Vol.2017-HPC-158 No.20
2017/3/9

メニーコアプロセッサ向け分割統治法の実装技術

廣田 悠輔[1,a)]　今村 俊幸[1]

概要：本研究では，メニーコアプロセッサ上で効率的に実行可能な分割統治法による固有値ソルバについて述べる．Intel Math Kernel Library（MKL）の分割統治法の実装は，メニーコア計算機において十分高い性能を示さない．そこで，Intel MKL の実装について仮定をおいたうえで，メニーコアプロセッサ上で高速に実行されない理由について考察を行った．また，その考察に基づき，分割ツリーの並列性とマージ処理内の二重の並列性を展開して並列化を行うメニーコアプロセッサ向けの高速化手法を提案する．数値実験により Intel MKL の実装の実行時間内訳を分析することにより考察の妥当性を確認した．また，提案する高速化手法の一部を取り入れた実装を作成して評価を行い，Intel MKL の実装い対して約 1.15 倍の高速化されることを確認し，手法の一部についての有効性を確認した．

Implementation Techniques of Divide-and-Conquer Method for a manycore processor system

Hirota Yusuke[1,a)]　Imamura Toshiyuki[1]

1. はじめに

本研究では，$n \times n$ 実対称三重対角行列 A が与えられるときに

$$A = Q\Lambda Q^{\top}$$

を満たす直交行列 Q，対角行列 Λ を求める問題（すなわち A の全固有対を求める問題）を解くソルバルーチン（三重対角行列固有値ソルバ）の実装について考える．

実対称密行列およびエルミート密行列の固有値問題（密行列固有値問題）を解くソルバ（密行列固有値ソルバ）では，三重対角行列固有値ソルバがその主要なサブソルバの 1 つとして使用される．密行列固有値ソルバは様々な計算科学アプリケーションの中核をなす重要な部品である．したがって，三重対角行列固有値ソルバの性能は，それらの応用アプリケーションの性能に間接的な影響を与えるものである．

一方，近年，Intel Xeon Phi Knights Corner（KNC），同 Xeon Phi Knights Landing（KNL）に代表されるメニーコアプロセッサが急速に普及しつつある．これらのプロセッサは，動作周波数が低く抑えられるなどの理由でシングルスレッド性能は低い傾向にある一方，従来のマルチコアプロセッサと比べて非常に多くのプロセッサコアをもち，また多数のスレッドを実行可能である．結果，プロセッサあたりの演算性能は従来のマルチコアプロセッサに比べて非常に高い．例えば，マルチコアプロセッサ Intel Xeon E5-2699A v4 が 22 コア，2.2 GHz，634 GFLOPS であるのに対して，メニーコアプロセッサ Intel Xeon Phi 7290 は 1.5 GHz，72 コア，6.9 TFLOPS である [1]．また，従来の CPU と同様に用いることが可能であり，GPU などと比べるとその柔軟性は高い．したがって，メニーコアプロセッサは，今後の有力な数値計算プラットフォームの一つになると考えられる．

本研究では，メニーコアプロセッサ上での三重対角行列固有値ソルバの性能について議論し，その高速化の手段を提案，評価する．

先行研究として，Pichon らによる分割統治法のマルチコアプロセッサ向けの並列実装に関する研究がある [2]．Pichon らの研究は後述する分割ツリーの各階層の並列性に着目している点は本研究と同様であるが，並列化と高性能

[1] 理化学研究所計算科学研究機構
RIKEN AICS, Kobe, Hyogo 650-0047, Japan
[a)] yusuke.hirota@riken.jp

化のアプローチには違いがある．アプローチの違いの詳細は後述する．また，Pichon らの研究はメニーコアプロセッサ上での動作を視野に入れてはいるものの，実際の評価も行われていない．本研究では，実際にメニーコアプロセッサ上で性能評価を行う．

本稿では，まず第 2 節で，密行列固有値ソルバをメニーコアプロセッサ上で効率的に実行するには，分割統治法実装の高速化の重要性が高いことを示す．第 3 節では，代表的な数値計算ライブラリ LAPACK の分割統治法 [3], [4] の実装について概説する．また，その実装がメニーコアプロセッサ上で効率的に実行されないことを示し，さらにその原因を考察する．第 4 節では，前節の考察をもとに，分割統治法の実装を高速化する手段について述べる．また，先行研究との違いについても述べる．その後，第 5 節では，第 3 節で述べた考察の妥当性，第 4 節で述べた高速化手段の一部の効果について，数値実験により確認する．最終節でまとめを述べる．

2. メニーコアプロセッサ上における分割統治法実装の高速化の重要性

メニーコアプロセッサ上での密行列固有値ソルバを効率的に実行するには，分割統治法の実装の高速化が重要である理由について述べる．

$n \times n$ 実対称密行列 B の固有値問題は，

(1) 直交行列 U_1 による B の三重行列 T への変換：$U_1^\top B U_1 \to T$（三重対角化），
(2) T のスペクトル分解：$T \to U_2 \Delta U_2^\top$（三重対角行列固有値問題），
(3) 直交行列 U の計算：$U \leftarrow U_1 U_2$（逆変換）

という 3 ステップを順次実行することで，B の固有値を対角要素にもつ対角行列 Δ，B の固有ベクトルを各列にもつ行列 U を求めるのが一般的である [5]．例えば，計算科学分野で標準的に使用される数学ライブラリである LAPACK[6] の密行列固有値ソルバ DSYEV，DSYEVD などでは，このような 3 ステップのアルゴリズムを用いられている．

三重対角行列固有値問題の解法には，分割統治法に加え，QR 法 [5]，MR^3[7] などがある．分割統治法は，固有値の近接度の高い問題に対する高速性や，行列積を中心的な演算とする現代的計算機で効率的な実装を作成しやすい特徴をもち，上述の解法の中でもっとも高速なものの 1 つとして知られている．このため，本研究では三重対角行列固有値問題の解法として分割統治法を用いる場合を考えている．

我々は予備実験として，Xeon E5-2660 プロセッサ 2 ソケット（以下では単に Xeon E5）上および Xeon Phi 3120P（ネイティブ実行，以下では単に Xeon Phi KNC）上で，三重対角行列固有値問題の解法に分割統治法を用いる密行列固有値ソルバ DSYEVD の実行時間の内訳を測定し，先述の 3 つのステップを実行するサブソルバ（順に DSYTRD, DSTEDC，DORMTR）の実行時間比重を比較した．その結果，メニーコアプロセッサ Xeon Phi KNC 上では，マルチコアプロセッサ Xeon E5 上の場合と比べ，密行列固有値ソルバ全体の実行時間に占める三重対角行列ソルバの実行時間は大きいことを見出した．Xeon E5 上および Xeon Phi KNC 上での予備実験の測定結果を，それぞれ図 1，図 2 に示す．Xeon E5 上では n の増大にしたがって DSTEDC の比重は低下し，$n \geq 4000$ では 3 ステップの中でもっとも実行時間が短い．一方，Xeon Phi KNC 上では，DSTEDC の比重は DSYTRD に次いで高く，次数大きい（$n \geq 3000$）行列では分割統治法ルーチンの実行時間の比重が Xeon E5 上で実行した場合よりも高くなっている．したがって，メニーコアプロセッサ上における密行列固有値ソルバを効率的に実行するには，分割統治法実装の高速化が重要度が高いといえる．

三重対角化（DSYTRD）の高速化もまた重要であるが，これは別のアプローチにより可能であると思われる．具体的には，先述の 3 ステップによる解法の中間形として，三重対角行列ではなく半帯幅が 2 以上の帯行列を用いるアプローチである．密行列から帯行列への変換 [8], [9] は，三重対角行列への変換と同程度の演算量でありながら要求する B/F 値が低い．このため中間形への変換がより短い時間で実行可能であると考えられる．また，逆変換の演算量は殆ど変わらず，実行効率も大きく変化しないと考えられるため，その実行時間は大きく変化しないと考えられる．一方，第 2 ステップでは三重対角行列固有値問題の代わりに帯行列の固有値問題を解くこととなり，その実行時間は増大する．したがって，三重対角行列ではなく帯行列を中間形に用いるアプローチを採る場合，中間形への変換（第 1 ステップ）はより高速化される一方，第 ii ステップの求解の比重はより大きくなると考えられる．帯行列の固有値問題に対する解法の 1 つに，帯行列向け分割統治法 [10], [11] がある．帯行列向け分割統治法は簡単に言えば，三重対角行列向けの分割統治法の繰り返しの適用である．したがって，帯行列を中間形に用いるアプローチで帯行列向け分割統治法を用いる場合には，三重対角行列向けの分割統治法の実装の高速化の重要性はますます高くなる．

3. LAPACK の分割統治法の実装とその性能

本節では，LAPACK の分割統治法ルーチン DSTEDC の実装について述べ，メニーコアプロセッサにおけるその性能および問題点について議論する．

3.1 分割統治法の概要

分割統治法では，以下の 3 つのステップにより，A の固有値（Λ の対角要素），固有ベクトル（Q の列ベクトル）を求める．

(1) 適当な分割サイズ n_1 を定め，

表 1　性能評価環境
Table 1　Computational environment.

CPU	Intel Xeon E5-2660（2.2 GHz，8 コア，ハイパースレッディング機能無効，141 GFLOPS）× 2 ソケット
主記憶	DDR3 メモリ 64 GB
アクセラレータ	Intel Xeon Phi 3120P（1.1 GHz，57 コア，228 スレッド, 1003 GFLOPS）
コンパイラ	Intel Fortran Compiler 17.0.0
BLAS/LAPACK	Intel Math Kernel Library 2017 および Netlib LAPACK 3.6.1

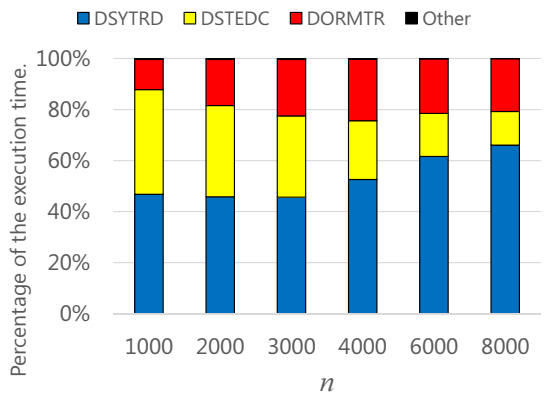

図 1　密行列固有値ソルバ DSYEVD の実行時間内訳（Xeon E5）
Fig. 1　Breakdown of the execution time of the dense eigensolver DSYEVD (on Xeon E5).

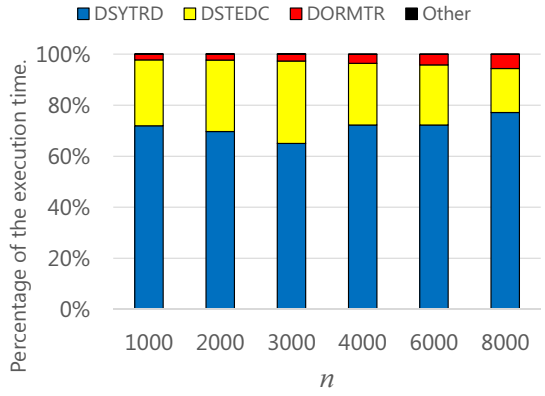

図 2　密行列固有値ソルバ DSYEVD の実行時間内訳（Xeon Phi KNC）
Fig. 2　Breakdown of the execution time of the dense eigensolver DSYEVD (on Xeon Phi KNC).

$$A = \begin{bmatrix} A_1 & O \\ O & A_2 \end{bmatrix} + \alpha \boldsymbol{v}\boldsymbol{v}^\top$$

を満たす三重対角の行列 $A_1 \in \mathbb{R}^{n_1 \times n_1}$，$A_2 \in \mathbb{R}^{(n-n1)\times(n-n1)}$，実数 α，実ベクトル $\boldsymbol{v} \times \mathbb{R}^n$ を求める（問題の分割）．

(2) A_1，A_2 の固有値問題

$$A_i = Y_i D_i Y_i^\top \quad (i = 1, 2)$$

を解く（小問題の求解）．

(3) 対角行列とランク 1 積の和で表される行列の固有値問題

$$\begin{bmatrix} D_1 & O \\ O & D_2 \end{bmatrix} + \alpha \boldsymbol{u}\boldsymbol{u}^\top = W \Lambda W^\top$$

の解を求める．ただし，

$$\boldsymbol{u} = \begin{bmatrix} Y_1 & O \\ O & Y_2 \end{bmatrix}^\top \boldsymbol{v}$$

である．その解と小問題の解の積

$$Q = \begin{bmatrix} Y_1 & O \\ O & Y_2 \end{bmatrix} W$$

が A の固有ベクトルとなる．（小問題の固有ベクトルのマージ）．

ステップ 2 の 2 つの小問題は互いに独立している．また，A_1，A_2 はともに実対称三重対角行列であるので，これらに対して再帰的に分割統治法を提供することが可能である．

小問題の固有ベクトルのマージは，さらに複数のステップに分割することができる．

(1) 次式を満たす置換行列 P_1, P_2，直交行列 G，対角要素が重複しない対角行列 $D' \in \mathbb{R}^{n' \times n'}$，対角行列 D''，ゼロ要素を含まないベクトル $\boldsymbol{u}' \in \mathbb{R}^{n'}$ を求める（デフレーション）．

$$P_2 G P_1 (D + \alpha \boldsymbol{u}\boldsymbol{u}^\top)(P_2 G P_1)^\top = \begin{bmatrix} D' & O \\ O & D'' \end{bmatrix} + \begin{bmatrix} \boldsymbol{u}' \\ \boldsymbol{0} \end{bmatrix} \left[(\boldsymbol{u}')^\top, \boldsymbol{0}^\top \right].$$

(2) $n' \times n'$ 行列の固有値問題

$$D' + \alpha \boldsymbol{u}' (\boldsymbol{u}')^\top = W' \Lambda' (W')^\top$$

を解く．このような特殊な対角行列とランク 1 積の和の n' 固有値 $\lambda'_1, \ldots, \lambda'_{n'}$ は，次の非線型方程式の n' 個の根と一致する．

$$f(\mu) = 1 + \alpha \sum_{i=1}^{n'} \frac{(u'_i)^2}{d'_i - \mu}$$

したがって，上記の方程式を何らかの解法で解くことにより固有値が求められる（secular 方程式の求解）．対応する固有ベクトル $\boldsymbol{w}'_1, \ldots, \boldsymbol{w}'_{n'}$ は

$$\left(D' + \alpha \boldsymbol{u}' (\boldsymbol{u}')^\top - \lambda_i I\right) \boldsymbol{w}'_i = \boldsymbol{0}$$

の解として求められる．実際には数値計算を安定に行うため，計算された固有値をもとに $\boldsymbol{u}'$ の再計算を行う必要があるが，本稿の内容に関係しないため省略する．このとき，固有値問題

$$D + \alpha \boldsymbol{u}\boldsymbol{u}^\top = W \Lambda W^\top$$

の解は

$$\Lambda = \begin{bmatrix} \Lambda' & O \\ O & \Lambda'' \end{bmatrix}, \ W = \begin{bmatrix} W' & O \\ O & I \end{bmatrix}$$

という形で表される．Λ'' は容易に求めることができるが詳細は省略する．

(3) A の固有ベクトルの

$$Q = \begin{bmatrix} Y_1 & O \\ O & Y_2 \end{bmatrix} P_1^\top G^\top P_2^\top W$$

を求める（固有ベクトル更新のための行列積）．行列積の順序には自由度が存在するが，実際の計算では疎性が失われないような順序を用いて計算量の削減などの工夫が行われる．

3.2 LAPACK の分割統治法の実装の概要

本副節では LAPACK の分割統治法ルーチン DSTEDC の実装の概要について述べる．Netlib による実装はソースコードが公開されているの一方，Intel MKL[12] を初めとする各種のプロプライエタリ実装は詳細が公開されていない．しかしながら，我々は，後者の実装は Netlib による実装をもとに最適化および並列化がなされていると信じている．そこで，本副節では，まず Netlib により公開されている実装について述べたあと，それを元にした Intel MKL での並列化がどのように行われているか推測を行う．

Netlib による DSTEDC の実装について説明する．Netlib による実装は，図 3 に示すように，(i) 小問題に対して問題の次数が十分小さくなるまで分割統治法を再帰的に適用して分割ツリーを構成し，(ii) 分割ツリーの葉の各小問題（葉問題）を QR 法によって解き，(iii) 小問題の固有ベクトルを再帰的にマージする，という 3 つのステップから構成される．以下では，分割ツリーの根に近い側を上層，分割ツリーの葉に近い側を下層と呼ぶことにする．ステップ iii の再帰的マージでは，分割ツリーの同じ階層に属する各マージ間には依存性がなく，並列に実行することが容易である．しかしながら，本実装は逐次処理環境に向けて作られており [13]，単純に最下層から幅優先でマージ処理を逐次実行していく．各マージ処理の中では，デフレーションを行うルーチン（DLAED2）secular 方程式の複数の根を求めるルーチン（DLAED4）の DO ループによる呼び出し，行列積

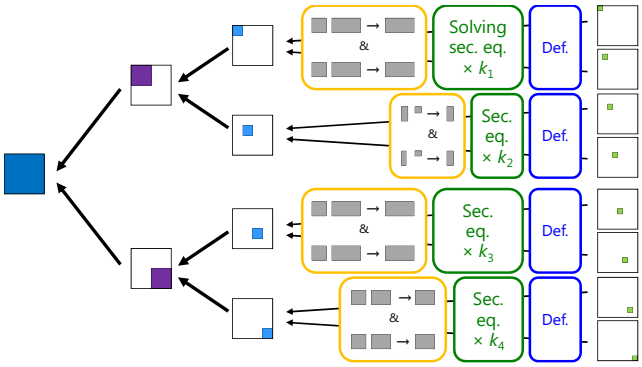

図 4 小問題の解の再帰的マージ

Fig. 4 Recursive merge of the solutions of the small problems in DSTEDC.

$$\begin{bmatrix} Q_{1,1} & Q_{1,2} \\ Q_{2,1} & Q_{2,2} \end{bmatrix} = \begin{bmatrix} \tilde{Y}_{1,1} & \tilde{Y}_{1,2} & O & \tilde{Y}_{1,4} \\ O & \tilde{Y}_{2,2} & \tilde{Y}_{2,3} & \tilde{Y}_{2,4} \end{bmatrix} \begin{bmatrix} \tilde{W} & O \\ O & I \end{bmatrix}$$

を求めるための行列積ルーチン（DGEMM）の 2 回の呼び出しが行われる．ただし，P_3 は

$$\begin{bmatrix} \tilde{W} & O \\ O & I \end{bmatrix} = P_3^\top \begin{bmatrix} W' & O \\ O & I \end{bmatrix},$$

$$\begin{bmatrix} \tilde{Y}_{1,1} & \tilde{Y}_{1,2} & O & \tilde{Y}_{1,4} \\ O & \tilde{Y}_{2,2} & \tilde{Y}_{2,3} & \tilde{Y}_{2,4} \end{bmatrix} = \begin{bmatrix} Y_1 & O \\ O & Y_2 \end{bmatrix} P_1^\top G^\top P_2^\top P_3$$

を満たす置換行列である．本実装を実行した場合，通常はステップ iii が実行時間の大部分を占めるため，デフレーション，secular 方程式の求解，行列積の各性能が分割統治法の実装全体に大きな影響を与える．

続いて，Intel MKL での並列化された実装について述べる．先述の通り，Intel MKL の各ルーチンは実装方式が公開されていないため．そのため，予備テストによる実行時間の振る舞いなどをもとに推測している．Intel MKL の実装では，BLAS レベルでのスレッド並列化と，secular 方程式の求解のスレッド並列化が行われていると予想される．したがって，行列積ルーチン DGEMM の実行の度にスレッドフォークが行われ，その並列に処理が実行されている．また，1 回のマージごとに（最大でマージ後の行列次数個の）secular 方程式の根を求めるが，根を一つ求めるルーチンを呼び出す DO ループには並列性があり，ループ並列化がなされている．一方，同じ階層に属する複数のマージの並列性は利用されていない．

3.3 マルチコアプロセッサ上およびメニーコアプロセッサ上での性能評価

表 1 の Xeon E5 上および Xeon Phi KNC 上でネイティブ実行により，Intel MKL の DSTEDC の性能評価を行う．テスト行列には，Frank 行列の逆行列である三重対角行列，

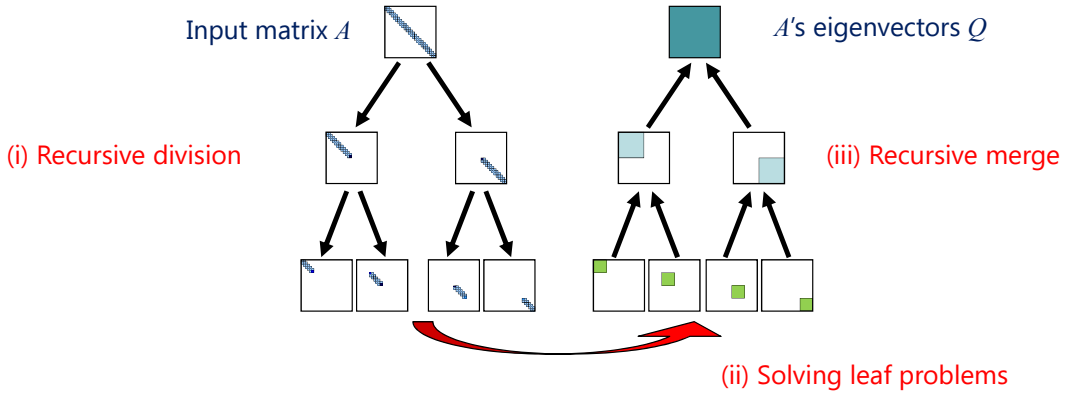

図 3 LAPACK の分割統治法実装
Fig. 3 LAPACK implementation of the divide-and-conquer method.

すなわち

$$A = \begin{bmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \quad (1)$$

を用いる．Frank 行列の逆行列は固有値の近接度が低く，分割統治法を適用した場合にデフレーションが発生が少ないという性質がある．

行列の次数を $n = 1000, 2000, 4000, 8000, 12000$ としたときの DSTEDC の実行時間を図 5 に示す．Xeon Phi KNC の浮動小数点演算性能は Xeon E5 の約 3.6 倍高速であるが，DSTEDC の実行時間は最大で 1.5 倍程度しか高速化されていない．また，$n \leq 4000$ の小さな問題ではかえって遅くなっていることが確認できる．以上の結果から，DSTEDC の実装には，マルチコアプロセッサ Xeon E5 上では効率的に動作するにも関わらず，メニーコアプロセッサ Xeon Phi KNC 上で効率的に動作しない問題があると考えられる．

3.4 メニーコアプロセッサ上での非効率性の原因

Intel MKL の DSTEDC が前述の推測のとおりに実装されていると仮定し，DSTEDC が Xeon Phi KNC 上で効率的に実行されない原因について，(i) デフレーション，(ii) secular 方程式の求解，(iii) 固有ベクトル更新のための行列積の 3 つの実行ステップに分けて考察する．本副節内では，サブツリーのマージ後の行列の次数を $m \times m$ とする．

3.4.1 デフレーション

デフレーション処理（DLAED2）では 2 個の m 要素の配列について，それぞれゼロ要素を含むか，重複要素をもつか判定を行い，そのような場合には，さらに要素の交換や Givens 回転などを適用する．これらの判定に関する処理は並列度は最大でも $O(m)$ しかない．ところが，分割ツリーの下層に進むにつれて次数 m は半減していく．表 1

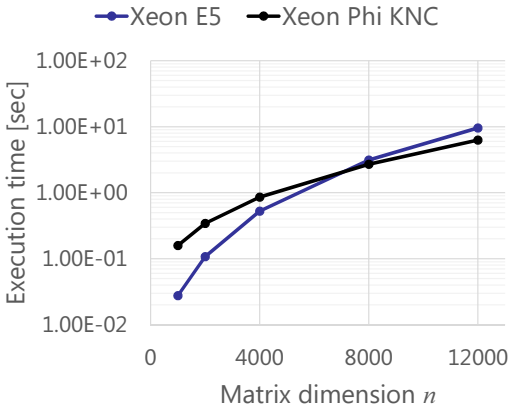

図 5 Xeon E5 上および Xeon Phi KNC 上での DSTEDC の実行時間
Fig. 5 Execution time of the routine DSTEDC on Xeon E5 and Xeon Phi KNC.

の環境では Intel MKL の DSTEDC を実行したとき，最下層のマージでは $m = 50$ 程度でしかない．このため，非常に多くのスレッド数をもつメニーコアプロセッサの並列性能を有効に利用できていないと考えられる．

また，処理の多くの分岐を含むことは，分岐ペナルティが相対的に大きい Xeon Phi KNC の性能を阻害する追加の要因となる．

これらの要因のため，Xeon Phi KNC のピーク浮動小数点演算性能は Xeon E5 の 3 倍以上高速であるにもかかわらず．デフレーション処理の実行時間はそれに見合った短縮がなされなかったと考えられる．

3.4.2 Secular 方程式の求解

デフレーションによる減次が発生しないとすれば，マージ後の固有値は m 個の secular 方程式の根として求められる．LAPACK の secular 方程式求解ルーチン（DLAED4）では，secular 方程式の根をある種の反復法 [14] により求めている．最大 m 個ある secular 方程式の根の計算は完全に独立に実行できる．一方，反復解法自体が逐次性をもつことに加え，根の求解が精度に敏感な計算であるために，反復法ルーチン DLAED4 自体の並列化を行うことは難し

い．したがって，secular 方程式の求解の実行ステップでは，解くべき secular 方程式の根の個数（最大 m）の並列度が存在することになる．ところが，第 3.4.1 節で述べたとおり，分割スリー下層に進むにれて m は減少して，最下層では $m = 50$ 程度でしかない．このため，デフレーションと同様，Xeon Phi KNC の並列性能を活かすことができないと考えられる．

また，Xeon Phi KNC は Xeon E5 と比べて長い SIMD 幅の命令を使用可能だが，DLAED4 ではそのような長い SIMD 幅の命令を利用することを考慮した設計となっていないため，その SIMD 長の増大による性能向上が得られない可能性がある．

これらの理由により，デフレーション処理同様，Xeon Phi KNC のピーク浮動小数点演算性能は Xeon E5 の 3 倍以上高速であるにもかかわらず．secular 方程式の求解の実行時間はそれに見合った短縮がなされないと考えられる．

3.4.3 固有ベクトル更新のための行列積

分割統治法では，分割ツリーのマージ処理において，2 回の密行列積ルーチン（DGEMM）が実行される．デフレーションを考えない場合，実行される行列積の次数は $C = AB, A \in \mathbb{R}^{k\times k}, B, C \in \mathbb{R}^{k\times 2k}$ となる．ただし，$k = m/2$ である．各階層におけるマージの回数は第 i 層では 2^{i-1} 回なので，各階層ごとの合計演算量は第 1 層では n^3，第 2 層合計は $(1/4)n^3$，第 3 層合計は $(1/4)^2 n^3$ と階層を下るにつれて 1/4 倍される．このため，行列積の性能が常に一定であれば分割統治法の行列積の実行時間の殆どは上層の行列積によって占められるはずである．

ところが，次数 k が小さくなると，次に述べる理由により，行列積の性能が低下する可能性がある．

- 行列の一方向（行または列）の並列度が低くなるため，処理のスレッドへの分割の SIMD 幅の大きい命令を効率的に使用することが難しくなる．
- 行列の各要素は $O(k)$ 回の再利用が可能であるが，k が小さくなるにつれてその回数は減少し，データを主記憶から取り出す時間が無視できなくなる．

Xeon Phi KNC は Xeon E5 よりも多くのスレッド数および長い SIMD 幅の命令を用いる．また，Xeon E5 の B/F が約 0.36 であるのに対して，Xeon Phi KNC では約 0.24 と相対的な主記憶アクセスコストが大きい．このため，k が小さくなったときの Xeon Phi KNC 上での行列積の性能低下率は，Xeon E5 上での低下率よりも大きくなると考えられる．図 6 に，Xeon E5 上および Xeon Phi KNC 上での DGEMM の性能を示す．Xeon Phi KNC 上での DGEMM の性能は，次数 k が大きい場合に Xeon E5 上での 3 倍近い性能を示すが，k が小さくなるにつれて性能は急激に低下し $k \leq 300$ では Xeon E5 上での性能よりも低くなっていることが確認される．

このように，Xeon Phi KNC 上では Xeon E5 上の場合

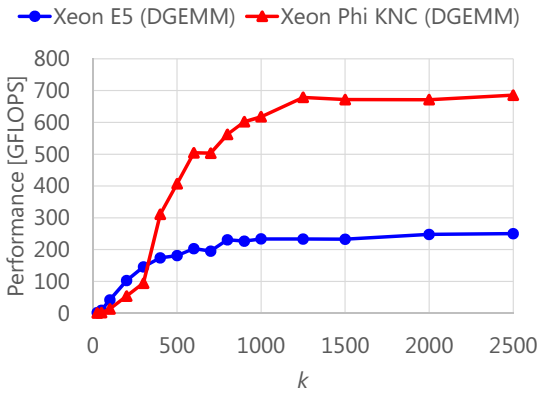

図 6 DGEMM の性能（$C = AB, A \in \mathbb{R}^{k\times k}, B, C \in \mathbb{R}^{k\times 2k}$）
Fig. 6 Performance of the DGEMM routine ($C = AB, A \in \mathbb{R}^{k\times k}, B, C \in \mathbb{R}^{k\times 2k}$).

と比べて，k の減少による行列積の性能低下率が高いという性質がある．したがって，Xeon Phi KNC 上では分割ツリー下層の固有ベクトル更新のための行列積が効率的に実行されず，DSTEDC の性能が低下する一因となっていると予想できる．

4. 実装の高速化

本節では，第 3.4 の考察に基づき，分割統治法実装の高速化の手段について述べる．また，提案する手法と先行研究の違いについて簡潔に述べる．

4.1 高速化の手法

Intel MKL の実装では，デフレーション，secular 方程式の求解，行列積のいずれでも，分割ツリーの下層に進むほど行列の次数が小さくなるため，メニーコアプロセッサのスレッド数に見合った並列性が確保できないことが性能低下の原因であると考察された．一方で，同階層に属する複数のマージ間の並列性はまったく使われていなかった．そこで，我々は，各マージ内の行列の次数に依存する並列性と，同階層の複数のマージ間の並列性の，二重の並列構造を展開して利用することを考える．具体的には，図 7 に示すように，同じ階層に属するマージ内のデフレーション，secular 方程式の求解，行列積を，それぞれの処理ごとにグループ化し，グループ内の処理を並列実行する．このとき各グループには（階層内のマージの個数）×（処理内の並列度）の（すなわち $O(n)$ 以上の）並列度が存在し，最初に与えられる行列の次数 n が大きければ十分な並列性が確保できる．また，同一の処理をまとめることは単に並列度を確保できる以上の長所が存在する．近年，Intel や NVIDIA などのハードウェアベンダーは，複数の行列積をまとめて実行する batch GEMM[15], [16] とよばれるルーチンを提供している．これらのルーチンでは，レイテンシの隠蔽や適切な負荷バランスの決定が内部的に行われ，単に行列積ルーチンを並列に実行する場合よりも高い性能を実現する

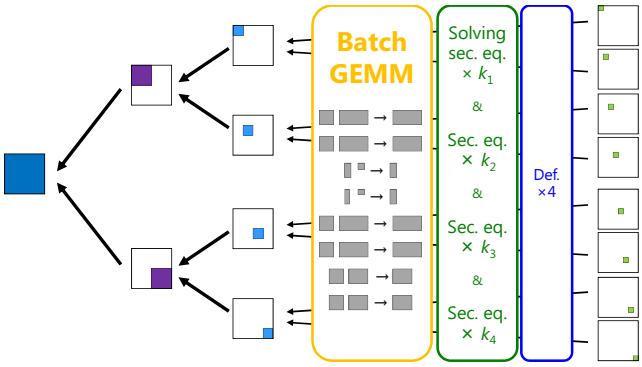

図 7 提案する実装における小問題のマージ
Fig. 7 Merge procedure in the proposed implementation.

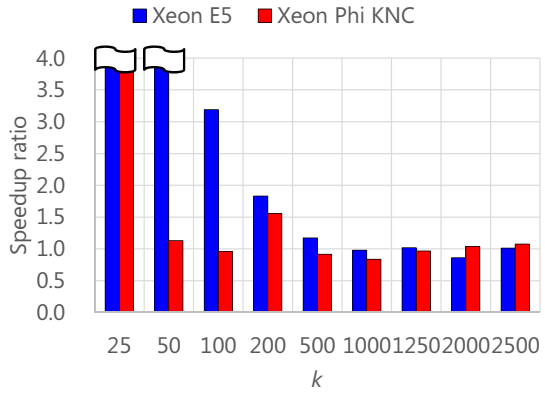

図 8 Batch DGEMM の DGEMM に対する高速化率 ($C_i = A_iB_i$, $A_i \in \mathbb{R}^{k\times k}$, $B_i, C_i \in \mathbb{R}^{k\times 2k}$, $i = 10000/k$)
Fig. 8 Speedup of batch DGEMM over DGEMM ($C_i = A_iB_i$, $A_i \in \mathbb{R}^{k\times k}$, $B_i, C_i \in \mathbb{R}^{k\times 2k}$, $i = 10000/k$).

ことを可能にしている．図 8 に batch DGEMM を使用したときの，DGEMM に対する高速化率を示す．いずれのサイズでも DGEMM と同程度以上の性能を示しており，特に，行列サイズが小さい場合の高速化率が高いことが確認できる．

同階層のマージ間の並列性を利用する他の方法としては，マージルーチンを並列に実行し，それぞれ複数のスレッドを割り当てるという方式も考えられる．しかしながら，マージのコストはデフレーションの結果に依存するため，先見的に各マージのコストを予想して適切な割り当てスレッド数を決定することは難しい．動的に割り当てる方法も考えられるが，ツリー下層ではマージ 1 回あたりの実行時間が短いため，レイテンシの増大を招くおそれがある．一方，先述の並列性を展開する手法では，そのような問題は発生しない．また，将来的に性能が非対称な並列プロセッサに移植する場合にも並列性を展開する手法は有利である．

4.2 先行研究との比較

Pichon らの先行研究の手法と提案手法を比較したとき，分割ツリーの各階層の並列性に着目した並列化を行っているという点は同様である．しかしながら，先行研究の手法では，彼らの並列化手法はマージ内で実行される一連の処理をより細かなタスクに分割し，動的スケジューラによりスレッドに割り当てて並列実行するというタスクフローモデルの並列化が行われている．一方，我々の提案する方法では，複数のマージ間の同一の処理（デフレーション，行列積など）をグループとして一つにまとめ，それらに対して fork/join モデルの並列化が行われている．我々の並列化手法は，全スレッド間の同期が $O(\log n)$ 回の必要となる短所があるが，同一処理のグループ化によって batch DGEMM による高性能化が促進されるという長所がある．また，短所である同期コストについても全体の演算量は $O(n^2)$ から $O(n^3)$ となるので重大なオーバーヘッドにはならないと考えている．

5. 数値実験

分割統治法の従来実装の詳細な性能評価を行い，第 3.4 節で述べた性能低下原因に対する考察の妥当性を確認する．また，第 4 節で提案した高速化技術の一部を導入した実装の性能評価を行い，その効果を示す．

本節の数値実験結果はすべて表 1 の Xeon E5 上での実行または Xeon Phi KNC 上でネイティブ実行して得られたものである．テスト行列には，$n = 12000$ の Frank 行列の逆行列である三重対角行列 (1) を用いる．

まず, DSTEDC の評価を行う．本評価では, DSTEDC の実行時間全体およびマージの実行時間内訳を測定する．測定対象とするのは, 分割ツリーの階層別のデフレーション実行時間 (DLAED2), secular 方程式の根の計算 (DLAED4), 固有ベクトル更新のための行列積（DGEMM）である．Intel MKL の DSTEDC を直接実行したのでは実行時間の内訳が得られないため，これら測定対象を呼び出す上位ルーチン DSTEDC，DLAED0，DLAED1，DLAED3 については Netlib による LAPACK の `DSTEDC.f`，`DLAED0.f`，`DLAED1.f`，`DLAED3.f` を改造したものを測定に使用する．改造の内容は，OpenMP による DO ループの副作用のない並列化ならびに，実行時間測定のためのタイミングルーチンの設置である．

Xeon E5 上での測定結果を図 9 に，Xeon Phi KNC 上での測定結果を図 10 に示す．ただし，これらのグラフでは分割ツリーの最上位層をレベル 1，その下の階層をレベル 2，以下同様にレベル 3，4，... と表現し，その階層ごとにマージ処理の実行時間とその内訳を表示している．また，レベル 6 以下の階層はそれらの内訳を合計して表している．Xeon E5 上では，実行時間の殆どが分割ツリーの上層のマージに費やされており，レベル 3 以下の下層の実行時間はマージ部全体の実行時間の 8.0%にすぎない．また，デフレーション（DLAED2）や secular 方程式の求解（DLAED4）の実行時間の占める割合は無視できるほど小

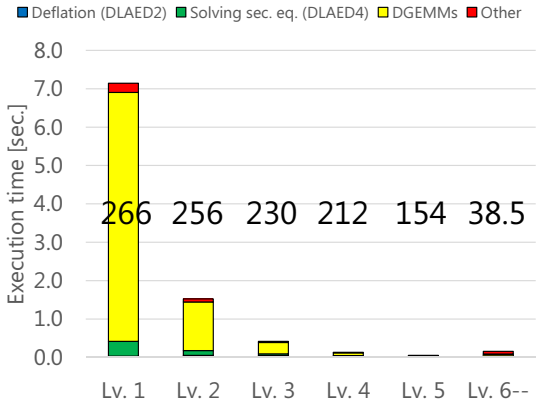

図 9 DSTEDC の分割ツリーの階層別の実行時間内訳（$n = 12000$, Xeon E5）．ただし，グラフ上の数値は DGEMM の性能（GFLOPS）を意味する．

Fig. 9 Breakdown of the execution time of DSTEDC by divide-and-conquer tree layer ($n = 12000$, on Xeon E5), where the numbers on the graph indicate the performance of DGEMM in GFLOPS.

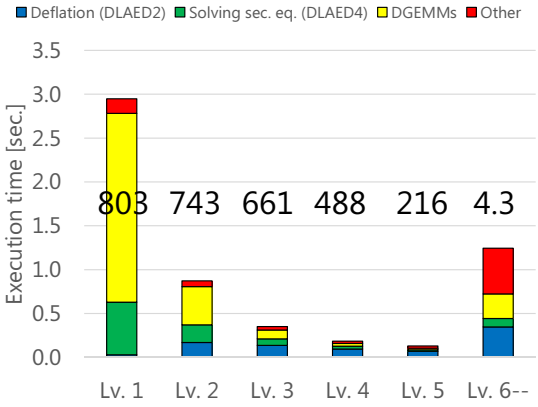

図 10 DSTEDC の分割ツリーの階層別の実行時間内訳（$n = 12000$, Xeon Phi KNC）．ただし，グラフ上の数値は DGEMM の性能（GFLOPS）を意味する．

Fig. 10 Breakdown of the execution time of DSTEDC by divide-and-conquer tree layer ($n = 12000$, on Xeon Phi KNC), where the numbers on the graph indicate the performance of DGEMM in GFLOPS.

さく，実行時間の殆どが DGEMM によって占められている．その DGEMM の性能は高く，行列次数が小さくなるレベル 6 以下の層でも平均して 40 GFLOPS の性能を示している．これに対して，Xeon Phi KNC 上では，DGEMM の性能はレベル 1 では 803 GFLOPS と高い性能を示しているが，ツリーの下層に進むほど（すなわち行列次数が小さくなるほど）急激に低くなり，レベル 6 以下の層では平均 4.3 GFLOPS でしかない．結果として，レベル 3 以下の下層の実行時間はマージ部全体の実行時間の 33%と長くなっている．また，DGEMM の性能が不十分であるにもかかわらず，デフレーションや secular 方程式の求解の実行時間の割合が Xeon E5 上での場合より高く，これらの部分が効率的に実行されていないことが確認できる．これらの結果は第 3.4 節で述べた原因から予想される振る舞いに一致しており，その妥当性を支持するものとなっている．

続いて，第 4 節で提案した高速化技術の一部を導入した実装の性能評価を行う．本節で評価する実装は，先述の Netlib による LAPACK と MKL を組み合わせた実装を元に，同階層の行列積の batch 化のみを取り入れている．テスト行列の次数 $n = 12000$ としたときの，提案実装のマージ実行時間の内訳を図 11 に示す．分割ツリーの上層（レベル 1 からレベル 5）では行列積の性能は殆ど変化していないが，下層（レベル 6 以下）の行列積の平均性能は 20.6FLOPS と従来の 4.8 倍高速化されており，行列積の階層別の batch 化による高速化の効果が確認できる．副次的な効果として，レベル 6 以下のデフレーションおよびその他の実行時間が高速化されている．詳細な解析はまだ行われていないが，行列積というメモリ上の広い範囲をアクセスする操作を階層別に最後にまとめて行うようにしたことで，その他の操作に必要なデータがキャッシュメモリか

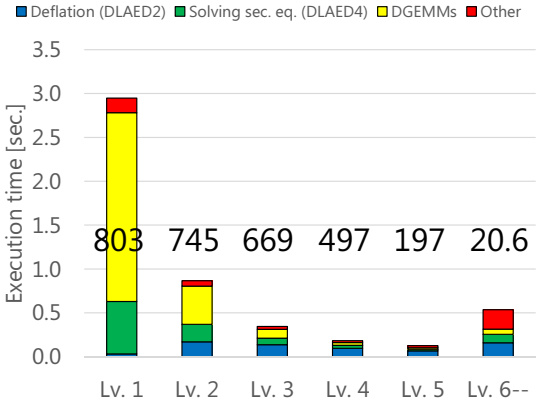

図 11 改良された実装の分割ツリーの階層別の実行時間内訳（$n = 12000$）．ただし，グラフ上の数値は DGEMM の性能（GFLOPS）を意味する．

Fig. 11 Breakdown of the execution time of the improved implementation by divide-and-conquer tree layer ($n = 12000$, on Xeon Phi KNC), where the numbers on the graph indicate the performance of DGEMM in GFLOPS.

ら追い出されずに再利用された効果ではないかと考えられる．分割統治法全体としての実行時間は，従来実装は 6.3 秒であったのに対し，改良実装は 5.4 秒となり約 1.15 倍の高速となった．したがって，提案した高速化技術の一つである行列積の階層別 batch 実行は，高速化の効果があることが確認できた．

6. おわりに

メニーコアプロセッサ上で密行列の固有値問題を効率的に解く上で，分割統治法の実装の高速化が重要になる理由について述べた．また，LAPACK の分割統治法の実装について概説し，その実装がメニーコアプロセッサ上で効率

的に実行されない理由について考察した．さらに，その考察に基づいて，メニーコアプロセッサ上で実装を高速化するための手段を提案した．数値実験を行い，考察が妥当であること，および提案手法の一部を実装に取り入れることにより，メニーコアプロセッサ上で高速化されることの 2 点を確認した．

今後は，行列積の batch 化以外の高速化手段についても実装を行い，それらの効果を評価することを予定している．また，最新のメニーコアプロセッサである Intel Xeon Phi KNL 上でも性能評価を行うことも予定している．

謝辞 本研究は JSPS 科研費 15H02709 の助成を受けている．

参考文献

[1] Intel ARK — Your Source for Intel(R) Product Specifications (online): `https://ark.intel.com/` (2017.02.06).
[2] Pichon, G., Haidar, A., Faverge, M. and Kurzak, J.: Divide and conquer symmetric tridiagonal eigensolver for multicore architectures, *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, IEEE, pp. 51–60 (2015).
[3] Cuppen, J.: A divide and conquer method for the symmetric tridiagonal eigenproblem, *Numerische Mathematik*, Vol. 36, No. 2, pp. 177–195 (1980).
[4] Dongarra, J. J. and Sorensen, D. C.: A fully parallel algorithm for the symmetric eigenvalue problem, *SIAM Journal on Scientific and Statistical Computing*, Vol. 8, No. 2, pp. s139–s154 (1987).
[5] Golub, G. H. and Van Loan, C. F.: *Matrix computations fourth edition*, Johns Hopkins University Press (2013).
[6] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D.: *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition (1999).
[7] Dhillon, I. S.: A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem, PhD Thesis, EECS Department, University of California, Berkeley (1997).
[8] Bischof, C. H., Lang, B. and Sun, X.: Algorithm 807: The SBR Toolbox—software for successive band reduction, *ACM Transactions on Mathematical Software (TOMS)*, Vol. 26, No. 4, pp. 602–616 (2000).
[9] Bischof, C. H., Lang, B. and Sun, X.: A framework for symmetric band reduction, *ACM Transactions on Mathematical Software (TOMS)*, Vol. 26, No. 4, pp. 581–601 (2000).
[10] Arbenz, P.: Divide and conquer algorithms for the bandsymmetric eigenvalue problem, *Parallel computing*, Vol. 18, No. 10, pp. 1105–1128 (1992).
[11] Gansterer, W. N., Schneid, J. and Ueberhuber, C. W.: A Divide-and-Conquer Method for Symmetric Banded Eigenproblems-Part I: Theoretical Results (1999).
[12] Intel Math Kernel Library (online): `https://software.intel.com/en-us/intel-mkl` (2017.02.06).
[13] Rutter, J. D.: A Serial Implementation of Cuppen's Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem, Technical Report UCB/CSD-94-799, EECS Department, University of California, Berkeley (1994).
[14] Li, R.-C.: Solving Secular Equations Stably and Efficiently, Technical Report UCB/CSD-94-851, EECS Department, University of California, Berkeley (1994).
[15] Introducing Batch GEMM Operations (online): `https://software.intel.com/en-us/articles/introducing-batch-gemm-operations` (2017.02.06).
[16] cuBLAS :: CUDA Toolkit Documentation (online): `http://docs.nvidia.com/cuda/cublas/#cublas-lt-t-gt-gemmbatched` (2017.02.06).