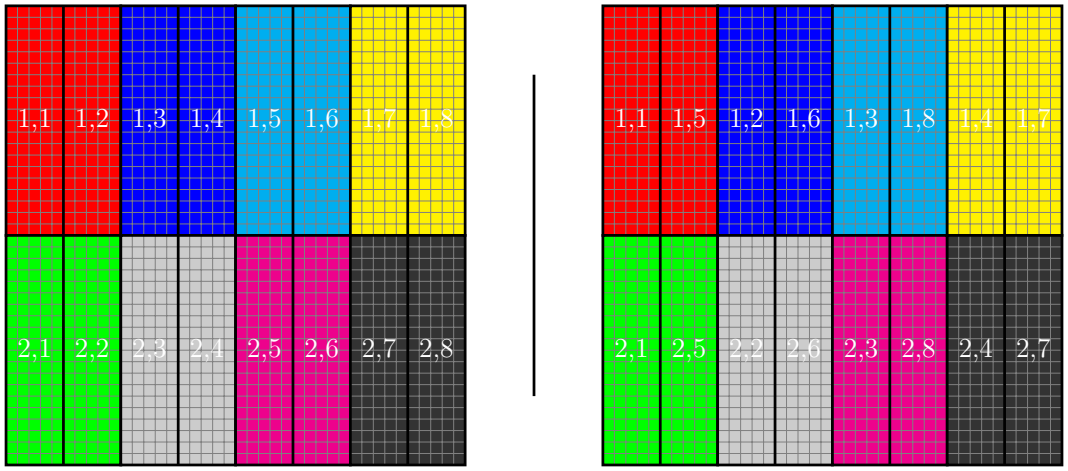

図5.10　二次元ブロック分割の例．左図は 40×40 行列を $p = 8$ のとき $k = 2, w = 8$ を用いた場合のデータ分散を表している．右図は並列ペアを $\{(1,5)(2,6)(3,8)(4,7)\}$ に変更したときのもの．並列ペアが変わった場合，データ移動を行うことで対処する．

持つため，これらを 1 つ目のグループとする．また，A_2 と A_4 は第 $3,4,7,8$ ノードが持ち，これを 2 つ目のグループとする．そして，このグループの中でデータの並び替えを行い二次元ブロックサイクリック分割にすることで，ScaLAPACK を用いることができる．この手順を図5.9に示す．

Variable Blocking によって列ブロックペアの直交化の並列化が可能となり，平均列ブロック幅を減少させることができる．また列ブロックペアの直交化の並列実装においては ScaLAPACK 内のアルゴリズムを用いることができることも利点である．しかし，この手法は通信網における局所性を考慮していないため，この点が欠点になる．先の例においては，第 1 のグループに $1,2,5,6$ が属しており，全結合や二次元メッシュのような通信網ならばよいが，1 次元結合の通信網ならば，$1,2$ 間と $5,6$ 間は直接連結しているが，$1,5$ 間のような組み合わせは直接つながっていない．このようなペアの間で通信を行えば，多ホップ数の通信となり，レイテンシーが増大する．また，第 2 のグループの中での通信と衝突し，バンド幅の減少も発生しやすい．この例ではノード数が少ないため，二次元メッシュ構造の通信網ならば問題を解決できるが，一般に大きな列ブロック数 w とノード数 p を用いた場合，二次元メッシュでは解決できず，全結合に近い通信網が必要となる．

5.2.3　二次元ブロック分割

二次元ブロック分割は 1 次元ブロック分割した行列を横に k 個に分割する，ごく単純な分割である．図5.10に例を示す．

二次元ブロック分割では 1 次元ブロック分割と同様に，データ移動を陽的に行うことで列ブロックペアを k 個のノードが属するノードグループに移動させる．このデータ移動では，縦方向に k 並列で列ブロックが横に移動することになる．一次元結合の通信網の場合は k 並列で動く通信同士が衝突してしまうが，二次元メッシュ構造であれば，縦方向には独立に通信が行われるため，縦同士での通信の衝突は発生しない．依然として，横移動の中では通信の衝突が発生するが，ここでの通信の衝突は一次元ブロック分割と比べて参加するノードが k 分の 1 となるため，緩和される．一方で 1 次元ブロック分割では列ブロックペアの直交化を通信なしに実行できたが，二次元ブロック分割では 1 つの列

```
1  subroutine CholQR([X_1, X_2, …, X_k]):
2          id = MPI_Comm_rank()
3          C = X_id^T X_id
4          R = chol(C) // C = R^T R
5          X = XR^-1
```

図5.11　一次元並列化された Cholesky QR 法の擬似プログラム

ブロックペアの直交化あたり k ノードを用いて計算するため，通信が発生することになる．

また，列ブロックペアの直交化におけるデータ分散も問題である．二次元ブロック分割においては，列ブロックペアの直交化を横に k 個に分割された行列上（すなわち一次元ブロック分割）で行うため，二次元ブロックサイクリック分割を用いる Variable Blocking と比べて，QR 分解のような行列分解を行う際の負荷分散が悪化する可能性がある．ただし，k 個のノードは Variable Blocking と異なり，通信網の構造を考慮して任意に割り当てることができるため，列ブロックペアの直交化内部の通信は Variable Blocking と比べてよいパターンになる可能性がある．

このように，通信の特徴を定性的に見ていけば二次元ブロック分割は決して他の手法と比べて優位ではないが，Cholesky QR 法のような，All-Reduce 型の通信を行う QR 分解と組み合わせるならば，二次元ブロック分割の通信パターンは大きく改善される．この点については次節で示す．

二次元ブロック分割の実装については，EVD 向けのブロックヤコビ法ではいくつか既存研究が存在し，古いものでは，Royo [118] によるものがある．大規模並列向けには高橋 [9] のものや，その実装改善を行った工藤 [10] のものがある．これらにおいて二次元ブロック分割が可能であったのは，EVD 向けのブロックヤコビ法では二次元的な並列性が明白であったからである．つまり，行列積として書けるブロック回転による行列の両側変換が，計算量の大部分を占めるため，それを並列化することは素直なアイデアである．しかし，片側ブロックヤコビ法ではこのような単純な並列性を見出すためには，QR 分解の並列化が必要であり，これは行列積の並列化と比べて複雑となる．そこで，この部分に対して Cholesky QR 法の並列性を利用した実装については新規となる．

5.2.4　二次元ブロック分割の手順

二次元ブロック分割の列ブロックペアごとのデータ配置は，縦方向の一次元ブロック分割に帰着するため，ScaLAPACK に実装された行列分解アルゴリズムでは効率的に動作しない．しかし，列ブロックペアのような縦長行列に対する縦方向の一次元ブロック分割は，Tall and Skinny 行列向けの QR 分解手法として知られるアルゴリズム，例えば Cholesky QR 法や，TSQR (Tall and Skinny QR) には適している．4 章で示した通り，Cholesky QR 法は，一般の行列に対しては不安定だが，前処理付きの片側ヤコビ法と Hari の V2 手法を組み合わせた場合には，問題なく動作する．また，特に縦長行列の場合には，Cholesky QR 法は Householder QR 法の約半分の演算量となり，行列積が中心の Cholesky QR 法は Householder QR 法と比べて高効率である．また，TSQR は Householder QR 法と比べて演算量はほぼ同等である一方，通信回数が減少するため並列計算に向くが，Cholesky QR 法と比べると演算効率は悪い．そこでここでは Hari の V2 手法と二次元ブロック分割とを組み合わせた場合について考える．このときに，Cholesky QR 法の並列化手法を参考にすることは有用である．

図5.11は縦方向に 1 次元ブロック並列化された行列に対する Cholesky QR 法の擬似プログラムである．プログラムは SPMD モデルで作成されており，MPI 並列を行うことを前提としている．ここでは

```
1  subroutine OSBJ-V2([A1,1, A1,2, …, A1,w, A2,1, …, A2,w, …, Ak,1, …, Ak,w], tol):
2          id = MPI_Comm_rank()
3          rowid = id % k
4          colid = id / k
5          do
6                  t = 0
7                  do r=1 to w
8                          [I, J] = modulus-pair(w, colid, r)
9                          X = [Arowid,I, Arowid,J]  // MPI_Send and Recv.
10                         C = X^T X
11                         MPI_AllReduce(C, MPI_SUM)
12                         offd = max_{i<j} |C_{i,j}| / sqrt(C_{i,i} C_{j,j})
13                         t = max(t, offd)
14                         if offd <= tol: continue
15                         R = chol(C)
16                         [U, S] = MTOSJ(R, tol)
17                         [Arowid,I, Arowid,J] = X(R^{-1}US)
18                         MPI_Barrier(MPI_COMM_WORLD)
19                 end
20                 MPI_AllReduce(t, MPI_MAX)
21         while t > tol
```

図5.12　Hari の V2 手法と二次元ブロック分割を用いたときの片側ブロックヤコビ法の擬似プログラム

行列 X を横方向に分割した部分行列 X_i を各 MPI プロセスに分割し，行列積 $X^{\mathsf{T}}X$ と XR^{-1} を並列化する．プログラムの入力は部分行列 X_i であり，MPI のプロセス ID を`MPI_Comm_rank` によって取得することで，行列積 $X^{\mathsf{T}}X$ は次のように分解できるため:

$$C = \sum_{i=1}^{k} X_i^{\mathsf{T}} X_i, \tag{5.3}$$

の各部分行列積 $X_i^{\mathsf{T}}X_i$ を $i = \mathrm{id}$ の MPI プロセスで行い，和を`MPI_AllReduce` によって計算する．一方，C の Cholesky 分解は各ノードで重複して計算する．そのため，通信は`MPI_AllReduce` の中でのみ行われ，Cholesky 分解の中で行われる，細かな通信は排除される．その代わりに，Cholesky 分解の演算は並列化されないため，行列が十分縦長でなければ，ボトルネックになる可能性がある．二次元ブロック分割を行った片側ブロックヤコビ法においてもこの考え方を用い，Hari の V2 手法を用いた列ブロックペアの直交化の内，行列積の部分だけを並列化し，小行列の Cholesky 分解や SVD は各プロセスで重複して行うことで，通信回数を減らすことにする．

図5.12に擬似プログラムを示す．プログラムの引数である $A_{i,j}$ は二次元分割した行列の部分行列を表し，tol は収束判定のための閾値を表す．MPI プロセスの ID は縦方向の並列化数 k によって割られ，これをノードの列 ID, また割った余りをノードの行 ID とする．プログラム内の`modulus-pair` は 3 章で指名した擬似巡回のオーダリングの 1 つである MMO のペアを生成するルーチンであり，w の並列ステップを持つため，内側ループの長さが`w` となっている．

プログラム中の 12,13,14 行目は収束判定のためのコードである．14 行目では，自分の列ブロックペアが十分直交化している場合，以降の処理をスキップする．そこで，各ノードで実用いたのばらつきが生じ，結果として，18 行目の動機の時間の増加につながる．

計算処理の部分はプログラム中の 8 行目から 17 行目だが，そのうち 9, 10, 15 行が Cholesky QR 法に相当する部分であり，それに 16, 17 行を組み合わせると Hari の V2 手法になる．Hari の V2 手法の並列化においては，Cholesky QR 法の並列化の考え方に則り，行列積の部分である 10 行目と 17 行目のみ並列化するため，通信は主に`MPI_AllReduce` の中でのみ行われる．そのため，列ブロックペア

```
subroutine OSBJ-DO([A_{1,1}, A_{1,2}, …, A_{1,w}, A_{2,1}, …, A_{2,w}, …, A_{k,1}, …, A_{k,w}], tol):
        id = MPI_Comm_rank()
        rowid = id % k
        colid = id / k
        do
                [I, J, offd] = dynamic-ordering(w, colid, r)
                if offd <= tol: break
                X = [A_{rowid,I}, A_{rowid,J}]  // MPI_Send and Recv.
                C = X^T X
                MPI_AllReduce(C, MPI_SUM)
                R = chol(C)
                [U, S] = MTOSJ(R, tol)
                [A_{rowid,I}, A_{rowid,J}] = X(R^{-1}US)
                MPI_Barrier(MPI_COMM_WORLD)
        end
```

図5.13　Hari の V2 手法と二次元ブロック分割を用い，さらに動的順序を組み合わせたときのの片側ブロックヤコビ法の擬似プログラム．

の直交化の内部での通信回数が少なく抑えられる．

片側ヤコビ法としての通信部分の詳細は行列 X に部分行列を格納する部分であるが，詳細はここでは省略している．実際の実装においては，次にほしい部分行列がどのノードにあるのか，今持っている部分行列をどのノードに送るのかの 2 種類の情報を，ペアの中の 2 つの部分行列それぞれに対して必要であり，その情報を用いて`MPI_Send`と`MPI_Recv`によって部分行列を交換する．このとき，MMO のように巡回ヤコビ法の内でも並列計算向けに設計されたものならば，それらの情報も簡単な計算で得られ，また，通信についてもホップ数が小さくなるように，今選択する列番号と次に選択する列番号と差が小さくなるものがある．

図5.13は MMO の代わりに Bečka の動的順序を用いたときの擬似プログラムである．動的順序では並列ペア生成と同時に収束判定のための情報生成を同時に行えるため，プログラムが比較的シンプルになる．またここでは列ブロックペアの交換の詳細を示しておらず，巡回ヤコビ法と比べて動的順序では複雑な手順になるが，次節に詳細を議論する．

5.2.5　計算量の比較

次に，これまで上げた 3 つの手法について，クリティカルパス上の計算量として演算量，通信量，そして通信回数を示す．ただしここでは議論を単純にするため，通信の衝突やホップ数などの，通信網の形状に依存する点は考慮しない．また，この議論では一対一通信と集団通信とを比較できるようにするため，集団通信はアルゴリズムを想定した上で，一対一通信の通信量・通信回数に換算する．また，一次元ブロック分割とそのほかの 2 手法では，縦方向の並列数 k によって 1 反復当たりの演算回数が異なるため，演算量の主要項を統一するために，MMO を用いたときの 1 巡回ごとの計算量を示す．つまり以下で求める計算量を，一次元ブロック分割の場合は $w = 2p$ 倍，他の 2 手法では $w = \frac{2p}{k}$ 倍にする．また，列ブロックペアの直交化手法は Hari の V2 手法に統一する．

この議論で出現する集団通信には，Broadcast, AllReduce, All-to-All の 3 種類がある．集団通信に参加するプロセッサ数を p_g, データの要素数を n_g とおく．まず Broadcast については，二分木型通信を用いると，クリティカルパス上で $\log_2 p_\mathrm{g}$ 回の通信が発生し，$n_\mathrm{g} \log_2 p_\mathrm{g}$ 要素のデータを移動するとする．また，AllReduce においては ReduceScatter & AllGather 型のアルゴリズムを用いると考え，$\log_2 p_\mathrm{g}$ 回の通信と，n_g 要素のデータ移動が発生するものとする．最後に All-to-All では p_g 回の通信と n_g 要素の

表5.1　3つのデータ分散における計算量の比較

		一次元ブロック分割	二次元ブロック分割	Variable Blocking
列ブロックの交換	通信量	$4n^2$	$4n^2/k$	
	通信回数	$8p$	$8p/k$	
データ並び替え	通信量			$O(n^2/k)$
	通信回数			$O(p)$
Gram 行列の生成	演算量	$2n^3/p$	$2n^3/p$	$2n^3/p$
	通信量		n^2k/p	$O(n^2 \log_2 k/\sqrt{k})$
	通信回数		$O(p \log_2 k/k)$	$O(p \log_2 k/\sqrt{k})$
Cholesky 分解	演算量	$\frac{2}{3}n^3/p^2$	$\frac{2}{3}n^3k^2/p^2$	$\frac{2}{3}n^3k/p^2$
	通信量			$O\left(\frac{n^2k \log_2 k}{bp}\right)$
	通信回数			$O(n \log_2 k/b)$
小行列の SVD	演算量	$O(n^3/p^2)$	$O(n^3k^2/p^2)$	$O(n^3k/p^2)$
	通信量			$O(n^2k \log_2 k/p)$
	通信回数			$O(n \log_2 k)$
行列方程式の計算	演算量	$2n^3/p^2$	$2n^3k^2/p^2$	$2n^3k/p^2$
	通信量			$O\left(\frac{n^2k \log_2 k}{bp}\right)$
	通信回数			$O(n \log_2 k/b)$
列ブロックの更新	演算量	$4n^3/p$	$4n^3/p$	$4n^3/p$
	通信量			$O(n^2/\sqrt{k})$
	通信回数			$O(p \log_2 k/\sqrt{k})$

データ移動が発生するものとする．これらの集団通信における通信量・通信回数は，理想的な通信網上でのものとなっており，現実の制限された通信網においては，これよりも悪化するものと考えられる．

一次元ブロック分割の場合，平均列ブロック幅が $l_{\mathrm{abs}} = \frac{n}{2p}$ となる一方で，行ブロックサイズは全体の行サイズと同じ n である．そこで，まず1ステップごとの通信回数と通信量を考えると，まず列ブロックペアの送信・受信を行うために4回の通信によって $4nl_{\mathrm{abs}} = \frac{2n^2}{p}$ 要素を移動する．一次元ブロック分割の場合には一度列ブロックペアが得られれば通信せずとも列ブロックペアの直交化を計算できるため，1ステップ内の通信は以上である．

一次元ブロック分割の演算量について考えると，まずグラム行列の計算に主要項で，$4nl_{\mathrm{abs}}^2 = \frac{n^3}{p^2}$ 演算かかる．また小行列の Cholesky 分解に主要項で，$\frac{8}{3}l_{\mathrm{abs}}^3 = \frac{n^3}{3p^3}$ の演算がかかる．また小行列の SVD の演算量は片側ヤコビ法の反復回数を考慮する必要があるが，$O(l_{\mathrm{abs}}^3) = O\left(\frac{n^3}{p^3}\right)$ とおく．次に，行列方程式の演算に主要項で $8l_{\mathrm{abs}}^3$ の演算が必要であり，最後の列ブロックペアの更新に，主要項で $8nl_{\mathrm{abs}}^2 = \frac{2n^3}{p^2}$ の演算量となる．以上をまとめると，表 5.1の3列目となる．

二次元ブロック分割の場合も同様にして考えることができる．この場合，縦方向の並列数 k によっ

表5.2　二次元ブロック分割において $k = \alpha\sqrt{p}$ と設定したときの計算量

	演算量	通信量	通信回数
列ブロックの交換		$\frac{4n^2}{\alpha\sqrt{p}}$	$\frac{8}{\alpha}\sqrt{p}$
Gram 行列の生成	$2n^3/p$	$\alpha n^2/\sqrt{p}$	$O(\sqrt{p}\log_2 p/\alpha)$
Cholesky 分解	$\frac{2}{3}\alpha^2 n^3/p$		
小行列の SVD	$O(\alpha^2 n^3/p)$		
行列方程式の計算	$2\alpha^2 n^3/p$		
列ブロックの更新	$4n^3/p$		

て，平均列ブロック幅が $l_{\mathrm{abs}} = \frac{nk}{2p}$ となり，また，平均行ブロック幅が $n_{\mathrm{b}} = \frac{n}{p}$ となる．そこで，通信量や通信回数，演算量については一次元ブロック分割のときのものに以上の平均ブロック幅を代入すれば，表 5.1の 4 列目が得られる．ただし，一次元ブロック分割からコストが追加される部分があり，これは列ブロックペアの直交化内で計算する Cholesky QR 法における総和のための集団通信であり，通信量が $2l_{\mathrm{abs}}^2 O\left(\log_2 k\right) = O\left(\frac{n^2 \log_2 k}{2p^2}\right)$ となり，通信回数が $O(\log_2 k)$ となる．

Variable Blocking の計算量の算出は少々複雑である．Bečka ら [117] は ScaLAPACK 内部の通信は“見えない”ため正確な計算をしていないが，例えば，行列積に PUMMA のような標準的なアルゴリズムを用いるなどの仮定をおくと，ある程度の予測が可能である．いま簡単のため k ノードは正方形に $\sqrt{k} \times \sqrt{k}$ ノードに分割され，また，ScaLAPACK のブロック幅 $b_{\mathrm{r}} = b_{\mathrm{c}} = b$ は行と列で共通に分割するものとする．PUMMA による行列積では，$O\left(\sqrt{k}\right)$ 回の集団通信 (Broadcast) と，$O\left(\sqrt{k}\right)$ 回の隣接通信を行い，それぞれの 1 回あたりの通信量は共通で $O\left(\frac{nl_{\mathrm{abs}}}{k}\right)$ 要素である．演算量については，理想的に均等に分割できていると想定して，Gram 行列の生成では $O\left(\frac{nl_{\mathrm{abs}}^2}{k}\right)$，列ブロックの更新ではその 2 倍とする．

次に問題となるのは小行列における計算である．Cholesky 分解の並列化においてはブロックごとに計算が進行するため，$O\left(\frac{l_{\mathrm{abs}}}{b}\right)$ 回の集団通信 (Broadcast) が行われ，1 回の通信ごとに $O(l_{\mathrm{abs}} b)$ 要素の移動が行われる．小行列の SVD についてはどの実装とデータ分散を用いるかで計算量が変わるが，単純な一次元ブロック分割を用いるならば，その計算量を当てはめることができ，また ScaLAPACK の SVD 手法を用いるならば，$O(l_{\mathrm{abs}})$ 回の集団通信 (Broadcast) が行われ，1 回の通信ごとに $O(l_{\mathrm{abs}})$ 要素の移動が行われる．ここでは Variable Blocking の考え方に則り，4 章のとおり収束性に問題が出るが，ScaLAPACK の SVD を用いることにする．また，行列方程式の計算量は Cholesky 分解と同等である．最後に，一次元ブロック分割から二次元ブロックサイクリック分割への並び替えが必要である．これは All-to-All 型の通信であり，$O\left(\frac{nl_{\mathrm{abs}}}{k}\right) = O\left(\frac{n^2}{p}\right)$ 要素を移動する集団通信となる．そこで表 5.1の 5 列目が得られる．

表 5.1において 3 手法を比較すると，3 手法ともに行列積（Gram 行列の生成と列ブロックの更新）における演算量は一致している．そこで $k \ll p$ であるならば，3 手法の主要項の演算量が一致する．しかし $k \approx \sqrt{p}$ となれば，二次元ブロック分割では小行列に対する演算の演算量が主要項とオーダーで一致する．小行列に対する計算は複雑な計算が多く行列積と比べて演算効率が低いため，大きな k を用いると二次元ブロック分割は性能低下することが予想される．そこで k の設定は重要であるが，小さな正定数 α を用いて $k = \alpha\sqrt{p}$ のように設定すれば，通信量をオーダーの意味で削減しつつ，大きな

表5.3　3つのデータ分散における1通信量・通信回数あたりの演算量

	一次元ブロック分割	二次元ブロック分割	Variable Blocking
1通信量あたりの演算量	$O(n/p)$	$O(nk/p)$	$O\left(\frac{n}{k\log_2 k}\right)$
1通信回数あたりの演算量	$O(n^3/p^2)$	$O\left(\frac{n^3 k}{p^2\log_2 k}\right)$	$O\left(\frac{n^2}{p\log_2 k}\right)$

k を用いることによる性能低下を緩和することができる．このときの計算量を表5.2に示す．この表の第2列を見ると，演算量はすべて p に反比例している．また小行列に対する演算は α^2 に比例した演算量となっており，α を適切に設定すれば，これらの計算によるオーバーヘッドを小さく抑えることができる．

一次元ブロック分割と二次元ブロック分割の通信を比較すると，まず表5.1では，二次元ブロック分割は列ブロックの交換における通信量・回数が $1/k$ 倍される代わりに，Gram 行列の生成において通信が増える．ただし，$2 \le k \le p/2$ であるため，通信量の和は減少するといえる．また通信回数については $\log_2 k/k$ が大きすぎなければ，一次元ブロック分割よりも小さくなる．また，一次元ブロック分割と二次元ブロック分割の両方で，通信回数が n に依存せず p や k などのノード数に依存することに注意が必要である．これは通信回数が n に比例する二重対角化を用いた SVD 手法とは対照的である．さらに，表5.2のように，二次元ブロック分割において $k = \alpha\sqrt{p}$ と設定すれば，通信量のオーダーは $O(n^2/\sqrt{p})$, 通信回数のオーダーは $O(\sqrt{p}\log_2 p)$ となり，通信量・通信回数の両方でオーダーの意味で一次元ブロック分割よりも小さくなる．

Variable Blocking と二次元ブロック分割とを比較すると，Varible Blocking の方が並列化する項目が多いため，小行列に対する演算において Variable Blocking の演算量のオーダーは二次元ブロック分割のものよりも優れている．しかし，その代わりに通信回数や通信量が増大し，並列度の高い環境においては不利にはたらくものと考えられる．

この小節の最後に，各データ分散における通信回数や通信量あたりの演算量を計算する．ここで $k \le \sqrt{p}$ を仮定している．1巡回あたりの演算量は，3つのデータ分割において共通で $O(n^3/p)$ である．一次元分割では，1巡回あたり $O(n^2)$ の通信量と $O(p)$ の通信回数が発生する．そこで，1通信量あたり $O(n/p)$ の演算となり，1通信回数あたり $O(n^3/p^2)$ の演算となる．二次元分割においては，1巡回あたり，$O(n^2/k)$ の通信量と $O(p\log_2 k/k)$ の通信回数が発生する．そこで，1通信量あたり $O(nk/p)$ の演算となり，1通信回数あたり $O\left(\frac{n^3 k}{p^2\log_2 k}\right)$ の演算となる．また Variable Blocking においては，1巡回あたり $O(n^2 k\log_2 k/p)$ の通信量と，$O(n\log_2 k)$ の通信回数が発生する．そこで，1通信量あたり $O\left(\frac{n}{k\log_2 k}\right)$ の演算となり，1通信回数あたり，$O\left(\frac{n^2}{p\log_2 k}\right)$ の演算となる．以上の結果をまとめたものが表 5.3である．これより，通信量あたりの演算量については，二次元ブロック分割と Variable Blocking が同じ程度 ($k^2\log_2 k < p$ かそうでないかによる) だが，通信回数あたりの演算量は二次元ブロック分割が3つの中で最も大きく，すなわち，通信粒度が大きいことがわかる．

また，表 5.3を使って，二次元ブロック分割において，1通信量あたりの演算量，または1通信回数あたりの演算量をそれぞれ c_{amount}, c_{count} に固定した場合における n と p の関係を計算すると，まず前者からは，

$$n = O\left(\frac{c_{\text{amount}}}{\alpha}\sqrt{p}\right) = O\left(\sqrt{p}\right) \tag{5.4}$$

が得られる．これは，n の増加（倍率）に対して p をその自乗倍だけ増加させても，1 通信量あたりの演算量は同じ比率となることを示している．また，後者について

$$n^3 = O\left(\frac{c_{\text{count}}}{\alpha} p^{\frac{3}{2}} \left(\frac{1}{2}\log_2 p + \log_2 \alpha\right)\right) = O\left(p^{\frac{3}{2}} \log_2(p)\right) \tag{5.5}$$

を得る．これは式に対数の項が含まれる分，前者と比べて関係がやや複雑であるが，対数の項の増大が緩やかだと考えれば，前者の式とよく似た傾向を示すと言える．すなわち，n の増加に対して p を自乗倍だけ増加させると，1 通信回数あたりの演算量がゆるやかに増大していくことを示している．そこで，計算時間が演算量に比例する項と，通信量に比例する項，通信回数に比例する項の和となっているような，ごく単純な計算モデルを考えることにする．すると，このモデル上では，全体の計算時間において通信が占める割合は c_{amount} と c_{count} の線形和で表せることがわかる．よって，この単純な計算モデルを前提として式 (5.4) と式 (5.5) の結果を読み替えれば，二次元ブロック分割では，行列サイズ n とノード数 p の自乗を比例させるように変化させた場合，演算量と計算時間の比率，つまり計算効率がほぼ一定（対数的に増大）となることがわかる．これはまた，演算量とノード数を比例させるように変化させた状況での性能である弱スケーリング性能は，n に比例して減少することを示している．ただしここでの比較は定数項である演算効率などの要素を省いたものとなっているため，実際の計算における挙動をどの程度，表現できているかについては興味深い．

5.2.6 大規模並列計算機を用いた性能検証

この節の最後に，二次元ブロック分割の性能検証を行う．ここでは 2 つの実験結果を示す．1 つは，比較的少ないノード数 (144) において，k を変化させながら行った実験，もう 1 つは，比較的大きなノード数 (〜 12,288) において，ScaLAPACK との性能比較を行う実験である．最初の実験では東京大学物性研究所にある SGI Altix ICE を用いた．利用者以外には非公開の利用マニュアルによれば，このシステムは機能が部分的に異なる 3 つのシステムの集合体となっているが，そのうち CPU ノードと呼ばれるものを用いている．このシステムは Intel の Xeon E5-2680v3 を Infiniband の通信網によって多数接続したクラスターマシンである．Xeon E5-2680v3 は 12 コアを持つマルチコア CPU であり，2.5GHz のクロック周波数で動作し，1CPU あたりのピーク演算性能は 480GFlop/s である．SGI Altix ICE では 1 ノード 2 ソケットのシステムとなっているが，1 ノードあたり 2MPI プロセスを走らせることで，1MPI プロセスあたり 1CPU となるように調整している．そこで，以降の結果ではノード数ではなく CPU 数と表記している．またこのシステムは Intel の CPU のクラスターであるため，Intel のコンパイラを用いており，バージョンは 16.0.4 である．後者の実験では，京コンピュータを用いている．京コンピュータの開発元である富士通が発行した Journal [119] によれば，京コンピュータは SPARC64 VIIIfx を独自の通信網 Tofu interconnect によって多数接続したクラスターマシンであり，1CPU あたり 8 つのコアを持ち，1CPU あたりのピーク演算性能は 128GFlop/s である．京コンピュータは最大 8 万ノード超まで利用できるが，ここではそのうち 12,288 ノードまでを用いており，計 98,304 コアを使うことになる．SPARC64 VIIIfx は富士通独自の CPU であり，専用のコンパイラ FCC が存在し，ここで利用したバージョンは `K-1.2.0-23` である．

最初の実験では，SGI Altix ICE の 144 CPU を用いて実験を行った．図 5.14, 図 5.15はそれぞれ $m = n = 4,320,\ 8,640$ において，ノード数 $p = 144$ と固定して，k を変化させたときの実行性能の内訳を示したものである．$k = 1$ の場合は一次元ブロック分割となり，$k > 1$ の場合，二次元ブロック分

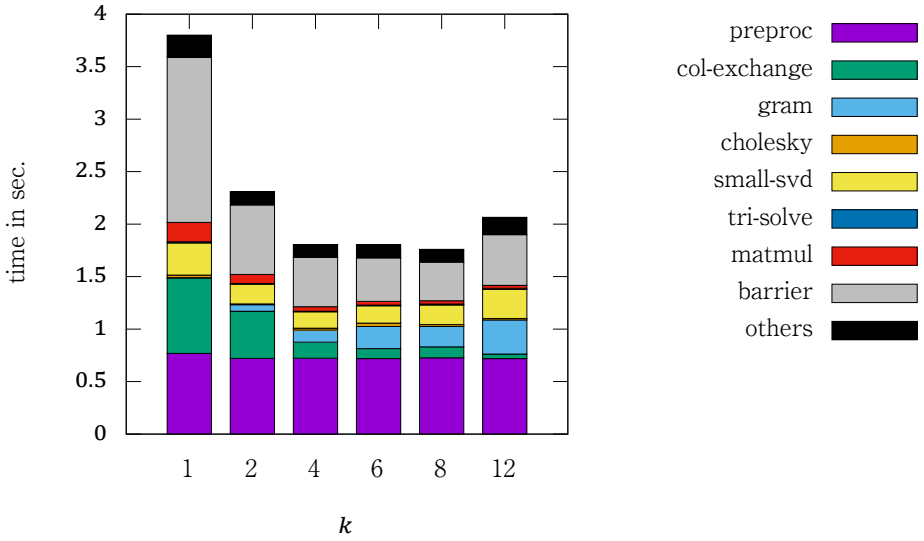

図5.14　SGI Altix ICE の 144CPU を用いて片側ブロックヤコビ法の実行時間を測定したときの，実行時間の内訳．行列は $m = n = 4,320$ であり，対数が一様分布する乱数によって特異値が決められた条件数が約 10^{10} の対角行列を乱数の直交変換によって両側から別々に変換したものである．

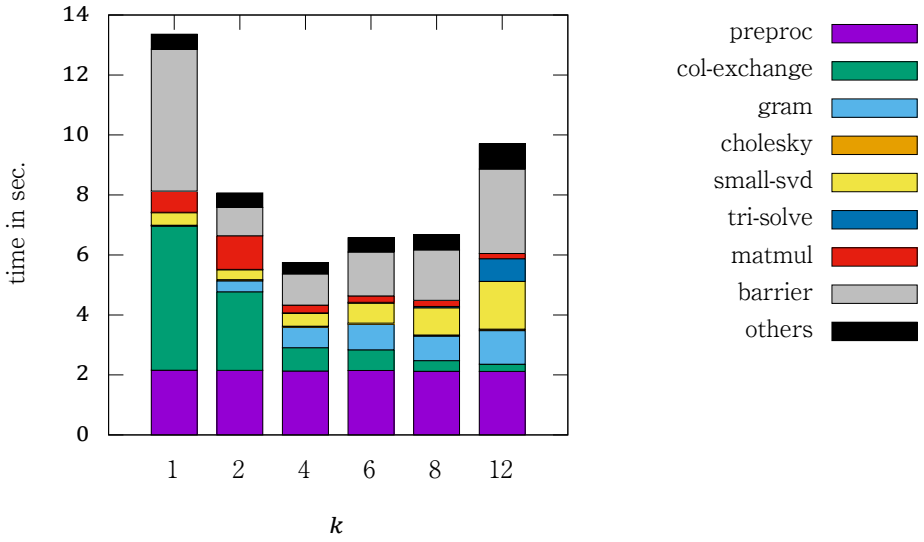

図5.15　SGI Altix ICE の 144CPU を用いて片側ブロックヤコビ法の実行時間を測定したときの，実行時間の内訳．行列は大きさが $m = n = 8,640$ となっていること以外，図 5.14と同じ．

表5.4　縦方向の並列数 k と反復回数

行列サイズ (n)	$k=1$	$k=2$	$k=4$	$k=6$	$k=8$	$k=12$
$4,320$	$2,016$	$1,008$	432	288	180	120
$8,640$	$2,016$	$1,008$	432	288	180	120

割となっている．実行時間の中で大きな部分を占めるものが前処理・後処理の時間 (preproc) と，同期 (barrier)，そして列ブロックの交換 (col-exchange) である．同期の時間の大きさは，各ノードの通信時間や演算時間のばらつきを表している．演算時間のばらつきは図5.12の擬似プログラムの 14 行目において，主に対象の列ブロックペアがすでに十分直交している場合に以降の処理 (cholesky, small-svd, tri-solve, matmul) をスキップするときに発生する．前処理・後処理は ScaLAPACK の二次元ブロックサイクリック分割をしており，k によってデータ分割は変化しないため，実行時間はほぼ一定となっている．また，実行時間の最も小さいところでは全体の約 4 割を占める．列ブロックの交換は，$k=1$ のときは前処理・後処理と同程度の実行時間となっているが，k に反比例して減少しており，理論計算量の議論に従って傾向を示している．

計算部分を見ると，Gram 行列の計算 (gram) は列ブロックの更新 (matmul) と同じ行列積の計算だが，前者の方の演算量は後者の約半分となっている．しかし，k が大きいときは前者の方が大きくなる．これには 2 つの理由があり，まず，Gram 行列の計算では AllReduce 型の集団通信が発生する一方で，列ブロックの更新では通信が発生しないためである．もう 1 つの理由は，前者と後者との演算性能の違いであり，k が大きい場合は，前者の演算性能が後者の約半分となっている．

また，次に大きな部分を占めるものが，小行列の SVD (small-svd) であり，k が大きくなるとくに割合として顕著となる．この部分は前節の通りスレッド並列化しているが，より高速化するならば，小行列の SVD に対しても V2 手法を用いた片側ブロックヤコビ法を適用するなどの改良が必要となると考えられる．

また，k に応じてヤコビ法部分の性能が大きく変わり，実験したものの中で最適な k は，$n=4,320$ の場合 $k=8$ であり，$n=8,640$ の場合，$k=4$ であった．これには，$O\left(\frac{n^3k^2}{p^2}\right)$ となる小行列に対する計算と，列ブロック交換の計算量や，ブロック幅増加による行列積の演算効率増大などの効果のトレードオフによって変化する．また，k を変化させることによって，収束するまでにかかる反復回数も増減する．表 5.4は，収束までにかかった並列ステップ数を示しているが，これに $\frac{k}{2p}$ をかけた値が MMO を用いたときの巡回回数となる．実際に計算すると，$k=1,2$ のときは 7 巡回だが，$k=4,6$ のときは 6 巡回，$k=8,12$ のときは 5 巡回となっており，k の増大に従って巡回回数が減少している．このことは一次元ブロック分割に対する二次元ブロック分割の優位性を表しており，また，適切な k の設定の重要さも表している．

図 5.16は京コンピュータ上の 12 ノードから 12,288 ノードを用いたときの，ScaLAPACK の PDGESVD と片側ブロックヤコビ法を用いたときの実行時間を示したものである．ここでは縦方向の分割数 $k \approx \sqrt{p/5}$ となるような整数とした．実際に用いた値は表 5.5に示す．行列サイズは $m=n=5,000$ と $m=n=10,000$ を用い，$m=n=5,000$ のときは $p=12,\ 48,\ 192,\ 768$, $m=n=10,000$ のときは $p=192,\ 768,\ 3,072,\ 12,288$ 用いた．京コンピュータ上での片側ブロックヤコビ法の性能は，ノード数が少ないほど ScaLAPACK の性能よりも高速であり，$m=n=5,000$,

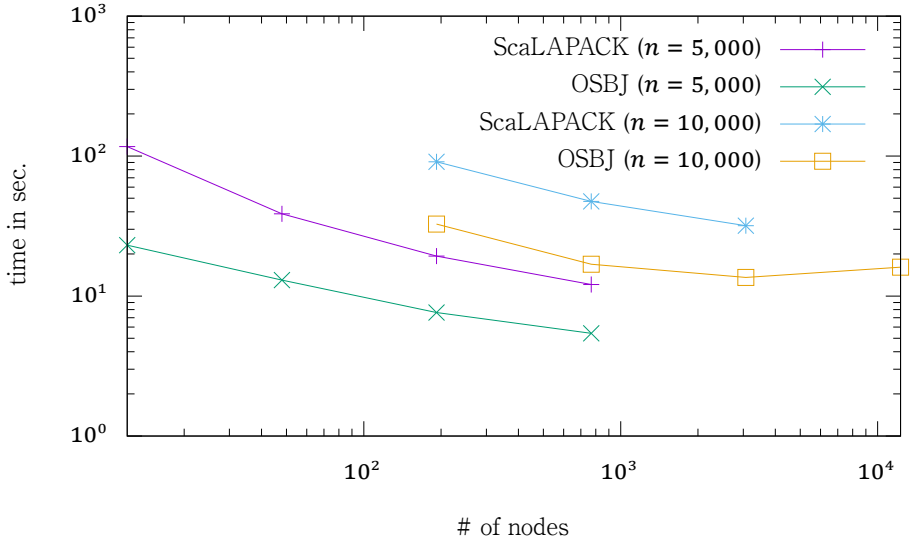

図5.16　京コンピュータ上での片側ブロックヤコビ法の強スケーラビリティ．行列は $m = n = 5,000,\ 10,000$ となっていること以外，図 5.14と同じ．

表5.5　京コンピュータ上での実験で用いたノード数 p と縦方向の並列数 k

ノード数 (p)	縦方向の並列数 (k)	横方向の並列数 ($p/k = w/2$)
12	1	12
48	3	18
192	6	32
768	12	64
3,072	24	128
12,288	48	256

$p = 12$ のとき，約 5 倍の性能となっており，また，$p = 768$ と増やすと，約 2.2 倍まで差が縮まる．$m = n = 10,000$ の場合は，$p = 3,072$ まで実行時間が減少するが，$p = 12,288$ になるとむしろ $p = 3,072$ よりも遅くなる．

図 5.16上で折れ線の傾きを見ると，どの手法のどの区間においても（両対数グラフ上での）傾きは -1 よりもずっとなだらかなものとなっていることがわかる．片側ブロックヤコビ法の折れ線を見ると，どの 2 点間を見ても傾きは $-\frac{1}{2}$ よりも小さい程度となっている．理想的にはノード数 p に対して実行時間は反比例，すなわち両対数グラフ上で傾き -1 となるべきだが，そこまでの並列化効率は達成できていないことがわかる．この原因には，実験におけるノード数，行列サイズでは実行時間の内，通信量や通信回数に依存する項が支配的になっていることが考えられる．また，計算量の算出では考慮していない，行列積などの演算における実行効率が，p の増大に従って列ブロック幅 l が減少することで，大きな影響を持つようになっていることも考えられる．

また式 (5.4) と式 (5.5) から，行列サイズ n を 2 倍にしたときノード数 p を 4 倍にすると，ごく単純な計算モデル上では計算効率がほぼ一定になることが期待される．このとき全体の演算量は 8 倍とな

るため，実行時間は 2 倍となると予測できる．そこで $m = n = 5,000$ における $p = 48, 192, 768$ の実行時間と $m = n = 10,000$ における $p = 192, 768, 3,072$ の実行時間との比を計算すると，それぞれ約 2.5, 2.2, 2.5 であり，予測値 2 と近い値となっている．ただし，グラフの傾き自体は計算量からの予測から外れたものとなっているため，これは，演算効率と通信効率が同じように低下したために発生したものだと考えられる．そこで式 (5.4) と式 (5.5) を用いた性能予測は一定の可能性を持つが，実際に用いるにはより詳細かつ幅広い試験が必要であると考えられる．

以上のように，大規模並列計算機における実験では片側ブロックヤコビ法は ScaLAPACK よりも高速となっており，片側ブロックヤコビ法の並列環境における優位性を実証できた．

5.3 Bečka の動的順序

Bečka の動的順序は，はじめ，ブロック化された Kobeliantz 法に対する並列オーダリング手法として開発された [8]．並列化可能な動的順序の提案自体は Foulser [89] の方が早いが，マッチングアルゴリズムとの関係や，片側ブロックヤコビ法への適用手法などの開発は Bečka らによって行われたもので，アルゴリズムの説明が洗練されている．このオーダリングは次の 2 つのアイデアに基づく．1 つは，当初の Jacobi が用いたオーダリングのブロック版への拡張であり，消去すべき非対角ブロックの中で最も収束を早めるものを貪欲法によって探索する．もう 1 つは並列オーダリングであり，同時に消去できるものを可能な限り多く選択する．すなわち，Bečka の動的順序は，貪欲法のアイデアで同時に消去可能な非対角ブロックの内，局所的に見て最も収束を早めるものを探索する．しかしこのオーダリングを片側ブロックヤコビ法に適用するためには次の 3 つの問題点がある．第一に，片側ブロックヤコビ法においては消去したいブロックが陽に計算されないため，探索を行うことが困難なこと，第二に，貪欲法によって探索するため得られた並列ペアが本当に良いものなのかどうかわからないこと，第三に，得られた並列ペアは行列によって任意に変化するため，どのペアをどのノードに割り当てれば通信が削減されるか，巡回ヤコビ法のときほど明白ではないことである．第一の問題点は Bečka によって近似手法 [117] が構築されたことにより，おおむね解決されているが，他の 2 つについては未解決である．そこでこの節ではまず Bečka のオーダリングについて詳細を示したのちに，第一の近似手法に対する改良手法の提案と，第二，第三の問題点の解決方法について述べる．

5.3.1 アルゴリズム

Bečka の手法には 2 つの構成要素がある．1 つは $A = \begin{bmatrix} A_1 & A_2 & \cdots & A_w \end{bmatrix}$ に対して $C = A^{\mathsf{T}}A$ の非対角ブロックの大きさ (フロベニウスノルム) を推定する手法，もう 1 つは，推定したノルムを用いて，貪欲法によって，収束を早めてかつ並列に選択可能なペアを列挙する手法である．前者を求める理由は，3 章の式 (3.36) の通り，ブロックヤコビ法の一次収束性が，消去した非対角ブロック分だけの非対角ブロックのフロベニウスノルムの自乗和が減少することによるためである．すなわち，最もフロベニウスノルムの大きい非対角ブロックを選択すれば，局所的に最も収束を早めることになる．当然これは最急降下法と同様に，あるステップにおいて局所的に最適な解を用いているだけであって，反復回数を最小化するとは限らないが，計算可能かつ容易であり，また，実験的にもうまくいく手法である．

ここで問題となることが C の計算であり，実際にすべての要素を計算するためには 1 並列ステップの 1 ノード当たり約 n^3/p の演算が必要となるが，これは他の演算のオーダーである $O(n^3k/p^2)$ より

も大きくなってしまう．そこで何らかの方法でこれを近似的に求める必要がある．

Bečka の手法では C の非対角ブロックのフロベニウスノルム $\left\|A_i^\top A_j\right\|_\mathrm{F}$ を計算する代わりに，1 ベクトル $\mathbb{1}$ を用いて $e = \frac{\mathbb{1}}{\|\mathbb{1}\|_2}$ によって

$$g_{i,j} \equiv \left\|A_i^\top A_j e\right\|_2 \tag{5.6}$$

を計算する．この計算は，1 つの非対角ブロックに対して，$O(nl_\mathrm{abs})$ となる．非対角ブロックは $\frac{w(w-1)}{2}$ 個あるが，p 並列で計算できるため，1 並列ステップの 1 ノード当たりの演算量は $O\left(\frac{n^2}{k}\right)$ となる．つまり，非対角要素すべてを計算するよりも演算量を抑えることができる．また，次の定理により，g_{ij} は $\left\|A_i^\top A_j\right\|_\mathrm{F}$ の良い近似になっていることが期待される．

定理 5.1 (山本ら [120], Theorem 3, 4). **C は $p \times q$ の実行列であり，$p \geq q$ かつ $\|C\|_\mathrm{F} = f$ を満たす．また，$\mathcal{M}_{p,q,f}$ は $p \times q$ のフロベニウスノルムが f の実行列の集合であるとする．すなわち，正の対角行列 $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ と，n 次の直交行列の集合 $\mathcal{O}(n)$ について，**

$$\mathcal{M}_{p,q,\sigma} = \left\{ U \begin{bmatrix} \Sigma \\ \mathbb{0}_{p-q,q} \end{bmatrix} V^\top \middle| U \in \mathcal{O}(p), V \in \mathcal{O}(q) \right\} \tag{5.7}$$

に対して，

$$\mathcal{M}_{p,q,f} \equiv \cup_{\mathrm{Tr}(\Sigma^2)=f} \mathcal{M}_{p,q,\Sigma} \tag{5.8}$$

と定義する．このとき，$C \in \mathbb{R}^{p \times q}$ を $\mathcal{M}_{p,q,f}$ からランダムにサンプルされた行列だとすると，次を満たす：

$$\mathrm{E}\left[\|Ce\|_2^2 \middle| C \in \mathcal{M}_{p,q,f}\right] = f^2, \tag{5.9}$$

$$\mathrm{V}\left[\|Ce\|_2^2 \middle| C \in \mathcal{M}_{p,q,f}\right] = \frac{2}{q+2} \sum_{i<j} \left(\sigma_i^2 - \sigma_j^2\right)^2. \tag{5.10}$$

系 5.2 (山本ら [120], Corollary 5). **いま，特異値のサンプル平均 $\hat{u}$ とサンプル分散 $\hat{s}$ を次のように定義する：**

$$\hat{u} = \frac{1}{q} \sum_{i=1}^{q} \sigma_i^2, \tag{5.11}$$

$$\hat{s} = \frac{1}{q-1} \sum_{i=1}^{q} \left(\sigma_i^2 - \hat{u}\right)^2 = \frac{1}{q(q-1)} \sum_{i<j} \left(\sigma_i^2 - \sigma_j^2\right)^2. \tag{5.12}$$

このとき，式 (5.10) より，

$$\begin{aligned} \frac{\sqrt{\mathrm{V}\left[\|Ce\|_2^2 \middle| C \in \mathcal{M}_{p,q,\Sigma}\right]}}{\mathrm{E}\left[\|Ce\|_2^2 \middle| C \in \mathcal{M}_{p,q,\Sigma}\right]} &= \sqrt{\frac{2}{q+2}} \cdot \frac{\sqrt{\sum_{i<j} \left(\sigma_i^2 - \sigma_j^2\right)^2}}{\sum_{i=1}^{q} \sigma_i^2} \\ &= \sqrt{\frac{2}{q+2}} \cdot \frac{\sqrt{q(q-1)}\hat{s}}{q\hat{u}} \\ &= \sqrt{\frac{2(q-1)}{q(q+2)}} \cdot \frac{\hat{s}}{\hat{u}} \approx \sqrt{\frac{2}{q}} \frac{\hat{s}}{\hat{u}}. \end{aligned} \tag{5.13}$$

以上の定理において，ランダムにサンプルされた行列 C は $A^\top A$ の非対角ブロック $C_{i,j} = A_i^\top A_j$ を想定しており，定理における行列サイズ p, q はそれぞれ A_i と A_j の列ブロック幅 n_i と n_j である．そこで，$C_{i,j}$ が $\mathcal{M}_{p,q,f}$ からランダムにサンプルされたものであれば，$g_{i,j}$ の期待値はほしかった値 $\left\|A_i^\top A_j\right\|_\mathrm{F}$ と一致し，また，$g_{i,j}$ の分散は，$\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_q^2\}$ の分散に比例してかつ $\sqrt{q}$ に反比例する．そのため，n_j が大きければ $g_{i,j}$ はよい近似となっていると考えられる．ただし，$C_{i,j}$ が本当に $\mathcal{M}_{p,q,f}$ からランダムにサンプルされたものと仮定できるのか，については議論の必要がある．実際には元の行列 $C = A^\top A$ がランダムにサンプルされたものであったとしても反復途中の行列の非対角ブロックがどのような構造を持っているのか，解析は困難である．そのため非対角ブロックのフロベニウスノルムが過小評価され，ヤコビ法の収束性が悪化したり，偽収束が発生したりする可能性がある．例えば，$C_{i,j}$ の 0 に近い固有値に対応する固有空間に $\mathbb{1}$ ベクトルが含まれる場合，ノルムが過小評価される．しかしながら，反復の途中でこのような過小評価される非対角ブロックがあったとして，$C_{i,J}$ が消去されなかったとしても，他のブロックの更新によって $C_{i,j}$ の値も書き換わり，固有ベクトルが変化する．そこで，過小評価されたブロックが再び過小評価されるという状況が起きなければ，偽収束に至ることはない．また以下の実験においては偽収束は観測されなかった．

Bečka は他に数種類の重みの定義を作成している [121] が，収束性との関係が最もわかりやすいものがこの定義である．他の定義においては，A_i と A_j それぞれの張る空間同士がどれだけ互いに傾斜しあっているかの度合いを測るため，A_i, A_j の代わりに，それらの列を正規化した U_i と U_j を用い，$\left\|U_i^\top U_j\right\|_\mathrm{F}$ を近似的に計算する．この場合においても近似手法に上記と同じ手法を用いることができる．

このように重みを定義できたとき，並列に消去可能でかつ重みが大きいものを探索したい．いま重み $g_{i,j}$ をフロベニウスノルムの近似としたので，この二乗和が最大になるような並列ペアを探索できれば，局所的に最も収束に近づくものを，近似的に求められると考えられる．これはグラフ理論の最大重みマッチングを用いれば定式化できる．いまグラフ G を w 個の頂点を持つ完全グラフと定義し，頂点 i と j の間の重み $\bar{g}_{i,j} = g_{i,j}^2$ とする．このとき，G 上の最大重みマッチング M_max とは，G 上のすべてのマッチングの集合 $\mathfrak{M}$ について

$$M_\mathrm{max} = \arg\max_{M \in \mathfrak{M}} \sum_{(i,j) \in M} \bar{g}_{i,j} \tag{5.14}$$

を求めることである．最大重みマッチングはマッチングの一種であり，マッチングは枝で結ばれる頂点が 2 度以上現れないような，つまり頂点のペアを集めたグラフである．そこで頂点のペア (i, j) を消去する非対角ブロック（直交化する列ブロックのペア）だと考えると，最大重みマッチングによって並列に消去可能でかつ重みの和が最大のペアを列挙できる．

最大重みマッチングは基本的なグラフ理論の問題の 1 つであり，Edmond による Blossom アルゴリズム [122] が有名である．Duan ら [123] に詳しいが，完全グラフ上の最大重みマッチングの計算量は $O(w^3)$ となり，また，行列計算とは異なり，計算が複雑で並列化が難しく，また計算効率も低い．そこで，Bečka は最大重みマッチングの厳密解を求める代わりに，高速な近似解法によって近似解を求める方法を選択している．これはそもそも重みが近似的に求められていること，最大重みマッチングの厳密解であっても，ヤコビ法の反復計算全体の厳密解ではないことから，最大重みマッチングの厳密解を求める意味が薄いためである．最大重みマッチングの近似解法については Avis らによる Survey [124] や，Duan ら [123] に詳しい．Bečka は，貪欲法に基づく近似解法を用いている．これは実装が単純でかつ，$O(w^2 \log_2 w)$ の計算量であり，厳密解の重みの和に対して $1/2$ 以上の重みを持つ

```
subroutine Greedy-Matching([g_{2,1}, g_{3,1}, g_{3,2}, …, g_{w,1}, …, g_{w,w-1}]):
        m = []
        s = [1, 2, …, w]
        G = sort([g_{2,1}, g_{3,1}, g_{3,2}, …, g_{w,1}, …, g_{w,w-1}])
        do
                g_{i,j} ← the first element of G.
                remove g_{i,j} from G.
                if i in s and j in s:
                        remove i and j from s.
                        append g_{i,j} to m.
        while s is not empty
        return m
```

図5.17　貪欲法を用いた最大重みマッチングの近似解法の擬似プログラム

解を得られることが知られている [124, Section II.B]. Duan らの手法のように，任意精度の解を得られるものも存在するが，ここではアルゴリズムの単純さと実行速度を考えて，貪欲法を用いる．このアルゴリズムを図 5.17に示す.

5.3.2　重み計算の高性能実装

以上の通り，重み計算の演算量は $O\left(\frac{n^2}{k}\right)$ である一方で，他の計算の演算量は $O\left(\frac{n^3 k}{p^2}\right)$ であるため，$\frac{k^2}{p^2}$ が小さくなるほど全体の計算時間のうち重み計算が占める割合が増えてしまう．しかし，k は小行列に対する演算のオーダー $O\left(\frac{n^3 k^2}{p^2}\right)$ を小さく抑えたいために p に対して小さな値となるよう設定したい．そのため，このトレードオフ関係を良い方向にずらすため，重み計算の実装を考えることに意味がある．しかし，Bečka らは重み計算の並列化手法については検討しているが，演算性能の向上については議論していない．そこで，Bečka らの重み計算の並列化手法と，筆者が関連論文 [b,c] において示した BLAS を利用する実装手法について説明する．また，前節で用いた二次元ブロック分割の場合における並列化手法を示す.

ここでは簡単のために，データ分散は一次元ブロック分割で行われており，また，列ブロックは初期状態の並びとなっており，第 1 ノードは A_1 と A_2 を，第 2 ノードは A_3 と A_4 などと，第 i 番目のノードが A_{2i} と A_{2i+1} を持っているものとする．Variable Blocking については，重み計算をする前に一次元ブロック分割に並び替えを行うため，このままの手法が用いられる．二次元ブロック分割については後で議論する．また列ブロックの並びについては，最終的に求められた重みを両側から並び替えることで任意の順序に対応できる.

重み計算は 2 つの行列ベクトル積に分かれており，第一のステップでは

$$h_j = A_j e \tag{5.15}$$

を計算し，第二のステップでは

$$\hat{h}_{i,j} = A_i^{\mathrm{T}} h_j \tag{5.16}$$

を計算する．最後に $\hat{h}_{i,j}$ のノルムを計算すれば $g_{i,j}$ を得る．この計算における第一の問題は並列化であり，どのノードがどの行列ベクトル積を計算するのか考えなくてはならない．並列化においては，ベクトル h_j と比べて列ブロック A_i, A_j の方がデータ量が大きいため，A_j を保持するノードで第一のステップを計算し，また，A_i を保持するノードへ h_j を移動することで第二のステップを計算するのが良い.

この計算における第二の問題は，$\hat{h}_{i,j}$ は $i < j$ が成り立つ要素のみが必要であるため，この三角構造

```
subroutine Weight-Computation([A_{1,1}, A_{1,2}, …, A_{1,w}, A_{2,1}, …, A_{2,w}, …, A_{k,1}, …, A_{k,w}], col_comm, row_comm):
        id = MPI_Comm_rank()
        r = id % k + 1
        c = id / k + 1
        e_1 = ones(col(A_{r, 2c−1}))
        e_2 = ones(col(A_{r, 2c}))
        e_1 ← e_1/‖e_1‖_2
        e_2 ← e_2/‖e_2‖_2

        h_{2c−1} = A_{r,2c−1} e_1
        h_{2c} = A_{r,2c} e_2
        [h_1, h_2, …, h_w] = MPI_AllGather([h_{2c−1}, h_{2c}], row_comm) // communicate over same row nodes

        Z_1 = [h_2, h_4, …, h_{2c−2}, h_{2c+1}, h_{2c+3}, …,h_{w−1}]
        Z_2 = [h_1, h_3, …, h_{2c−1}, h_{2c+2}, h_{2c+4}, …, h_w]
        [x_1, x_2, …, x_{w/2−1}] = A^T_{r,2c−1} Z_1
        [x_{w/2}, x_{w/2+1}, …, x_{w−1}] = A^T_{r,2c} Z_2
        [x̂_1, x̂_2, …, x̂_{w−1}] = [‖x_1‖_2^2, ‖x_2‖_2^2, …, ‖x_{w−1}‖_2^2]

        Gc = [y_1, y_2, …, y_{w−1}] = MPI_AllReduce([x̂_1, x̂_2, …, x̂_{w−1} ], MPI_SUM, col_comm)
        G = MPI_AllGather(Gc, row_comm)
```

図5.18　重み計算のアルゴリズム．引数の col_comm と row_comm はそれぞれ縦，横方向のコミュニケータ．

をうまく並列化して演算量を均等に分割することである．そこで，偶数と奇数のペアを考えることで，分割することを考える．これは i が奇数の場合は，i より小さな偶数の j と，i より大きな奇数の j について計算し，i が偶数の場合は，i より小さな奇数の j と，i より大きい偶数の j について計算するものである．例えば $w = 10$ のとき，あるノードが $i = 5$ と $i = 6$ の列ブロックを持っていたとすると，このノードは

$$\begin{bmatrix}\hat{h}_{5,2} & \hat{h}_{5,4} & \hat{h}_{5,7} & \hat{h}_{5,9}\end{bmatrix} = A_5^{\mathrm{T}}\begin{bmatrix}h_2 & h_4 & h_7 & h_9\end{bmatrix} \tag{5.17}$$

$$\begin{bmatrix}\hat{h}_{6,1} & \hat{h}_{6,3} & \hat{h}_{6,5} & \hat{h}_{6,8} & \hat{h}_{6,10}\end{bmatrix} = A_6^{\mathrm{T}}\begin{bmatrix}h_1 & h_2 & h_3 & h_8 & h_{10}\end{bmatrix} \tag{5.18}$$

を計算する．このように 1 つのノード当たり $w-1$ 個の独立のペアを $p = w/2$ ノードで計算するため，すべてのペアを重複なく計算できる．

またここからわかる通り，式 (5.15) や式 (5.18) の計算はそれぞれ行列ベクトル積と行列行列積の形でかけるため，Level-2 BLAS の xGEMV と Level-3 BLAS の xGEMM を用いると高性能に計算できる．

二次元ブロック分割の場合，行列 A_j のデータが縦に分散しているため，多少の変更が必要である．実際には，A_j の行ブロックの分割に合わせて h_j の行ブロックを分割すると，それぞれの行ブロックを並列で計算できる．また，A_i の行ブロックの分散も h_j の行ブロックの分散と一致しているため，行ブロックごとに積を計算し，最後に総和を計算すれば，$\hat{h}_{i,j}$ を得る．これをプログラムにまとめたものを図 5.18に示す．

5.3.3　収束性の理論的解析

このように Bečka の動的順序は最大重みマッチングを使うことで局所的に最も良い並列ペアを選択するが，そもそも最大重みマッチングをすることに意味があるのかどうか，それによって並列ペアを選択するが，それがどれだけ良いものなのか，わかっていない．実際の計算では 2 つの近似（重みの近似とマッチングに対する貪欲法）を行っているが，ここではそれらを無視して，正確な重みと正確なマッチングを計算すると仮定した上で，最大重みマッチングが収束性にどれだけ影響を与えるのかを検証する．

3 章の通り，行列 C のブロック要素 $C_{p,q}$ を消去したとき，C の非対角ブロックのフロベニウスノルムの自乗和が減少する．すなわち $C^{(k)}$ を $C^{(k+1)}$ に変換したとき，

$$\sum_{i\neq j}\left\|C_{ij}^{(k+1)}\right\|_{\mathrm{F}}^{2}=\sum_{i\neq j}\left\|C_{i,j}^{(k+1)}\right\|_{\mathrm{F}}^{2}-2\left\|C_{p,q}^{(k)}\right\|_{\mathrm{F}}^{2}. \tag{5.19}$$

そこで，全体の非対角ブロックのフロベニウスノルムの自乗和に対する消去したブロック要素のフロベニウスノルムの自乗の比を r を

$$r\equiv\frac{\left\|C_{p,q}^{(k)}\right\|_{\mathrm{F}}^{2}}{\sum_{i<j}\left\|C_{i,j}^{(k+1)}\right\|_{\mathrm{F}}^{2}} \tag{5.20}$$

と定義すると

$$\sum_{i\neq j}\left\|C_{ij}^{(k+1)}\right\|_{\mathrm{F}}^{2}=(1-r)\sum_{i\neq j}\left\|C_{i,j}^{(k)}\right\|_{\mathrm{F}}^{2}. \tag{5.21}$$

よって，r の下限を求めることによって一次収束の係数 $1-r$ の上限を求めることができる．

最大重みマッチングにおいて，非対角ブロックのフロベニウスノルムの自乗和は枝の重みの和の 2 倍 $2\sum_{(i,j)\in G}\hat{g}_{i,j}$ に一致し，消去したブロック要素のフロベニウスノルムの自乗和は，マッチングにおける枝の重みの和の 2 倍 $2\sum_{(i,j)\in M_{\max}}\hat{g}_{i,j}$ に一致する．そこで，あるグラフ X における枝の重み和 $S(X)$ を次のように定義する：

$$S(X)=\sum_{(i,j)\in X}\hat{g}_{i,j}. \tag{5.22}$$

このときグラフ G における最大重みマッチング $M_{\max}$ について $r=S(M_{\max})/S(G)$ を計算することが問題である．そこで以下の定理を証明する．

定理 5.3. グラフ G の枝の重みは $\hat{g}_{i,j}\geq 0$ を満たし，G の頂点数 w は偶数であると仮定する．$M_{\max}$ がグラフ G における最大重みマッチングであるとき，

$$r=\frac{S(M_{\max})}{S(G)}\geq\frac{1}{2w-3}. \tag{5.23}$$

ただし $S(G)=0$ のときは $r=1$ と定義する．

証明. グラフ G は不足する枝を重み零の枝で補い，完全グラフになっていると仮定する．この証明では，貪欲法によって G のマッチングを計算することによって厳密解の下限を求める．

いま頂点数 $w=2$ のとき，$S(M_{\max})=S(G)=\hat{g}_{1,2}$．よって $r=1\geq\frac{1}{2w-3}$．次に頂点数 2 から $w-2$ まで上式が成り立つと仮定する．このとき頂点数 w のグラフ G_w に対して貪欲法によって重み最大の枝を選択したとする．この枝を (i,j) とし，残余グラフ $G_{w-2}=G_w\backslash(i,j)$ とおき，差グラフ $\bar{G}_w=G_w-G_{w-2}$ とおく．G_w は完全グラフであるため，$\bar{G}_w$ の枝数は $2w-3$ である．$\hat{g}_{i,j}$ は G_w 全体における最大重みの枝であるため，$\hat{g}_{i,j}=0$ の場合，$S(G_w)=0$ である．そうでない場合，$\hat{g}_{i,j}$ は $\bar{G}_w$ における最大重みの枝でもあるため，

$$\frac{\hat{g}_{i,j}}{S(\bar{G}_w)}\geq\frac{1}{2w-3}. \tag{5.24}$$

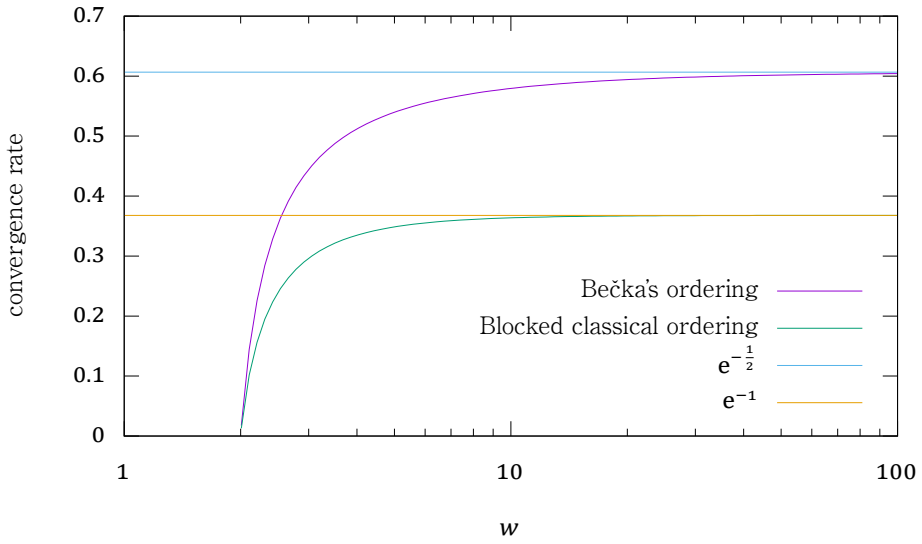

図5.19　Bečka の動的順序とブロック版古典的ヤコビ法の収束率の上限の比較

そこで $\frac{S(\tilde{G}_w)}{S(G_w)} = t$ とおいたとき，帰納法の仮定を用いると

$$r = \frac{t}{2w-3} + \frac{1-t}{2w-7}. \tag{5.25}$$

これは $t = 1$ のとき最小値をとるため，$r \geq \frac{1}{2w-3}$. □

よって r の下限を計算できた．またこの証明では貪欲法によって真の解の下限を得ているため，同じ結果が貪欲法においても成り立つ．

この r の下限は，並列化の効果を表している．並列化を行わない動的順序の場合，つまり，1 ステップごとにフロベニウスノルム最大の非対角ブロックを消去するブロック版古典的ヤコビ法における消去ブロックの比は $r_{\mathrm{bc}} = \frac{2}{w(w-1)}$ である．そこで両者を比較するため 1 巡回における収束率の極限を調べると，Bečka のオーダリングの場合，

$$\lim_{w\to\infty} (1-r)^{w-1} = \mathrm{e}^{-\frac{1}{2}}. \tag{5.26}$$

一方，ブロック版古典的ヤコビ法の場合，

$$\lim_{w\to\infty} (1-r_{\mathrm{bc}})^{\frac{w(w-1)}{2}} = \mathrm{e}^{-1}. \tag{5.27}$$

よって両者は反復数 2 倍分だけ異なることになる．言い換えると，それぞれにおける最悪のケース同士を比較すると，Bečka の動的順序は相手の 2 倍の計算量になる代わりに並列化可能である．実際には，w が現実的に小さな数値である場合には収束率の差はもっと縮まる．図 5.19にそれぞれの 1 巡回における収束率の上限を示しているが，w が小さいところでは差が縮まっていることがわかる．

5.3.4　ペアの割り当て手法

動的順序では，並列ペアの選択が行列に依存して任意に変化する．そのため，列ブロックペアの交換における通信パターンが固定化されない．これは巡回ヤコビ法向けの並列オーダリングが，通信パターンも考えて設計されているのとは対照的である．Bečka らが使っていた Variable Blocking ではデータをノードに固定していたため，これが問題になることはなかったが，一次元ブロック分割や二次元ブロック分割を用いるのならば，どの並列ペアをどのノードに割り当てれば通信が効率的になるのかを選択しなければならない．ここでは，列ブロックペアの交換における通信ホップ数を削減する手法を考える．通信速度を考えるには通信の衝突なども重要な要因となり，すなわち，全体の通信手順を考えなければ正確な予測は不可能であるが，通信ホップ数を削減できれば結果として通信の衝突も減らせるため，ここでは簡単に計算可能な通信ホップ数のみを対象とする．

ここで今の並列実行可能なペアの集合を Ω_c, 列ブロックの添え字の集合 $\mathcal{P} = \{1, 2, \dots, w\}$，ノード番号の集合 $\mathcal{N} = \{1, 2, \dots, w/2\}$, 並列ペアのノードへの割り当てを $\pi_\mathrm{c} : \Omega_\mathrm{c} \mapsto \mathcal{N}$ とおく．またあるノード $\varpi \in \mathcal{N}$ から別のノード $\varrho \in \mathcal{N}$ へ列ブロックを移動するときのコストを $c(\varpi, \varrho) : \mathcal{N}^2 \mapsto \mathbb{R}$ と定義する．このコストの定義は，行き先・送り元のみに依存するため，ホップ数のようなコストを表現できるが，通信の衝突のような，他の通信に依存するものなどは表現できない．しかし，通信網の構造には依存しないため，このようなホップ数を定義できる通信網であればどんなものにでも適用できる．

Ω_c は動的順序で求められたペアを想定しており，$\Omega_\mathrm{c} = \{(i_1, j_1),\ (i_2, j_2)\ \cdots\ (i_w, j_w)\}$ の形をしている．また，1 つのペアは必ず 1 つのノードに割り当てられ，どちらにもあまりがないため，valid な割り当て π_c は逆像 π_c^{-1} を持つ．

以上を用いてある列ブロックがどのノードに割り当てられているかを示す関数 $\bar{\pi}_\mathrm{c} : \mathcal{P} \mapsto \mathcal{N}$ を定義する．すなわち，ある i に対して j が存在して $\pi_\mathrm{c}((i,j)) = \varpi$ であった場合，$\bar{\pi}_\mathrm{c} = \varpi$ であり，逆に，ある j がに対して i が存在して $\pi_\mathrm{c}((i,j)) = \varrho$ であった場合，$\bar{\pi}_\mathrm{c} = \varrho$ と定義する．

いま新しいペアの集合 Ω_n が与えられたとき，このペアのノードに対する割り当て全体の集合を $\Pi_\mathrm{n} = \left\{\pi_\mathrm{n} \middle| \pi_\mathrm{n} : \Omega_\mathrm{n} \mapsto \mathcal{N} \text{ and } \pi_\mathrm{n}^{-1} \text{ exits}\right\}$ と定義すると，ある割り当て $\pi_\mathrm{n} \in \Pi_\mathrm{n}$ のコスト $c(\pi_\mathrm{n})$ を

$$\hat{c}(\pi_\mathrm{n}) = \sum_{(i,j)\in\Omega_\mathrm{n}} c\left(\pi_\mathrm{n}\left((i,j)\right), \bar{\pi}_\mathrm{c}(i)\right) + c\left(\pi_\mathrm{n}\left((i,j)\right), \bar{\pi}_\mathrm{c}(j)\right) \tag{5.28}$$

と定義できる．そこで，最適な割り当てを探索する問題を

$$\pi_\mathrm{opt} = \mathrm{argmin}_{\pi\in\Omega_\mathrm{n}} c(\pi) \tag{5.29}$$

と定義できる．

これは割り当て問題であり，言い換えると，二部グラフ上の最小重み完全マッチング問題である．割り当て問題には，Blossom 法のほかに，二部グラフの構造を用いたハンガリアン法と呼ばれるアルゴリズムが知られており，これは $O(w^3)$ の計算量となる [125]. そこで，具体的にこの問題を解くことが可能である．当然，マッチング問題であるため，最大重みマッチングにおいても用いた貪欲法が適用できるが，Avis [124, Theorem 1] や Reingold [126] によれば，最小重み完全マッチング問題における貪欲法による解の精度，つまり，真の解の重み和 (C_min) に対する貪欲法の重み和 (C_GR) の比率は，重み

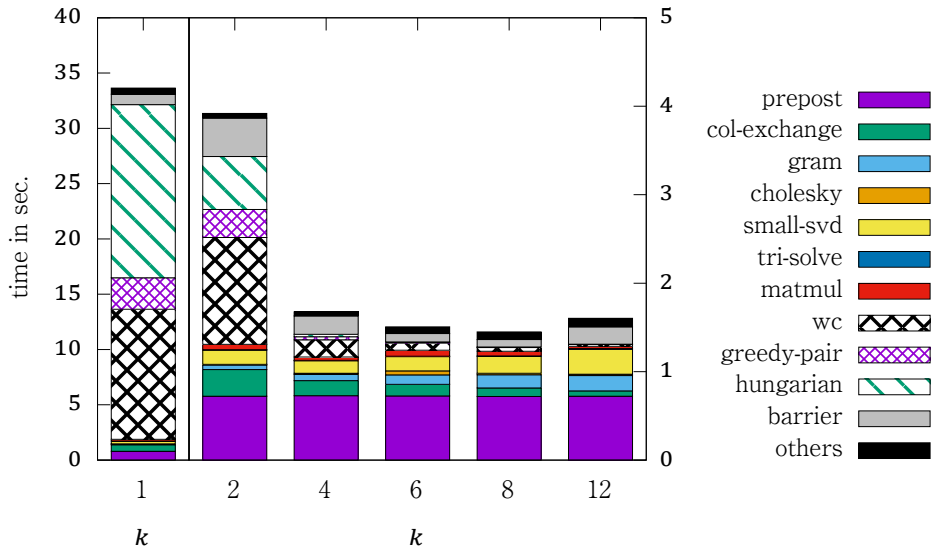

図5.20　SGI Altix ICE の 144CPU を用いて Bečka の動的順序を用いた片側ブロックヤコビ法の実行時間を測定したときの，実行時間の内訳．行列は図 5.14と同じ．$k = 1$ の結果は左軸，それ以外のものは右軸を用いている．

が三角不等式を満たしかつ頂点が偶数個の場合，

$$\frac{C_{\mathrm{G}}}{C_{\min}} \leq \frac{4}{3}\,(w/2)^{\log_2 1.5} - 1, \tag{5.30}$$

となり，しかもこの上限を達成する例が存在する．これは，定数で下限を抑えられる最大重みマッチングとは対照的である．そこで割り当て問題においては近似解法を用いず厳密解法であるハンガリアン法を用いることにする．

コスト $c(\varpi, \varrho)$ の定義は重要である．ホップ数を最小化したいという考えでは，コストをホップ数と定義し，その最大値の最小化を行うことになるが，これでは上の割り当て問題に当てはめられず，また，通信の衝突を削減することを考えれば，ホップ数最大のもののみを見て，他のものをおろそかにすることは，実際の通信速度を劣化させることになる．一方，ホップ数の和の最小化とした場合だと，ホップ数が大きなものと小さなものが混在できてしまう．そこで，それらの中間を目指す意味で，ホップ数の自乗をコストとするものが適切だと考えられる．

5.3.5　性能検証

この節の最後に，Bečka の動的順序を用いたときの性能測定結果を示す．ここで用いたプログラムでは，重みの計算を，単純なフロベニウスノルムの近似ではなく，Bečka が用いている，列ブロックを正規化したものに対する近似を行っている．また，重みは近似による誤差によって一定以上 0 に近づかないため，`tol` の値を，巡回順序を用いたときの値 $\sqrt{n}\mathbf{u}$ から $\sqrt{nl_{\mathrm{abs}}}\mathbf{u}$ に変更している．図 5.20と図 5.21は，前節と同様に SGI Altix ICE を用いて片側ブロックヤコビ法を実行したときの，実行時間の内訳である．前節の結果と違い，動的順序を用いているため，重み計算や貪欲法によるマッチング，ハンガリアン法による割り当ての時間が追加されている．そのため，$k = 1$ のときにおける実行時間

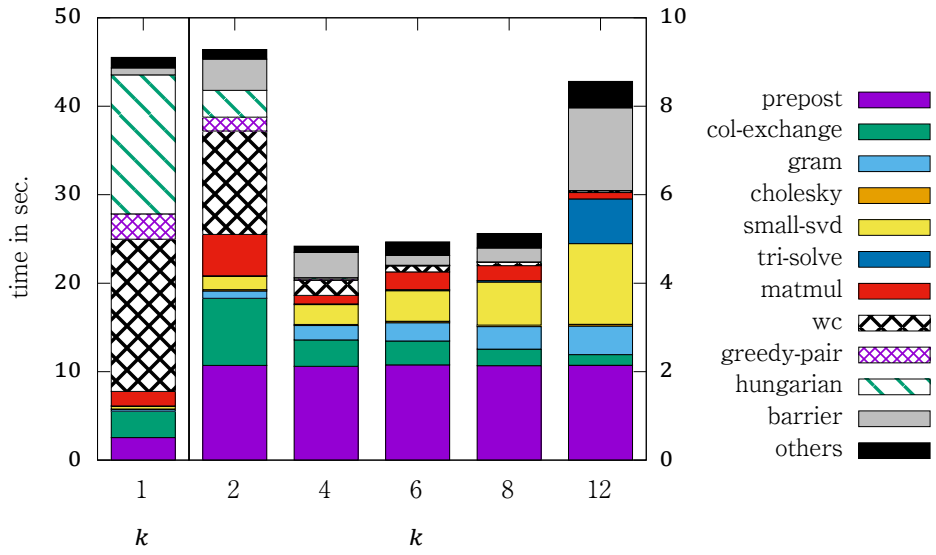

図5.21　SGI Altix ICE の 144CPU を用いて Bečka の動的順序を用いた片側ブロックヤコビ法の実行時間を測定したときの，実行時間の内訳．行列は大きさが $m = n = 8,640$ となっていること以外，図 5.14と同じ．$k = 1$ の結果は左軸，それ以外のものは右軸を用いている．

表5.6　Modified Modulus 順序と Bečka の動的順序の場合における縦方向の並列数 k と反復回数

行列サイズ (n)	手法	$k = 1$	$k = 2$	$k = 4$	$k = 6$	$k = 8$	$k = 12$
4,320	MMO	2,016	1,008	432	288	180	120
	Bečka	961	464	225	145	106	70
8,640	MMO	2,016	1,008	432	288	180	120
	Bečka	966	463	224	147	108	71

が増大したため，$k = 1$ の場合のみ左軸を用い，他は右軸を用いている．

この結果から，k が小さい場合は動的順序で追加された計算，特に重み計算とハンガリアン法による割り当ての時間が大きな時間を占めてしまっていることがわかる．貪欲法によるマッチングはそれらよりも小さいが，これは近似解法を用いているためであり，ハンガリアン法と同じオーダーとなる Edmond の Blossom 法によって厳密解を求めた場合はハンガリアン法と同程度の時間となることが予想される．しかしこれらの計算時間は k が増大するに従い劇的に減少する．これは予測された結果であり，理論計算量に従うと，重み計算の計算量は k に反比例し，貪欲法によるマッチングは k^2 の反比例よりも速く減少，ハンガリアン法は k^3 に反比例するためである．

表 5.6は動的順序を用いたときの並列ステップ数である．ただし比較のため MMO の並列ステップ数についても再掲している．MMO のものと比べると，約半分のステップ数となっており，動的順序の効果がわかる．しかしながら，全体の実行時間自体はあまり変化しておらず，$n = 4,320$ のとき，MMO では 1.76 秒であったが，動的順序の場合，1.45 秒，$n = 8,640$ のとき，MMO では 5.75 秒であったが，動的順序の場合，4.84 秒であった．これは，前処理・後処理の実行時間は両者で変化しないことや，重み計算などの追加計算があること，そして，列ブロック交換における実行時間が増加したことによる．

表5.7　Modified Modulus 順序と Bečka の動的順序における 1 並列ステップあたりの列ブロック交換の実行時間 (msec)

行列サイズ (n)	手法	$k=1$	$k=2$	$k=4$	$k=6$	$k=8$	$k=12$
4,320	MMO	0.357	0.445	0.353	0.331	0.584	0.362
	Bečka	0.622	0.650	0.754	0.909	0.900	0.865
8,640	MMO	2.39	2.60	1.80	2.40	1.99	2.03
	Bečka	3.08	3.27	2.68	3.66	3.48	3.40

表 5.7は MMO と Bečka の動的順序における，1 並列ステップ当たりの列ブロック交換にかかった時間である．実行時間が短いため，単位はミリ秒としている．この表からわかる通り，動的順序によって列ブロック交換にかかる時間は増大し，最大で 3 倍弱の時間となっていることがわかる．

以上の通り，Bečka の動的順序について，収束性の理論的解析を行い，また，Level-3 BLAS を活用した重み計算の実装や，二次元ブロック分割に適した計算手法と，列ブロックのノードへの割り当て手法を開発し，実行時間の比較とその内訳を検証した．その結果，縦方向の並列数 k が小さい場合，動的順序に必要な新たな計算の負荷によってむしろ遅くなるが，最適な k を用いた場合における実行時間では MMO を上回る性能となった．将来的にノード数が増大した場合には，重み計算の計算速度の改善や，ハンガリアン法の並列化，近似解法の適用などの改良が必要になると推測される．

第 6 章

部品の性能評価と実装技術

この章では，将来の高性能計算機へ向けた実装を考えるため，片側ブロックヤコビ法に用いられる部品の内，Gram 行列の計算を取り出して議論する．Gram 行列の計算は前章の通り，片側ブロックヤコビ法におけるオーダーの意味で支配的な 2 つの計算の内 1 つであるため，行列サイズ n やノード数 p がより大きくなった場合には重要となるものと考えられる．Gram 行列の計算は BLAS で定義された行列積のパターンの 1 つである xSYRK であり，通常の行列積 xGEMM と同等の性能が期待されるが，xSYRK の持つ対称構造のために性能効率が低く，前章の結果の中では，2 倍の演算量である列ブロックの更新と同程度の実行時間となってしまっている．これは，将来の計算機に頻繁に利用されると考えられるメニーコア CPU においてはより顕著となる．

そこでここでは xSYRK について，メニーコア CPU の 1 つである Xeon Phi (Knights Corner, KNC) への実装手法と，性能向上手法について議論する．KNC は，単体では遅い速度のコアを多数集積することで，全体としては高い性能を実現するものであり，1 つのコア当たりの速度は倍精度において約 17GFlop/s であるが，これを 60 個程度，集積することで，1 プロセッサでは約 1TFlop/s の高い理論ピーク性能を持つ．このように多数のコアを持つため，KNC において高い性能を実現するためには，高い並列性を持つプログラムが必要であり，xSYRK の実装においてもこのことを考慮しなければならない．

KNC に対する行列積 (DGEMM) の実装手法については Smith ら [127, 128] が研究しており，BLIS という名前を持つ KNC などのメニーコア向け BLAS 実装も公開しているが，xSYRK については十分な研究がされていない．そこで，Smith らの研究や，他にマルチコアプロセッサ向けの Level-3 BLAS 実装についての後藤ら [129] の研究をベースに，KNC への DSYRK の並列化手法の実装と，性能解析を行う．またここでは倍精度向けの DSYRK について議論するが，並列化手法やキャッシュ利用手法などは他の精度や複素数においても役立つものと考えられる．

この章の内容は関連論文 [d] に基づくものであり，DSYRK において 3 つの並列化軸すべてを用いた並列化手法を示し，実際に KNC 上で DSYRK の高速化を実現している点が新規である．また，将来的に DSYRK を片側ブロックヤコビ法に用いる際に役立つものだと考えられる．

6.1 Gram 行列の計算

DSYRK は Fortran 形式で格納された倍精度実数の $m \times n$ 行列 A について

$$C = AA^{\mathsf{T}} \tag{6.1}$$

を計算するものである．DSYRK は別の計算パターンとして $C = A^{\mathsf{T}}A$ を計算することもできるが，本稿ではこの形のものについて考える．また，C の上三角・下三角部分のどちらを計算するか，を選択することもできるが，本稿では上三角部分のみを計算するものを考える．この式からすぐに C は $m \times m$ 行列であって，$C = AA^{\mathsf{T}} = \left(AA^{\mathsf{T}}\right)^{\mathsf{T}} = C^{\mathsf{T}}$ が成り立つことがわかる．しかし，DSYRK は結果が対称となることを除けば単なる行列積であり，DGEMM と多くの点で同じ議論ができる．

6.1.1 ブロック行列積

高性能な行列積の実装には並列化だけでなく，キャッシュブロック化や，データアクセス性能向上のためのデータパッキング，SIMD 命令などのプロセッサの機能を活用することが不可欠である．これらを意識した実装をするうえでベースとなる考え方は，行列積をブロック行列積として書き直すことである．

いま $C = \{Z_{i,j}\}$，$A = \{X_{i,k}\}$，$A = \{Y_{j,k}\}$ とブロック分割する．簡単のためそれぞれのブロックの大きさは，$Z_{i,j}$ が $b_i \times b_j$，$X_{i,k}$ が $b_i \times b_k$，$Y_{j,k}$ が $b_j \times b_k$ のように固定とする．このとき，$C = AA^{\mathsf{T}}$ は

$$Z_{i,j} = \sum_k X_{i,k} Y_{j,k}^{\mathsf{T}} \tag{6.2}$$

と書ける．つまり全体の行列積を部分行列積の繰り返しとして書ける．ブロック幅は任意に設定できるため，データ量をキャッシュサイズに合った大きさにしたり，並列化のときに演算量を均等化したりすることができる．また部分行列積もまた再帰的にブロック行列積として書くことができるため，さらに小さなブロックに分割することで，SIMD レジスタ幅に合わせた大きさにすることもできる．

ここで，部分行列積における，メモリからキャッシュへのデータ移動量と演算量を計算する．いま，部分行列積における部分和の計算の 1 つ

$$Z_{i,j} \leftarrow Z_{i,j} + X_{i,k} Y_{j,k}^{\mathsf{T}} \tag{6.3}$$

における，メモリからキャッシュへのデータ移動量は $8(b_i b_j + b_i b_k + b_j b_k)$ Byte であり，キャッシュからメモリへのデータ移動量は $8 b_i b_j$ Byte である．また，演算量は $2 b_i b_j b_k$ Flop である．よって，部分行列積におけるメモリ・キャッシュ間データ移動量と演算量の比（B/F 値）を B とおくと，

$$B = \frac{4}{b_i} + \frac{4}{b_j} + \frac{8}{b_k} \tag{6.4}$$

となり，キャッシュのサイズが許す限りブロック幅を大きくすることで，B/F 値を小さくすることができる．実際のブロック幅はキャッシュサイズからおおむね決定される上限の大きさを決めておいて，端数処理や並列化のために，ブロックごとに異なるサイズをとれるようにする．

6.1.2 ブロック行列積の手順

いま，C が $n_i \times n_j$ ブロックであり，また，A の列ブロック数が n_k であるとする．これはブロック幅が固定の場合，それぞれ $n_i = \lceil m/b_i \rceil$，$n_j = \lceil m/b_j \rceil$，$n_k = \lceil m/b_k \rceil$ である．このとき，ブロック行列積全体は図 6.1のようなプログラムとして書くことができる．

`blocked-matmul` の 3 つ目の j に関するループは，C の上三角部分 (tri(C)) に重なるもののみを処理するために，開始位置が変化する．そのため，j ループの開始位置を常に 1 とすれば，このプログラム

```
subroutine blocked-matmul({Z_{i,j}}, {X_{i,k}}, {Y_{j,k}})
        Z_{:,:} ← O
        do i = 1:n_i
                do k=1:n_k
                        do j = 1:n_j  if Z_{i,j} overlaps with triu(C)
                                Z_{i,j} ← Z_{i,j} + X_{i,k} × Y^T_{j,k}
                        end
                end
        end
```

図6.1　ブロック行列積の擬似プログラム

```
subroutine blocked-matmul2({Z_{i,j}}, {X_{i,k}}, {Y_{j,k}}):
        Z_{:,:} ← O
        do i = 1:n_i
                do k=1:n_k
                        X̄ ← X_{i,k}
                        do j = 1:n_j  if Z_{i,j} overlaps with triu(C)
                                Ȳ ← Y_{j,k}
                                Z_{i,j} ← Z_{i,j} + X̄ × Ȳ
                        end
                end
        end
```

図6.2　データパッキングを追加したブロック行列積の擬似プログラム．

を DGEMM の計算に用いることができる．また補足として，本稿での実験では $m \leq b_i$ の範囲でしか性能測定していないため，このプログラムの最外側ループは 1 度のみしか実行しない構造となっている．そこで本稿では最外側ループを無視した形での議論をしているが，$m > b_i$ の領域を考える場合には，以下の議論を多少拡張する必要がある．

`blocked-matmul` はメモリ上の Fortran 配列として格納された行列データに直接アクセスするが，CPU によっては CPU に合わせたデータ順序でなければ高い性能を引き出せないことがある．そこで，最内側処理のデータアクセス順序に合わせてデータの順序を並び替える処理（データパッキング）を追加したものが図 6.2のプログラムであり，$X_{i,k}$ と $Y_{j,k}$ のパッキングを実装している．

6.1.3　データパッキングの効果

データパッキングを行うと，データアクセスが増える分のコストが発生する．そのため，データ順序が適切になったことによる内部処理の性能向上がデータパッキングのための性能劣化を上回らなければ，実装する価値がない．一方，パッキングを行ったデータはその直後に使用されるため，$\bar{X}$ や $\bar{Y}$ をキャッシュにのる大きさにすれば，性能劣化を小さく抑えることができる．そこで，データパッキングにおいては，パッキングのコストと，パッキングのためのデータ量が重要である．

まず全体の計算コストに対するパッキングの相対的なコストを考えるため，パッキングしたデータの再利用回数を計算する．`blocked-matmul2` でのパッキングでは，$\bar{X}$ は 1 要素当たり最大 m 回，再

利用されるが，$\bar{Y}$ は 1 要素当たり $\min(b_i, m)$ 回，再利用される．またパッキングのためのデータ量を計算すると，$\bar{X}$ を保存するために $8b_ib_k$ Byte，$\bar{Y}$ を保存するために $8b_jb_k$ Byte の領域が必要である．よって，b_j は $\bar{Y}$ が L2 キャッシュのような高い階層のキャッシュにのるように設定し，一方で b_i は $\bar{Y}$ の再利用回数を増やすため，大きな値とすることになり，$\bar{X}$ はラストレベルキャッシュ，もしくはメモリ上に置くことになる．

実際に KNC に対する実装で用いた値は $b_j = 144$，$b_i = 40,000$ である．これらの値は Smith ら [128] の実装における $b_j = 120$，$b_i = 14,400$ と異なるが，b_j は後述の，レジスタブロック幅の違いと手動チューニングの結果によるものである．b_i は，後に示す内積方向分割のときに，$\bar{Y}$ を分割して利用するため，分割したときでも十分な大きさとなる値としている．b_i の値を $14,400$ と $40,000$ とのどちらの値に設定したとしても，$\bar{X}$ のデータ量は KNC のキャッシュに収まらない大きさとなり，$\bar{X}$ はメモリ上に置くことになる．このため，$\bar{X}$ をパッキングするコストは $\bar{Y}$ のものと比べて大きくなる．そこで，$\bar{X}$ については，パッキングをする場合とそうでない場合の性能差が興味深い．

6.1.4 ブロック行列積の並列化

`blocked-matmul2` に対する並列化の議論は Smith ら [128] によって詳しく解説されている．まず，最内側処理である部分行列積（$Z_{i,j} \leftarrow Z_{i,j} + \bar{X} \times \bar{Y}$）は，$b_i$ が十分に大きいため十分な演算量があり，並列化に適している．また，j に関するループは，b_j が小さな値であり，ループ回数が大きいため，並列化に適している．一方，k に関する 2 番目のループは，b_k が小さな値であるためループ回数は大きいが，総和のための同期の問題や，スレッドごとに $\bar{X}$ の領域を確保する必要があるなどの問題があるため，あまり並列化に適していない．i に関する最外側ループは，$\bar{X}$ の領域に関する問題があり，また，最内側処理の並列化と並列化軸が同じであるため，積極的に採用する理由がない．よって Smith らは Xeon Phi に対しては，再内側処理と j に関するループの並列化を行っている．本研究ではこれを DSYRK に拡張すること，さらに，k に関するループの並列化との組み合わせを考える．

6.2 Knights Corner に対する部分行列積の実装

6.2.1 Knights Corner の詳細

KNC については Jeffers [130] に概要はあるが，詳細なアーキテクチャに関しては Intel の命令リファレンス [131] における実行可能な命令の一覧や Rahman [132] による命令スケジューリングの解説などを理解し，現実の状況に照らし合わせることでおおむね理解できる．ここでは行列積の実装のために必要なことに焦点をあて，簡単に説明する．

KNC は，57 以上のコアを持つメニーコアプロセッサであり，512bit 幅の SIMD 演算（FMA を含む）を毎サイクル行う演算器を持ち，そして，1 コアあたり 4 つのスレッドを 1 サイクルごとに切り替えて実行する機構を持つ．

KNC は多数のコアが 1 つの Bus につながれており，キャッシュコヒーレンシが自動的にとられる．そのため，通常のマルチコア CPU のように使用できる．それぞれのコアが L1I/L1D キャッシュと L2 キャッシュをもち，Bus につながれる Memory Controller には GDDR5 メモリが接続されている．参考のため，Fang ら [133] による，Xeon Phi 5100 シリーズにおける，各種メモリ性能を測定した結果を表 6.1に示す．今回用いた Xeon Phi 3120A は表の Xeon Phi 5100 シリーズと比べて，メモリーチャネ

表6.1　Xeon Phi 51xx のメモリ性能．Fang らの結果 [133, Table 2] を抜き出したものである．

	Latency	Bandwidth	Capacity
L1D cache	3 cycle	64 Bytes/cycle	32KB
L2 cache	24 cycle	12 GB/s	512KB
Remote L2	250 cycle		
Memory	302 cycle	164 GB/s	

ルの幅が 3/4 倍であり，理論ピークメモリバンド幅も約 3/4 倍となっているため，メモリバンド幅は表で示した値よりも小さな値となるが，それ以外の値は参考になる．

要点を書くと，コア内部にあるキャッシュメモリへのアクセスは高速であるが，コアの外部にあるデータへのアクセスは一様に遅い．そのため，コア間で共有するデータが少ないようにプログラムすることが重要である．

SIMD 演算は，8 要素の SIMD レジスタを 32 本持つ構成となっており，1 サイクルに最大 1 つの SIMD 積和演算命令を実行できる．そのため，1 サイクルで最大 16 演算を行うことができる．SIMD データ移動命令ではデータのアライメントがそろっている場合は 1 つの命令で 8 要素のデータをロードまたはストアすることができる．アライメントがそろっていない場合，2 つのキャッシュラインをまたぐデータ移動となるため，2 つのデータ移動命令を行う必要があり，性能低下の要因となる．KNC のパイプラインは 2 本（uv-pipe）あるが，一部の SIMD 命令を除き片方のパイプライン（u-pipe）にしか流せないため，SIMD 積和命令の機能をうまく活用して，少ない命令数で行列積を作らなければならない．

KNC の命令デコーダーは 1 コアにつき，2 つ以上 4 つまでのスレッドが実行されることを前提としており，1 スレッドのみでは最大の命令供給性能（2 命令/cycle）の半分の性能しか出せない．そこで，1 コアに 2 つ以上のスレッドを走らせなければならないが，これらのスレッドでコア内のキャッシュを共有することになる．そのため，コア内のキャッシュにはこれらのスレッド間で共有するデータを入れるようにすると，キャッシュをスレッドで分断することがなくなり，都合がよい．一方，コア外部のキャッシュについては，複数のコアでデータを共有すると，同じデータがそれらのコアのキャッシュに配置されるため，コア間でデータを共有することに利点がない．よってコア内部での並列化とコア間での並列化とでは共有データの違いを意識する必要がある．

6.2.2　計算カーネルの設計

計算カーネルの設計方法については，BLIS の設計の中で語られており，また，インラインアセンブラのコードが公開されている [134]．より詳しくは，Intel の研究者らによる計算カーネルの解説がある [135]．

$Z_{:,:}$，$X_{:,:}$，$Y_{:,:}$ についての部分行列積をさらに分割して，SIMD 演算に適した大きさにする．SIMD レジスタは 8 要素，32 本あるため，ワークレジスタの本数を最小限にし，部分行列を SIMD レジスタ上

```
subroutine kernel8x24(w, u, v):
        r(1:8, 1:24) ← O
        do k = 1:b_k
                s ← u(1:8, k)
                do j = 1:24
                        r(1:8, j) ← r(1:8, j) + v(k, j) × s
                end
                w ← w + r
        end
```

図6.3　計算カーネルの擬似プログラム.

に乗せることで，メモリ・キャッシュとレジスタ間のデータ移動命令を最小限にする．ここでは，

$$Z_{\cdot,\cdot} = \left[\begin{array}{c|c|c|c} w_{1,1} & w_{1,2} & \cdots & w_{1,s_j} \\ \hline w_{2,1} & w_{2,1} & \cdots & w_{2,s_j} \\ \hline \vdots & & \ddots & \vdots \\ \hline w_{s_i,1} & \cdots & & w_{s_i,s_j} \end{array}\right] \tag{6.5}$$

$$X_{\cdot,\cdot} = \left[\begin{array}{c|c|c|c} u_1^\mathsf{T} & u_2^\mathsf{T} & \cdots & u_{s_i}^\mathsf{T} \end{array}\right]^\mathsf{T} \tag{6.6}$$

$$Y_{\cdot,\cdot} = \left[\begin{array}{c|c|c|c} v_1 & v_2 & \cdots & v_{s_j} \end{array}\right] \tag{6.7}$$

のように分割し，$w_{i,j}$ を SIMD レジスタ上に置く方法を採用する．$w_{i,j}$ を 8×24 行列とすると，$w_{i,j}$ の 1 列に 1 つのレジスタを用いることで，24 本の SIMD レジスタで表現できる．このとき計算の主要部分は $8 \times b_k$ 行列 u_i と $b_k \times 24$ 行列の v_j との積 $w_{i,j} \leftarrow u_i \times v_j$ であり，これを KNC に最適化する．

図 6.3のプログラムは KNC 上で高速な $w_{i,j} \leftarrow u_i \times v_j$ の計算手順である．このプログラムは，外側ループが 1 つ進むたびに u の 1 列と v の 1 行を読み込んで，r のすべての要素を更新する形となっている．r をレジスタ上に格納するため，内側ループは固定長になっており，完全なループアンロールを行う．最内側処理は，r の 1 列（1 つのレジスタ）に，s（1 つのレジスタ）を $v(k,j)$（メモリ・キャッシュ上の 1 要素）でスカラー倍したものを足しこむ操作となっている．これは KNC の命令`vfmadd231pd`のアドレッシングのオプションを使えば，1 命令で実行できる．よって，内側ループは 24 命令で 24 回の積和を行うことになる．そして，外側ループのループ制御命令などをうまく配すれば，外側ループの 1 ステップあたり 26 サイクルで実行できる．よって外側ループの実行効率は，すべての命令が 1cycle ごとに投入される最も理想的な状態においても最大で $24/26 \approx 0.923$ となる．実際には，w への結果の足しこみ処理や，データパッキングのような計算カーネル以外の処理も行わなければならないため，DSYRK 全体の実行性能は 0.923 を上限としてこれよりも小さな値となる．

レジスタのうち w の格納以外に 2 本のレジスタを使うため，内側ループの長さは最大で 30 まで伸ばすことができる．逆に内側ループ長の短いものも作ることで端数処理に対応できる．BLIS では幅 30 のもののみを実装しているが，本研究では，縦と横の大きさの比が単純となり実装が簡単となるため，8 の倍数の幅のもの（8, 16, 24）を作成した．

6.2.3　キャッシュブロック化

```
subroutine subblock-matmul({w_{i,j}}, {u_i}, {v_j}):
        do i = 1:b_i/8
                do j = 1:b_j/24
                        kernel8x24(w_{i,j}, u_i, v_j)
                end
        end
```

図6.4　計算カーネルを用いた部分行列積の擬似プログラム．

前節の計算カーネルを用いると，部分行列積は図 6.4のようなプログラムとなる．実際にはこの 2 重ループのうち内側ループは上三角の構造に合わせて開始位置を調整し，計算量を抑える．このプログラムにおいて u_i は内側ループで繰り返し使われる．$\bar{X}$ のパッキングを行わない場合は，この u_i へのアクセスは行列 A へ直接アクセスすることになる．v_j は外側ループで繰り返し使われるため，キャッシュに乗せる．前者の格納に必要なデータ量は $64b_k$ Byte であり，後者は $8b_jb_k$ Byte である．よってデータ量だけを考えて後者を L2 キャッシュに収めるようなブロック幅を計算すると，b_j, b_k を約 181 にすればよいが，計算カーネルのブロック幅や，b_j と b_k の性能への影響の違い，キャッシュ制御機構の影響を考慮したうえで最適な b_j, b_k を理論的に求めることは困難である．そこで，手動でチューニングした結果，性能の良かった $b_j = 144, b_k = 200$ を用いる．この場合，v_j のデータ量は 225KB，u_i は 12.5KB となり，u_i は十分 L1D キャッシュに収まる大きさとなるが，次に示すコア内での並列化のため，u_i も L2 キャッシュに収めるようにする．

このように部分行列積ではキャッシュの使用量を考えるため，キャッシュを共有するコア内部での並列化を同時に取り扱うと都合がよい．subblock-matmul において，外側ループを並列化すると，u_i は共有できないが，v_j は共有でき，L2 キャッシュのデータの共有ができる．この並列化においては，内側ループの長さの違いや，スレッド間の多少の実行時間の違いを吸収するため，外側ループ番号の i をアトミック変数として動的な並列化を行う．

6.3　DSYRK の並列化

以上の通りレジスタ・キャッシュブロック化やデータパッキングを行うので，DSYRK の並列化では，ブロックサイズやコア間での共有データを考慮する．

行列積の並列性には 2 種類あり，1 つは結果の行列 C の要素ごとの並列性であり，もう 1 つは内積計算の並列性である．前者は処理間に依存性がない並列性であるが，データパッキングをする場合には結局同期が必要となる．また，C の上三角部分のみ計算するため，分割したときに処理のバランスをとることが難しい．後者を並列化した場合には，どのタイミングで同期をとり総和を計算するかが問題となる．また，部分和を保存しておくためのメモリ領域が必要となる．

そこで，まず内積計算の並列性を用いた分割を考える．これは，単独で用いた場合，必要なメモリ量が非現実的な値になりやすいという問題があるため，他の並列化手法と組み合わせて利用する．次に，結果の行列の要素ごとの並列性を用いた手法 2 つを示す．これらの 2 手法は計算時間を均等化することが難しいが，内積方向の分割と組み合わせると，この問題を緩和できる．

6.3.1 内積方向分割

内積方向分割は，$A = \begin{bmatrix} A_1 & A_2 & \cdots & A_{N_\mathrm{I}} \end{bmatrix}$ と N_I 個に分割しておき，より小さな DSYRK である $C_l = A_l A_l^\mathsf{T}$ をすべて並列に行った後，最後に総和 $C = \sum_l C_l$ を計算するものである．これはプログラム`blocked-matmul2`において，中央の k に関するループを並列化したことに相当する．

この並列化手法の利点は，演算量や計算時間を均等化しやすい点である．A の分割を 1 列単位で行えば，各 C_l 間の演算量の違いは 1 列分以下となるため，n が N_I と比べて十分大きければ，演算量の違いは小さくなる．また，各 C_l の計算領域は A_l の列数の違いを除けば同じとなるため，計算時間の違いも小さくなると予想される．

一方，問題点は，分割によって内積方向のデータ再利用性が低下することと，総和の計算のためのコストが増えること，そして C_l のためのメモリ領域が必要になることである．A_l の列数がキャッシュブロックの大きさ b_k よりも小さくなると，その分だけデータ再利用性が低下するため，性能が劣化する．また，A_l の列数が b_k よりも多少大きい場合でも，端数処理のために b_k よりも小さい列数のブロックを計算するため，同じ問題が起きる．総和の計算は，行列積の内積のうちいくつかの足し算を最後に処理することであるため，行列積の演算量を変化させない．しかし積和命令を持つような CPU では，積と和を同時に行えないため演算量が増えることと同然となる．また，総和の計算は演算量に対してデータ移動量が大きいため，データ移動時間を演算時間の中に隠蔽することができない．このため総和の計算の実行効率は行列積自体と比べて小さくなり，相対的に総和のコストが大きく見えやすい．以上 2 つの問題点は n が N_I と比べて十分大きければ無視できるようになる．

C_l を格納するために必要なメモリ量は，ユーザーから与えられた C を活用すると 1 つ分だけ省略できるので，$8m^2(N_\mathrm{I} - 1)$ Byte となる．ただし，今回は実装していないが，C_l の対称性を利用すればメモリ量を約半分にすることも可能である．この式を用いて実際に必要なメモリ量を計算すると，例えば $m = 5{,}000$，$N_\mathrm{I} = 56$ の場合，約 10GB のメモリ領域を必要とすることになり，現実的でない．またデータパッキングを行う場合，$\bar{X}$ や $\bar{Y}$ もコアごとに用意しなければならない．とくに $\bar{X}$ はもとから大きな領域となっているため，問題である．

そこで，内積方向の分割は単独では利用せず，別の並列化手法（以下で説明する 1 次元，2 次元分割）と組み合わせて利用する．例えば $N_\mathrm{I} = 10$ としておいて，各 C_l を別の並列化手法を用いて，$N_\mathrm{O} = 5$ または $N_\mathrm{O} = 6$ 並列で実行する．こうすれば，必要なメモリ量を大幅に小さくでき，なおかつ，以下の並列化手法においても現れる，並列度が大きい場合のデメリットを緩和できる．

ここで，N_I の決定基準が問題となる．N_I の値は，メモリ量や計算時間にも影響するが，ここでは，使用するメモリ量の最大値を決めて，その範囲で最も大きな N_I を選択する，固定量メモリ基準を使う．いま全体のコア数を N_C，C_l のために用意するメモリ量の最大値を要素数として $M = 25 \times 10^6$ のように決めたとき，

$$\bar{m} = 8\lceil m/8 \rceil \tag{6.8}$$

とおき，

$$N_\mathrm{I} = \min\left(\left\lfloor \frac{M}{\bar{m}^2} \right\rfloor + 1, N_\mathrm{C} \right) \tag{6.9}$$

と決める．$\bar{m}$ はアライメントを調整した m であり，$\bar{m} \times \bar{m}$ の大きさの行列が M の中に何個入るか計算して，それに 1 を足した値を N_I としている．ただし N_C より大きくならないように調整する．これ

は $N_{\mathrm{C}} = 56$ だとすると，$m = 672$ のときの最大値 56 をとり，m が大きくなるにつれ段階的に減少し，$m \geq 5{,}000$ のとき最小値 1 をとる．このとき $\bar{X}$ を格納するために必要な領域は，$8\bar{m}N_{\mathrm{I}}b_k\mathrm{Byte}$ である．$\bar{m}N_{\mathrm{I}}$ の最大値は M によって決まるため，$b_i \geq \max(\bar{m}N_{\mathrm{I}})$ として $\bar{X}$ を 1 つ用意しておき，$N_{\mathrm{I}} > 1$ の場合は $\bar{X}$ のメモリ領域を N_{I} 個に分割して用いるようにすれば，$\bar{X}$ のためのメモリ領域も固定サイズになる．

固定量メモリ基準によって N_{I} を m^2 に反比例するように決めることは，使用するメモリ量を一定にできるだけでなく，次の観点で合理性がある．まず，m が十分大きい場合は，以下の並列化手法を用いても十分高い性能が実現できるため，小さな N_{I} でよいが，逆に m が小さい場合は，以下の並列化手法の性能が落ちるため，N_{I} を大きくすべきである．ただし，この基準は n を計算に使用していないため，「n が小さく，総和のコストが相対的に大きくなる状態では，m が小さくても N_{I} を小さくする」などの調整は行えない．このような計算時間の最小化を行うためには，計算時間のモデル化が必要であるため，今後の課題とする．

6.3.2 1 次元分割

1 次元静的分割手法は内積方向分割を行った後の行列 C_l を N_{O} 個の列ブロックに分割し，それぞれの列ブロックに対する計算を並列に行う手法である．（内積方向分割と組み合わせない場合は $C_1 = C$ をコア数だけ分割する．）すなわち C_l を：

$$C_l = \begin{bmatrix} D_1 & D_2 & \cdots & D_{N_{\mathrm{O}}} \end{bmatrix}, \tag{6.10}$$

のように分割する．これはプログラム`blocked-matmul2` において最内側ループの j に関するループを並列化したものである．ただし C_l は上三角部分のみを計算するため，列ブロック幅を均等に分割すれば演算量のばらつきが大きくなってしまう．そこで列ブロック幅をうまく選択することで実行時間のばらつきを小さくしたいが，正確な実行時間を予測するためには性能モデルが必要であるため，ここではコアごとの演算量を均等化することを目標とする．

いまそれぞれの列ブロックの列幅を $c_1, c_2, \ldots, c_{N_{\mathrm{O}}}$ とおき，列ブロックの開始位置 $j_s = \sum_{t=1}^{s} c_t$（ただし $j_0 = 0$）とおく．j_s は本来，整数だが，実数の範囲で考えると最適値は簡単に計算でき，

$$\frac{j_{s+1}^2 - j_s^2}{2} = D \tag{6.11}$$

と漸化式で求められる．ただし，$D = \frac{m^2}{2N_{\mathrm{O}}}$ であり，上三角部分の面積をコア数で割った値である．GotoBLAS や BLIS で実装されている手法は，単純にこれを整数で丸めたものであり，次の漸化式を使っている．

$$j_{s+1} = \mathrm{round}\left(\sqrt{j_s^2 + 2D}\right). \tag{6.12}$$

ただし，終端は行列サイズと一致するよう $j_{N_{\mathrm{O}}} = m$ とする．round は丸め関数で，計算カーネルのブロック幅 8 へ丸める．このように求めた c_s は b_j より大きくなりうるが，その場合は b_j ごとに分割して，同じコアで計算させればよい．

j_s を整数の範囲で考えた場合，最適値を求めることは整数最適化問題となる．すなわち

$$\begin{aligned} &\min. && \max\left\{\textstyle\sum_{i=i_{s-1}}^{i_s-1} i \,\middle|\, s = 1, 2, \ldots, N_{\mathrm{O}}\right\}, \\ &\text{sub. to.} && 0 = i_0 \leq i_1 \leq i_2 \leq \cdots \leq i_{N_{\mathrm{O}}} = m, \end{aligned} \tag{6.13}$$

の最適解を求めることになる．これは一見複雑な問題だが，実際には，逆問題を考えることで簡単に解くことができる．つまり，「コア数（の上限）が与えられたとき，演算量の最大値を最小化する問題」の代わりに，逆問題「1 コアあたりの演算量の上限が与えられたときに，必要となるコア数の最小値を求める問題」を考える．この逆問題は，1 番目のコアから順に上限を満たす中で最も大きな列幅を選択する貪欲解法によって計算可能である．なぜならば，残りの列が多くなればなるほど必要となるコア数が増大するからである．また，この逆問題を，演算量の上限が与えられたときのコア数の最小値を返す関数だと考えると，これは単調非増加関数である．そこで演算量の上限を二分探索することでコア数が N_{O} 以下のものの内，もっとも小さな演算量の上限を得ることができ，これは元の整数最適化問題の最適値となる．このアルゴリズムは，列幅を単語の長さと見ることで，word wrapping の逆問題として理解することができ，その解説がインターネット上に存在する [136].

1 次元分割の問題点は，列ブロックの幅が小さくなり，データの再利用性が低くなりやすいことである．式（6.11）で概算すると，$N_{\mathrm{O}} = N_{\mathrm{C}} = 56$ の場合 $m = 5000$ のとき $c_{N_{\mathrm{O}}}$ は約 45 となる．これはキャッシュブロック幅 b_j の約 3 分の 1 である．この問題は内積方向分割との組み合わせで緩和できる．

6.3.3　2 次元分割

1 次元分割では，1 列ブロック当たり 1 コアを割り当てるので，右端の列ブロックの幅が小さくなった．2 次元分割では，右の列ブロックに行くほど演算量を増やす代わりに多くのコアを割り当てることで，列ブロックの幅を均等にする．1 つの列ブロックをさらに行ブロックに分割することで，列ブロック内での並列化を行う．これはプログラム`blocked-matmul2` において，最内側処理を並列化したことに相当する．例として，図6.5に分割の様子を示す.

まず，列ブロックの個数を N_{X}，s 番目の列ブロックに割り当てるコアの数を m_s とおく．このとき，1 コアあたりの演算量がなるべく等しくなるように分割するため，1 次元静的分割の問題を拡張した次の問題の最適解を，列ブロックの分割位置とする．

$$\begin{aligned} &\min. \quad \max\left\{\sum_{j=j_{s-1}}^{j_s-1} \frac{j}{m_s} \middle| s = 1, 2, \ldots, N_{\mathrm{X}}\right\}, \\ &\text{sub. to.} \quad 0 = j_0 \le j_1 \le j_2 \le \cdots \le j_{N_{\mathrm{X}}} = m, \end{aligned} \tag{6.14}$$

この問題に対しても 1 次元分割の時と同じアルゴリズムが使えるので効率的に計算できる．N_{X} や m_s の設定の仕方が問題であるが，分割した領域が正方形となることを目指して，$N_{\mathrm{X}} \approx 2\sqrt{N_{\mathrm{O}}}$，$m_s \approx s$ を整数に丸めた値とする．

6.4　性能測定

本実験のプログラムは C++ と SIMD 演算の intrinsic 命令を用いて作成した．コンパイラは icpc の 15.0.1 である．とくに計算カーネルは命令スケジューリングが大きく性能に影響するため，1 コア当たり 4 つのスレッド専用に設計しており，他のスレッド数では現状動作しない．Xeon Phi はコプロセッサとしての実行形態として，ホストマシンと独立した計算機として動作する native モードや，ホストマシンから呼び出される形で動作する offload モードなどがあるが，このプログラムは native モードのみを対象としている．また，並列化のためのスレッド生成や affinity の設定などに OpenMP を用いている．また，同期機構については，OpenMP や Pthread のバリアではなく，独自に実装したバタフライバリアを用いている．これは，コア内のスレッド間での同期や内積方向分割における各 C_l ごとの

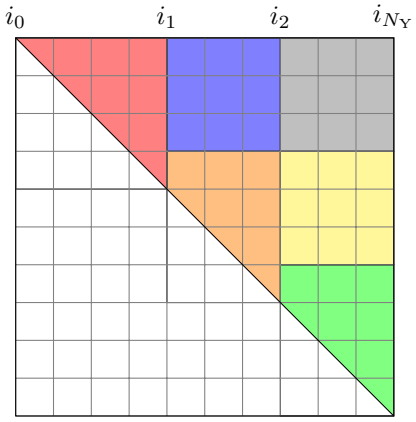

図6.5　2 次元静的分割の例

同期など，複雑な制御が必要であるため，OpenMP のバリアでは実装が難しく，また，Pthread のバリアは性能上の問題があったためである．

性能測定は Xeon Phi 3120A を用いた．これは 57 個のコアを持つが，Jeffer[intel3] などで解説されているとおり，1 つのコアを OS やホストマシンとの通信で使われる分として残し，56 コアのみを計算に用いた．このときの理論ピーク性能は 985.6GFlop/s である．スレッド数は，各コア 4 スレッドずつの 224 とし，環境変数を`KMP_AFFINITY=granularity=fine,compact` と設定し affinity を指定した．実行性能は演算量 $E_{\text{all}} = m(m+1)(2n-1)/2$ を実行時間で割ったものとして計算しており，実際にプロセッサ上で行った浮動小数点演算の量とは異なる．

測定の対象は，まず比較対象として MKL の version 11.2.1 に含まれる DSYRK を用いた．また，今回実装したものとして，1 次元分割のみを使うもの（1D），1 次元分割で $\bar{X}$ のパッキングを行うもの (1D+X)，それぞれについて，内積方向分割を組み合わせたもの（別の次元を補助的に使うという意味でそれぞれ 1.5D, 1.5D+X），それらの 2 次元分割に置き換えたもの（2D, 2D+X, 2.5D, 2.5D+X）を対象とした．行列 A と C はどちらもそれぞれの列のアライメントがそろっているものを用いている．

6.4.1　実行性能

計算する行列の大きさに関して 2 つのパラメータ m，n があるため，それぞれのパラメータの実行性能への影響を見るために，$n = 10,000$ に固定して m を 8 から $5,600$ まで変化させたときの性能と，$m = 2,400$ に固定して n を 8 から $10,000$ まで変化させたときの性能をそれぞれ調べた．

まず，$n = 10,000$ と固定して m を変化させたときの性能を図 6.6と図 6.7に示す．図 6.6は 1 次元分割と，それに内積方向分割を組み合わせたもの，図 6.7は 2 次元分割と内積方向分割を組み合わせたものである．1 次元分割で最も速いものは，1.5D であり，小さな m においても高い性能となり，最大 746GFlop/s となっている．内積方向分割を組み合わせないものは m が小さい時の性能が低くなっている．2 次元分割では，内積方向分割を組み合わせなくても，小さな m で高い性能となるが，全体として最も高速であるのは 2.5D であり，最大 769GFlop/s に到達している．

$\bar{X}$ のデータパッキングを行ったものは，むしろ性能が減少している．これはデータパッキングのコストがデータアクセス速度向上の効果より大きくなっているためだと思われる．しかし，この実験で

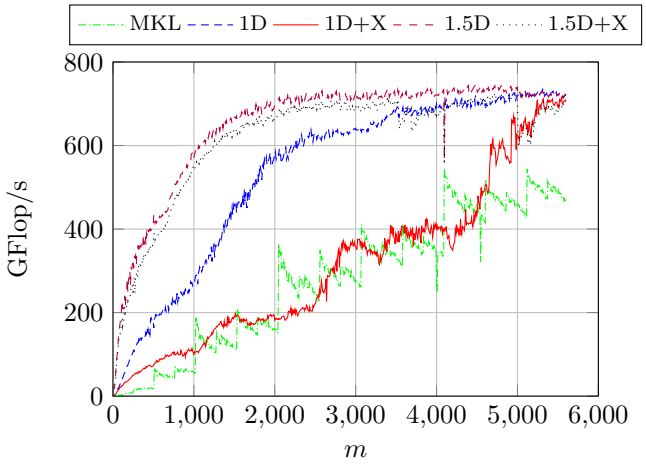

図6.6　n を固定したときの実行性能 (1D)

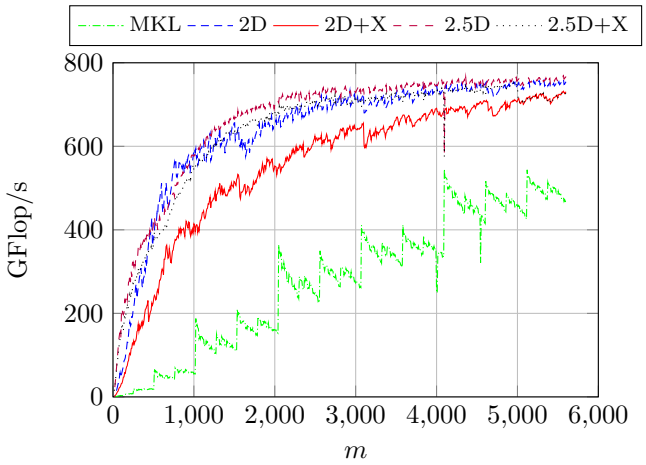

図6.7　n を固定したときの実行性能 (2D)

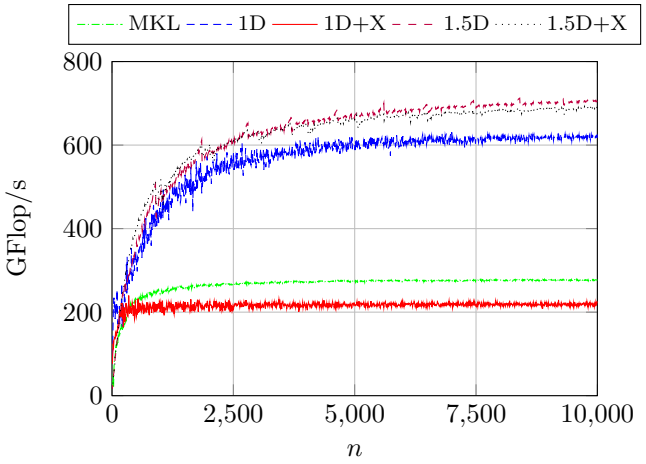

図6.8　m を固定したときの実行性能 (1D)

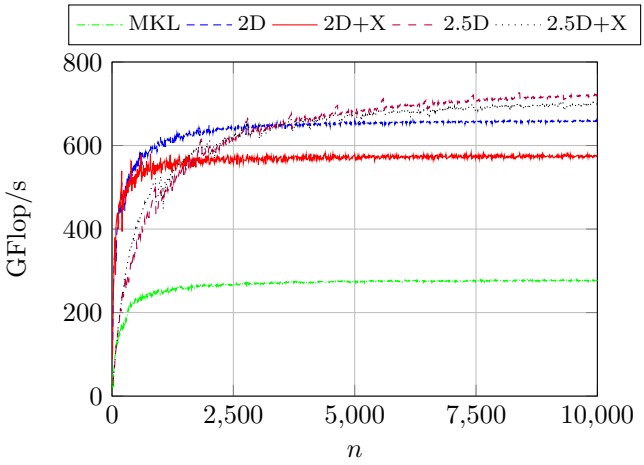

図6.9　m を固定したときの実行性能 (2D)

は A のアライメントがそろっているものを用いているが，そうでない場合，A へのアクセスコストが上昇するため，この差が逆転する可能性がある．

次に，$m = 2,400$ と固定して n を変化させたときの性能を図 6.8と図 6.9に示す．図 6.8は 1 次元分割と，それに内積方向分割を組み合わせたもの，図 6.9は 2 次元分割と内積方向分割を組み合わせたものである．この図から，内積方向分割を行わないものは，$n > 1,000$ 程度の領域でほぼ性能が一定となっていることがわかる．一方で，内積方向分割を組み合わせたものは，n によって性能が大きく変動しており，総和などのオーバーヘッドの影響が見て取れる．ただし例外として 1D については，内積方向分割を行うものと同じように n によっての性能変化が大きい．

6.4.2　性能解析

次にプログラムを各ステップに分けて時間測定を行うことで実行時間の内訳を示したものが図 6.10である．ここでは行列サイズとして，$m = 1,200$ または $2,500$，$n = 2,500$，または $5,000$ の組み合わせとして 4 種類用いた．ステップは，並列化手法によって多少異なるが，計算カーネル部分（`kernel8x24` に相当）と，$\bar{X}$ のパッキング（行うもののみ），$\bar{Y}$ のパッキング，C の総和（内積方向分割のみ），その他（同期や初期化処理）に分けている．

図 6.6などの結果と対応して，図 6.10においても 1D+X がとくに遅い．実行時間の内訳をみると，1D と比べて，単に同期や $\bar{X}$ のパッキングの時間が増えているだけではなく，計算カーネルの実行時間自体が増えていることがわかる．1D と 1D+X とでは計算領域の形状が一致するが，$\bar{X}$ をコア間で共有する 1D+X は 1D より頻繁に同期を行う．この結果，複数のコアで共通のデータへのアクセスが頻繁に発生し，単純なメモリアクセスより低速となるデータブロードキャストが頻繁に発生したことが原因ではないかと考えられる．

1D+X 以外についてみると，まず 1D は 1D+X ほどではないが計算カーネルの実行時間が大きい．とくに $m = 1,200$ のときは 2D のものと比べて約 2 倍となっているが，$m = 2,400$ のときは約 1.2 倍まで低下する．これは純粋に，1D の欠点である，列ブロックの幅が小さくなりやすく，データ再利用性が低下する問題が表れているものと考えられる．

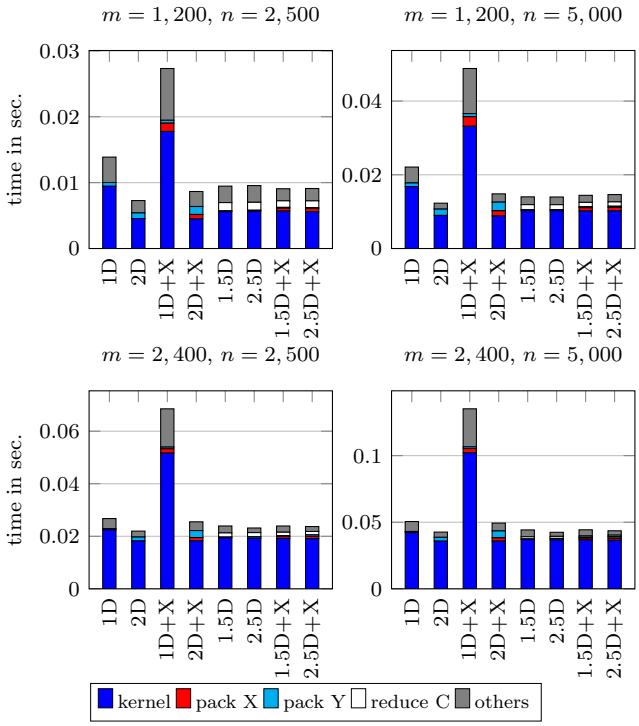

図6.10　実行時間の内訳

2D や 2D+X は他と比べて顕著に $\bar{Y}$ のパッキングにかかる時間が大きい．これらはblocked-matmul2 の再内側処理を複数のコアで並列化するが，このとき $\bar{Y}$ のパッキングについては並列化していないことが影響していると考えられる．

また，全体として，m が増加したとき $\bar{X}$ や $\bar{Y}$ のパッキングの割合が減少し，n が増加したとき，C の総和の割合が減少していることが確認できる．C の総和の割合は m が増加したときも減少している．これは，$m = 1,200$ のときは内積方向の分割数 $N_{\mathrm{I}} = 18$ であるのに対して，$m = 2,400$ のときは $N_{\mathrm{I}} = 5$ となり，総和の演算量の占める割合が減少することを反映していると考えられる．

以上の通り，将来利用されるアーキテクチャとしてメニーコア CPU の 1 つである KNC に対して，片側ブロックヤコビ法の計算においてオーダーの意味でボトルネックとなっている Gram 行列計算の実装手法について検討した結果，並列化手法の変更によって，ピーク演算性能の 7 割以上に達する高性能を実現できることが分かった．ここでの実装手法は KNC を対象としているが，キャッシュや共有メモリなどの構造を持つ他のメニーコア CPU や将来のものに対しても応用可能な手法であると考えられる．そこで，性能モデル化などと組み合わせた自動チューニング機構などを開発することで，より汎用性を高めることが課題である．これらのテクニックは将来に片側ブロックヤコビ法をメニーコア CPU に移植する場合に役立つものと考えられる．

第 7 章

結論

本論文は，非常に高い並列性を持つ現代と将来の高性能計算機に適する SVD 計算手法を開発するため，片側ヤコビ法に着目した．片側ヤコビ法は LAPACK に実装された優れた SVD 計算手法の一つであるが，ブロック化や並列化などの拡張手法と組み合わせることで，高い計算効率と，粗粒度な並列性を持たせることができ，高性能計算機に適すると考えられる．しかし，ブロック化や並列化については理論的・実験的検証が不十分であり，また現代の高い並列性を持つ高性能計算機に向けた実装の研究も進んでいなかった．

そこで本研究では第一に高性能計算に適した Hari の V2 手法 [7] と呼ばれる片側ヤコビ法のブロック化手法に対して理論的な誤差解析を行うことで，計算結果の直交性に対する誤差上界を求めた．直交性に対する誤差は片側ブロックヤコビ法の収束性に影響する．経験的には Hari の V2 手法の直交性の誤差が小さいことは知られていたが，理論的になぜそうなるかは不明であった．我々の結果は，行列計算における誤差を詳細に追跡していくものであり，この証明によって，片側ブロックヤコビ法における列スケーリングした条件数が小さいという性質が直交性の誤差の小ささに影響していることが判明した．

次に本研究では，Bečka らの並列動的オーダリングを用いたときの，一次収束性に対するより新しい上界を求めた．これまでの研究では，この上界が並列数に関係せず一定であったが，我々の上界は並列数に反比例する形となっているため，我々は，Bečka らの手法を用いたときの計算時間の上限が並列化によって減少することを示したことになる．

第三に，我々は Hari の V2 手法を用いたときの片側ブロックヤコビ法の計算の特徴を利用したデータ分散手法・並列化手法である，二次元ブロック化手法を開発し，大規模並列計算機において，高い並列性を有することを実証した．このデータ分散手法はごく単純なものであるが，並列化において，AllReduce 型と呼ばれる，通信量や回数の削減を重視した計算手順となっている．そこで，理論的に通信量や回数を算出した後に，並列計算機上で検証を行い，理論から予測された結果に従った結果を得た．また，京コンピュータを用いた大規模並列計算機上での性能検証を行い，$10,000$ 次元の行列の SVD を計算するときに $3,072$ 並列という，行列の次元数と近いノード数まで計算が加速することを確かめた．

そして最後に，Hari の V2 手法を用いた片側ブロックヤコビ法において主要な演算の 1 つとなる DSYRK という対称な形を持つ行列積について，将来使用されるであろうと目されるアーキテクチャを持つメニーコア CPU，Xeon Phi (Knights Corner, KNC) を用いて性能検証と，実装手法の研究を行った．KNC は Intel が開発したメニーコア CPU であり，Intel が提供する計算ライブラリには最適化

されたDSYRKも含まれているが，ピーク演算性能と比べるとずっと小さな性能しか得られなかった．我々は，KNCの持つ多数の並列性を活かすDSYRKの並列化手法を検討し，行列積の持つ3次元的な並列化軸をすべて利用する並列化アルゴリズムを用いることで，通常の行列積DGEMMに近い，高い性能を得た．これは将来の計算機において片側ブロックヤコビ法を実装するときに役立つものと期待される．

今後，実際に片側ブロックヤコビ法を高性能計算機上で活用していくためには，次のことが課題になると考えている．第一に，収束性の証明であり，我々のHariのV2手法に対する誤差解析によって，多くの謎が解決したが，実際に，誤差を持った環境における収束性の証明にはまだたどり着いていない．誤差のない環境におけるブロックヤコビ法の収束性の証明自体がごく最近に行われたものであるが，それを，本研究の成果と組み合わせることで，この証明が可能になるものと考えられる．

また，高性能計算機に向けた実装の改善・強化や性能解析はさらに発展させる必要がある．本論文で提案した二次元ブロック分割手法はデータ分散と列ブロック数を決定するパラメータαがあり，これを最適化することで性能改善を行えるが，最適なαの値を決定するには事前の性能測定か，あるいは性能モデル化が必要である．現実には，5章での京コンピュータにおける実験のように大規模計算を行う場合は，事前の性能測定がコスト面などの制約のため困難な場合もあるため，詳細な性能解析に基づく良い性能モデルの構築が今後の課題となる．また，現代の高性能計算機の発展は著しく，とくに並列度の増大や性能バランスの変化は急速である．我々は行列の次元数とノード数が近い状態においても加速するほどの性能を得たが，将来の計算機に対応するためには，さらに高度な並列性能が必要になる．それに対しては，本研究で行ったDSYRKの実装技術の研究のように，個々の部品に対する研究も必要になってくると考える．実際に現状においてはQR前処理という，ScaLAPACKの実装を用いた部分が大きな部分を占めるものとなっており，単に片側ブロックヤコビ法だけでなく，そこに用いる部品のアルゴリズムも含めた改善・強化が必要になってくると考える．

謝辞

本研究にあたり，学部・修士を含めて 6 年にわたり，指導教員としてご指導くださった山本有作教授に感謝します．また，指導教員として研究室生活についてご相談に乗っていただき，また，審査委員を引き受けてくださった緒方秀教教授に感謝します．修士時代にご指導いただき，また外部審査委員を引き受けてくださった神戸大学大学院システム情報学研究科の横川三津夫教授に感謝します．また，審査委員を引き受けてくださり，公聴会や予備審査においては貴重な意見やご指導をくださった山本野人教授，成見哲教授，山﨑匡准教授に感謝します．また，関連論文 [b,c] の共著者として有益なコメントやアドバイスを下さった，スロヴァキア科学アカデミーの Bečka 教授と，ザルツブルグ大学の Vajteršic 教授に感謝します．

この研究は日本学術研究振興会科研費（26286087, 15H02708, 15H02709, 16KT0016, 17H02828, 17K19966）の補助を受けている．また，日本学術振興会特別研究員 DC2（17J07747）の補助を受けている．

参考文献

[1] J. W. Demmel and K. Veselić, "Jacobi's Method is More Accurate than QR," *SIAM Journal on Matrix Analysis and Applications*, vol. 13, no. 4, pp. 1204–1245, Oct. 1992. DOI: 10.1137/0613074.

[2] Z. Drmač and K. Veselić, "New Fast and Accurate Jacobi SVD Algorithm. I," *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 4, pp. 1322–1342, Jan. 2008. DOI: 10.1137/050639193.

[3] ——, "New Fast and Accurate Jacobi SVD Algorithm. II," *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 4, pp. 1343–1362, 2008. DOI: 10.1137/05063920X.

[4] J. W. Demmel, "SIAM/LA Prize Recognizes More Accurate, Faster SVD Algorithm," *SIAM News*, vol. 43, no. 1, 2010.

[5] M. Planitz, "LAPACK Users Guide," *The Mathematical Gazette*, vol. 79, no. 484, p. 210, 1995. DOI: 10.2307/3620088.

[6] Top500, *TOP500 Supercomputing Sites*, 1993. [Online]. Available: http://www.top500.org/ (visited on 10/04/2017).

[7] V. Hari, S. Singer, and S. Singer, "Full block J-Jacobi method for Hermitian matrices," *Linear Algebra and Its Applications*, vol. 444, pp. 1–27, 2014. DOI: 10.1016/j.laa.2013.11.028.

[8] M. Bečka, G. Okša, and M. Vajteršic, "Dynamic ordering for a parallel block-Jacobi SVD algorithm," *Parallel Computing*, vol. 28, no. 2, pp. 243–262, Feb. 2002. DOI: 10.1016/S0167-8191(01)00138-7.

[9] Y. Takahashi, Y. Hirota, and Y. Yamamoto, "Performance of the block Jacobi method for the symmetric eigenvalue problem on a modern massively parallel computer," *Proceedings of the Conference Algoritmy*, pp. 151–160, 2012.

[10] 工藤周平, "実対称固有値問題に対するブロックヤコビ法の京コンピュータ上での実装と性能評価," 修士論文, 神戸大学, 2015.

[11] J. D. McCalpin, *SC16 Invited Talk Spotlight Dr. John D. McCalpin Presents Memory Bandwidth and System Balance in HPC Systems*, 2016. [Online]. Available: http://sc16.supercomputing.org/2016/10/07/sc16-invited-talk-spotlight-dr-john-d-mccalpin-presents-memory-bandwidth-system-balance-hpc-systems/ (visited on 10/12/2017).

[12] ——, *SC16 Invited Talk Memory Bandwidth and System Balance in HPC Systems*, 2016. [Online]. Available: http://sites.utexas.edu/jdm4372/2016/11/22/sc16-invited-talk-memory-bandwidth-and-system-balance-in-hpc-systems/ (visited on 10/12/2017).

[13] J. D. C. Little and S. C. Graves, "Chapter 5 Little ' s Law," *Operations Management*, vol. 115, pp. 81–100, 2008. DOI: 10.1007/978-0-387.

[14] 総合科学技術・イノベーション会議評価専門調査会, "国家的に重要な研究開発の評価「フラッグシップ 2020 プロジェクト（ポスト「京」の開発）」に係る基本設計評価の確認結果," Tech. Rep., 2017. [Online]. Available: http://www8.cao.go.jp/cstp/tyousakai/hyouka/hyokapj%7B%5C_%7Dh25.html.

[15] M. Feldman, *First US Exascale Supercomputer Now On Track for 2021*, 2016. [Online]. Available: https://top500.org/news/first-us-exascale-supercomputer-now-on-track-for-2021/ (visited on 10/12/2017).

[16] ——, *Retooled Aurora Supercomputer Will Be America's First Exascale System*, 2017. [Online]. Available: https://top500.org/news/retooled-aurora-supercomputer-will-be-americas-first-exascale-system/ (visited on 10/12/2017).

[17] ——, *The Four-Way Race to Exascale*, 2016. [Online]. Available: https://top500.org/news/the-four-way-race-to-exascale/ (visited on 10/12/2017).

[18] *Co-designed Innovation and System for Resilient Exascale Computing in Europe: From Applications to Silicon*, 2017. [Online]. Available: http://cordis.europa.eu/project/rcn/210095%7B%5C_%7Den.html (visited on 10/12/2017).

[19] T. Trader, *EU Funds 20 Million Euro ARM+FPGA Exascale Project*, 2017. [Online]. Available: https://www.hpcwire.com/2017/09/07/eu-funds-20-million-euro-armfpga-exascale-project/ (visited on 10/12/2017).

[20] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray ® XC ™ Series Network," Tech. Rep., 2012, pp. 1–28.

[21] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker, "The IBM Blue Gene/Q Interconnection Fabric," *IEEE Micro*, vol. 32, no. 1, pp. 32–43, Jan. 2012. DOI: 10.1109/MM.2011.96.

[22] J. Lien and C. Pospiech, "IBM System Blue Gene Solution Blue Gene/Q Application Development," Tech. Rep., 2013.

[23] Y. Ajima, T. Inoue, S. Hiramoto, and T. Shimizu, "Tofu: Interconnect for the K computer," *Fujitsu Scientific and Technical Journal*, vol. 48, no. 3, pp. 280–285, 2012. DOI: 10.1109/MM.2011.98.

[24] Z. Pang, M. Xie, J. Zhang, Y. Zheng, G. Wang, D. Dong, and G. Suo, "The TH Express high performance interconnect networks," *Frontiers of Computer Science*, vol. 8, no. 3, pp. 357–366, 2014. DOI: 10.1007/s11704-014-3500-9.

[25] X. K. Liao, Z. B. Pang, K. F. Wang, Y. T. Lu, M. Xie, J. Xia, D. Z. Dong, and G. Suo, "High Performance Interconnect Network for Tianhe System," *Journal of Computer Science and Technology*, vol. 30, no. 2, pp. 259–272, 2015. DOI: 10.1007/s11390-015-1520-7.

[26] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and G. Yang, "The Sunway TaihuLight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, no. 7, 2016. DOI: 10.1007/s11432-016-5588-7.

[27] Message Passing Interface Forum, *MPI: Message-Passing Interface Standard. Version 3.1*. Stuttgart: High Performance Computing Center Stuttgart, 2015, p. 868. [Online]. Available: http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[28] L. Grigori, J. W. Demmel, and H. Xiang, "CALU: A Communication Optimal LU Factorization Algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1317–1350, Oct. 2011. DOI: 10.1137/100788926.

[29] G. Ballard, J. W. Demmel, O. Holtz, and O. Schwartz, "Communication-optimal Parallel and Sequential Cholesky Decomposition," *SIAM Journal on Scientific Computing*, vol. 32, no. 6, p. 29, 2009. DOI: 10.1137/090760969.

[30] G. Ballard, D. Becker, J. Demmel, J. Dongarra, A. Druinsky, I. Peled, O. Schwartz, S. Toledo, and I. Yamazaki, "Communication-Avoiding Symmetric-Indefinite Factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 4, pp. 1364–1406, Jan. 2014. DOI: 10.1137/130929060.

[31] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal Parallel and Sequential QR and LU Factorizations," *SIAM Journal on Scientific Computing*, vol. 34, no. 1, A206–A239, Jan. 2012. DOI: 10.1137/080731992.

[32] J. W. Demmel, L. Grigori, M. Gu, and H. Xiang, "Communication Avoiding Rank Revealing QR Factorization with Column Pivoting," *SIAM Journal on Matrix Analysis and Applications*, vol. 36, no. 1, pp. 55–89, Jan. 2015. DOI: 10.1137/13092157X.

[33] J. A. Duersch and M. Gu, "Randomized QR with Column Pivoting," *SIAM Journal on Scientific Computing*, vol. 39, no. 4, pp. C263–C291, Jan. 2017. DOI: 10.1137/15M1044680.

[34] P. R. Willems, B. Lang, and C. Vömel, "Computing the Bidiagonal SVD Using Multiple Relatively Robust Representations," *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 4, pp. 907–926, Jan. 2006. DOI: 10.1137/050628301.

[35] O. Marques and P. B. Vasconcelos, "Computing the Bidiagonal SVD Through an Associated Tridiagonal Eigenproblem," *High Performance Computing for Computational Science – VECPAR 2016. VECPAR 2016. Lecture Notes in Computer Science*, vol. 10150, no. 1, pp. 64–74, 2017. DOI: 10.1007/978-3-319-61982-8_8.

[36] Y. P. Hong and C.-T. Pan, "Rank-revealing QR factorizations and the singular value decomposition," *Mathematics of Computation*, vol. 58, no. 197, pp. 213–213, 1992. DOI: 10.1090/S0025-5718-1992-1106970-4.

[37] J. W. Demmel, "3. Linear Least Squares Problems," in *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Jan. 1997, pp. 101–138. DOI: 10.1137/1.9781611971446.ch3.

[38] IEEE Computer Society, *IEEE Std 754™-2008 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic*, August. 2008, vol. 2008, pp. 1–58. DOI: 10.1109/IEEESTD.2008.4610935.

[39] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*. 2002, pp. 1–663. DOI: 10.2307/2669725.

[40] J. W. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač, "Computing the singular value decomposition with high relative accuracy," *Linear Algebra and its Applications*, vol. 299, no. 1-3, pp. 21–80, Sep. 1999. DOI: 10.1016/S0024-3795(99)00134-2.

[41] ——, "Computing the singular value decomposition with high relative accuracy," *Linear Algebra and its Applications*, vol. 299, no. 1-3, pp. 21–80, 1999. DOI: 10.1016/S0024-3795(99)00134-2.

[42] K. V. Fernando and B. N. Parlett, "Accurate singular values and differential qd algorithms," *Numerische Mathematik*, vol. 67, no. 2, pp. 191–229, 1994. DOI: 10.1007/s002110050024.

[43] J. Choi, J. Dongarra, R. Pozo, and D. Walker, "ScaLAPACK: a scalable linear algebra library for distributed memory concurrent computers," *[Proceedings 1992] The Fourth Symposium on the Frontiers of Massively Parallel Computation*, pp. 120–127, 1992. DOI: 10.1109/FMPC.1992.234898.

[44] G. H. Golub and H. A. Van Der Vorst, "Eigenvalue computation in the 20th century," *Journal of Computational and Applied Mathematics*, vol. 123, no. 1-2, pp. 35–65, 2000. DOI: 10.1016/S0377-0427(00)00413-1.

[45] J. G. F. Francis, "The QR Transformation A Unitary Analogue to the LR Transformation–Part 1," *The Computer Journal*, vol. 4, no. 3, pp. 265–271, Mar. 1961. DOI: 10.1093/comjnl/4.3.265.

[46] ——, "The QR Transformation–Part 2," *The Computer Journal*, vol. 4, no. 4, pp. 332–345, Apr. 1962. DOI: 10.1093/comjnl/4.4.332.

[47] V. Kublanovskaya, "On some algorithms for the solution of the complete eigenvalue problem," *USSR Computational Mathematics and Mathematical Physics*, vol. 1, no. 3, pp. 637–657, Jan. 1962. DOI: 10.1016/0041-5553(63)90168-X.

[48] B. Cipra and B. Cipra, "The best of the 20th century: editors name top 10 algorithms," *SIAM News*, vol. 33, no. 4, pp. 05–00, 2000.

[49] G. H. Golub and F. Uhlig, "The QR algorithm: 50 years later its genesis by John Francis and Vera Kublanovskaya and subsequent developments," *IMA Journal of Numerical Analysis*, vol. 29, no. 3, pp. 467–485, Jul. 2009. DOI: 10.1093/imanum/drp012.

[50] G. H. Golub and W. Kahan, "Calculating the Singular Values and Pseudo-Inverse of a Matrix," *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, vol. 2, no. 2, pp. 205–224, Jan. 1965. DOI: 10.1137/0702016.

[51] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numerische Mathematik*, vol. 14, no. 5, pp. 403–420, 1970. DOI: 10.1007/BF02163027.

[52] T. F. Chan, "An Improved Algorithm for Computing the Singular Value Decomposition," *ACM Transactions on Mathematical Software*, vol. 8, no. 1, pp. 72–83, 1982. DOI: 10.1145/355984.355990.

[53] 森正武, 数値解析第二版, 2nd ed., 古屋茂, 一松信, and 赤摂也, Eds. 東京都: 共立出版, 2002.

[54] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling, "Block reduction of matrices to condensed forms for eigenvalue computations," *Advances in Parallel Computing*, vol. 1, no. C, pp. 215–227, 1990. DOI: 10.1016/B978-0-444-88621-7.50015-3.

[55] G. H. Golub and C. F. Van Loan, *Matrix Computations third edition*, 3rd ed. Maryland: The Johns Hopkins University Press, 1996.

[56] R. Ralha, "One-sided reduction to bidiagonal form," *Linear Algebra and Its Applications*, vol. 358, no. 1-3, pp. 219–238, 2003. DOI: 10.1016/S0024-3795(01)00569-9.

[57] J. L. Barlow, N. Bosner, and Z. Drmač, "A new stable bidiagonal reduction algorithm," *Linear Algebra and its Applications*, vol. 397, no. 1-3, pp. 35–84, Mar. 2005. DOI: 10.1016/j.laa.2004.09.019.

[58] J. Demmel and W. Kahan, "Accurate Singular Values of Bidiagonal Matrices," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 5, pp. 873–912, Sep. 1990. DOI: 10.1137/0911052.

[59] M. Gu and S. C. Eisenstat, "A Divide-and-Conquer Algorithm for the Bidiagonal SVD," *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 1, pp. 79–92, Jan. 1995. DOI: 10.1137/S0895479892242232.

[60] C. G. J. Jacobi, "ÜBER EIN LEICHTES VERFAHREN DIE IN DER THEORIE DER SÄCULARSTÖRUNGEN VORKOMMENDEN GLEICHUNGEN NUMERISCH AUFZULÖSEN.," *Journal für die reine und angewandte Mathematik*, vol. 30, pp. 51–94, 1846.

[61] H. H. Goldstine, F. J. Murray, and J. von Neumann, "The Jacobi Method for Real Symmetric Matrices," *Journal of the ACM*, vol. 6, no. 1, pp. 59–96, Jan. 1959. DOI: 10.1145/320954.320960.

[62] J. Matejaš and V. Hari, "On high relative accuracy of the Kogbetliantz method," *Linear Algebra and Its Applications*, vol. 464, pp. 100–129, 2015. DOI: 10.1016/j.laa.2014.02.024.

[63] K. V. Fernando, "Linear convergence of the row cyclic Jacobi and Kogbetliantz methods," *Numerische Mathematik*, vol. 56, no. 1, pp. 73–91, 1989. DOI: 10.1007/BF01395779.

[64] M. R. Hestenes, "Inversion of Matrices by Biorthogonalization and Related Results," *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, no. 1, p. 51, 1958.

[65] B. A. Chartres, "Adaptation of the Jacobi Method for a Computer with Magnetic-tape Backing Store," *The Computer Journal*, vol. 5, no. 1, pp. 51–60, Jan. 1962. DOI: 10.1093/comjnl/5.1.51.

[66] H. F. Kaiser, "The JK method: a procedure for finding the eigenvectors and eigenvalues of a real symmetric matrix," *The Computer Journal*, vol. 15, no. 3, pp. 271–273, Mar. 1972. DOI: 10.1093/comjnl/15.3.271.

[67] J. C. Nash, "A one-sided transformation method for the singular value decomposition and algebraic eigenproblem," *The Computer Journal*, vol. 18, no. 1, pp. 74–76, Jan. 1975. DOI: 10.1093/comjnl/18.1.74.

[68] H. Rutishauser, "Handbook Series Linear Algebra The Jacobi Method for Real Symmetric Matrices *," vol. 0, 1966.

[69] A. A. Anda and H. Park, "Fast Plane Rotations with Dynamic Scaling," *SIAM Journal on Matrix Analysis and Applications*, vol. 15, no. 1, pp. 162–174, Jan. 1994. DOI: 10.1137/S0895479890193017.

[70] 杉原正顯 and 室田一雄, 線形計算の数理. 東京都: 岩波書店, 2009.

[71] 木村英紀, 線形代数数理科学の基礎. 東京都: 東京大学出版会, 2003.

[72] P. Henrici, "On the Speed of Convergence of Cyclic and Quasicyclic Jacobi Methods for Computing Eigenvalues of Hermitian Matrices," *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, no. 2, pp. 144–162, Jun. 1958. DOI: 10.1137/0106008.

[73] R. T. Gregory, "Computing Eigenvalues and Eigenvectors of a Symmetric Matrix on the ILLIAC," *Mathematical Tables and Other Aids to Computation*, vol. 7, no. 44, p. 215, Oct. 1953. DOI: 10.2307/2002832.

[74] E. R. Hansen, "On Cyclic Jacobi Methods," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 448–459, 1963.

[75] G. E. Forsythe and P. Henrici, "The cyclic Jacobi method for computing the principal values of a complex matrix," *Transactions of the American Mathematical Society*, vol. 94, no. 1, pp. 1–23, Jan. 1960. DOI: 10.1090/S0002-9947-1960-0109825-2.

[76] H. P. M. van Kempen, "On the quadratic convergence of the special cyclic Jacobi method," *Numerische Mathematik*, vol. 9, no. 1, pp. 19–22, Nov. 1966. DOI: 10.1007/BF02165225.

[77] V. Hari, "On sharp quadratic convergence bounds for the serial Jacobi methods," *Numerische Mathematik*, vol. 60, no. 1, pp. 375–406, 1991. DOI: 10.1007/BF01385728.

[78] R. P. Brent and F. T. Luk, *The Solution of Singular Value and Symmetric Eigenvalue Problems on Multi-processor Arrays*, 1985. DOI: DOI:10.1137/0906007.

[79] A. H. Sameh, "On Jacobi and Jacobi-Like Algorithms for a Parallel Computer," *Mathematics of Computation*, vol. 25, no. 115, p. 579, Jul. 1971. DOI: 10.2307/2005221.

[80] F. T. Luk and H. Park, "On parallel jacobi orderings," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 1, pp. 18–26, Jan. 1989. DOI: 10.1137/0910002.

[81] G. W. Stewart, "A Jacobi-Like Algorithm for Computing the Schur Decomposition of a Nonhermitian Matrix," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, no. 4, pp. 853–864, Oct. 1985. DOI: 10.1137/0906058.

[82] F. T. Luk and H. Park, "A Proof of Convergence for two Parallel Jacobi SVD Algorithms," *IEEE Transactions on Computers*, vol. 38, no. 6, pp. 806–811, 1989. DOI: 10.1109/12.24289.

[83] S. Singer, S. Singer, V. Novaković, A. Uščumlić, and V. Dunjko, "Novel modifications of parallel Jacobi algorithms," *Numerical Algorithms*, vol. 59, no. 1, pp. 1–27, 2012. DOI: 10.1007/s11075-011-9473-6.

[84] P. J. Eberlein and H. Park, "Efficient implementation of Jacobi algorithms and Jacobi sets on distributed memory architectures," *Journal of Parallel and Distributed Computing*, vol. 8, no. 4, pp. 358–366, Apr. 1990. DOI: 10.1016/0743-7315(90)90134-B.

[85] M. Mantharam and P. J. Eberlein, "New Jacobi-sets for parallel computations," *Parallel Computing*, vol. 19, no. 4, pp. 437–454, Apr. 1993. DOI: 10.1016/0167-8191(93)90056-Q.

[86] C. D. LaBudde, "A new algorithm for diagonalizing a real symmetric matrix," *Mathematics of Computation*, vol. 18, no. 85, pp. 118–118, Jan. 1964. DOI: 10.1090/S0025-5718-1964-0160319-5.

[87] A. Ruhe, "The relation between the Jacobi algorithm and inverse iteration and a Jacobi algorithm based on elementary reflections," *BIT*, vol. 20, no. 1, pp. 88–96, Mar. 1980. DOI: 10.1007/BF01933589.

[88] C. F. Van Loan, "The Block Jacobi Method for Computing the Singular Value decomposition," *Computational and combinatorial methods in systems theory*, pp. 245–256, 1986.

[89] D. E. Foulser, "A Blocked Jacobi Method for the Symmetric Eigenproblem," *Research Report of Yale University*, 1989.

[90] Y. Nakatsukasa and N. J. Higham, "Stable and Efficient Spectral Divide and Conquer Algorithms for the Symmetric Eigenvalue Decomposition and the SVD," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, A1325–A1349, Jan. 2013. DOI: 10.1137/120876605.

[91] Y. Yamamoto, Z. Lan, and S. Kudo, "Convergence analysis of the parallel classical block Jacobi method for the symmetric eigenvalue problem," *JSIAM Letters*, vol. 6, pp. 57–60, 2014. DOI: 10.14495/jsiaml.6.57.

[92] Z. Drmač, "A global convergence proof for cyclic Jacobi methods with block rotations," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 3, pp. 1329–1350, 2009. DOI: 10.1137/090748548.

[93] C. H. Bischof, "Computing the singular value decomposition on a distributed system of vector processors," *Parallel Computing*, vol. 11, no. 2, pp. 171–186, 1989. DOI: 10.1016/0167-8191(89)90027-6.

[94] V. Hari, "Accelerating the SVD Block-Jacobi Method," *Computing*, vol. 75, no. 1, pp. 27–53, Jul. 2005. DOI: 10.1007/s00607-004-0113-z.

[95] K. Vince Fernando and B. N. Parlett, "Implicit Cholesky algorithms for singular values and vectors of triangular matrices," *Numerical Linear Algebra with Applications*, vol. 2, no. 6, pp. 507–531, Nov. 1995. DOI: 10.1002/nla.1680020604.

[96] K. Veselić and V. Hari, "A note on a one-sided Jacobi algorithm," *Numerische Mathematik*, vol. 56, no. 6, pp. 627–633, 1989. DOI: 10.1007/BF01396349.

[97] G. H. Golub, "Bounds for eigenvalues of tridiagonal symmetric matrices computed by the LR method," *Mathematics of Computation*, vol. 16, no. 80, pp. 438–438, 1962. DOI: 10.1090/S0025-5718-1962-0163430-6.

[98] P. Businger and G. H. Golub, "Linear least squares solutions by householder transformations," *Numerische Mathematik*, vol. 7, no. 3, pp. 269–276, 1965. DOI: 10.1007/BF01436084.

[99] N. J. Higham, "QR factorization with complete pivoting and accurate computation of the SVD," *Linear Algebra and its Applications*, vol. 309, no. 1-3, pp. 153–174, 2000. DOI: 10.1016/S0024-3795(99)00230-X.

[100] M. Gu and S. C. Eisenstat, "Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization," *SIAM Journal on Scientific Computing*, vol. 17, no. 4, pp. 848–869, 1996. DOI: 10.1137/0917055.

[101] V. Hari, S. Singer, and S. Singer, "Full block J-Jacobi method for Hermitian matrices," *Linear Algebra and its Applications*, vol. 444, pp. 1–27, Mar. 2014. DOI: 10.1016/j.laa.2013.11.028.

[102] M. Bečka and G. Okša, "New approach to local computations in the parallel one-sided Jacobi SVD algorithm," *Lecture Notes in Computer Science*, vol. 9573, no. January, pp. 605–617, 2016. DOI: 10.1007/978-3-319-32149-3.

[103] T. Fukaya, Y. Nakatsukasa, Y. Yanagisawa, and Y. Yamamoto, "CholeskyQR2: A Simple and Communication-Avoiding Algorithm for Computing a Tall-Skinny QR Factorization on a Large-Scale Parallel System," in *2014 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, IEEE, Nov. 2014, pp. 31–38. DOI: 10.1109/ScalA.2014.11.

[104] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya, "Roundoff Error Analysis of the CholeskyQR2 Algorithm," *Electronic Transactions on Numerical Analysis*, vol. 44, pp. 306–326, 2015.

[105] ——, "Roundoff error analysis of the CholeskyQR2 algorithm in an oblique inner product," *JSIAM Letters*, vol. 8, pp. 5–8, 2015. DOI: 10.14495/jsiaml.8.5.

[106] D. J. Higham, "Condition numbers and their condition numbers," *Linear Algebra and its Applications*, vol. 214, no. 1995, pp. 193–213, Jan. 1995. DOI: 10.1016/0024-3795(93)00066-9.

[107] J. W. Demmel, "On Floating Point Errors in Cholesky," *LAPACK Working Note*, vol. 14, pp. 1–7, 1989.

[108] A. van der Sluis, "Condition numbers and equilibration of matrices," *Numerische Mathematik*, vol. 14, no. 1, pp. 14–23, 1969. DOI: 10.1007/BF02165096.

[109] S. Singer, S. Singer, V. Novaković, D. Davidović, K. Bokulić, and A. Uščumlić, "Three-level parallel J-Jacobi algorithms for Hermitian matrices," *Applied Mathematics and Computation*, vol. 218, no. 9, pp. 5704–5725, Jan. 2012. DOI: 10.1016/j.amc.2011.11.067.

[110] P. P. M. de Rijk, "A One-Sided Jacobi Algorithm for Computing the Singular Value Decomposition on a Vector Computer," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 2, pp. 359–371, Mar. 1989. DOI: 10.1137/0910023.

[111] W. F. Mascarenhas, "On the Convergence of the Jacobi Method for Arbitrary Orderings I," *SIAM J. Matrix Anal. Appl.*, vol. 16, no. 4, pp. 1197–1209, 1995.

[112] N. Auger, C. Nicaud, and C. Pivoteau, "Merge Strategies: from Merge Sort to TimSort," *HAL*, 2015.

[113] J. Choi, J. J. Dongarra, and D. W. Walker, "PB-BLAS: a set of Parallel Block Basic Linear Algebra Subprograms," *Proceedings of the Scalable High-Performance Computing Conference, 1994*, pp. 534–541, 1994. DOI: 10.1109/SHPCC.1994.296688.

[114] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, "The Design and Implementation of ScaLAPACK LU, QR, and Cholesky," *Scientific Programming*, vol. 5, pp. 173–184, 1996. DOI: 10.1.1.38.7758.

[115] M. Bečka, G. Okša, M. Vajteršic, and L. Grigori, "On data layout in the parallel block-jacobi svd algorithm with pre–processing," *Proceedings of ALGORITHMY 2009*, pp. 449–458, 2009.

[116] M. Bečka and G. Okša, "On variable blocking factor in a parallel dynamic block - Jacobi SVD algorithm," *Parallel Computing*, vol. 29, no. 9, pp. 1153–1174, 2003. DOI: 10.1016/S0167-8191(03)00096-6.

[117] ——, "Parallel One–Sided Jacobi SVD Algorithm with Variable Blocking Factor," *Parallel Processing and Applied Mathematics. PPAM 2013. Lecture Notes in Computer Science*, vol. 8384, no. September 2013, pp. 57–66, 2014. DOI: 10.1007/978-3-642-55224-3_6.

[118] D. Royo, M. Valero-Garcia, and A. Gonzalez, "A Jacobi-based algorithm for computing symmetric eigenvalues and eigenvectors in a two-dimensional mesh," in *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing - PDP '98 -*, IEEE Comput. Soc, 1998, pp. 463–469. DOI: 10.1109/EMPDP.1998.647234. [Online]. Available: http://ieeexplore.ieee.org/document/647234/.

[119] 宮崎博行, 草野義博, 新庄直樹, 庄司文由, 横川三津夫, and 渡邊貞, "スーパーコンピュータ「京」の概要," *FUJITSU*, vol. 63, no. 3, pp. 237–246, 2012.

[120] Y. Yamamoto and S. Kudo, "Probabilistic analysis of an estimator for the Frobenius norm of a matrix product," *JSIAM Letters*, vol. 9, pp. 9–12, 2015. DOI: 10.14495/jsiaml.9.9.

[121] M. Bečka, G. Okša, and M. Vajteršic, "New Dynamic Orderings for the Parallel One–Sided Block-Jacobi SVD Algorithm," *Parallel Processing Letters*, vol. 25, no. 02, p. 1 550 003, 2015. DOI: 10.1142/S0129626415500036.

[122] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, pp. 449–467, 1965. DOI: 10.4153/CJM-1965-045-4.

[123] R. Duan and S. Pettie, "Approximating maximum weight matching in near-linear time," *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pp. 673–682, 2010. DOI: 10.1109/FOCS.2010.70.

[124] D. Avis, "A survey of heuristics for the weighted matching problem," *Networks*, vol. 13, no. 4, pp. 475–493, 1983. DOI: 10.1002/net.3230130404.

[125] J. Munkres, "Algorithms for the Assignment and Transportation Problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, Mar. 1957. DOI: 10.1137/0105003.

[126] E. M. Reingold and R. E. Tarjan, "On a greedy heuristic for complete," *SIAM Journal on Computing*, vol. 10, no. 4, pp. 676–681, 1981. DOI: 10.1137/0210050.

[127] F. G. V. Zee, V. Austel, J. A. Gunnels, L. Killough, T. M. Smith, B. Marker, T. M. Low, R. A. V. D. Geijn, F. D. Igual, M. Smelyanskiy, X. Zhang, and M. Kistler, "The BLIS Framework," *ACM Transactions on Mathematical Software*, vol. 42, no. 2, pp. 1–19, Jun. 2016. DOI: 10.1145/2755561.

[128] T. M. Smith, R. V. D. Geijn, M. Smelyanskiy, J. R. Hammond, and F. G. V. Zee, "Anatomy of High-Performance Many-Threaded Matrix Multiplication," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IEEE, May 2014, pp. 1049–1059. DOI: 10.1109/IPDPS.2014.110.

[129] K. Goto and R. A. V. D. Geijn, "High-performance implementation of the level-3 BLAS," *ACM Transactions on Mathematical Software*, vol. 35, no. 1, pp. 1–14, Jul. 2008. DOI: 10.1145/1377603.1377607.

[130] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High Peformance Programming*, 1st ed. Massachusetts: Morgan Kaufmann, 2013.

[131] Intel Corporation, *Intel Xeon Phi Coprocessor Instruction Set Architecture Reference Manual*, 2012. [Online]. Available: https://software.intel.com/sites/default/files/forum/278102/327364001en.pdf (visited on 10/25/2017).

[132] R. Rahman, *Intel Xeon Phi Coprocessor Vector Microarchitecture*, 2012. [Online]. Available: http://software.intel.com/sites/default/files/article/393199/intel-xeon-phi-coprocessor-vector-microarchitecture.pdf (visited on 10/25/2017).

[133] J. Fang, H. Sips, L. Zhang, C. Xu, Y. Che, and A. L. Varbanescu, "Test-driving Intel Xeon Phi," *Proceedings of the 5th ACM/SPEC international conference on Performance engineering - ICPE '14*, no. 11272352, pp. 137–148, 2014. DOI: 10.1145/2568088.2576799.

[134] FLAME GROUP, *BLAS-like Library Instantiation Software Framework*. [Online]. Available: https://github.com/flame/blis (visited on 10/25/2017).

[135] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A. G. Shet, G. Chrysos, and P. Dubey, "Design and implementation of the Linpack benchmark for single and multi-node systems based on Intel® Xeon Phi™ coprocessor," *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, pp. 126–137, 2013. DOI: 10.1109/IPDPS.2013.113.

[136] Pete T, *Word wrap to X lines instead of maximum width (Least raggedness)*, 2011. [Online]. Available: http://stackoverflow.com/questions/6426017/word-wrap-to-x-lines-instead-of-maximum-width-least-raggedness.

関連論文

[a] S. Kudo, Y. Yamamoto, “On using the Cholesky QR method in the full-blocked one-sided Jacobi algorithm,” *Proceedings of PPAM2017, Lecture Notes in Computer Science*, vol. 10777, 2018, to appear. (4 章の内容に関連する)

[b] S. Kudo, Y. Yamamoto, M. Bečka, M. Vajteršic, “Performance of the Parallel One-Sided Block Jacobi SVD Algorithm on a Modern Distributed-Memory Parallel Computer,” *Proceedings of PPAM2015, Lecture Notes in Computer Science*, vol. 9573, Apr. 2016. (5 章の内容に関連する)

[c] S. Kudo, Y. Yamamoto, M. Bečka, M. Vajteršic, “Performance analysis and optimization of the parallel one-sided block Jacobi SVD algorithm with dynamic ordering and variable blocking,” *Concurrency and Computation: Practice and Experience*, vol. 29, issue 9, Dec. 2016. (5 章の内容に関連する)

[d] 工藤周平, 山本有作, “Xeon Phi における DSYRK の並列化手法と性能解析,” 情報処理学会論文誌コンピューティングシステム (ACS), vol. 9, no. 4, pp. 15–24, Nov. 2016. (6 章の内容に関連する)