実対称行列に対する多分割の分割統治法の分散メモリ型並列計算機 HA8000 への移行

田村 純一
株式会社 OKI ソフトウェア

坪谷 怜
コンピュートロン株式会社

桑島 豊　　重原 孝臣
埼玉大学大学院 理工学研究科

1 はじめに

近年様々な分野で，数値シミュレーションの大規模化が進んでいる．計算化学においては，タンパク質の分子軌道計算を行う際に数万次元の行列の固有値問題を扱うことがある．他にも構造解析や波動方程式など様々な問題を扱う際には，大規模な実対称行列の固有値問題が現れることが多く，その高速な解法に対する需要は大きい．

実対称固有値問題を解く際は，対象行列の固有対を直接計算するのではなく，三重対角化を行ってから本計算を行うのが一般的な手順である．実対称三重対角行列の固有対を求める数値解法には，以下のものが知られている．古典的な二分法・逆反復法 (BII) や QR 法，最近の解法として二分割の分割統治法 (DC2)[1] や MRRR 法 [2] などが挙げられる．近年では，DC2 を拡張した多分割の分割統治法 (DCK) が提案されている [3, 4]．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．そのため，DCK は DC2 の自然な拡張となっている．

DCK は逐次計算機や共有メモリ型並列計算機に実装されている．共有メモリ型並列計算機においても，DCK の持つ並列性を活用し，ほかの解法と同等以上の性能となることが示されている [5]．DCK でより大規模な問題を扱うにあたり，分散メモリ型並列計算機への実装が不可欠である．

本稿では，若手利用者推薦制度の支援のもと行われた，分散並列化した DCK の T2K スパコン (HITACH HA8000) への移行について報告する．

2 節でまず DCK の逐次アルゴリズムについて述べてから，3 節で分散並列版 DCK について述べ，4 節で前回の報告からの差異を補足的に述べる．5 節で数値実験と考察を行い，6 節でまとめを行う．

2 DCK の概要

DCK は，主対角要素 a_j $(1 \le j \le n)$ と副対角要素 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 を入力とし，n 次直交行列 Q と実対角行列 Λ を出力するアルゴリズムである．ただし Q と Λ は $T = Q\Lambda Q^T$ を満たす．以下に DCK の概要を示す．説明を簡単化するため，以降では $n = m \times k$ (m は正整数) と仮定する．また n 次単位行列 I_n の第 ℓ 列ベクトルを $\boldsymbol{e}_\ell^{(n)}$ と表記する．

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，$V \equiv (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$，$\boldsymbol{v}_j \equiv \boldsymbol{e}_{jm} + \mathrm{sign}(b_{jm})\boldsymbol{e}_{jm+1}$，$C \equiv \mathrm{diag}(|b_m|, \ldots, |b_{(k-1)m}|)$
2. 固有分解 $T_j = Q_j \Lambda_j Q_j^T$ $(j = 1, \ldots, k)$ を計算
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 を用いて，$D + UCU^T$ を r 次実対称行列 $\tilde{D} \equiv D_1 + U_1 C U_1^T$ と $n-r$ 次実対角行列 D_2 との直和に変形し $P_d(D + UCU^T)P_d^T = \tilde{D} \oplus D_2$ を得る．この操作を deflation と呼ぶ
5. $\tilde{D}$ 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．ここで T の全固有値 $\Lambda \equiv \tilde{\Lambda} \oplus D_2$ が求まる
6. $\tilde{D}$ 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

7. $\tilde{D}$ の固有ベクトルのうち，再直交化が必要なベクトル同士をグループにまとめる
8. 非直交な固有ベクトル同士を再直交化する．$\tilde{\boldsymbol{q}}'_j$ 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
9. $Q \equiv PP_d(\tilde{Q}' \oplus I_{n-r})$ を計算

DC2 と比較して，分割数 k を大きくできることによる利点は，行列積計算 (ステップ 9.) の演算量を削減できることである．これは，k が大きいと直交行列 P の非ゼロ要素数が減少するため，その情報を活用して無駄な処理を行わずに行列積を計算できることによる．ステップ 9. はアルゴリズム全体で唯一 $O(n^3)$ かかるため，このステップの高速化がアルゴリズム全体の速度に貢献する．また，実装時には deflation と呼ばれる操作により，実対角行列と低階数摂動の和 $D+UCU^T$ から自明な固有対を取り除く．これは，D の第 j 対角成分を d_j，U の第 j 行ベクトルを $\underline{\boldsymbol{u}}_j$ とすると，

$$\underline{\boldsymbol{u}}_j = \boldsymbol{0}^T \quad \Rightarrow \quad (D+UCU^T)\boldsymbol{e}_j^{(n)} = d_j \boldsymbol{e}_j^{(n)}$$

が成立し，$D+UCU^T$ の固有対 $(d_j, \boldsymbol{e}_j^{(n)})$ をただちに得られることによる．この操作により $D+UCU^T$ の実質的なサイズが減少し，かつ行列積の演算量が減少する．この操作は，DC2 と 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行えることが知られており，k を大きくすると効果を得られない場合がある．

実対角行列と階数 $k-1$ の摂動の和 $\tilde{D} = D_1 + U_1 C U_1$ の固有値を計算する際は，実対角行列と階数 1 の摂動の和 $D + \boldsymbol{u}\boldsymbol{u}^T$ の固有値問題に帰着させる．帰着後の問題は DC2 にも現れ，この問題を $k-1$ 回繰り返し解くことで $\tilde{D}$ の固有値を得ることが出来る．

$\tilde{D}$ 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 $\tilde{F}(\tilde{\lambda}_j)$ $(k-1 \leq \alpha < n+k)$ の核空間を用いて求められる．$\tilde{F}(\tilde{\lambda}_j)$ のサイズは，数学的には $k-1$ で良いが，求める固有ベクトルの数値的精度を向上させるためにこれより大きくする場合がある．拡大後のサイズは $\tilde{\lambda}_j$ に依存するがほとんどは k の数倍程度で，固有ベクトルを求める演算量が大きく増加することはない．このような手法を用いても直交精度の低い固有ベクトルの組が存在する場合，再直交化により直交精度を確保する．

行列積計算 (ステップ 9.) ではブロック対角行列 P と密行列 $\tilde{Q}$ の積を計算するが，前述の通り P の構造を利用することによって $2n^3/k$ の演算量で計算できる．

なお，単純に k を大きくしてもアルゴリズム全体を高速化できないことに注意する．k が大きいと $\tilde{D}$ の摂動項が増えるため，固有値・固有ベクトル計算の演算量が大きくなってしまう．また，行列の種類にも依存するが，deflation は k が大きいときにはあまり行えない．これらの要因から，DCK を用いて固有値問題を高速に解く際には，行列ごとに適切な分割数を与える必要があるが，この件については他文献が詳しく扱っている [6]．k が十分小さい ($k \simeq 2$) ときに DCK が高速であるならば DC2 を用いれば十分であるため，本研究では分割数 k を 2 より大きくすると高速に解ける状況に関心を置いている．

3 分散並列 DCK

本章では，前章で述べた逐次 DCK の分散並列アルゴリズム (PDCK) について述べる．本研究では MPI によるプロセス間並列を行い，OpenMP によるプロセス内並列，ならびに両者を組み合わせるハイブリッド並列化は行っていない．以降ではプロセス数を p とし，各プロセスを順に $p_0, p_1, \ldots, p_{p-1}$ と呼ぶ．本研究の実装では，分割数 k はプロセス数 p の倍数であること $k = c \times p$ (c は正整数) を仮定している．また入力 T, k は計算開始前に全プロセスで共有している．

DCK の逐次アルゴリズムのうち，小行列の固有分解 (ステップ 2.) と $\tilde{D}$ の固有値計算 (ステップ 5.)，$\tilde{D}$ の固有ベクトル計算 (ステップ 6.) は本質的に並列性が高い．よって，これらのステップは並列化を自然に行える．一方，他のステップは並列性が自明ではない．非直交ベクトルのグループ化 (ステップ 7.) や直交化計算 (ステップ 8.) については効率的な分散並列化の手法が自明ではないため，入力行列に応じて負荷を適切に分散するような手法が必要である．また，行列積計算 (ステップ 9.) は計算量的に負荷が最も高いス

テップであり，速度の面で重要であるため，詳しく述べる．以下の説明では，簡単のため deflation 後のサイズ r を p の倍数とする．

3.1 比較的自明な部分の並列化

小行列の固有分解 (ステップ 2.) では T_j $(j = 1, \ldots, k)$ の固有分解を行う．この計算は各 j について独立であり，プロセス毎に k/p 個の問題を解くことで並列化できる．本研究では，プロセス p_j に $T_{p\ell+j+1}$ $(\ell = 0, \ldots, k/p-1)$ を担当させ，各プロセスは逐次版 LAPACK の DC2 を用いて固有分解を計算する．入力行列 T を全プロセスがそれぞれ保持しているため，この計算の間に通信を行う必要はない．T_j の固有分解がすべて終了した時点で，D，U，C を全プロセスで共有する．

$\tilde{D}$ の固有値計算 (ステップ 5.) は，対角行列と階数 1 の摂動の和の固有値問題を $k-1$ 回解くことにより計算できる [4]．対角行列と階数 1 の摂動の和は，固有値・固有ベクトルを独立に求めることができる．そこで，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，変化する部分を各プロセスが計算し全体で共有する．$\tilde{D}$ の全固有値 $\tilde{\lambda}_j$ $(j = 1, \ldots, r)$ 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，固有ベクトルの再直交化を効率よく行うための準備である．

$\tilde{D}$ 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 $\tilde{F}(\tilde{\lambda}_j)$ $(k-1 \leq \alpha < n+k)$ の核を用いて求めることができる [4]．$\tilde{F}$ は $\tilde{\lambda}_j$ によって異なるため，対応する固有ベクトルを独立に計算できる．本研究では，プロセス p_j が $jr/p+1$ 番目から $(j+1)r/p$ 番目までの r/p 本の固有ベクトルを計算することで，$\tilde{D}$ の固有ベクトル計算 (ステップ 6.) を並列化する．この固有ベクトルの番号は，整列後の固有値の順番に対応する．計算をこのように分担することで，$\tilde{D}$ の固有ベクトルは列ブロック分割形式でデータ分散された状態で各プロセスが保持することになる．固有ベクトルを計算する際は，正規化する前の長さと $\tilde{F}(\tilde{\lambda}_j)$ の最小特異値を保存する [4]．全固有ベクトルを計算したあとに，これらを全プロセスで共有する．これらの値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

グループ化 (ステップ 7.) は固有ベクトル間の直交性の検査と非直交グループの構成に分けることができる．これらと直交化計算 (ステップ 8.) を合わせたものが，$\tilde{D}$ の固有ベクトルの再直交化である．以降では，これら三つの処理の並列化について述べる．

3.2.1 直交性検査の並列化

一般に，固有ベクトル同士の直交性が悪い場合，対応する固有値が近接している場合が多い．PDCK では，整列済みの固有値の，ある連続した範囲に対応する固有ベクトルを各プロセスが計算している．これにより，非直交な固有ベクトルはプロセス番号が近いか同じプロセスに局在していることが期待できる．その結果，後述の [詳細検査] においてデータ通信量を抑制することが可能である．

本ステップで計算する固有ベクトル同士の直交性検査の結果は，二値行列 B に格納する．行列 B は $\tilde{D}$ の固有ベクトルと同様に，列ブロック分割形式で各プロセスにデータ分散させる．プロセス p_j が保持する部分を B_j と表記する $(B = (B_0, \ldots, B_{p-1}),\ B_j \in \{0,1\}^{r \times r/p})$．行列 B の (i,j) 成分 b_{ij} は，ステップ 6. 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)．

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

固有ベクトルの組み合わせで要素の値が決まるため，B は対称行列となる．また前述の通り，非直交な組み合わ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．

固有ベクトルの直交性判定は，次の二段階のステップで実行する．まず，簡易検査を行う．ここで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．各検査は以下の通り．

■簡易検査　固有ベクトル計算 (ステップ 6.) で計算した正規化前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 [4] に譲る．

■詳細検査　ベクトル同士の内積計算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．

直交性検査の進行の様子を，$p = 4$ の場合について図示したものを図 1 に示す．三つ並んだ大きな正方形が各段階における二値行列 B を表しており，左から右に向かって検査が進行する．色のついたブロックは B のその領域を計算したことを表す．正方形の上部にある p_j は j 番のプロセスを指し，p_j の直下のブロック列を B_j とする．

はじめに左の図の最上行を説明する．ここでの目的は p_2 と書かれた黄色の $(1,3)$ ブロックの成分を調べることである．まず p_2 は p_0 の持つベクトルと自分が持つベクトルの間で簡易検査を行い，このブロックの成分を得る．簡易検査の結果，このブロックに 1 となる成分が存在しないならばこのブロックの検査を終わる．そうでない場合，簡易検査で得た直交性の悪い組合せのうち p_0 の持つベクトルの番号を列挙し，最も若い番号 $f_{0,2}$ と大きな番号 $\ell_{0,2}$ を調べる．この二つの数字を p_0 に送信すると，$f_{0,2}$ から $\ell_{0,2}$ までの固有ベクトルが p_0 から p_2 へと送信される．p_2 は受信したベクトルを用いて，簡易検査で直交性が悪いと判定された組の詳細検査を行い，B の成分を得る．これで $(1,3)$ ブロックの検査は終了である．なお正方形上部の矢印は詳細検査のためのベクトルデータの流れを表している．

次の行では，p_0 が p_1 の持つベクトルとの組を検査し緑色の $(2,1)$ ブロックを，p_3 が p_1 の持つベクトルとの組を検査し赤色の $(2,4)$ ブロックをそれぞれ調べる．上の行で行われる検査で発生する通信とこの行で起こる通信はほとんど被らないため，これらの検査はほぼ並列に行われる．

次に中央の図のように，第 3 行と第 4 行の検査を行い，各ブロックの成分を調べていく．上述の通り，上下に隣接する二行は検査を並列に行える．最後は右の図のように，対角ブロックの成分を調べる．このステップでは各プロセスが自分の持つベクトル同士で検査を行う．この検査は完全に独立であり通信も起こらないが，詳細検査が最も多く行われると予想される．空白のブロックが残っているが，それらの対角線対称なブロックは全て色付きであり，必要な検査は全て実行されている．

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．すべてのプロセスが，この手順を同時に実行する．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無いならば $f_{i,s} = \ell_{i,s} = -1$ 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**

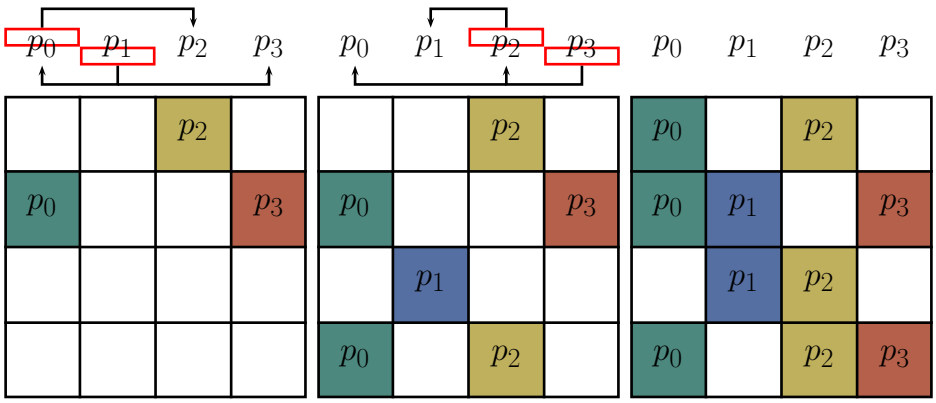

図 1　直交性検査の進行の様子 ($p = 4$ のときの p_0 が行う計算)

17:　　**end for**
18:　**end if**
19: **end for**
20: $\tilde{Q}'_s$ のベクトル同士の直交性を検査する

3.2.2　グループ化

前節で作成した二値対称行列 B に基づき，非直交なベクトル同士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 [7] 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，二値対称行列 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．プロセス p_j は，B の部分行列 B_j を参照することになる．グループ情報は，各固有ベクトルの所属をグループごとに異なる整数値で表し，一本の n 次整数ベクトルに格納する．それぞれのプロセスがグループ情報を構成したあとに，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末尾から降順に繰り返すことで，最終的に全プロセスのグループ情報が 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 $O(n)$ である．これを $p-1$ 回繰り返して最後に全体に送信するため，全体で $O(pn)$ の通信を行う．

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．ベクトル数の少ないグループの直交化は一プロセスのみに担当させ，それを各プロセスで並列に実行することで，並列化を行う．一方グループ内のベクトル数が多い場合は，データ分散などにかかる通信コストより直交化計算の負荷分散によるコスト減を重視し，全プロセスで実行する．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は次の通りである．単一プロセスを用いる場合は，そこにグループ内の全てのベクトルを集め，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データ分散し，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．

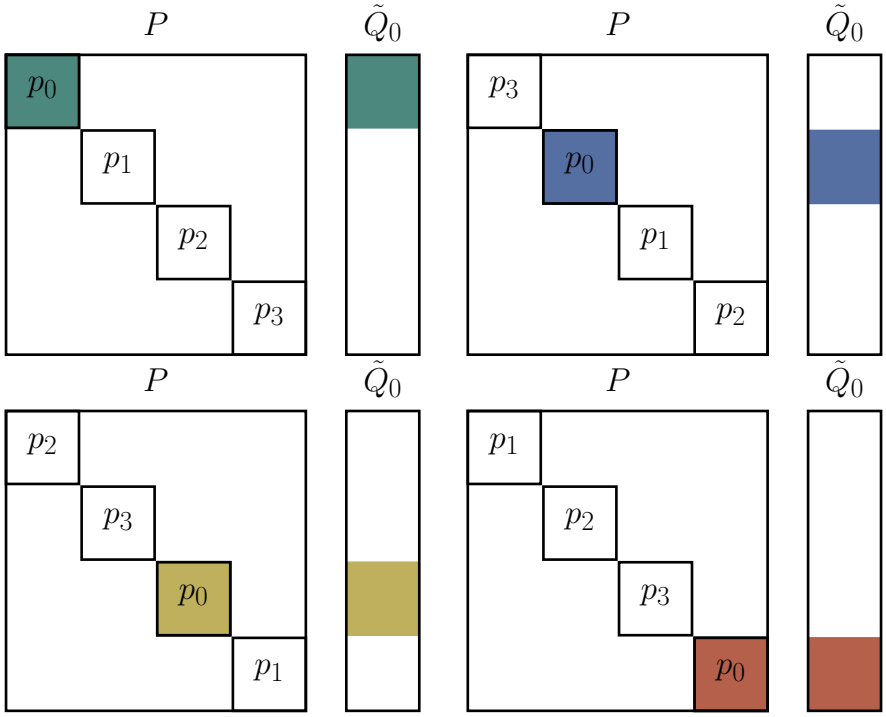

図 2　行列積の進行の様子 ($p = 4$)

3.3　行列積の並列化

T_j の固有分解で得た直交行列 Q_j の直和からなる P と $D + UCU^T$ の固有ベクトルを並べた行列 $\tilde{Q}$ との行列積の並列化について述べる．P と $\tilde{Q}$ はデータ分散されており，とくに P の構造を利用して分散並列計算を行う．

プロセス p_j は P の対角に並ぶブロックのうち $j, p+j, \ldots, (k/p-1)p+j$ 番目の行列と，$\tilde{Q}_j$ を持っている．そこで，まず始めに，各プロセスは P と $\tilde{Q}$ のうち自プロセスが持つデータのみを用いて積をとる．次に，p_j は P の対角ブロックを p_{j-1} に送信し，p_{j+1} の持つ P の対角ブロックを受信する．送信先ないし受信元のプロセス番号を p の法の下で扱う事で，p 個のプロセスが環状に並びデータを送受信することになる．これを繰り返す事で P と $\tilde{Q}$ の行列積を計算できる．

図 2 に $k = p = 4$ の例を示す．ここでは p_0 が計算する様子を表しており，左上から右下に向かい進行する．P の下にある正方形は直交行列 P を表し，対角に並ぶブロックは ステップ 2. でいう Q_j となる．ブロック中に表記されたプロセス番号でそのブロックを保持するプロセスを表している．

まず，p_0 は自分が持つ P の対角ブロック Q_1 と $\tilde{Q}_0$ の対応する箇所の行列積を計算する．これは左上の図で緑色に塗った領域同士で積を取ることに対応する．次に p_0 は，Q_1 を p_3 に送信し，p_1 から Q_2 を受信する．他のプロセスでも同様に，環状に並んだプロセスがバケツリレーを行うように Q_j を送受信する．ブロックの送受信を終えると，受信した P の対角ブロックと $\tilde{Q}_0$ の対応する部分とで積を取る．これは，図右上の青く塗った部分に対応する．続いて，左下，右下の図へと同様の手順を行い，計算ステップが進行する．

以上のような，直交行列 P の対角ブロックを交換し，受け取ったブロックと $\tilde{Q}_j$ の対応する部分とで積を取るという操作を全プロセスで繰り返すことにより，行列積 $P\tilde{Q}$ を分散並列計算できる．

4 前回の若手利用との差異

二期前にも若手利用者推薦制度を利用して研究を行った．当時とはいくつかの部分の並列化手法に変更があるので，その箇所について述べる．本節では，当時の手法を旧 PDCK と呼ぶ.

4.1 固有ベクトル計算の負荷分散

旧 PDCK には固有ベクトルを計算する際の負荷が偏る問題があった.

DCK は deflation により自明な固有ベクトルを得ることで計算すべき固有ベクトル数を r に減らすことができる．分散並列版 DCK は，この r 本を p プロセスに負荷分散して計算することになる．旧 PDCK では，番号の若いプロセスから n/p 本ずつ分担するようにしていた．このとき，もし p_0 から p_s の分担する本数が r となると，p_{s+1} 以降のプロセスは計算を担当しない．そのため旧 PDCK は，deflation が多く起こる状況では負荷分散が行えない．よって，deflation が起きても固有ベクトル計算のステップの計算時間は短縮されなかった.

現在は，deflation 後の本数をプロセス数で割った r/p だけ各プロセスが担当する．これにより，deflation の効果に応じて負荷分散が行われる.

4.2 固有ベクトルの再直交化の負荷分散

分散並列 DCK では，非直交なグループを構成した後にグループごとに直交化計算を行う，旧 PDCK においてはサイズが小さいグループは全てルートプロセス (p_0) のみが担当していた．p_0 が小グループの直交化を行うあいだ他のプロセスは何もしないため，負荷が p_0 に偏っていた.

そこで，小グループの直交化計算を p_0 以外も担当するよう変更することで，直交化計算の負荷を全プロセスに分散させる.

4.3 行列積での PBLAS の使用

ステップ 9. の行列積計算において，旧 PDCK では通信と計算を自前で制御していた．一方 PBLAS には分散並列で行列積を行うルーチンが用意されており，このルーチンを利用する事でも行列積計算は可能である.

簡単なベンチマークにより両者を比較したところ，プロセス数に近い分割数をとる場合は，PBLAS を用いる方が高速という結果を得た．一方，プロセス数に比較して大きな分割数をとる場合は，自前で制御する方が高速だった．後者の場合，分割数が大きいために固有ベクトル計算や再直交化などの他のステップの所要時間が大きく増加する事が経験上予想される．そこで，本研究では PBLAS を用いて計算するようにした.

4.4 自明解の保持方法の変更

旧 PDCK は，行列積を計算する際に P の対角ブロックと合わせて自明解の情報を送受信し，自明解を保持すべきプロセスが必要な対角ブロックを受け取ると，自明解を適切な領域に保持していた．$D + UCU^T$ の自明な固有ベクトルは I_{n-r} のいずれかの列ベクトルなので，これを T の固有ベクトルへと相似変換する際に P との積を計算する必要は無く，適切な P の列をコピーすれば良い.

しかし，ある自明解に対応する対角ブロックの列はブロックの送受信を行う前に判断でき，そのようなブロックを持つプロセスが自明解を保持する事でそもそも通信を行う必要がなくなる．現在はそのように変更することで通信を削減した.

5 数値実験

分散メモリ型並列計算機 HITACHI HA8000 上で数値実験を行い，本稿で述べた PDCK の性能を調べる．世界的に広く使用されている数値線形代数ライブラリ ScaLAPACK に収録されている解法とも比較を行う．

5.1 実験方法

本研究の PDCK を用いて行列の固有値問題を解き，その計算時間と計算精度を調べる．参考とする比較対象として，世界的に標準として利用されている数値計算ライブラリである LAPACK(BLAS) [8] の分散並列版ルーチンが収録されている ScaLAPACK(PBLAS) [9] に実装された解法を利用する．

具体的には，PDCK 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し，計算した解の相対残差と直交誤差を調べる．同様に二分法・逆反復法 (BII)，二分割の分割統治法 (DC2) の計算時間と計算精度も調べ，提案手法と比較する．どちらの解法も ScaLAPACK のルーチン (それぞれ `pdstebz`/`pdstein`, `pdstedc`) を用いる．

5.2 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である:

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

5.3 対象行列

数値実験には，以下の三種類の行列を用いた．

平均 0，分散 1 の正規乱数を成分とする実対称行列を三重対角化した行列 [10] 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 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．

また二つの同じ行列 QC を対角に並べ，継ぎ目にあたる副対角成分の値を 1 とした実対称三重対角行列を行列 SQ と呼ぶ．この行列は，固有値同士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，固有ベクトルの数値的直交性を確保しづらい．

主対角成分 $a_j = j \times 10^{-6}$，副対角成分 $b_j = 1$ の実対称三重対角行列を行列 DS と呼ぶ．行列 DS は deflation 率の低い，人為的に構成された行列である．

5.4 実験環境

東京大学情報基盤センターの HITACHI HA8000 を 128 CPU (8 ノード) 利用した．HA8000 は，4 コア持つ CPU を 4 個 (計 16 コア) 搭載したノードが高速ネットワークで接続された，分散並列計算機である．システムの諸元を 表 1 に示す．実験時は，プロセス数と同数の CPU を使用する．

表 1　HITACHI HA8000 のスペック

1 コアあたりの理論性能	9.2 GFLOPS
1 ノードあたりのコア数	16
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	32 GB(一部 128GB)

5.5 実験条件

数値計算ライブラリとして世界的な標準として利用されている LAPACK と，その下位で基礎線形代数演算を担う BLAS を用いる．本研究では分散並列計算を行うため，同ライブラリの分散並列版である ScaLAPACK と PBLAS も用いる．ScaLAPACK(PBLAS) は線形代数演算向けメッセージ交換ライブラリ BLACS [11] を通して MPI [12] による通信を行うが，PDCK も同様に BLACS を用いる．MPI 通信には，MPI-2 通信ライブラリを用いる．いずれのライブラリも並列計算機に備え付けのものを利用した．

プログラムは C 言語で実装し，HITACHI 最適化 C コンパイラでコンパイルした．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．

5.6 実験結果

本節では，行列ごとに実験結果を述べる．

5.6.1 行列 QC

行列サイズ $n = 20000, 40000, 60000$ の行列 QC の固有分解を計算した際の所要時間を 図 3 に示す．横軸は行列サイズ，縦軸は計算時間で，縦軸のみ対数軸である．`BII` は二分法・逆反復法，`DC2` は二分割の分割統治法，`DCK128` と `DCK256` は PDCK(分割数 $k = 128, 256$) である．行列 QC は deflation があまり起こらないため，分割数 k を大きくすることで行列積の演算量を削減できる DCK が DC2 より有効となる．

次に，行列 QC の固有分解の相対残差を 図 4 に，直交誤差を 図 5 に示す．どちらも横軸は行列サイズ，縦軸は誤差であり，縦軸のみ対数軸とした．

行列 QC の固有分解の計算時間を解法ごとに示す図 3 を見ると，PDCK は DC2 より高速に計算している．$n = 60000$ においては，DC2 は約 940 秒であるのに対し DCK256 は約 45 秒と，1/20 以下の時間で計算している．これは分割数を 2 より大きく取ることで行列積の演算量を削減できるという DCK の特徴によるものである．一方，並列性の高い BII は，同サイズで約 30 秒とさらに高速であった．なお，DCK128 が $n = 60000$ で計算時間が大きくなっているが，これは行列積計算の遅延によるものである．この原因は調査中であるが，PBLAS の通信に遅延が出ていることによる可能性がある．

同行列の固有分解の相対残差を示す図 4 によると，PDCK，DC2，BII はいずれも 10^{-13} より小さく，十分な相対残差である事が分かる．また直交誤差を記載した 図 5 によると，PDCK は 10^{-12} 程度であり，BII より 1 桁弱良いという結果が得られた．PDCK は再直交化によって直交精度を確保しており，DC2 はレブナーの定理を用いた精度確保の手法が確立しているため，十分な精度で計算できている．一方 BII はプロセス間の再直交化を行わないため，精度が低い．

5.6.2 行列 SQ

行列 SQ は，差がマシンイプシロン程度になる固有値の組が存在するため，固有ベクトルの直交性を確保しにくい行列である．また deflation はあまり起こらない．図 6 は行列 SQ の固有分解の計算時間を示す図である．続いて行列 SQ の固有分解の相対残差を 図 7 に，直交精度を 図 8 に示す．

行列 SQ の固有分解の計算時間を示す図 6 を見ると，定性的には行列 QC の実験結果と同様であった．DCK128 の遅延も同様である．

行列 SQ の固有分解の直交精度を示す図 8 によると，BII は 10^{-11} より低い．全プロセス間で再直交化を行う提案手法は，いずれの次数でも DC2 と同等の精度で計算できている．DCK128 が $n = 60000$ で大きく精度を下げているが，これは現在調査中である．相対残差を示す 図 7 を見ると，BII を含むいずれの解法も ϵ_r は十分小さく，良好である．

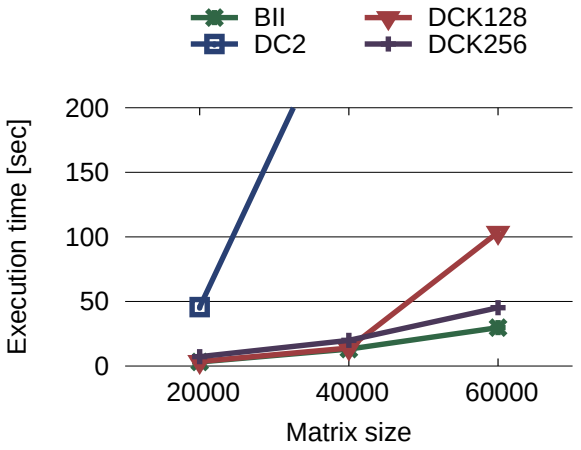

図 3　計算時間 (行列 QC)

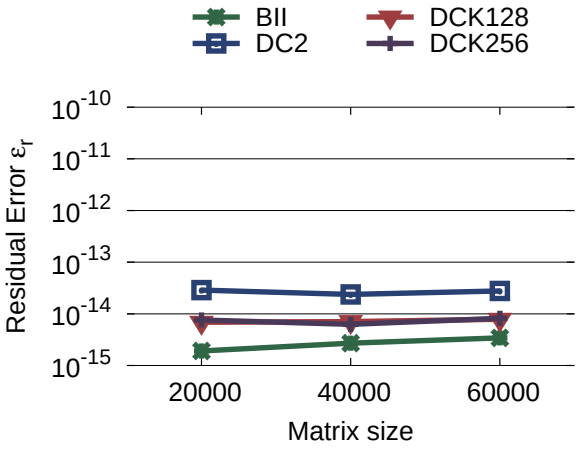

図 4　相対残差 (行列 QC)

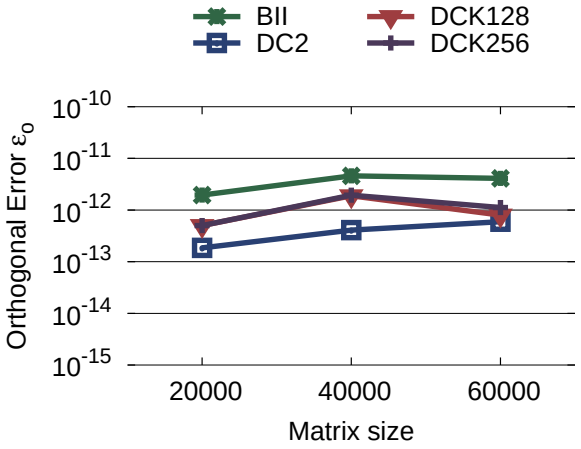

図 5　直交誤差 (行列 QC)

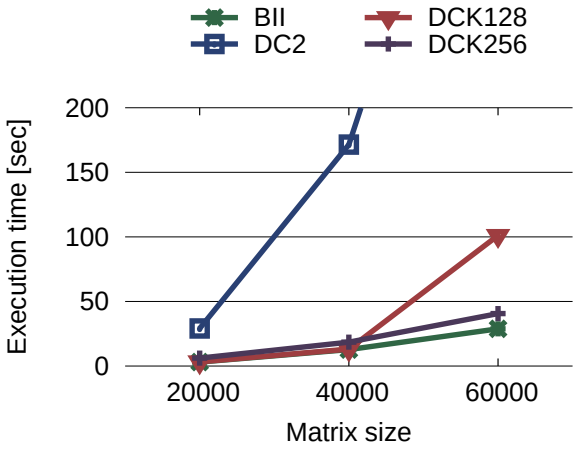

図 6　計算時間 (行列 SQ)

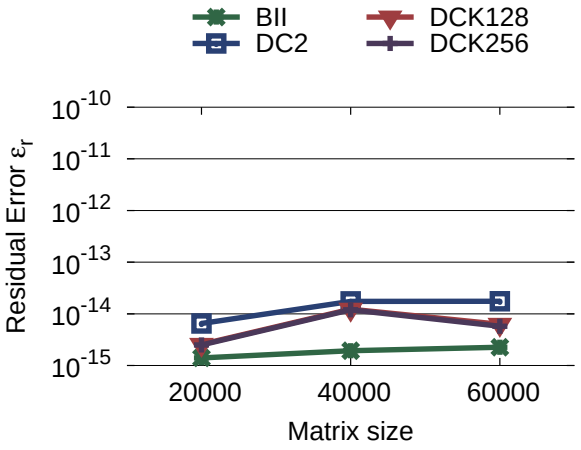

図 7　相対残差 (行列 SQ)

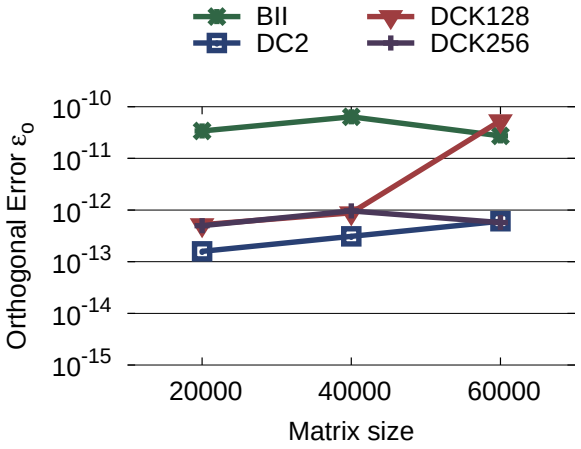

図 8　直交誤差 (行列 SQ)

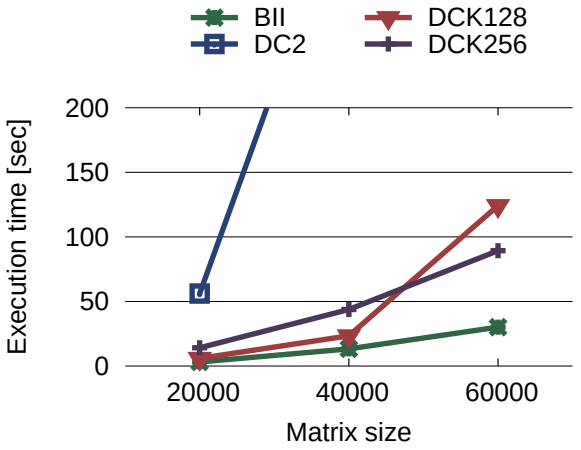

図 9　計算時間 (行列 DS)

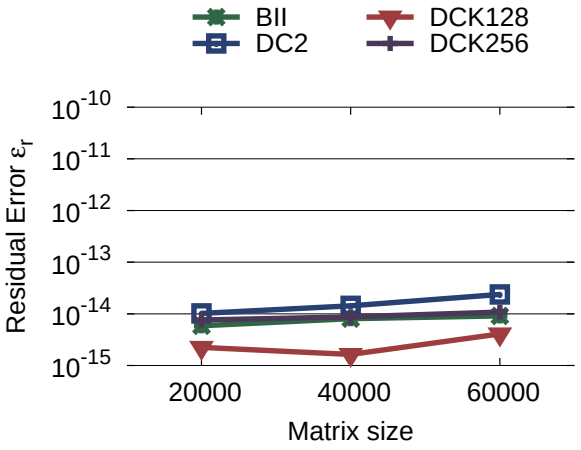

図 10　相対残差 (行列 DS)

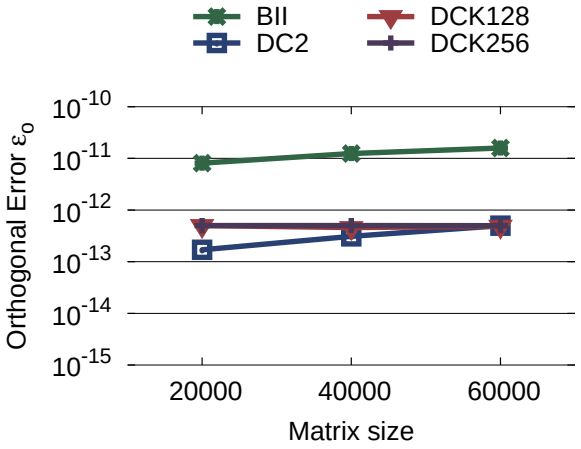

図 11　直交誤差 (行列 DS)

5.6.3 行列 DS

行列 DS の固有分解の計算時間を 図 9 に示す．行列 QC と同様に deflation はほとんど起こらないため，分割数を大きくできる提案手法が DC2 より有効である．続いて行列 DS の固有分解の相対残差を 図 10 に，直交精度を図 11 に示す．

行列 DS の固有分解の計算時間を示す図 9 を見ると，BII は行列 QC の結果と同様の振る舞いをしている．一方 DCK と DC2 は行列 QC の 1.5 ～ 2 倍の時間がかかっている．行列 DS の固有分解を PDCK で行うと，固有ベクトル計算で用いる $\tilde{F}(\lambda_j)$ のサイズが大きくなり，(行列 QC の場合と比較して 3 倍程度)，直交性検査の詳細検査を行う回数が多い (行列 QC の 2 倍程度) ため，このような結果となったと考えられる．またこの行列でも DCK128 の遅延が $n = 60000$ にて起きている．

同行列の固有分解の直交精度を示す図 11 を見ると，PDCK と DC2 は BII と比較して一桁強ほど良い直交精度で計算できている．図 10 に示す相対残差はいずれの解法も十分小さく，良好である．PDCK は BII の倍の時間を要するが直交精度は 2 桁良いため，直交精度を重視する場合には提案手法が有用である．

6 まとめ

本研究では，多分割の分割統治法 (DCK) の分散並列版アルゴリズム (PDCK) を HA8000 環境に移行した．前回の若手利用 (H21 前期) での報告と比較し，固有ベクトル計算や直交化計算などの負荷分散を改善している．

数値実験では，PDCK が二分割の分割統治法 (DC2) に対して計算時間の点で有効であることが示された．一方，二分法・逆反復法 (BII) に対しては 1.5 から 3 倍程度の計算時間となった．これには，高精度に固有ベクトルを計算するための処理が影響している．

精度面については，相対残差は実験で用いたどの解法も十分小さいという結果を得た．直交精度は，行列 SQ と行列 DS に対しては PDCK が BII より二桁ほど良く，行列 QC に対しては同等であることが示された．DC2 とはいずれの行列も同程度の精度が得られた．PDCK はプロセス間の再直交化を高速に行う処理があるため，十分な精度が得られている．

以上より，精度に重きを置く場合は PDCK が有効であると言える．

今後の課題として挙げられるのは，行列積において起こる遅延への対処である．より詳細なベンチマークを行い，適切な手法を選択する必要がある．

謝辞

本研究に対する東京大学情報基盤センターの若手利用者推薦制度による支援に，感謝申し上げます．

情報基盤センターへ要望を挙げるとするならば，LAPACK や ScaLAPACK，MATRIX/MPP 以外の数値計算ライブラリを HA8000 上で提供していただけると良いと思いました．コンパイル時のオプション等も公開していただけると，自前でコンパイルする際に一つの参考にできるのではないかと思いました．

参考文献

[1] J.J.M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik*, Vol. 36, pp. 177–195, 1980.

[2] I.S. Dhillon. A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem. Technical Report UCB/CSD-97-971, University of California at Berkeley, Berkeley, 1997.

[3] 桑島豊, 重原孝臣. 実対称三重対角固有値問題の分割統治法の拡張. 日本応用数理学会論文誌, Vol. 15, No. 2, pp. 89–115, 2005.

[4] 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.

[5] 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.

[6] Y. Ishikawa, J. Tamura, Y. Kuwajima, and T. Shigehara. *Software Automatic Tuning*, chapter 6. Springer, 2010.

[7] T.H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, 浅野哲夫, 岩野和生, 梅尾博司, 山下雅史, 和田幸一. アルゴリズムイントロダクション 改訂 2 版 第 2 巻. 近代科学社, 2007.

[8] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, third edition, 1999.

[9] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[10] I. Dumitriu and A. Edelman. Matrix models for beta ensembles. *Journal of Mathematical Physics*, Vol. 43, No. 11, pp. 5830–5847, November 2002.

[11] Jack J. Dongarra and R. Clint Whaley. A user's guide to the blacs v1.1. Technical report, 1997.

[12] MPI Forum. *MPI-2: Extensions to the Message-Passing Interface*, 2003.