情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 2　20–29 (June 2010)

実対称固有値問題に対する多分割の分割統治法の分散並列アルゴリズムの提案

田村純一[†1,*1]　坪谷怜[†2]
桑島豊[†1]　重原孝臣[†1]

本稿では，最近提案された実対称固有値問題に対する多分割の分割統治法（DCK）の分散並列アルゴリズムを提案する．特に，DCK で必要とされる再直交化を高性能で実行するための方法について詳しく述べる．HITACHI SR11000 を利用した数値実験で，速度と精度を他の実対称固有値問題解法と比較することにより，提案手法の有効性を確認する．比較対象としては，ScaLAPACK に実装されている解法と HPCS2009 において報告した共有並列の DCK を用いる．

Proposal of Distributed Parallel Algorithm of Multiple Division Divide-and-conquer for Real Symmetric Eigenproblem

Junichi Tamura,[†1,*1] Satoshi Tsuboya,[†2]
Yutaka Kuwajima[†1] and Takaomi Shigehara[†1]

For a recently proposed multiple division divide-and-conquer (DCK) algorithm for real symmetric eigenproblem, we propose a parallel algorithm suitable for distributed-memory parallel computers. A special emphasis is put on a treatment to keep the performance at high level in the reorthogonalization procedure required in DCK. The efficiency of the proposed algorithm is confirmed through numerical experiments on HITACHI SR11000, by comparing the accuracy and the performance with the solvers for real symmetric eigenproblem available in ScaLAPACK as well as our parallel implementation of DCK for shared-memory parallel computers reported in HPCS2009.

1. はじめに

実対称固有値問題を数値的に解く際には，まず実対称三重対角行列に相似変換した後，固有対を求めることが一般的である．固有対を求める数値解法としては，古典的には二分法・逆反復法（BII），QR 法が知られ，最新の方法としては二分割の分割統治法[1)]（DC2）と MRRR 法[2)] が知られている．さらに，DC2 を拡張した解法として，多分割の分割統治法[3)]（DCK）が提案されている．DCK は分割数 k をパラメータに持ち，$k = 2$ の場合に DC2 を含む DC2 の自然な拡張となっている．

DCK はすでに逐次計算機や共有メモリ型並列計算機向け[4)] に実装されている．共有メモリ型並列計算機上では，DCK の持つ並列性を活用し他の解法と同等以上の性能となることが示されている．より大規模な問題を扱うためには分散メモリ型並列計算機への実装が不可欠であるが，これまでに DCK の分散並列実装は行われていない．

そこで本稿では，分散メモリ型並列計算機への効率的な DCK の実装を提案する．DCK の分散並列実装においては，固有ベクトルの再直交化に対する効率的な分散並列化が鍵となる．DC2 では固有ベクトルの直交性を確保するために Löwner の定理に基づいた手法を用いて計算を行う．他方 DCK ではそれに相当する定理が未発見であるため，再直交化を行い直交性を向上させる．この再直交化の分散並列化手法は，一般には自明ではない．ScaLAPACK の BII で行われる再直交化は，同一プロセスの持つ固有ベクトルのみで行っており，固有分解全体としては精度が低い．しかし，本稿で提案する再直交化手法は，他プロセスの持つ固有ベクトルとも再直交化を行う．しかも，本稿の提案手法は共有並列の DCK と同等の性能を持ち，全体としても ScaLAPACK に実装された BII や DC2 より高速であることを数値実験により示す．

以降では，まず 2 章で DCK の逐次アルゴリズムを示し，3 章でその各ステップの並列化手法について述べる．4 章では数値実験により提案手法の有効性を検証する．5 章ではまとめと今後の課題を述べる．

†1 埼玉大学大学院理工学研究科
Graduate School of Science and Engineering, Saitama University
†2 コンピュートロン株式会社
Computron Co., Ltd.
*1 現在，沖ソフトウェア株式会社
Presently with Oki Software Co., Ltd.

2. DCK のアルゴリズムと deflation

DCK は，主対角要素 a_j（$1 \le j \le n$），副対角要素 b_j（$1 \le j \le n-1$）を持つ n 次実対称三重対角行列 T と，分割数 k（$2 \le k \ll n$）を入力として，T の固有分解 $T = Q\Lambda Q^T$ を与える n 次直交行列 Q と実対角行列 Λ を出力する．**図 1** に DCK のアルゴリズムの概要を示す．$n = m \times k$（m は正整数）とする．説明を簡単にするため，以降でもこの仮定をおく．重要なステップには `Tj`，`eval` などの名前を付け，n 次単位行列 I_n の第 l 列ベクトルを $\boldsymbol{e}_l$ と表記する．

DCK では，指定する分割数 k を大きくすることで，ステップ (3) の直交行列 P の非零要素数が減少する．P の非零要素数が減少することで，ステップ (9) で T の固有ベクトルを計算する際に，行列積の演算量を削減できる．DCK の計算量のうち唯一 $O(n^3)$ かかるのがこの行列積であり，この演算量を削減することで高速化を図ることができる．ただし，k が大きいと $D + UCU^T$ の固有分解の演算量が大きくなる．実装の際には，ステップ (4) の deflation と呼ばれる操作により，実対角行列と低階数摂動の和 $D + UCU^T$ から自明な固有値・固有ベクトルを取り除く．この操作によって，$D + UCU^T$ の固有分解の実質的なサイズが減少し，かつステップ (9) の行列積の演算量が減少する．この操作は DC2，DCK の速度向上に大きく貢献している．ただし deflation は k が小さいときに多く起こることが知られており，単純に k を大きくするだけでは高速化することはできない．

(1)　$T = \bigoplus_{j=1}^{k} T_j + VCV^T$ と分割．ただし T_j は m 次実対称三重対角行列，
$V \equiv (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{k-1})$，$\boldsymbol{v}_j \equiv \boldsymbol{e}_{jm} + \mathrm{sign}(b_{jm})\boldsymbol{e}_{jm+1}$，
$C \equiv \mathrm{diag}(|b_m|, \ldots, |b_{(k-1)m}|)$

(2)　固有分解 $T_j = Q_j \Lambda_j Q_j^T$（$j = 1, \ldots, k$）を計算（`Tj`）

(3)　$T = P(D + UCU^T)P^T$ と変形．ただし
$P \equiv \bigoplus_{j=1}^{k} Q_j$，$D \equiv \bigoplus_{j=1}^{k} \Lambda_j$，$U \equiv P^T V$

(4)　適当な置換行列 P_d により $D + UCU^T$ を r 次実対称行列 $D_1 + U_1 C U_1^T$ と $n-r$ 次実対角行列 D_2 との直和の形に変形，すなわち
$P_d^T(D + UCU^T)P_d = (D_1 + U_1 C U_1^T) \oplus D_2$ とする．この操作を deflation と呼ぶ

(5)　$D_1 + U_1 C U_1^T$ のすべての固有値 $\tilde{\lambda}_j$ を計算し $\tilde{\Lambda} \equiv \mathrm{diag}(\tilde{\lambda}_1, \ldots, \tilde{\lambda}_r)$ とする（`eval`）
ここで T の全固有値 $\Lambda \equiv \tilde{\Lambda} \oplus D_2$ が求まる

(6)　$D_1 + U_1 C U_1^T$ のすべての固有ベクトル $\tilde{\boldsymbol{q}}'_j$ を計算し，$\tilde{Q}' \equiv (\tilde{\boldsymbol{q}}'_1, \ldots, \tilde{\boldsymbol{q}}'_r)$ とする（`evec`）

(7)　固有ベクトル $\tilde{\boldsymbol{q}}'_j$ の中で再直交化が必要なベクトルを見つけ出し，グループにまとめる（`group`）

(8)　非直交な固有ベクトルどうしを再直交化する．$\tilde{\boldsymbol{q}}'_j$ に対応する再直交化後の固有ベクトルを $\tilde{\boldsymbol{q}}_j$ とし，$\tilde{Q} \equiv (\tilde{\boldsymbol{q}}_1, \ldots, \tilde{\boldsymbol{q}}_r)$ とする（`reortho`）

(9)　$Q \equiv PP_d(\tilde{Q} \oplus I_{n-r})$ を計算（`prod`）

図 1　DCK のアルゴリズム
Fig. 1　DCK algorithm.

deflation 発生率を δ としたとき，計算時間は $(1-\delta)^2$ に比例し，δ が大きい場合には，行列積の演算量を大幅に削減できるため，速度の面で $k = 2$ すなわち DC2 が最適である．一方 δ が小さい場合は，k が大きいときに計算時間が最小となるが，最適な分割数を選択しても δ が大きい場合と比較して計算時間が長い．このため，行列ごとに最適な k を設定することが重要であるが，これについては本稿では扱わない．分割数の自動チューニングについては，他誌[5] を参照されたい．本稿では，分割数 k を 2 より大きくすることで高速化を見込める，δ が小さい状況に興味を置いている．

3. DCK の分散並列化手法

本章では，DCK を分散メモリ型並列計算機へ実装する方法を提案する．並列化には MPI[6] のみを使用し，OpenMP[7] を併用するハイブリッド並列化は行っていない．以降では用いるプロセスの数を p とし，各プロセスを $p_0, p_1, \ldots, p_{p-1}$ と呼ぶ．本実装は分割数 k が p の倍数であることを仮定している．入力 T，k は全プロセスが保持しているものとする．

DCK のアルゴリズムのうち，`Tj`，`eval`，`evec` には本質的に高い並列性が存在するため，自然に並列化することができる．3.1 節でこれらのステップの並列化について述べる．一方，`group`，`reortho` については効率のよい並列化手法が自明でなく，問題に応じて適切に負荷が分散するような手法を考える必要がある．3.2 節ではこの並列化手法について述べる．また，`prod` は演算時間の多くを占め，速度の面で重要であるため，3.3 節で詳しく述べる．以下では，簡単のため r は p の倍数であるとする．

3.1 比較的自明な部分の並列化

`Tj` で計算する各小問題（T_j の固有分解）は完全に独立しているため，各プロセスに k/p 個の小問題を解かせることで並列化することができる．本実装では，プロセス p_j は T_{pl+j+1}（$l = 0, \ldots, k/p-1$）を担当し，それぞれ逐次版 LAPACK の DC2 を呼んで固有分解を行う．入力行列 T を全プロセスが保持しているため，このステップ中に通信は必要ない．T_j

の固有分解がすべて完了した時点で通信を行い D，U，C を全体に放送する．

`eval` は，対角行列と階数 1 の摂動の和の固有値問題を $k-1$ 回解くことで計算できる[3]．DC2 の内部にも現れる対角行列と階数 1 の摂動の和は，各固有値・固有ベクトルを独立に求められるため，各プロセスが r/p 個の固有値を計算することで並列化する．対角行列と階数 1 の摂動の和の固有値問題を解くごとに他の摂動が変化するため，その変化する部分を各プロセスが計算し全体で共有する．$D_1+U_1CU_1^T$ の全固有値を求めたら，固有値を昇順に整列して全体に放送する．これは，固有ベクトル計算後の再直交化の効率を上げるための準備である．

$D_1+U_1CU_1^T$ の固有ベクトルは，D_1，U_1，C から構成される α 次実対称行列 $\tilde{F}(\lambda_j)$（$k-1\leq\alpha<n+k$）の核を用いて求められる[3]．$\tilde{F}(\lambda_j)$ のサイズは数学的には $k-1$ 次でよいが，求める固有ベクトルの精度を向上させるために大きくする場合がある．拡大後のサイズは λ_j により異なるが，ほとんどの場合は k の数倍程度となる．本実装では固有値を昇順に整列したことをふまえ，プロセス p_j は $jr/p+1$ 番目から $(j+1)r/p$ 番目までの固有値に対応する r/p 個の固有ベクトルを計算する．計算をこのように分担することで，$D_1+U_1CU_1^T$ の固有ベクトルは列ブロック分割形式で分散して各プロセスに格納される．固有ベクトル計算時には正規化前の長さと $\tilde{F}(\lambda_j)$ の最小特異値を保存しておき[3]，全固有ベクトル計算後にそれらを全プロセスで共有する．これは固有ベクトルの直交性検査（後述の簡易検査）に用いられる．

3.2　再直交化の並列化

再直交化は計算した固有ベクトル間の直交性の検査，非直交グループの構成，直交化計算の 3 つのステップからなる．以下，これらのステップの分散並列化を行う．3.2.1 項で固有ベクトル間の直交性検査，3.2.2 項でグループ化，3.2.3 項でグループの直交化計算について述べる．

なお，これらの手法は共有並列と同等の性能を持つことを 4 章の数値実験で示す．

3.2.1　直交性の検査

一般的に，直交性を損なう固有ベクトルの対は，対応する固有値が近接している場合が多い．そのため，固有値を事前にソートし，列ブロック分割を用いたことで，非直交な固有ベクトルはプロセス番号が近い（または同じ）プロセスに局在化していることが期待できる．その結果，後述の［詳細検査］におけるデータ通信量を抑制することが可能である．

固有ベクトルどうしの直交性検査の結果は 2 値行列 B に保存する．行列 B は $\tilde{Q}'$ と同様に，列ブロック分割形式で各プロセスにデータを分散させ，プロセス p_j が持つ部分を B_j と表記する（$B_j\in\{0,1\}^{r\times r/p}$）．行列 B の (i,j) 成分 b_{ij} は，ステップ (6)（`evec`）で求めた固有ベクトル $\tilde{\boldsymbol{q}}'_i$，$\tilde{\boldsymbol{q}}'_j$ と微小パラメータ ϵ を元に次のように計算する（具体的手順は後述）．

$$b_{ij}=\begin{cases}0 & (|(\tilde{\boldsymbol{q}}'_i,\tilde{\boldsymbol{q}}'_j)|\leq\epsilon)\\ 1 & (\text{otherwise})\end{cases}$$

固有ベクトルの組合せで要素の値が決まるため，B は対称行列となる．また非直交な組合せは隣り合った固有ベクトルで多く起こるため，B は対角成分近辺に 1 が多い行列となる．

行列 B の成分を計算するための，固有ベクトルどうしの直交性判定は 2 段階で行う．まず，簡易検査を行い，直交性が不十分と判定された固有ベクトルの対にのみ内積を用いた詳細検査を行う．内積の前に簡易検査を行うことで，演算量と通信量を削減している．各検査は以下のとおり．

［簡易検査］固有ベクトルを求める際に得たそのベクトルの長さと最小特異値を利用して，固有ベクトルの内積を $O(1)$ の計算で過大評価する．必要なデータは全プロセスが保持しているため，簡易検査中に通信は不要である．簡易検査の詳細は文献 3) に譲る．

［詳細検査］簡易検査で直交性が不十分とされた場合，内積により陽に直交性を検査する．内積計算を別プロセスの保持するベクトルと行う際は通信が必要である．

以下，まず 6 プロセス用いる場合（$p=6$）の例を述べたあとで，アルゴリズムとして直交性検査のステップを示す．**図 2** はこの例での計算ステップの進行を示す．以降，特に明記しない限り，ベクトルは $D_1+U_1CU_1^T$ の固有ベクトルを意味する．また $D+U_1CU_1^T$ の固有ベクトルを並べた $\tilde{Q}'$ のうちプロセス p_j が計算した部分を $\tilde{Q}'_j=(\tilde{\boldsymbol{q}}'_{jr/p+1},\ldots,\tilde{\boldsymbol{q}}'_{(j+1)r/p})$ と表記する．まず，プロセス p_0 の持つベクトル $\tilde{Q}'_0$ と p_2 の持つベクトル $\tilde{Q}'_2$ との直交性を次のように調べる．

(1)　p_2 が簡易検査によって，$\tilde{Q}'_2$ の各ベクトルと非直交となる $\tilde{Q}'_0$ のベクトルの対の候補を求める．候補がなければ，B の (1, 3) ブロックに 0 を代入し，直交性の検査を終了する．候補がある場合，以下の (2)〜(4) のステップに進む．

(2)　(1) で求めた非直交なベクトルの対の候補の中で，$\tilde{Q}'_0$ の中の最初と最後のベクトルの番号 $f_{0,2}$，$\ell_{0,2}$ を p_2 から p_0 へ送信する．

(3)　p_0 は，$\tilde{Q}'_0$ のうち $f_{0,2}$ 番目から $\ell_{0,2}$ 番目までのベクトルを p_2 に送信する．

(4)　p_2 は，受信したベクトルのうち非直交の可能性のあるベクトルの対に対して内積計算を行い，2 値行列 B の $(1,3)$ ブロックの値を得る．

これで p_2 は，p_0 の持つベクトルとの直交性検査を終え，2 値行列 B の $(1,3)$ ブロックを

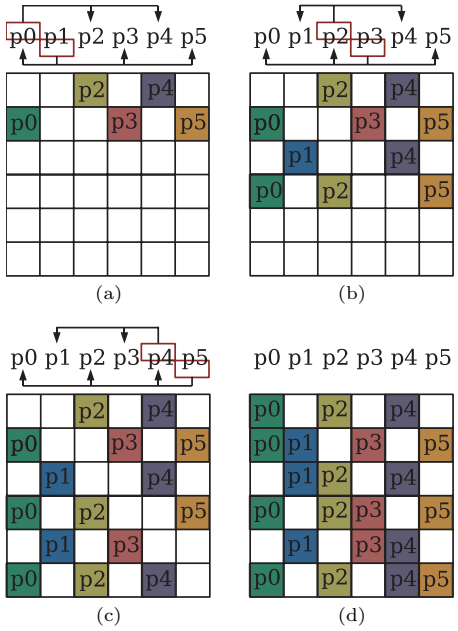

図 2　直交性検査（行列 B の構成）の進行
Fig. 2　Flow of orthogonality check (how matrix B is constructed).

計算できた．同じ処理を p_0 と p_4 の間でも行うことで，図 2 (a) の最上行のうちプロセス名の記載されているブロックの計算ができたことになる．また p_1 と p_0，p_1 と p_3，p_1 と p_5 の間でも同じように行う（いずれも p_1 がベクトルデータを送信する）と，図 2 (a) のように，上から 2 行目を計算できる．図 2 の四角の中の文字は，その部分の直交性検査を行うプロセスを示している．矢印はベクトルデータの流れを示しており，図上部の枠で囲まれたプロセスから送信することを表す．ここで，行列 B の上から 2 ブロック分の部分を構成するステップは，p_0 と p_1 の間を除いて通信が競合せず，各プロセスの行う処理は独立なことから，並列に実行できると考えられる．

2 値対称行列 B の構成は，図 2 (a) から (b)，(c) へと進行する．先ほど述べたように，このステップは行ごとに並列に実行可能である．

最後に各プロセス p_j が持つベクトル $\tilde{Q}'_j$ どうしの直交性を調べる（図 2 (d)）．固有値の並びと固有ベクトル計算の分配方法から，ある 1 つのプロセスが持つベクトルどうしの直交性検査には内積計算が必要になることが多いと考えられる．この部分を最後に行うのは，図 2 のステップ (a) から (c) のように，通信が必要なときにその部分を計算すると遅延が発生する恐れがあり，これを避けるためである．

一般的には，以下のアルゴリズムに従い直交性検査を行う．ここで，整数 s ($s = 0, \ldots, p-1$) を自分のプロセス番号とする．

1: **for** $i = 0$ to $p - 1$ **do**
2:　**if** $s - i$ が正の偶数 or 負の奇数 **then**
3:　　$\tilde{Q}'_i$ と $\tilde{Q}'_s$ に対して簡易検査を行う
4:　　簡易検査の結果に基づき，$\tilde{Q}'_s$ と非直交な $\tilde{Q}'_i$ の最初と最後のベクトルの番号を $f_{i,s}$, $\ell_{i,s}$ とする
5:　　直交性の悪い組合せがないならば $f_{i,s} = \ell_{i,s} = -1$ とする
6:　　p_i に $f_{i,s}$，$\ell_{i,s}$ を送信する
7:　　**if** $\tilde{Q}'_i$ と $\tilde{Q}'_s$ に直交性の悪い組合せがある **then**
8:　　　p_i から $\tilde{Q}'_i$ の $f_{i,s}$ 列目以降 $\ell_{i,s}$ 列目までを受信するまで待つ
9:　　　受信後，直交性の悪い組合せについてのみ内積計算を行い，直交性を判定する
10:　　**end if**
11:　**else if** $s = i$ **then**
12:　　**for** $j = 0$ to $p - 1$ **do**
13:　　　**if** $j - s$ が正の偶数 or 負の奇数 **then**
14:　　　　p_j から $f_{i,j}$，$\ell_{i,j}$ を受信するまで待つ
15:　　　　受信後，$f_{i,j} \neq -1$ ならば，p_j へ $\tilde{Q}'_s$ の $f_{i,j}$ 列目以降 $\ell_{i,j}$ 列目までを送信する
16:　　　**end if**
17:　　**end for**
18:　**end if**
19: **end for**
20: $\tilde{Q}'_s$ のベクトルどうしの直交性を検査する

3.2.2　グループ化

前項で作成した 2 値対称行列 B に基づき，非直交なベクトルどうしをまとめたグループを

構成する．グループ情報は互いに素な集合（Disjoint sets）を扱うデータ構造 Union-Find [8] を用いて管理する．この構造は，ある元の属する集合を求める操作と集合の合併が高速に行えるという特徴を持つ．

まず各プロセスは，2 値対称行列 B のうち自分が計算した部分のみを参照してグループを作る．つまりプロセス p_j が B の部分行列 B_j を参照する．グループ情報としては，各固有ベクトルがどのグループに属するかをグループごとに異なる整数値で表せばよいため，n 次整数ベクトル 1 つの領域があれば十分である．次に，p_{p-1} は作成したグループ情報を p_{p-2} に送信する．p_{p-2} は，受信した p_{p-1} のグループ情報を自身の持つグループ情報と合併させ，それを p_{p-3} に送信する．これをプロセス番号の降順に繰り返すことで，最終的に全プロセスからのグループ情報が p_0 に集まる．p_0 は最終結果を全プロセスに送信する．

本ステップでプロセスが通信する情報は毎回 $O(n)$ であり，これを $p-1$ 回繰り返して最後に全体に送信するため，$O(pn)$ の通信が必要となる．

3.2.3　直交化計算

グループごとにレイリー・リッツ法を用いてベクトルを直交化する．直交化は，グループ内のベクトル数があらかじめ設定した閾値以下であれば，単一のプロセスが行う．そうでなければ，すべてのプロセスを用いて行う．単一のプロセスが直交化を行う場合，担当するグループをプロセス p_0 から順に割り振る．あるグループを p_j が担当するとき，その次のグループは p_{j+1}（$j+1=p$ ならば p_0）が担当する．

グループの直交化の手順を述べる．一般にグループ内のベクトルは異なるプロセスが保持しているため，単一プロセスを用いる場合は，そこにグループ内のすべてのベクトルを集めて該当プロセスが直交化計算を行う．全プロセスを用いる場合は，適切なプロセスにベクトルを送信してデータ分散を行い，分散並列ルーチンを用いて直交化計算を行う．最後に，直交化前のベクトルを保持していたプロセスに直交化後のベクトルを送信する．なお直交化計算には，単一プロセスを用いる場合は逐次版 LAPACK の `DSYEV` を，全プロセスの場合は ScaLAPACK の `PDSYEV` を使用する．

3.3　行列積の並列化

本節では，`prod` で計算する行列積 $Q \equiv PP_d(\tilde{Q} \oplus I_{n-r})$ の分散並列化について述べる．

まず記法を定義する．A_1 を $m_1 \times n_1$ 行列，A_2 を $m_2 \times n_2$ 行列としたとき，$(m_1+m_2)\times(n_1+n_2)$ 行列 $A_1 \overrightarrow{\oplus} A_2$，$A_1 \overleftarrow{\oplus} A_2$ を

$$A_1 \overrightarrow{\oplus} A_2 \equiv \begin{pmatrix} A_1 & \\ & A_2 \end{pmatrix}, A_1 \overleftarrow{\oplus} A_2 \equiv \begin{pmatrix} & A_1 \\ A_2 & \end{pmatrix}$$

とする．本節ではこれを適宜利用する．

$PP_d(\tilde{Q} \oplus I_{n-r})$ を計算するにあたり，まず PP_d を計算し，その後 $\tilde{Q} \oplus I_{n-r}$ との積を計算するという手順をとる．`prod` の直前においては，$P \equiv \bigoplus_{j=1}^{k} Q_j$，$\tilde{Q} \equiv (\tilde{Q}_0, \ldots, \tilde{Q}_{p-1})$ としたとき，プロセス p_j は Q_{pl+j+1}（$l = 0, \ldots, k/p-1$）と $\tilde{Q}_j$ を保持していることに注意する．

n 次置換行列 $P_d \equiv (P_d^{(1)}\ \ P_d^{(2)})$ は以下の性質を持つ．$P_d^{(1)}$ は $n \times r$ 行列で，$i<j$ に対して $p(i)<p(j)$ を満たす数列を用いて，$P_d^{(1)} \equiv (\boldsymbol{e}_{p(1)}, \ldots, \boldsymbol{e}_{p(r)})$ と表せる．$P_d^{(2)}$ は $n \times (n-r)$ 行列で，$i<j$ に対して $q(i)>q(j)$ である数列を用いて，$P_d^{(2)} \equiv (\boldsymbol{e}_{q(1)}, \ldots, \boldsymbol{e}_{q(n-r)})$ と表現できる．

よって，

$$PP_d \equiv (Z^{(1)}\ \ Z^{(2)})$$
$$(Z^{(1)} \in \boldsymbol{R}^{n\times r},\ Z^{(2)} \in \boldsymbol{R}^{n\times(n-r)})$$

としたとき，

$$Z^{(1)} = Z_1^{(1)} \overrightarrow{\oplus} \cdots \overrightarrow{\oplus} Z_k^{(1)},$$
$$Z^{(2)} = Z_1^{(2)} \overleftarrow{\oplus} \cdots \overleftarrow{\oplus} Z_k^{(2)}$$

となる．ただし

$$Z_j^{(1)} \in \boldsymbol{R}^{m\times r_j},\ Z_j^{(2)} \in \boldsymbol{R}^{m\times(m-r_j)},\ \sum_{j=1}^{k} r_j = r$$

で，適当な m 次置換行列 Π_j を用いて，$Q_j\Pi_j = (Z_j^{(1)}\ \ Z_j^{(2)})$ を満たす．この操作は P に対する列の入れ替えのみであるため，プロセス p_j に

$$Z_{pl+j+1}^{(1)},\ Z_{pl+j+1}^{(2)} (l = 0, \ldots, k/p-1)$$

を保持させ，通信は行わない．

続いて，$Q = PP_d(\tilde{Q} \oplus I_{n-r}) \equiv (W\ \ Z^{(2)})$ としたときの $W \in \boldsymbol{R}^{n\times r}$ を計算する．

$$W \equiv (W_{(1)}^T, \ldots, W_{(k)}^T)^T,\ \tilde{Q} \equiv (\tilde{Q}_{(1)}^T, \ldots, \tilde{Q}_{(k)}^T)^T$$
$$(\ W_{(j)} \in \boldsymbol{R}^{m \times r},\ \tilde{Q}_{(j)} \in \boldsymbol{R}^{r_j \times r})$$

とすると，$W = Z^{(1)}\tilde{Q}$ の計算は行列積 $W_{(j)} = Z_j^{(1)}\tilde{Q}_{(j)}$（$j = 1, \ldots, k$）に帰着される．$W$ を列ブロック分割形式で格納するため，$\tilde{Q}_{(j)}$ と同様に $W_{(j)}$ は全プロセスに分散しており，本実装ではこの行列積を PBLAS を用いて計算する．最終的にプロセス p_j は T の固有ベクトルとして，W_j と $\hat{Z}_{pl+j+1}^{(2)}$（$l = 0, \ldots, k/p - 1$）を保持する．ただし，

$$\hat{Z}_i^{(2)} \equiv E_i Z_i^{(2)},\ I_n = (E_1, \ldots, E_k),\ (E_j \in \boldsymbol{R}^{n \times m})$$

とする．$Z_i^{(2)}$ から $\hat{Z}_i^{(2)}$ への変換はメモリコピーのみで達成できる．

自分のプロセス番号を s（$s = 0, \ldots, p - 1$）としたとき，アルゴリズムは以下のようになる．通信は 6 行目でのみ発生する．

1: **for** $j = 1$ to k **do**
2:　**if** $j - 1$ を p で割った余りが s **then**
3:　　置換行列 Π_j によって Q_j の列を入れ替え，$(Z_j^{(1)}\ \ Z_j^{(2)}) \equiv Q_j\Pi_j$ とする
4:　　$\hat{Z}_j^{(2)} \equiv E_j Z_j^{(2)}$ を作成
5:　**end if**
6:　PBLAS で $W_{(j)} = Z_j^{(1)}\tilde{Q}_{(j)}$ を計算
7: **end for**

4. 数値実験

分散メモリ型並列計算機 HITACHI SR11000 上で数値実験を行い，本稿で提案した DCK の分散並列化の効率を確かめる．また，実対称三重対角行列に対する他の固有値問題解法と速度，計算精度を比較する．

具体的には，分散並列化した DCK を用いて，n 次実対称三重対角行列の全固有対 $(\lambda_j, \boldsymbol{q}_j)$ を計算する時間を計測する．同様に二分法・逆反復法（BII），二分割の分割統治法（DC2）の計算時間も計測し，DCK と比較する．どちらの解法も ScaLAPACK のルーチン（それぞれ `pdstebz`/`pdstein`，`pdstedc`）を用いる．計算時間の計測には `xclock` ルーチンを利用した．

以下，4.1 節で対象行列，4.2 節で実験条件，4.3 節で実験結果を述べ，4.4 節で考察を行う．

表 1　HITACHI SR11000 の諸元
Table 1　Specifications of HITACHI SR11000.

ノード数	128（16CPU/ノード）
1CPU あたりの理論性能	9.2 GFLOPS
1 ノードあたりの理論性能	147.2 GFLOPS
1 ノードあたりの主記憶	128 GB

4.1 対象行列

数値実験には，平均 0，分散 1 の正規乱数を要素とする実対称行列（を三重対角化した行列）を用いた．以降この行列を行列 QC と呼ぶ．この行列は，量子カオス系のモデルとして用いられ，deflation 発生率 $\delta \equiv 1 - r/n$ の低い行列の典型となっている．再直交化の負荷も経験的には標準的な大きさとなっている．

また 2 つの同じ行列 QC を対角に並べ，継ぎ目にあたる副対角成分の値を 1 とした行列を行列 SQ と呼ぶ．この行列は，固有値どうしの差がマシンイプシロン程度になる固有値の対が存在するため，特に BII にとっては精度が悪くなる悪条件問題である．

4.2 実験条件

数値計算ライブラリとして LAPACK（BLAS），その分散並列版である ScaLAPACK[9]（PBLAS）を用いる．ScaLAPACK と PBLAS は，線形代数ライブラリ用メッセージ交換（通信）ライブラリ BLACS[10] を経由して MPI 通信を行うが，DCK で通信を行う際も同様に BLACS を用いる．MPI による通信には，MPI-2 通信ライブラリを利用する．いずれのライブラリも並列計算機に備え付けのものを使用した．

比較対象として共有並列の DCK を用いる実験（`group`/`reortho` の性能確認）があるが，そこではベンダによりスレッド並列化が行われたライブラリを用いている．共有並列の DCK は OpenMP[7] を用いて並列化を行ったものである[4]．

プログラムは C 言語で実装し，HITACHI 最適化 C コンパイラ（バージョンは `Hitachi C Compiler Release 01-03-/C`）でコンパイルした．MPI を用いて通信するため，コンパイルは `mpicc -64 -O4 +Op -noparallel` のようにした．

SR11000 は 1 ノードあたり 16CPU を搭載したノードを複数台備えた並列計算機である．1 ノードを共有または分散メモリ型並列計算機として扱うことができる．実験では 8 ノードまで使用した．システムの諸元を**表 1** に示す．

4.2.1 精度評価

解の相対残差 ϵ_r，直交誤差 ϵ_o は次の式で評価する．ここで δ_{ij} はクロネッカーのデルタ

表 2　共有と分散の比較（行列 QC，40000 次，64 分割，単位：秒）
Table 2　Stepwise times [sec] by shared and distributed parallel DCK (Matrix QC, $n = 40000$, $k = 64$).

	all	Tj	eval	evec	group	reortho	prod
共有	72.8	10.0	16.3	12.8	6.5	4.3	22.2
分散	61.6	0.6	15.7	12.9	5.3	1.8	22.8

表 3　行列サイズ n ごとの計算時間（行列 QC，単位：秒）
Table 3　Dependence of execution time [sec] on matrix size n (Matrix QC).

n	BII	DC2	DCK128	DCK256
20000	4.0	28.0	3.4	6.0
40000	15.7	221.6	11.8	18.8
60000	35.1	770.3	29.6	44.0

である：

$$\epsilon_r = \max_{1\le j\le n} \frac{\|T\boldsymbol{q}_j - \lambda_j \boldsymbol{q}_j\|_2}{\|T\|_2},\ \epsilon_o = \max_{1\le i\le j\le n} |\boldsymbol{q}_i^T \boldsymbol{q}_j - \delta_{ij}|.$$

4.2.2　従来法に対する設定

逆反復法は，大規模な問題に対して十分な精度を得るためには（固有ベクトルの）再直交化が不可欠である．しかし，`pdstein` は異なるプロセスにあるベクトルどうしの再直交化は行わず，同じプロセスにあるベクトルどうしの再直交化も行わない設定で実験を行ったため，高速ではあるが精度は低い．

実験時は，プロセス数と同数の CPU を使用する．

4.3　実 験 結 果

本稿で行った実験の結果を本節に掲載し，考察は次節で述べる．

分散並列と共有並列の DCK を用いて，40000 次の行列 QC を 64 分割で解いた際の所要時間の内訳を**表 2** にあげる．`all` は全体時間を表し，それ以外は図 1 の名前付けに従う．分散・共有ともに 1 ノード内（16CPU，分散は $p = 16$）で計算を行った．

以降の実験はすべて分散並列のみである．128 プロセスを用いて行列サイズ n を変化させたときの計算時間を BII，DC2 と合わせて**表 3** に示す．DCK は 128 分割と 256 分割での結果をあげる．**図 3** は表 3 をグラフで示したもので，横軸が行列サイズ，縦軸が計算時間であり，両対数としている．

また，行列サイズを 40000 次に固定し，プロセス数 p を変えた際の計算時間を**表 4** に示す．表 4 をグラフ化したものが**図 4** である．図 4 の横軸はプロセス数，縦軸は計算時間であり，両対数としている．

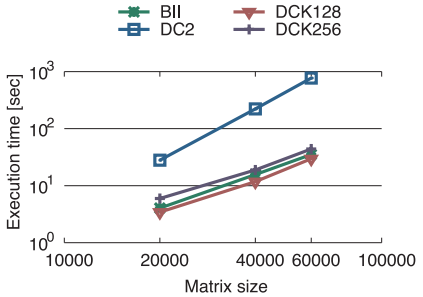

図 3　行列サイズ n ごとの計算時間（行列 QC）
Fig. 3　Dependence of execution time [sec] on matrix size n (Matrix QC).

表 4　プロセス数 p ごとの計算時間（行列 QC，単位：秒）
Table 4　Dependence of execution time [sec] on number of processes p (Matrix QC).

p	BII	DC2	DCK128	DCK256
16	117.3	1609.4	–	–
32	59.9	817.7	35.5	60.1
64	30.7	369.1	19.9	34.1
128	15.7	221.6	11.8	18.8

次に，128 プロセスを用いて行列 QC の固有分解を計算したときの DCK（128 分割）と BII，DC2 の計算精度を示す．**表 5** は相対残差 ϵ_R，**表 6** は直交誤差 ϵ_O である．

最後に，行列 SQ の固有分解を 128 プロセス用いて計算した際の解の直交誤差を**表 7** に，また計算時間を**表 8** に示す．DCK128（nochk）は固有ベクトルの直交性を調べる際に簡易検査のみ行うが，DCK128 は詳細検査も行っている．

4.4　考　　察

4.4.1 項で再直交化の分散並列化について検証し，4.4.2 項で計算時間，4.4.3 項で計算精度の従来法との比較を述べる．

4.4.1　再直交化の分散並列化の検証

まず，本稿で提案した再直交化の分散並列化が十分な効果をあげていることを述べる．一般に，分散並列化により共有並列並みの並列効果を得ることは容易ではない．このため，

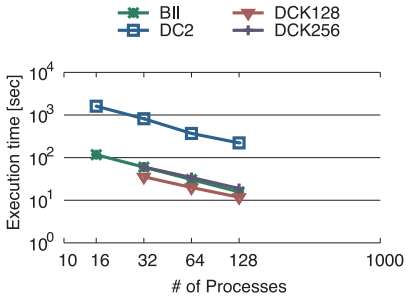

図 4　プロセス数 p ごとの計算時間（行列 QC）
Fig. 4　Dependence of execution time [sec] on number of processes p (Matrix QC).

表 5　相対残差 ϵ_r（行列 QC）
Table 5　Residual error ϵ_r (Matrix QC).

n	BII	DC2	DCK128
20000	1.7e-15	9.7e-15	8.6e-15
40000	2.7e-15	2.5e-14	7.4e-15
60000	3.6e-15	3.2e-14	9.9e-15

表 6　直交誤差 ϵ_o（行列 QC）
Table 6　Orthogonal error ϵ_o (Matrix QC).

n	BII	DC2	DCK128
20000	2.3e-12	1.5e-13	5.1e-13
40000	3.0e-12	3.5e-13	4.8e-13
60000	7.3e-12	5.2e-13	1.3e-12

`group` と `reortho` が分散並列でも共有並列と同等の性能を得られているならば，本稿で提案した分散並列化は有効であるといえる．

表 2 によれば，分散並列の `group` と `reortho` は，いずれも共有並列より高速に計算できており，本稿の分散並列化が有効であることが分かる．他のステップの計算時間も共有並列と同等である．ただし，共有並列の `Tj` は 10.0 秒と分散並列の 0.6 秒と比較して大きく遅延しているが，これは共有並列で使用したスレッド並列ライブラリの影響である．

4.4.2　従来法との速度比較

128 プロセスを用いて行列サイズ n を変化させたときの計算時間を BII，DC2 と比較する．

表 7　直交誤差 ϵ_o（行列 SQ）
Table 7　Orthogonal error ϵ_o (Matrix SQ).

n	BII	DC2	DCK128	DCK128（nochk）
20000	1.1e-7	1.5e-13	4.1e-13	1.3e-13
40000	7.3e-1	2.7e-13	3.0e-13	2.6e-13
60000	1.0e-0	4.9e-13	4.1e-13	–

表 8　計算時間（行列 SQ，単位：秒）
Table 8　Computation time [sec] (Matrix SQ).

n	BII	DC2	DCK128	DCK128（nochk）
20000	3.9	17.7	2.4	248.8
40000	14.7	115.1	10.1	2303.2
60000	32.9	400.6	24.8	–

図 3 によると，DCK は DC2 より傾きが小さいが，これは DCK が 2 より大きな分割数をとることで演算量を削減できるためであり，より大次元の問題に対しても DCK の方が高速であると考えられる．また 表 3 によると，60000 次では，DC2 は 770.3 秒かかるが 128 分割の DCK は 29.6 秒と 1/25 以下の時間で計算できており，35.1 秒要する BII と比較しても DCK が 15%程度高速という結果が得られた．

次に，行列サイズを 40000 次に固定し，プロセス数 p を変えた際の計算時間を BII，DC2 と比較する．図 4 によると，DCK を含むいずれの解法も同程度の傾きであり，DCK は他の解法と同様の並列効率を持つといえる．表 4 に記載したプロセス数においては DCK が最も高速であり，プロセス数をさらに増やしても DCK が最も高速であると予想できる．

4.4.3　従来法との精度比較

表 5 によると，いずれの解法も相対残差 ϵ_r は十分小さい．一方，表 6 によると，直交誤差 ϵ_o は DCK と DC2 が同程度で，BII はそれらより 1 桁悪い精度となっている．

DCK には精度に関するパラメータ ϵ があり（3.2.1 項），この値が全体の性能に影響を与える．本稿では一貫してこの値を $\epsilon = 5 \times 10^{-14}$ と設定し DC2 と同等の精度で計算しており，この意味で表 6 の結果は当然といえる．本稿の重要な主張は，DC2 と同等の精度を保ちつつ DC2 の実行性能を大幅に上回る DCK の分散並列実装を実現した点にある．

4.4.4　悪条件問題への対応と詳細検査の意義

表 7 によると，DCK の直交誤差は 20000 次で 10^{-13} 程度であり，DC2 と同等の十分な精度が得られている．他方 BII は約 10^{-7} と精度は悪い．これは行列 SQ が近接固有値を持

つ悪条件問題であることによる．さらに表 8 によると，DCK は BII より 2〜3 割ほど高速である．より大きな次数ではこれらの傾向が顕著であり，DCK は悪条件問題に対して BII より有効であるといえる．

また表 8 によると，簡易検査のみ行う DCK128 (nochk) は，詳細検査も行う DCK128 と比べて計算時間が 100 倍以上である．これは，DCK128 (nochk) は簡易検査のみ行うためグループサイズが $n/3$〜$n/2$ 程度まで大きくなってしまい，直交化計算に大きな時間を要し，計算時間が増大している．一方 DCK128 は詳細検査により不要な直交化計算を省くことで高速に計算している．なお表 7 によると，両者の精度は約 10^{-13} と同等である．以上より，詳細検査により直交化計算の高速化が実現しているといえる．

5. ま と め

実対称固有値問題に対する多分割の分割統治法（DCK）の分散並列アルゴリズムを提案し，実装，評価した．本稿では，一般には分散並列手法が自明ではない再直交化に対して，DCK の特徴を利用した効率的な手法を提案した．

数値実験により，本稿で提案した DCK の分散並列手法が有効であることが確かめられた．分散並列化された再直交化の性能は共有並列の DCK と同等以上であり，十分な並列化効率が出ている．ScaLAPACK に実装された二分割の分割統治法（DC2）と比較すると，128 プロセス用いた場合，DC2 と同等の精度で 8〜26 倍高速に計算できた．また二分法・逆反復法（BII）と比較すると，BII より 1 桁程度良い精度で 15%以上高速に計算できており，近接固有値を持つ行列に対しては，BII は十分な精度で解けないが DCK は DC2 と同程度の精度で高速に計算できている．以上より，DCK が実対称固有値問題に対する既存の標準的な分散並列ルーチンの性能を上回ることが示された．

今後の課題としては，通信トポロジの二分木構造化，別の手法による行列積計算などがあげられる．

再直交化でのグループ情報の合併などにおいて，プロセスを一列に並べた形で通信を行っている箇所がいくつかある．この通信を二分木構造で行うことで，通信時間をプロセス数に関する線形時間から対数時間に削減できる．

また，3.3 節で行列積の分散並列手法を述べたが，簡易的なベンチマークにより，$Z_j^{(1)}$ のサイズによっては PBLAS を用いない別の手法で行列積を計算する方が良い場合があることが分かっている．そこで，分割数 k や deflation により決まる $Z_j^{(1)}$ のサイズによって行列積の計算方法を変えることで，高速化できると考えられる．詳細は別稿にて報告する．

謝辞　本研究の一部は，科学研究費補助金（課題番号：19560058）のもとで行われた．

田村，坪谷は東京大学情報基盤センターのスーパーコンピュータ若手利用者推薦制度のもとで研究を行った．研究へのご支援に感謝いたします．

参 考 文 献

1) Cuppen, J.: A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem, *Numerische Mathematik*, Vol.36, pp.177–195 (1980).
2) Dhillon, I.S. and Parlett, B.N.: Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices, *Linear Algebra Appl.*, Vol.387, pp.1–28 (2004).
3) 桑島　豊，重原孝臣：実対称三重対角固有値問題に対する多分割の分割統治法の改良，日本応用数理学会論文誌，Vol.16, No.4, pp.453–480 (2006).
4) 田村純一，坪谷　怜，桑島　豊，重原孝臣：実対称固有値問題に対する多分割の分割統治法の共有メモリ型並列計算機における有効性，HPCS2009 論文集，pp.97–104 (2009).
5) 石川祐輔，田村純一，桑島　豊，重原孝臣：実対称固有値問題に対する多分割の分割統治法における準最適分割数の自動決定について，第 38 回数値解析シンポジウム講演予稿集，pp.17–20 (2009).
6) MPI Forum: *MPI-2: Extensions to the Message-Passing Interface* (2003).
7) OpenMP ARB: *OpenMP C and C++ Application Program Interface* (2002).
8) Cormen, T.，浅野哲夫ほか：アルゴリズムイントロダクション第 2 巻，近代科学社 (1995).
9) Blackford, L.S., et al.: *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA (1997).
10) Dongarra, J.J. and Whaley, R.C.: A User's Guide to the BLACS v1.1, Technical report (1997).

(平成 21 年 10 月 2 日受付)
(平成 22 年　1 月 7 日採録)

田村　純一

昭和 60 年生．平成 20 年埼玉大学工学部情報システム工学科卒業．平成 22 年同大学大学院理工学研究科数理電子情報系専攻修了．同年沖ソフトウェア株式会社入社．専門分野はハイパフォーマンスコンピューティング，並列数値計算．

坪谷　　怜

昭和 59 年生．平成 19 年埼玉大学工学部情報システム工学科卒業．平成 21 年同大学大学院理工学研究科数理電子情報系専攻修了．同年コンピュートロン株式会社入社．専門分野はハイパフォーマンスコンピューティング，並列数値計算．

桑島　　豊

昭和 55 年生．平成 14 年埼玉大学工学部情報システム工学科卒業．平成 19 年同大学大学院情報数理科学専攻修了．同年より埼玉大学大学院理工学研究科助教．博士（工学）．専門分野は数値線形代数，ハイパフォーマンスコンピューティング．日本応用数理学会会員．

重原　孝臣（正会員）

昭和 35 年生．昭和 58 年東京大学理学部物理学科卒業．昭和 63 年同大学大学院理学系研究科物理学専攻修了．同年より東京大学大型計算機センター研究開発部助手，平成 9 年より埼玉大学工学部情報システム工学科講師を経て，現在，同大学大学院理工学研究科教授．理学博士．専門分野は数値線形代数，ハイパフォーマンスコンピューティング，数理物理．日本応用数理学会，電子情報通信学会，日本物理学会各会員．