対称三重対角固有値ソルバにおける多固有値多分法の開発[1]

東京大学　情報基盤センター　スーパーコンピューティング部門
特任准教授　片桐孝洋
E-mail: katagiri@cc.u-tokyo.ac.jp

1. はじめに

二分法（Bisection Method）は、対称三重対角行列の固有値を計算するために広く用いられている方法です。しかしながら、計算すべき固有値が密集している場合、計算時間が増加するという問題があります。これは、幅広く用いられている数値計算ライブラリ LAPACK の二分法のルーチン[1-5]においても、例外ではありません.

この問題を解決するため、二分法の並列化が強く要請されています。特に、共有メモリ型並列計算機での並列化方式が、多くのスーパーコンピュータの計算ノードが共有メモリ型並列計算機となっているため、特に重要な技術となっています。

本稿では、複数固有値を計算するための二分法における新並列化方式を提案することを目的としています。対象は共有メモリ型並列計算機です。この手法の主なアイデアは、複数固有値計算に必要な処理の自然な並列性と、固有値の存在区間における複数の探索点の並列性の２並列性を利用し、性能を極限まで抽出することにあります。我々はこの新手法を、**多固有値多分(Multi-section with Multiple Eigenvalues method、MME)**と呼んでいます。

二分法を並列化するため、複数の探索点を利用することは新しいアイデアではありません。この方法は、多分法（Multisection Method）と呼ばれています。たとえば、Lo ら[6]や Simon[7]の論文で、多分法の提案がなされています。しかしながらこれらの手法は、ベクトル型計算機を対象とした上で、複数の探索点を利用しベクトル化を行うものでした。ベクトル化をするため、演算中心となるループ(カーネル・ループ)において、IF 分がその中心に位置されています。

これら従来手法に対し我々が提案する手法は、多数のプロセッサ・エレメント（PE）を有する共有メモリ型並列計算機によるスレッド実行を意識し、カーネル・ループにおける最外ループの自然な並列性に着目するものです。加えて、複数固有値を計算する際の並列性も利用します。この２並列性を利用することで、最外ループの長さ（ループ長）を従来手法に対して長く保つことができます。それにとどまらず、探索点が少数しかとれない場合[2]においても、最外ループ長を長く保てるという長所があります。これらの特徴はいずれも、共有メモリ型並列計算機において高並列効率を達成するため、極めて重要な要素となります。現在のスーパーコンピュータでは、ノード内 PE 数は 16PE 程度が普通であり、今後さらにノード内 PE 数が増加していくという計算機アーキテクチャのトレンド[3]を考慮すると、ノード内並列性を少しでも高く保つ

---

[1] 本稿は、2006 年 2 月 27 日、北海道大学で開催された第 13 回「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ（HOKKE-2006）において発表された、「Multi-section with Multiple Eigenvalues Method for Computing Eigenvalues in Symmetric Tridiagonal Eigensolvers」(Takahiro Katagiri, Christof Voemel, James Demmel)の原稿をもとに、加筆修正を行ったものです。

[2] 探索点を多くすると並列性が増すのですが、深い探索を必要としない固有値分布のとき、余分な計算が増すことで、並列化の効果がなくなることがあります。つまり、最適な探索点数は固有値分布に依存するので、従来法では最外ループ長を長く保てない場合があります。

[3] マルチコアとか、メニーコアとか言われている高並列環境のことです。

ことが、高性能計算で決定的要因となることがお分かりいただけると思います。

この原稿は、以下の構成となっております。2 章では、多固有値多分法の説明をします。3 章では、多固有値多分法の性能評価を、最新の LAPACK ルーチンで提供されている対象三重対角行列固有値ソルバで使われている MRRR アルゴリズムをベンチマークとして利用し、HITACHI SR8000 を用いて性能評価を行います。このベンチマークとは、LAPACK3.1.1 で提供されている xGTEGR ルーチンのことです。最後に、本稿での知見を述べます。

2. 多固有値多分法の演算カーネル

2.1 LAPACK dlarrb ルーチン

ここで示される二分法の実装は、LAPACK 3.1.1 の DSTEGR ルーチン中で呼び出されている二分法を用いた固有値計算ルーチンの dlarrb ルーチン(dlarrb.f)を基にしています。具体的に、図 1 は dlarrb ルーチンで使われる二分法の演算カーネルである dlaneg 関数（dlaneg.f）の概略を示したものです。

```
< 0 >  S = 0; P=0; NEG1=0; NEG2=0; NEGCNT=0;
< 1 > I) 上三角行列部分
< 2 > do J = 1, R-1
< 3 >   T = S - σ
< 4 >   DPLUS = D( J ) + T
< 5 >   S = T * LLD( J ) / DPLUS
< 6 >  if ( DPLUS .lt. ZERO ) NEG1 = NEG1 + 1
< 7 >  enddo
< 8 >  if (S .eq. NaN) 上記ループより遅いバージョンを使う;
< 9 >  NEGCNT = NEGCNT + NEG1
< 10 > II) 下三角行列部分
< 11 > do J = N-1, R, -1
< 12 >   DMINUS = LLD( J ) + P
< 13 >   P = P*D( J ) / DMINUS - σ
< 14 >   if ( DMINUS .lt. ZERO ) NEG2 = NEG2 + 1
< 15 > enddo
< 16 > if (P .eq. NaN) 上記ループより遅いバージョンを使う;
< 17 > NEGCNT = NEGCNT + NEG2
< 18 > III) ねじれ部分
< 19 > GAMMA = S + P
< 20 > if (GAMMA .lt. ZERO) NEGCNT = NEGCNT+1
< 21 > return (NEGCNT)
```

図 1　LAPACK 3.1.1 dSTEGR における二分法の演算カーネル。変数 R は、三重対角行列 T のねじれ分解における「ねじれ点」を意味している。

図 1 のカーネルでは、σ より小さな固有値数を返します。ここで図 1 のカーネルでは、ターゲットとなる固有値１つについて、探索点σ以下の固有値数が計算される仕様となっています。また MRRR アルゴリズムの特徴から、dlarrb ルーチンで計算される固有値数は、入力された三

重対角行列に対する Relatively Robust Representation (RRR) [3]という形式を基に決まります。σの値は、対象となる固有値の存在範囲から決まります。この存在範囲は、MRRR アルゴリズムの場合、入力された三重対角行列に対する Representation Tree[3]で決定されます。

dlarrb ルーチンでは、MRRR アルゴリズムで必要となるＬＤＬ^Ｔ分解に「特化された」二分法を実行します。すなわち、以下の３つの部分があります：
I. ＬＤＬ^Ｔ － σ Ｉ = L_+ D_+ L_+^T に関する「上三角行列部分」
II. ＬＤＬ^Ｔ － σ Ｉ = U_- D_- U_-^T に関する「下三角行列部分」
III. ねじれ部分

また高速化のため、NaN などを考慮した IEEE-754 浮動小数点形式に特化した処理[8]が実装されています。

図 1 のσの値は、探索中の固有値に対する二分探索の基準点です。このσは、現在の固有値存在区間を[a,b]とした場合、a+(b-a)/2 で計算されます。

我々は、二分法のカーネルを、I 部分の< 2 >～< 7 >行と思うことにします。なぜなら、II 部分においても、計算に必要な変数のアクセスパターンはＩ部分と同様ですから。

2.2 二分法カーネルはなぜ並列化できないか？

図 2 は、いま対象としている一つの固有値に対する二分法のカーネルとします。

```
< 0 >   S=0; NEG1=0;
< 1 >   do J = 1, R-1
< 2 >     T = S - σ
< 3 >     DPLUS = D( J ) + T
< 4 >     S = T*LLD( J ) / DPLUS
< 5 >     if ( DPLUS .lt. ZERO ) NEG1 = NEG1 + 1
< 6 >   enddo
```

図 2　二分法のカーネル

図 2 のカーネルは、そのままでは並列化ができません。これは、変数 S について、<4>行で定義された値が J-ループをまたいで、<2>行で参照されるためです。このようなデータの依存関係を、**ループ伝搬フロー依存 (loop-carried flow dependency)** と呼びます。

図 2 のループを並列化するためには、実は、いろいろな方法があります。二分法については、2 種の方法が知られています。

まずはじめの方法は、二分法で用いる固有値の存在区間を分割し、ぞれぞれについて並列に探索していく方法です[9]。しかしながらこの方法では、図 2 のカーネル自体が並列化されません。さらにベクトル型計算機においてはカーネルがベクトル化できないので、演算効率は悪くなります。

もう一つの方法は、並列接頭辞法（parallel prefix method）という方法を用いて、固有方程式 p_k(x) = det(T_k - x I) に関する＜二分木状の＞処理を並列化する方法です[10]。
この方法は並列性を抽出でき、さらにベクトル計算機においてもベクトル化ができるようになるのですが、残念なことに数値安定性が悪く、固有値数のカウントを誤ることがあります。したがって実用上、使われません。

2.3 多分法のカーネル

図 2 のカーネル自体を並列化する方法は、多分法を用いることです。図 3 に、対象の一つの固有値を計算する際に ML 点の探索点を使う場合における、多分法のカーネルを示します。

```
< 0 >   S(1:ML)=0; NEG1(1:ML)=0;
< 1 >   do I = 1, ML
< 2 >     do J = 1, R-1
< 3 >       T( I ) = S( I ) - σ( I )
< 4 >       DPLUS( I ) = D( J ) + T( I )
< 5 >       S( I ) = T( I )*LLD( J ) / DPLUS( I )
< 6 >       if ( DPLUS( I ) .lt. ZERO ) NEG1( I ) = NEG1( I ) + 1
< 7 >     enddo
< 8 >   enddo
```

図 3 多分法のカーネル

図 3 の配列σ(1:ML) には、現在の固有値存在区間を[a,b]とすると、σ(i) = a + h * i (i=1,2,...,ML)、 h は (b-a)/(ML+1)と定義したもの、が設定されます。
図 3 のカーネルは、最外ループである I ループについて、並列化ができることに注意してください。なぜなら、ループ変数 I に関連する変数（配列）は、まったくデータ依存がないからです。これは、明らかですね。

しかしながら、多分法のカーネルにも問題があります。もし並列度を上げるために、最外ループ I の長さ（ループ長）を長くしたいとします。このとき、図 3 の探索点数 ML を、できるだけ大きく設定しなくてはなりません。ところが ML に大きな値を設定すると、今度は二分法の探索効率が悪くなる場合が出てきてしまいます。なぜかというと、もし従来の二分法において探索１回で見つかる固有値分布の場合、多分法で探索点を増やすと、探索点を増やした分だけ余分な計算をすることになり、結果として二分法に対し実行時間が増えてしまいます。

つまり多分法には、**〈並列化効率〉と〈探索効率〉のドレードオフ** があります。

本稿で提案する方法は、この多分法のトレードオフを解消するものです。つまり、多分法の探索点数を低く抑えつつも、同時計算する固有値の計算処理の並列性を使うことで、高速実行と高探索効率を同時に実現する方式です。我々はこの新しい方式を、**多固有値多分法（Multi-section with Multiple Eigenvalues method、MME 法）**と呼びます。

2.4 多固有値多分法のカーネル

図 4 に、多固有値多分法のカーネルを載せます。

```
< 0 > S(1:ML,1:EL)=0; NEG1(1:ML,1:EL)=0;
< 1 > do K = 1, EL
< 2 >   do I = 1, ML
< 3 >     do J = 1, R-1
< 4 >       T( I, K ) = S( I, K ) - σ( I, K )
< 5 >       DPLUS( I, K ) = D( J ) + T( I, K )
< 6 >       S( I, K ) = T( I, K )*LLD( J ) / DPLUS( I, K )
< 7 >       if ( DPLUS( I, K ) .lt. ZERO ) NEG1( I, K ) = NEG1( I, K ) + 1
< 8 >     enddo
< 9 >   enddo
< 10 > enddo
```

図 4　多固有値多分法のカーネル

図 4 の配列σ(1:ML, 1:EL) には、k 番目の固有値の存在区間を[a_k,b_k]とするとき、σ(i,k) = a_k + h_k * i (i=1,2,...,ML)で、h_k は (b_k - a_k)/(ML+1)で定義される値、が設定されます。

図 4 の K ループと I ループは、融合することができます。図 5 に、融合した場合のカーネルを載せます。

```
< 0 >   S(1:ML*EL) = 0; NEG1(1:ML*EL) = 0;
< 1 >   do I = 1, ML*EL
< 2 >     do J = 1, R-1
< 3 >       T( I ) = S( I ) - σ( I )
< 4 >       DPLUS( I ) = D( J ) + T( I )
< 5 >       S( I ) = T( I )*LLD( J ) / DPLUS( I )
< 6 >       if ( DPLUS( I ) .lt. ZERO ) NEG1( I ) = NEG1( I ) + 1
< 7 >     enddo
< 8 >   enddo
```

図 5　ループ融合した多固有値多分法のカーネル

図 5 のループ融合した多固有値多分法のカーネルは、多分法に対して以下の利点があります。

第一に、最外の I ループのループ長は、多分法に対し並列処理のオーバヘッドを隠ぺいできるほど、長く取ることができます。なぜなら、普通の多分法のループ長である ML に対して、EL 倍長くすることができるからです。

次に、探索点数 ML を小さく取っても、最外ループ長 EL を適切に設定することによって、長く保てる可能性があります。このことは先述のとおり、探索効率を高く保てることを意味しています。

しかしながら、欠点もあります。それは明らかなことですが、もし同時に計算できる固有値が上位のルーチン（本稿での対象の場合は dlarrb ルーチン）中にないとき、多固有値多分法は多分法と同じになります。

2.5 多固有値多分法の処理の概略

図 6 は、多固有値多分法の処理の概略を示しています。

```
< 1 > if (固有値数 .gt. 1) then
< 2 >    多固有値多分法ルーチンを以下の条件で呼ぶ:
           EL = 固有値数; ML = 適切な値;
< 3 > else
< 4 >    普通の多分法のルーチンを以下の条件で呼ぶ:
           ML = 適切な値;
< 5 > endif
```

図 6 多固有値多分法の概略

ここで、図 6 に現れる同時に計算可能な固有値数は、たとえ入力行列に対するすべての固有値計算が必要な場合でも、事前にわからないことに注意してください。なぜなら一般に、二分法ルーチンで計算される固有値数は、二分法ルーチンを呼び出す上位のルーチンで用いられるアルゴリズムに強く依存するからです。ここでは、MRRR アルゴリズムに依存するので、入力行列に対する Representation Tree によって決定される固有値数で決まります。

したがってパラメタの EL と ML に対する最適な値は、ルーチン実行前に決定できません。より適切なパラメタの設定は、何らかの実行時最適化が必要になります[4]。

図 7 は、LAPACK の dlarrb ルーチンにおける多固有値多分法の詳細を示しています。

図 7 では、固有値計算に関して、以下の３つの部分があります:

I. 左側固有値存在区間 LEFT_j に関する計算部分 (< 3 >行～< 9 >行)

II. 右側固有値存在区間 RIGHT_j に関する計算部分 (< 10 >行～< 16 >行)

III. 全固有値存在区間 [LEFT_j,RIGHT_j]に対する、精度改善のための計算部分 (< 17 >行～< 23 >行)

なお、 j = J, J+1, ..., J+EL-1 です。

もし、固有値がとても密集している場合、< 17 >行～< 23 >行の III の部分が、時間を要する部分となります。

3. 性能評価

3.1 計算機環境

ここでは、スーパーコンピュータ HITACHI SR8000 (2 ノード/16PE モデル)を利用し、多固有値多分法の性能評価を行います。この計算機の詳細は以下のとおりです。

- 8PE / 1 ノード
- ジョブタイプ:E8E。これは、1 ノードを 8PE で独占するジョブであることを示しています。
- コンパイラ:HITACHI OFORT90 V01-04-/B。
- コンパイラオプション:-O4 -parallel=4。これは、ノード内自動並列化を行うことを意味しています。
- 時間計測関数:xclock。これは、日立により提供されている高精度タイマーです。

[4] 我々は、この EL と ML の実行時最適化方式 (実行時自動チューニング方式) について、動的計画法を用いた方法を提案しています[14]。

```
< 1 >  do J = 1, 固有値数, EL
< 2 >     j 番目の固有値 (j=J,...,J+EL-1)の存在区間[LEFT_j,RIGHT_j] の確認;

< 3 >     I) L_+ D_+ L_+^T = L D L^T - LEFT_j  から  NEGCNT_j  を計算する ;
< 4 >     while (すべての固有値の存在区間が十分小さい)
< 5 >       現在の区間 LEFT_j から、σ(1:EL) を設定する;
< 6 >       EL、ML を設定して多固有値多分法カーネルを呼ぶ;
< 7 >       σ(1:EL)の点から、戻り値を利用して区間 LEFT_j を修正する ;
< 8 >       すべての固有値の存在区間が十分小さいかチェックする;
< 9 >     endwhile

< 10 >    II) L_+ D_+ L_+^T = L D L^T - RIGHT_j から NEGCNT_j を計算する ;
< 11 >    while (すべての固有値の存在区間が十分小さい)
< 12 >      現在の区間 RIGHT_j から、σ(1:EL) を設定する;
< 13 >      EL、ML を設定して多固有値多分法カーネルを呼ぶ;
< 14 >      σ(1:EL)の点から、戻り値を利用して区間 RIGHT_j を修正する ;
< 15 >      すべての固有値の存在区間が十分小さいかチェックする;
< 16 >    endwhile

< 17 >  III) まだ収束していない区間がある
< 18 >  while (すべての固有値の存在区間が十分小さい .or. (反復回数 .gt. MAXITER))
< 19 >    現在の区間[LEFT_j, RIGHT_j]から、σ(1:EL) を設定する;
< 20 >    EL、ML を設定して多固有値多分法カーネルを呼ぶ;
< 21 >    σ(1:EL)の点から、戻り値を利用して区間 [LEFT_j,RIGHT_j]を修正する ;
< 22 >    すべての固有値の存在区間が十分小さいかチェックする;
< 23 >  endwhile

< 24 > enddo
< 25 > if (固有値数が EL で割り切れない)
          残りの固有値に対して、ML を設定して多分法カーネルを呼ぶ ;
```

図 7　多固有値多分法の詳細。これは、多固有値多分法を実装した LAPACK dlarrb ルーチンの詳細でもある。

3.2 ベンチマーク

性能評価のための入力行列について、以下の 4 種類の行列を用いました。

- 行列サイズ: 2100
- **行列 1:** (-1,2,-1) 行列
- **行列 2:** 0 ～ 1 の値で生成される一様乱数.
- **行列 3:** Wilkinson 行列 W^+_2100
- **行列 4:** 「副対角要素を循環シフトした」Glued Wilkinson Matrix W^+_21。
  この行列は以下のように定義されます: Diag(T) = Diag(Glued Wilkinson Matrix W^+_21)。
  Sub(T) は (δ, 1,..,1)_21 を 100 回繰り返したもの。glue value の δ は 1e-1。

表 1 は、dlarrb ルーチンが呼ばれたときにおける、それぞれのテスト行列の固有値分布です。

表 1　dlarrb ルーチンにおける固有値数の分布（DQDS モード）

(a)行列 1

固有値数	頻度[回]	全体に対する割合[%]
1	22	95.6
789	1	4.34

(b)行列 2

固有値数	頻度[回]	全体に対する割合[%]	固有値数	頻度[回]	全体に対する割合[%]
1	850	68.4	13	1	0.08
2	145	11.6	14	4	0.32
3	78	6.28	15	3	0.24
4	45	3.62	16	2	0.16
5	31	2.49	17	1	0.08
6	21	1.69	18	3	0.24
7	15	1.20	19	1	0.08
8	11	0.88	20	1	0.08
9	7	0.56	21	1	0.08
10	9	0.72	22	1	0.08
11	5	0.40	24	1	0.08
12	4	0.32	35	1	0.08

(c)行列 3

固有値数	頻度[回]	全体に対する割合[%]
1	2093	66.7
2	1043	33.2
102	1	0.31

(d)行列 4

固有値数	頻度[回]	全体に対する割合[%]
1	235	82.7
2	27	9.50
3	1	0.35
99	7	2.46
100	12	4.22
200	2	0.70

対象アプリケーションは、LAPACK 3.1.1 の DSTEGR ルーチンです。詳細は以下のとおりです。

- **対象ルーチン**：多固有値多分法を DSTEGR ルーチンの二分法ルーチンである dlarrb ルーチンに実装したもの。

- **対象計算**：全固有値および全固有ベクトルの計算

- **モード**：DSTGR には、以下の２つのモードがあります。
  - DQDS モード（LAPACK の DLASQ1[11]）
    - このモードでは、DQDS 法を固有値計算部分で利用します。また、固有値の精度を改良するため 、固有ベクトル計算部分で二分法を利用します。
  - 積極的な二分法モード
    - このモードでは、固有値計算部分および固有ベクトル計算部分の両方で、二分法を利用します。

  以上の２モードを、本性能評価で利用します。

- **測定対象**：多固有値多分法を実装した dlarrb ルーチンの総実行時間を計測します。

- **パラメタに関する「静的固定方式」**：EL と ML について、DSTEGR ルーチンが終了するまで固定値で実行する方式です。なお、この固定値は、以下の値について変動させて時間を計測します。最高速となるパラメタのものを、性能評価対象の各方式（多分法、多固有値多分法）の実行時間とします。
  - EL ∋ [1, 2, 3, 4, 8, 16, 32]: 7 種類
  - ML ∋ [1, 2, 4, 8, 16, 24] : 6 種類

  すなわち、EL * ML = 42 種類のうち最適なパラメタによる実行時間とします。

3.3 多分法に対する静的固定方式による多固有値多分法の効果

表 2 は、多分法に対する静的固定方式による多固有値多分法の効果を示しています。
表 2 より、多固有値多分法を利用することで、二分法に対して最大で 7.7 倍、多分法に対して最大で 4.3 倍の速度向上を得ました。

行列 1 と 4 に関して、多固有値多分法では 16 個の固有値同時計算が選択されました。この理由は、789 個もしくは 200 個の固有値を同時に求める呼び出しが dlarrb ルーチンでなされており、この処理の高速化が有効であったからです（表 1)。

一方で、行列 2 と 3 については、2 個の固有値同時計算が選ばれました。この理由は、行列 1 と 4 に比べ、delarrb ルーチンの多くの呼び出しが少数固有値を同時に計算する処理であることによります。たとえばこれらの行列では、多くても 35 個もしくは 102 個の固有値だけが 1 回だけ計算されます（表 1)。

表 2　多分法に対する静的固定方式による多固有値多分法の効果

(a) dlarrb ルーチンの総実行時間（DQDS モード）

方法／行列種類	行列１	行列２	行列３	行列４
二分法[秒]	0.347	1.83	15.2	8.20
多分法[秒]	0.323	1.45	5.90	3.30
最適 ML	8	8	16	16
多固有値多分法[秒]	0.075	1.16	5.34	1.75
最適(EL, ML)	(16,2)	(2,8)	(2,8)	(16,1)
二分法に対する速度向上	4.6x	1.5x	2.8x	4.6x
多分法に対する速度向上	4.3x	1.2x	1.1x	1.8x

(b) dlarrb ルーチンの総実行時間（積極的な二分法モード）

方法／行列種類	行列１	行列２	行列３	行列４
二分法[秒]	3.94	7.11	23.9	17.2
多分法[秒]	2.11	3.69	8.70	6.19
最適 ML	16	16	16	16
多固有値多分法[秒]	0.51	2.93	7.89	2.75
最適(EL, ML)	(16,1)	(2,8)	(2,8)	(16,1)
二分法に対する速度向上	7.7x	2.4x	3.0x	6.2x
多分法に対する速度向上	4.1x	1.2x	1.1x	2.2x

4.　おわりに

本稿では対称三重対角固有値ソルバにおける固有値計算において、新しい実装方式を提案しました。この新方式の鍵となるアイデアは、複数固有値計算の自然な並列性と、多分法における一つの固有値計算に必要な区間における探索点処理の並列性の２つを同時利用することにあります。この方式を、多固有値多分法と命名しました。本稿では、LAPACK 3.1.1 の DSTEGR ルーチン中の dlarrb ルーチンに多固有値多分法を実装しました。性能評価の結果、従来方式に対し、1 ノードあたり 8PE の HITACHI SR8000 において、最大で 7.7 倍の速度向上を得ました。本手法は、DSTEGR ルーチンに特化した方法ではなく、汎用的なルーチンとして利用可能であることに注意してください。

この原稿で示した、多固有値多分法のパラメタに対する「静的固定方式」は、最適ではありません。なぜなら、同時に計算する固有値数は dlarrb ルーチンが呼ばれるたびに変化するからです。我々は「動的固定方式」と呼ぶ、実行時パラメタチューニング方式を提案しており、静的固定方式に対して、さらなる速度向上が得られることを確認しています[13]。

多固有値多分法の並列化の効果は、プロセッサ台数を増すとさらに増すように思えます。今後の重要な課題は、より PE 数（コア数）の多い並列計算機で多固有値多分法の並列化の効果を検証することです。

謝辞

本研究を遂行するにあたり、米国カリフォルニア大学バークレー校の James Demmel 教授、および米国ローレンスバークレー国立研究所の Cristof Voemel 博士らにご協力を頂きました。ここに感謝の意を表します。

参考文献

[1] Parlett, B.N. and Dhillon, I.S.: Fernando's Solution to Wilkinson's Problem: An Application of Double Factorization, Linear Algebra and Appl.,Vol.267, pp.247-279 (1997).

[2] Parlett, B.N. and Dhillon, I.S.: Relatively Robust Representations of Symmetric Tridiagonals, Linear Algebra and Appl., Vol.309, pp.121-151 (2000).

[3] Parlett, B.N. and Dhillon, I.S.: Multiple Representations to Compute Orthogonal Eigenvectors of Symmetric Tridiagonal Matrices, Linear Algebra and Appl.,Vol.387, pp.1-28 (2004).

[4] Parlett, B.N. and Dhillon, I.S.: Orthogonal Eigenvectors and Relative Gaps, SIAM J. Matrix Anal. Appl., Vol.25, No.3, pp.858-899 (2004).

[5] Dhillon, I.S., Parlett, B.N. and Voemel, C.: LAPACK Working Note 162: The Design and Implementation of the MRRR Algorithm, Technical Report UCBCSD-04-1346, University of California, Berkeley (2004).

[6] Lo, S.-S., Philippe, B. and Sameh, A.: A Multiprocessor Algorithm for the Symmetric Tridiagonal Eigenvalue Problem, SIAM J. Sci. Stat. Comput., Vol.8, No.2, pp.s155-s165 (1987).

[7] Simon, H.D.: Bisection Is Not Optimal on vector Processors, SIAM J. Sci. Stat. Comput., Vol.10, No.1, pp.205-209 (1989).

[8] Marques, O.A., Riedy, J. and Voemel, C.: Benefits of IEEE-754 features in Modern Symmetric Tridiagonal Eigensolvers, Technical Report UCBCSD-05-1414, University of California, Berkeley (2005).

[9] Demmel, J.W., Dhillon, I. and Ren, H.: On the Correctness of Parallel Bisection in Floating Point, Technical Report UCB/CSD-94-805, University of California, Berkeley (1994).

[10] Ren, H.: On the Error Analysis and Implementation of Some Eigenvalue Decomposition and Singular Value Decomposition Algorithms, Technical Report UT-CS-96-336, University of California, Berkeley (1996).

[11] Parlett, B.N. and Marques, O.: An Implementation of The DQDS Algorithm (Positive Case),Linear Algebra and Appl., Vol.309, pp.217-259 (2000).

[12] Blackford, L.S., Cleary, A., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Petitet, A., Ren, H., Stanley, K. and Whaley, R.C.: Practical Experience in the Dangers of Heterogeneous Computing, Technical Report UT-CS-96-330, University of California, Berkeley (1996).

[13] Katagiri,T., Voemel,C., Demmel, J.W.:Effect on Run-time Auto-tuning for Multi-section with Multiple Eigenvalues Method,2006 年並列／分散／協調処理に関する『高知』サマー・ワークショップ (SWoPP2006)、高知商工会館、情報処理学会研究報告 2006－HPC－107、pp.187-192 (2006)