ブロックヤコビ法による超並列固有値計算プログラムの開発

高橋佑輔，廣田悠輔，山本有作

神戸大学大学院システム情報学研究科計算科学専攻

１．はじめに

実対称固有値問題の解法は，化学や工学の多くの分野で必要とされるものである。その中でも分子軌道法などの量子化学計算では，中規模（ N ～10000）の実対称固有値問題を解くのに，多くの計算時間を必要としている。ゆえにこの解法の高速化が強く求められている。

実対称固有値問題を解くための一般的な手法として，三重対角化を用いた方法が挙げられる[7]。この手法では，まず入力された行列を直交変換で実対称三重対角化行列に変換をし，変換された行列に対して固有値と固有ベクトルを求め，固有ベクトルを逆変換することで元の行列の固有ベクトルを求める。この手法は，計算量の観点からは最適であり，LAPACK[1]に採用され広く利用されている。また，大規模並列分散メモリマシン向けに，ScaLAPACK[5]のような高性能実装も行われている。しかし，これらの解法は，超大規模問題向けに設計されたものであり，数千コアの並列計算機の性能を十分に引き出すためには，行列サイズが数十万元以上と大きい必要がある。

ここでは，中規模の実対称固有値問題の解法で数千コアを活用することを考える。そのために本研究ではブロックヤコビ法を利用する。ブロックヤコビ法は三重対角化を用いた解法に比べて，数倍もの計算量を必要とするデメリットがある。しかし，それ以外の点で様々なメリットがある。まず，大粒度並列処理が可能であり，ブロックサイズを L とした時，各プロセッサで $O(L^3)$ の浮動小数点演算を同期なしで行うことが出来る。次に，ほとんどの計算を行列乗算で実行でき，これは，現在のマイクロプロセッサの性能を引き出すのに特に適している。最後に，アルゴリズムが単純であり，演算が小規模行列の固有値問題の計算と行列乗算の2つだけで成り立っているという点である。本研究の目的は，ヤコビ法を基にした実対称固有値問題を大規模な分散メモリ型並列計算機場で実装し，性能上のボトルネックを同定するとともに，その改善法を検討することである。

以下，2章では，ヤコビ法とその並列化手法を解説する。並列化手法には，1次元分割と2次元分割を利用する方法があり，どちらがより大規模並列計算に適しているかを比較する。3章では，T2Kコンピュータを利用し，1024コアまで利用した結果を記載する。最後に4章では結果について考察を行う。

２．ブロックヤコビ法とその並列化手法

２．１ 巡回ブロックヤコビ法

実対称行列 $A \in \boldsymbol{R}^{N \times N}$ を考える。ブロックヤコビ法では，行列 A をサイズ $L \times L$ のブロック単位に分割する。いま， $W = N/L$ とおく。また， (I, J) 番目にある行列Aのブロック行列を

A_{IJ} と表す。巡回ヤコビ法では，$A_{IJ}(I<J)$ となる上三角非対角ブロックの1つを選択し，直交変換を行うことで，このブロックの要素を消去する[7]。この処理を，$W(W-1)/2$ 個の非対角ブロックに行い，これを1反復とする。あるブロックの消去を行うと，既に消去したブロックの要素が非ゼロ要素になってしまう場合があるので，行列全体がほぼ対角行列になったと言える状態になるまでこの反復計算を繰り返す必要がある。非対角ブロック A_{IJ} を消去する直交行列を $P^{(I,J)}$ とする。いま，

$$\tilde{A}=\begin{bmatrix} A_{II} & A_{IJ} \\ A_{JI} & A_{JJ} \end{bmatrix} \in \boldsymbol{R}^{2L\times 2L}$$

と定義し，$\tilde{A}$ を対角化する直交行列を $\tilde{P}\in\boldsymbol{R}^{2L\times 2L}$ とする。すなわち，

$$\tilde{P}^{T}\tilde{A}\tilde{P}=\tilde{\Lambda} \quad .$$

この $\tilde{P}$ を $L\times L$ の部分行列に分割して，指定されているブロックに組み込み，それ以外の要素は単位行列 I_N の $(I,I),(I,J),(J,I),(J,J)$ ブロックに埋め込んでできる $N\times N$ 行列を $P^{(I,J)}$ とする。上の式より，(I,J) 番目と (J,I) 番目の部分行列は，$(P^{(I,J)})^{T}AP^{(I,J)}$ の計算によって零行列になり，対角行列が得られることが分かる。計算の手順はまず，$(P^{(I,J)})^{T}$ を左側から掛けることによって，行列 A の I 番目及び J 番目のブロック行が更新される。今度は同様に $P^{(I,J)}$ を右側から掛けることによって，行列 A の I 番目及び J 番目のブロック列が更新される。

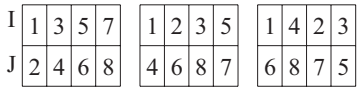
[アルゴリズム1:巡回ブロックヤコビ法]

1. $P=I_N$
2. for $n=1,2,\cdots$ do
3. for $(I,J)=(1,2),(1,3),\cdots,(1,W),(2,3),\cdots,(W-1,W)$ do
4. $\tilde{A}=\begin{bmatrix} A_{II} & A_{IJ} \\ A_{JI} & A_{JJ} \end{bmatrix}$ を対角化する直交行列 $\tilde{P}$ を求める
5. $\tilde{P}$ を $N\times N$ の行列 $P^{(I,J)}$ に拡張
6. $A\leftarrow(P^{(I,J)})^{T}A$
7. $A\leftarrow AP^{(I,J)}$
8. $P\leftarrow PP^{(I,J)}$
9. end for
10. end for

ここで，巡回ブロックヤコビ法のアルゴリズムをアルゴリズム1に示す。全ての非対角ブロックは1反復毎に必ず1度選ばれて消去が行われる。計算が収束した後は，行列 A の対角要素は固有値にな

り，行列 P の列ベクトルは固有ベクトルになる。また，このアルゴリズムは， $2L\times 2L$ の固有ベクトルの導出（ステップ4）と，行列乗算（ステップ6〜8）の2つの演算で成り立っている。演算量に関して，ステップ4は $32L^3$ ，ステップ6〜8はそれぞれの計算が $24L^2N$ となっており，

$L \ll N$ であれば，行列乗算が演算のほとんどを占めることになる。これより，巡回ブロックヤコビ法は現代の高性能アーキテクチャに適した方法であると言える。



(a)Step1　(b)Step2　(c)Step3

図２．１．巡回ロビン法

２．２ 巡回ブロックヤコビ法の並列化

巡回ブロックヤコビ法で， (I_1, J_1) 番目の非対角ブロックの後に消去するブロックを (I_2, J_2) 番目とし，ここで I_1, I_2, J_1, J_2 は全て異なる整数であるとする。この場合， $P^{(I_2, J_2)}$ を決める際の4つの部分行列 $A_{I2,I2}, A_{I2,J2}, A_{J2,I2}, A_{J2,J2}$ は， $P^{(I_1, J_1)}$ を利用する直交変換の更新に関係しないので，

$P^{(I_1, J_1)}$ の計算の順番は $P^{(I_2, J_2)}$ の前に来ても，後に来ても，どちらでも良いことになる。これより，(I_1, J_1) 番目と (I_2, J_2) 番目の $\tilde{P}$ を利用した計算は並列化できる，と言える。更に，ステップ6での $\left(P^{(I_1, J_1)}\right)^T$ と $\left(P^{(I_2, J_2)}\right)^T$ を利用した乗算も並列化でき，ステップ7の乗算やステップ8の P の更新も同様である。よって，このアルゴリズム1のステップ3のループにおいて， (I_1, J_1) に対する処理と (I_2, J_2) に対する処理は並列に実行出来る。

次に $I_1, I_2, \ldots, I_{W/2}$ と $J_1, J_2, \ldots, J_{W/2}$ は全て異なる整数であると仮定する。ステップ4での (I, J) の組み合わせは， $(I_1, J_1), (I_2, J_2), \ldots, (I_{W/2}, J_{W/2})$ の計 $W/2$ 個になる。ステップ6〜8は前述の内容と同様に並列化できる。上三角行列には全部で $W(W-1)/2$ 個の非対角ブロックがあるので，ステップ3で消去する非対角ブロックを選択する回数は $W-1$ 回になる。

$(I_1, J_1), (I_2, J_2), \ldots, (I_{W/2}, J_{W/2})$ を決めるには，ラウンドロビン巡回法を利用する[6]。 $W=8$ として，始めに，図2.1(a)にあるように (I, J) の組み合わせは $(1,2), (3,4), (5,6), (7,8)$ を利用して， $A_{12}, A_{34}, A_{56}, A_{78}$ を同時に消去する。次に移る前に，1以外の整数の位置を図2.1(b)の様に右回りに移動させて，この図より， $(1,4), (2,6), (3,8), (4,5)$ の組み合わせを選び，

$A_{14}, A_{26}, A_{38}, A_{45}$ を同時に消去する。

更に次に移る前に，先ほどと同じく 1以外の整数の位置を図2.1(c)の様に右回りに移動させて，この図

より，$A_{16}, A_{48}, A_{27}, A_{35}$ を同時に消去する。以上の手順を $W-1=7$ 回繰り返すことで，全ての非対角ブロックの組み合わせを選択して消去することが出来る。

並列巡回ブロックヤコビ法のアルゴリズムをアルゴリズム2に示す。

[アルゴリズム2:並列巡回ブロックヤコビ法]

1. $P=I_N$
2. for $n=1,2,\cdots$ do
3. for $K=1,2,\ldots,W-1$ do
4. for $(I,J)=(I_1^{(K)},J_1^{(K)}),(I_2^{(K)},J_2^{(K)}),\ldots,(I_{W/2}^{(K)},J_{W/2}^{(K)})$ do
5. $\tilde{A}=\begin{bmatrix} A_{II} & A_{IJ} \\ A_{JI} & A_{JJ} \end{bmatrix}$ を対角化する直交行列 $\tilde{P}$ を求める
6. $\tilde{P}$ を $N\times N$ の行列 $P^{(I,J)}$ に拡張
7. $A\leftarrow(P^{(I,J)})^T A$
8. $A\leftarrow AP^{(I,J)}$
9. $P\leftarrow PP^{(I,J)}$
10. end for
11. end for
12. end for

２．３ 行列データの分割法

２．３．１ 1次元分割

並列計算機上で並列巡回ブロックヤコビ法を実装するために，データの分割方法を考える必要がある。従来法として用いられているのは１次元分割である[4][6][9][10]。この方法では $W/2$ 個のプロセッサが利用される。 l 番目 $(l=1,2,\ldots,W/2)$ のプロセッサが $I_l^{(K)}$ 列と $J_l^{(K)}$ 列のブロックを受け持つとして，アルゴリズム2のステップ5～9の (I,J) を $(I,J)=(I_l^{(K)},J_l^{(K)})$ とする。ステップ5及びステップ8・9は $I_l^{(K)}$ 番目及び $J_l^{(K)}$ 番目のブロック列を利用して計算するため，これらは通信なしに実行出来る。しかし，ステップ7を実行する場合， l 番目のプロセッサは $(I,J)=(I_1^{(K)},J_1^{(K)}),(I_2^{(K)},J_2^{(K)}),\ldots,(I_{W/2}^{(K)},J_{W/2}^{(K)})$ となる $P^{(I,J)}$ が必要であり，そのために全体全通信を行う必要がある。ステップ4～10の処理を終えた後は，ラウンドロビン法に従うと， l 番目のプロセッサは， $I_l^{(K+1)}$ 番目及び $J_l^{(K+1)}$ 番目の行と列を入れ替える必要がある。それぞれ $(I_1^{(K)},J_1^{(K)}),(I_2^{(K)},J_2^{(K)}),\ldots,(I_{W/2}^{(K)},J_{W/2}^{(K)})(K=1,2,\ldots,W-1)$ となるような組を選択すると，全ての

l と K において $(I_1^{(K)}, J_1^{(K)})$ と $(I_1^{(K+1)}, J_1^{(K+1)})$ の要素を入れ替えることになる。この時，互いのプロセッサは2つのブロック行の代わりに，1つのブロック行を送受信する必要がある。また，この時プロセス内の通信は円環状になり，隣接したプロセッサとのブロック列を入れ替えるため，通信に時間がかかる。

1次元分割はアルゴリズム2のステップ5〜9の処理の実行をする際にアイドル時間が無いという点では有利である。一方で欠点も有り，その中でプロセス数 p が $p \leq N/2$ と制限されてしまう点がある。更に行列乗算部分を実行する際，level-3 BLASの性能を引き出すには，行列の行および列の数を100以上にしなければならない。これより， $N=10,000$ の問題を解く時は，プロセス数はおよそ100に制限されてしまい，中規模問題を数千プロセスを用いて解くという今回の目的に反する。また，1次元分割では， $(I,J)=(I_1^{(K)}, J_1^{(K)}), (I_2^{(K)}, J_2^{(K)}), \ldots, (I_{W/2}^{(K)}, J_{W/2}^{(K)})$ となるを $P^{(I,J)}$ 全てのプロセッサで必要とするために全体全通信をしたり，プロセス間でブロック列全体を交換する必要がある，といった問題がある。これらの通信パターンは超並列環境には適していない。

２．３．２ 2次元分割

これらの問題を避けるために，本研究では2次元分割を提案する。この手法では， $W^2/4$ 個のプロセッサを利用して， l 行 m 列目 $(l, m=1,2,\ldots,W/2)$ のプロセッサが，4つのブロックとなる $A_{I_l^{(K)}, I_m^{(K)}}, A_{I_l^{(K)}, J_m^{(K)}}, A_{J_l^{(K)}, I_m^{(K)}}, A_{J_l^{(K)}, J_m^{(K)}}$ を持つものとする。対角にあるプロセッサ $(l=m)$ がアルゴリズム2のステップ5を実行し，その後，対角行列 $\tilde{P}$ をブロック列の番号が同じプロセッサとブロック行の番号が同じプロセッサに対して，ブロードキャストする。この時，全てのプロセッサで対角行列を受け取るので，ステップ7〜9の処理が実行される。2次元分割を用いた並列巡回ヤコビ法のアルゴリズムに関して示したものが，アルゴリズム3である。ここではMPIのような記述を用いて， $W^2/4$ 個のプロセッサを用いる方法を表している。

2次元分割にはいくつもの利点が挙げられる。まずは，多くのプロセッサを使いやすいという点である。行列乗算の際の行列サイズを 100×100 と制限しても， $N=10,000$ の問題を解く場合， $10,000$ プロセッサを利用することが出来る[1]。次に，全体全通信の必要が無いという点である。対角行列 $\tilde{P}_l^{(K)}$ の通信は，同じブロック行かブロック列のプロセッサ群毎に独立して行われる。更に，プロセッサ間で行う1回の通信のサイズは，ブロック行全体ではなく，4つのブロック行列の大きさの分のみである。特に通信に関しての利点によって，プロセッサ間通信の回数や通信量を大幅に減らすことが出来る。

---

1 4つのブロックを取ることから，各プロセスの持つ行列のサイズを $2L \times 2L$ とする。この場合，$L=50$ とすると，プロセス数は $W^2/4 = N^2/4L^2 = 10,000$ となる。

［アルゴリズム3:2次元分割を用いた並列巡回ブロックヤコビ法］

1. プロセス番号を取得する (l,m) 。
2. $\tilde{A}=\begin{bmatrix} A_{I_l^{(K)},I_m^{(K)}} & A_{I_l^{(K)},J_m^{(K)}} \\ A_{J_l^{(K)},I_m^{(K)}} & A_{J_l^{(K)},J_m^{(K)}} \end{bmatrix}$
3. if $l=m$ then
4. $\tilde{P}=I_{2L}$
5. else
6. $\tilde{P}=O_{2L}$
7. end if
8. for $n=1,2,\cdots$ do
9. for $K=1,2,\ldots,W-1$ do
10. if $(l=m)$ then
11. $\tilde{A}$ を対角化する直交行列 $\tilde{P}$ を導出
12. l 行のプロセッサに $\tilde{P}_l^{(K)}$ を送信
13. l 列のプロセッサに $\tilde{P}_l^{(K)}$ を送信
14. else
15. l 行 l 列のプロセッサから $\tilde{P}_l^{(K)}$ を受信
16. m 行 m 列のプロセッサから $\tilde{P}_m^{(K)}$ を受信
17. end if
18. $\tilde{A}\leftarrow(\tilde{P}_l^{(K)})^T\tilde{A}$
19. $\tilde{A}\leftarrow\tilde{A}\,\tilde{P}_m^{(K)}$
20. $\tilde{P}\leftarrow\tilde{P}\,\tilde{P}_m^{(K)}$
21. $I_l^{(K+1)},J_l^{(K+1)},I_m^{(K+1)},J_m^{(K+1)}$ を計算
22. 他のプロセッサの持つブロックと交換

    $A_{I_l^{(K+1)},I_m^{(K+1)}},A_{I_l^{(K+1)},J_m^{(K+1)}},A_{J_l^{(K+1)},I_m^{(K+1)}},A_{J_l^{(K+1)},J_m^{(K+1)}}$
23. end for
24. end for

一方で，2次元分割には，対角にあるプロセッサがステップ11を実行する際に，それ以外のプロセッサがアイドル状態になる，という欠点がある。これにより，プロセッサ利用率の割合においては，1次元分割より劣る。

Table 2.1. アルゴリズム3のステップ11，18〜20の実行時間の予測

ステップ	演算処理	計算量	プロセッサ利用数	実行時間
11	$\tilde{A}$ の対角化	$32L^3\times p^{\frac{1}{2}}$	$p^{\frac{1}{2}}$	$32L^3$
18	$\tilde{A}\leftarrow(\tilde{P}_l^{(K)})^T\tilde{A}$	$8L^2N\times p^{\frac{1}{2}}$	p	$16L^3$
19	$\tilde{A}\leftarrow\tilde{A}\tilde{P}_m^{(K)}$	$8L^2N\times p^{\frac{1}{2}}$	p	$16L^3$
20	$\tilde{P}\leftarrow\tilde{P}\tilde{P}_m^{(K)}$	$8L^2N\times p^{\frac{1}{2}}$	p	$16L^3$

Table 2.2. アルゴリズム3のステップ12/15，13/16，22の通信時間

ステップ	演算処理	データサイズ（バイト）	ステップ数	通信時間
12/15	l 行への $\tilde{P}_t^{(K)}$ のブロードキャスト	$4L^2\times 8$	$\frac{1}{2}\log_2 p$	$(16L^2\log_2 p)/s$
13/16	l 列への $\tilde{P}_t^{(K)}$ のブロードキャスト	$4L^2\times 8$	$\frac{1}{2}\log_2 p$	$(16L^2\log_2 p)/s$
20	ブロックの入れ替え	$2\times 4L^2\times 8$	1	$64L^2/s$

そこで，全体のパフォーマンスにおいて，ステップ11の影響を測るために，弱スケーリングによる解析を行う。ここで，1プロセッサに割り当てる行列のサイズを $2L\times 2L$ とし，また K をプロセッサ数 p として，ステップ11とステップ18〜20の実行時間をモデル化する。この時，全体の行列のサイズは $N=2Lp^{\frac{1}{2}}$ となる。予測の結果を表したものがTable 2.1.である。表にある実行時間というのは，1プロセッサ当たりの実行時間のことである。表から，ステップ11の実行全体に対する時間の割合は， p とは独立で一定であるということが分かる。これは，2次元分割のプロセッサ利用の面において不利になる点ではあるが，弱スケーラビリティは低下しない。一方で，通信時間の結果を表したものがTable 2.2.である。 s はプロセッサ間通信でのバンド幅(Bytes/sec)を表し，表にある通信時間とは，通信完了に必要となる時間である。ここで，ブロードキャストは二分木を利用し，1ステップで1ブロック分入れ替えが出来るものとしている。この表から，全体の実行時間に対する通信時間の影響は $\frac{1}{L}\log_2 p$ 程度の小さいものであるということが分かる。これらの解析から，2次元分割は超並列環境に適していると言える。

３．計算結果

３．１ 実行環境

アルゴリズム3で示した方法に基づく並列巡回ブロックヤコビ法のプログラムを実装し，その性能を

評価する。プログラムはC言語を利用し，MPIを用いた並列化を行う。対角ブロックである$\tilde{A}$の対角化はLAPACKの関数dsyevを利用し，行列乗算にはBLASのDGEMMを利用する。プログラムは，東京大学のスーパーコンピューターT2Kで実行される。T2Kの性能はTable3.1.に表す。テスト行列として，0～10の範囲の乱数を利用した実対称行列を準備する。ブロックサイズは，$L=125$とし，行列サイズはプロセッサ数pを用いて，$N=2Lp^{\frac{1}{2}}$とする。pの数は4，16，64，256，1024と設定し，Nは500，1000，2000，4000，8000と定まる。ブロックヤコビ法の終了条件は，非対角要素の最大値が10^{-10}以下になった時とする。

Table 3.1. 東京大学のスーパーコンピューターT2Kのスペック

アイテム	演算処理
ノード	AMD Opteron Processor（2.3GHz，4 Cores）×4 64ノード利用で最大1024コア
メモリ	ノード当たり32Gbytes
OS	Red Hat Enterprise Linux
コンパイラ	PGI C/C++ コンパイラ
BLAS・LAPACK	AMD計算ライブラリ

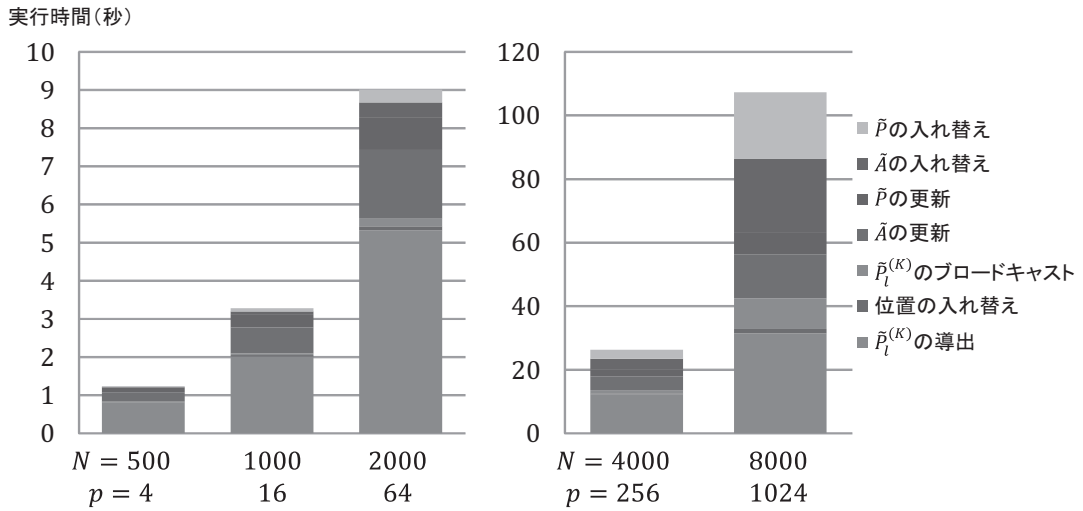


Fig.3.1 T2Kを用いた並列巡回ブロックヤコビ法の実行時間

３．２ 結果

pを変えてプログラムを実行した結果をグラフにしたものがFig. 3.1.である。グラフより，pが小さい時，$\tilde{A}$の対角化が全体の計算に対して支配的であることが分かる。そして，pが増加するにつれて，ブロードキャストや入れ替えの時間が増えていく。この結果は，$\tilde{A}$の対角化の全実行

時間に対する割合は，　p　が増えても一定値以内に収まるという2.3章の内容と一致する。演算量が　$O(N^3)$　，利用するコア数は　$O(N^2)$　ということから，全体の実行時間は　N　に比例して増加すると予想した。しかし，グラフでは，実行時間は　N　が2倍になった時，実行時間は2倍より更に増加している。これには2つの原因があり，まず，　p　の増加につれて，ブロードキャストや入れ替えの時間が増加しているという点がある。

これはプロセッサ間のネットワークで競合が発生したのが原因だと考えられる。もう1つは，　p　の増加によって収束までの反復回数も増加しているという点が挙げられる。実際には，アルゴリズム3の8行目の反復は，　$p=$　4，16，64，256，1024の時，それぞれ5，6，7，8，10となる。これより，反復回数は　$O(\log_2\sqrt{p})$　で増加していると考えられる。これらの考察を基に考えられる今後の改良点を次項に示す。

３．３ 改良点

前項における考察を踏まえ，今後出来る改良点については3点挙げられる。

３．３．１　$\tilde{A}$ の対角化の時間の短縮

Fig. 3.1より，　$p=1024$　の時，　$\tilde{A}$　の対角化は全体の実行時間のおよそ半分を占めていることが分かる。この実装では，対角化にQR法を元にしたLAPACKのdsyevを利用している。これをより高速な分割統治法やMR³法に変えることで，実行時間の短縮を図る。他にも並列化向きの対角化の方法は存在するが，目標とする問題の例として，　$\tilde{A}$　のサイズが100のような小さい場合，並列化をしても大幅な高速化は見込めない。別の方法として，対角化にヤコビ法を利用するということも考えられる。この方法は，演算量が大きくなる代わりに，高い並列性を持つ。今後の研究課題である。

３．３．２ ブロードキャスト及び入れ替えの時間の短縮

前節より，　p　が増えるにつれて，ブロードキャストや入れ替えの時間が増加していることが分かる。これは，実装の際に通信の衝突が存在しているということを示唆する。この衝突はネットワークトポロジーを考慮して，プロセッサに部分行列を割り当てることによって回避できると考えられる。

３．３．３ 収束の加速

ブロックヤコビ法の収束は前処理にQR分解を用いることで高速化出来るという研究がある[3][8]。また，非対角ブロックをフロベニウスノルムの大きな順に消去する動的オーダリング[2][8]は，収束の高速化に有用な方法となるだろう。

４ まとめ

本研究では，ブロックヤコビ法を用いて中規模の固有値問題を超並列環境で解く手法について考察した。従来の１次元分割法を用いた場合と提案手法である2次元分割法を用いた場合で，それぞれ並列

化向けブロックヤコビ法を比較し，プロセス数が大きい場合，提案手法は通信量及び通信回数の点で有用であるということが分かった。1次元分割法と比べて，2次元分割法は $\sqrt{p}$ プロセスしか利用できない部分もあるが，解析によって，この部分の影響で弱スケーラビリティが悪化することはないことを示した。この解析は，東京大学のスーパーコンピューターT2Kで1024コアまでを利用した性能評価によって裏付けを得た。また，3つの視点からプログラムの性能を上げるための方法を検討した。本研究で使用したプログラムは，スーパーコンピューター「京」に移植済みであり，今後，更に評価と改良を行って，1万コア以上を活用できる超並列固有値ソルバを実用化したいと考えている。

参 考 文 献

[1] E Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen. *LAPACK Users Guide.* SIAM, 1992.

[2] M. Becka, G. Oksa and M. Vajtersic. Dynamic ordering for a parallel block Jacobi SVD algorithm. *Parallel Computing*, 28:243-262, 2002

[3] M. Becka, G. Oksa, M. Vajtersic and R. Grigori. On iterative QR preprocessing in the parallel blockJacobi SVD algorithm. *Parallel Computing*, 36:297-307, 2010

[4] C. H. Bischof. *The two-sided block Jacobi method on a hypercube*, pages 612-618. SIAM, 1988.

[5] 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.* SIAM, 1997.

[6] R. P. Brent and F. T. Luk. The solution of the singular value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. Sci. Statist. Comput*., 6:69-84, 1985.

[7] G. H. Golub and C. F. van Loan. *Matrix Computations.* Johns Hopkins University Press, third edition, 1996.

[8] G. Oksa and M. Vajtersic. Ecient preprocessing in the parallel blockJacobi SVD algorithm. *Parallel Computing*, 32:166-176, 2006.

[9] M. Vajtersic and M. Becka. BlockSVD algorithms and their adaptation to hypercubes and rings. In N. Mirenkov, Q.-P. Gu, S. Peng and S. Sedukhin, editors, *Proceedings of the nd Aizu International Symposium on Parallel Algorithms / Architecture Synthesis,* pages 175-181. IEEE Computer Society Press, 1997.

[10] B. B. Zhou and R. P. Brent. A parallel ring ordering algorithm for ecient onesided Jacobi SVD computations. *J. of Parallel and Distributed Computing, 42:1-10, 1997.*