Vol.2013-ARC-207 No.20
Vol.2013-HPC-142 No.20
2013/12/17

再直交化付きブロック逆反復法のライブラリ利用に基づくGPU実装についての検討

石上 裕之[1,2,a)]　木村 欣司[1,b)]　中村 佳正[1,c)]

概要：再直交化付きブロック逆反復法は，著者らによって提案された，行列乗算によって大半の計算を行うことのできる固有ベクトル計算法である．本論文では，数値計算ライブラリ CULA が提供するルーチンを用いた，同時逆反復法および再直交化付きブロック逆反復法の GPU 実装について検討を行う．実対称 3 重対角行列に対する数値実験により，実装コードの性能を評価する．

1. はじめに

$n \times n$ 実対称密行列 A に対する標準固有対問題

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i, \quad i = 1, \ldots, n$$

を考える．ここで，$\lambda_i \in \mathbb{R}$ は A の固有値 ($\lambda_1 < \cdots < \lambda_n$)，$\boldsymbol{v}_i \in \mathbb{R}^n$ は λ_i に対応する固有ベクトルである．実対称密行列の固有対計算は多くの科学技術計算で現れる基本的な線形計算である．振動解析における固有モード計算や蛋白質などの構造解析で用いられる分子軌道計算では，全固有対ではなく，一部の固有対のみが必要とされる．以下，このような問題を部分固有対問題と呼ぶ．

近年，これらのシミュレーションは大規模かつ複雑なものになり，部分固有対問題の対象となる行列も大規模化している．大規模な問題を高速に解く方法の一つとして，GPU や MIC アーキテクチャといったアクセラレータを用いた並列計算が注目を浴びている．これらアクセラレータは次世代スーパーコンピュータに不可欠なものと考えられており，事実，近年の TOP500 上位のスーパーコンピュータには，GPU や MIC アーキテクチャを搭載したものが多く存在する．このような時代背景から，部分固有対問題のソルバについて，アクセラレータの性能を十分に引き出す計算手法を確立することは重要な課題と言える．本研究では，マルチコア CPU と GPU で構成される計算機環境（以下，CPU-GPU ヘテロジニアス環境，と呼ぶ）上での並列計算について考える．

一般に，実対称行列の固有対計算は，前処理として直交変換によって 3 重対角化した後，3 重対角行列の固有対を計算する．ここで，元の行列の固有ベクトルは，3 重対角化の逆変換を施すことによって得られる．これらの手法については，CPU-GPU ヘテロジニアス環境上で高い性能の得られるアルゴリズムが提案されている [3], [17].

実対称 3 重対角行列の一部の固有対のみを計算する手法として，本研究では，2 分法と逆反復法を組み合わせた方法 [14] に着目する．この方法では，求めたい固有値を 2 分法により計算し，得られた固有値を基に，対応する固有ベクトルを逆反復法によって計算する．2 分法による固有値計算は，並列化が容易であることが知られており，CPU-GPU ヘテロジニアス環境上においても効果的な方法が提案されている [18].

その一方で，逆反復法はマルチコア CPU 上においても効果的な並列化が難しく，GPU による高速化については効果的な手法は提案されていなかった．これは，逆反復法のクラスター固有値に対応する固有ベクトル計算において，ベクトル演算中心の再直交化計算を行っており，並列化による高速化のボトルネックとなっていたからである．この問題に対して，著者らは逆反復法の効果的な GPU 実装を提案している [11]．この GPU 実装では，ベクトル演算よりも GPU による演算性能の向上が期待できる，行列-ベクトル乗算を用いた再直交化アルゴリズムを採用し，再直交化計算のみを GPU で計算するという手法をとっている．これに加え，CPU-GPU 間のデータ転送量を削減することで，高速な GPU 実装を実現している．

しかしながら，上記の結果は単体の GPU を搭載した計算機 1 ノードにおける成果である．複数の GPU を搭載したノードで構成されるシステムにおいては，行列-ベクトル乗算中心のアルゴリズムでは性能をうまく引き出すことが

[1] 京都大学大学院情報学研究科
[2] 日本学術振興会特別研究員（DC）
[a)] hishigami@amp.i.kyoto-u.ac.jp
[b)] kkimur@amp.i.kyoto-u.ac.jp
[c)] ynaka@i.kyoto-u.ac.jp

できないと予想される．そこで，キャッシュヒット率や並列化効率の点で行列-ベクトル乗算よりも計算機の性能を引き出せる，行列乗算中心のアルゴリズムであれば，そのようなシステム上でも計算機の性能をうまく引き出せると期待できる．

本研究では，行列乗算を用いた固有ベクトル計算法である著者らが提案した，再直交化付きブロック逆反復法 [12] の GPU 実装について検討を行う．また，再直交化付きブロック逆反復法の基のアイディアである同時逆反復法の GPU 実装についても見当を行う．ここで，GPU 実装については，数値計算ライブラリ CULA を利用したものについて考える．本論文の構成は以下の通りである．2 章では，GPU 実装を用いた数値計算について述べる．3 章では，本論文で GPU 実装の対象とする，同時逆反復法および再直交化付きブロック逆反復法を導入する．4 章では，再直交化付きブロック逆反復法の CULA のライブラリ関数を利用した実装法を示す．5 章では，マルチコア CPU と GPU で構成される計算機上において数値実験を行い，提案実装の性能評価を行う．また，性能評価の結果について，考察を加える．6 章はまとめである．

2. GPU を用いた数値計算

GPU を用いた並列計算は，NVIDIA 社が提供する CUDA[13] などの開発環境を利用することで実現できる．しかしながら，CUDA プログラミングにおいては，GPU の特性に注意したプログラミングを行わなければ，高速に計算を行えるコードを作成するのは難しい．このことは，マルチコア CPU 上での並列計算にも言える．

一方，多くの数値線形計算アルゴリズムは，LAPACK（Linear Algebra PACKage）[1] のようなライブラリに実装されている．また，行列-ベクトル乗算や行列乗算といった基本線形演算は BLAS（Basic Linear Algebra Subroutines）と呼ばれ，LAPACK は下位ルーチンに BLAS を利用している．BLAS や LAPACK ルーチンは，各種プロセッサ向けにチューニングされたライブラリが多くのベンダーによって開発されている．これらライブラリで提供されているルーチンを利用することで，高速に計算のできるコードをユーザが実装することが可能となる．本研究では，このようなライブラリとして，Intel Math Kernel Library（以下，Intel MKL）[9] と CULA[8] を用いた実装について考える．

Intel MKL は，マルチコア CPU 向けにチューニングされた BLAS および LAPACK ルーチンを提供しており，その多くのルーチンがスレッド並列化されている．同様に，CULA[8] は GPU 向けにチューニングされた BLAS および LAPACK ルーチンを提供している．CULA では，Intel MKL との組み合わせにより，マルチコア CPU と GPU 両方を用いた協調的な並列計算を行う実装ルーチンを提供している．このような協調計算を行う場合，本来であればマルチコア CPU と GPU にどのように計算を割り当てるかという負荷分散を考える必要があるが，CULA ではルーチン内部において自動的に負荷分散を行う．これにより，容易に高性能な協調演算を実行することが出来る．これは，CULA に実装されている LAPACK ルーチンのみならず，一部の BLAS についても同様である．

最後に，CULA の提供するルーチンを利用する際の注意点について述べる．GPU プログラミングにおける大きな特徴の一つとして，メインメモリと GPU のメモリが独立していることが挙げられる．このため，GPU 上で計算を行うには GPU のメモリにデータを転送する必要がある．CULA の提供するルーチンでは，メインメモリのデータを参照して必要なデータを GPU のメモリ側に転送し，GPU での計算終了後にはメインメモリにデータを転送するように実装されている．しかし，CPU と GPU 間のデータ転送の速度は，現代のプロセッサの浮動小数点計算に比べて低速であり，データ転送の量や回数が多ければ多いほどプログラムの実行時間は遅延することとなる．したがって，CULA の提供するルーチンを利用する際には，データ転送を自動化できる一方，データ転送によって実行時間が遅延する可能性がある．

3. 固有ベクトルの計算アルゴリズムとその実装

本研究では，n 次元実対称 3 重対角行列 T の固有ベクトル計算について考える．ここで，T の固有値を $\lambda_j \in \mathbb{R}$（$\lambda_1 < \lambda_2 < \cdots < \lambda_n$）とし，$\boldsymbol{q}_j \in \mathbb{R}^n$ を λ_j に対応する T の固有ベクトルとする．以下では，逆反復法と同時逆反復法，それらを基に著者らが提案した再直交化付きブロック逆反復法について説明する．

本章に記述する各アルゴリズムの擬似コードは，[12] におけるものと全く同じであるが，擬似コード中の QR 分解について異なる実装を利用している．これについては，3.4 章に詳細を記述している．

3.1 逆反復法

逆反復法では，λ_j の近似固有値 $\tilde{\lambda}_j$，一様乱数により生成した n 次元ベクトル $\boldsymbol{v}_j^{(0)}$ に対して，以下の連立方程式を反復して解くことによりベクトル $\boldsymbol{v}_j^{(i)}$ を得る．

$$\left(T - \tilde{\lambda}_j I\right)\boldsymbol{v}_j^{(i)} = \boldsymbol{v}_j^{(i-1)}, \quad i = 1, 2, \ldots, \tag{1}$$

ここで，I は n 次元単位行列である．T の固有値が互いに十分離れている場合，$i \to \infty$ において，$\boldsymbol{v}_j^{(i)}$ は固有ベクトル $\boldsymbol{q}_j$ に収束する．実装上では，計算誤差の影響を抑えるため，連立方程式の求解には部分ピボット選択付き LU 分解を利用する．

T の固有値が近接している場合には，式 (1) の求解による反復計算のみでは，固有ベクトルの直交性が失われてしまうことが知られている．この問題に対して，逆反復で得

られたベクトルを再直交化する手法が提案されている [10]. 以下では，近接している固有値をクラスターと呼ぶ.

本研究では，Peters-Wilkinson の判定基準 [15] をクラスターの判定として用いる．この判定基準は, LAPACK の実対称 3 重対角行列向け逆反復法ルーチン xSTEIN[10] においても採用されている．この判定基準では，$|\tilde{\lambda}_{j-1} - \tilde{\lambda}_j| \leq 10^{-3}\|T\|$ $(2 \leq j \leq m)$ を満たすとき，λ_{j-1} と λ_j は同じクラスターに属するとみなす．Peters-Wilkinson の判定基準を用いた場合，数千以上のサイズの行列の固有値の大半が一つの固有値クラスターに属してしまうことが知られている [6]．このため，逆反復法による固有値計算を行う場合には，固有値クラスターに着目した並列化をするのではなく，反復内の演算においての並列化が前提となる．

3.2 同時逆反復法

逆反復法における再直交化計算は，ベクトル演算や行列-ベクトル乗算の実装となるため，計算ユニットの多い並列計算では性能向上に限界がある．これに対して，同時逆反復法 [4] は，クラスター固有値に対応する固有ベクトルの計算に行列乗算を利用できるアルゴリズムである．[4] では，行列 T の重複固有値 σ に対する定式化が示されている．本研究では，行列 T のあるクラスター固有値 $\lambda_1, \ldots, \lambda_{m_c}$ に対して，それぞれの近似値 $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_{m_c}$ を用いて定式化したアルゴリズムを同時逆反復法とする.

同時逆反復法では，まず，$n \times m_c$ 行列 $V^{(0)}$ を乱数により生成し，QR 分解 $V^{(0)} := Q^{(0)}R^{(0)}$ により正規直交行列 $Q^{(0)}$ を計算する．$Q^{(0)}$ を初期行列とした以下の 2 式で表される計算を反復することで，$\lambda_1, \ldots, \lambda_{m_c}$ に対応する固有ベクトル $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{m_c}$ を計算することができる.

$$\left(T - \tilde{\lambda}_k I\right)\boldsymbol{v}_k^{(i-1)} = \boldsymbol{q}_k^{(i-1)},\ k = 1, \ldots, m_c, \tag{2}$$

$$V^{(i)} := Q^{(i)}R^{(i)}. \tag{3}$$

ここで，$V^{(i)} = \begin{bmatrix}\boldsymbol{v}_1^{(i)} & \cdots & \boldsymbol{v}_{m_c}^{(i)}\end{bmatrix}$，$Q^{(i)} = \begin{bmatrix}\boldsymbol{q}_1^{(i)} & \cdots & \boldsymbol{q}_{m_c}^{(i)}\end{bmatrix}$ $\boldsymbol{v}_k^{(i)}$ と $\boldsymbol{q}_k^{(i)}$ は，それぞれ $V^{(i)}$ と $Q^{(i)}$ の第 k 列ベクトルである．同時逆反復法は，式 (2) において，逆反復法における式 (1) で表される m_c 本の連立方程式の求解を行い，再直交化計算の代わりに式 (3) の QR 分解をすることにより固有ベクトル計算を行う.

ここで，連立方程式の求解における計算量は，$O(m_c n)$ である．この演算は，各 k について独立に計算できるため，同期なしの並列計算が可能である．一方，QR 分解の計算量は $O(m_c^2 n)$ であるが，行列乗算中心の実装が可能であるため，キャッシュヒット率や並列化効率の点で高性能な演算となる．したがって同時逆反復法を用いることで，逆反復法よりも高速な並列計算が期待できる.

以上をまとめると，同時逆反復法の擬似コードは Algorithm 1 のようになる．5 行目では $T - \tilde{\lambda}_j I$ の部分ピボット選択付き LU 分解を行っており，得られた P_j，L_j，U_j を用いて，12 行目で連立方程式の求解を行っている.

Algorithm 1 同時逆反復法（SInv）

```
1:  function SInv(T, λ̃_1, ..., λ̃_{m_c})
2:      Generate V^(0) = [v_1^(0) ... v_{m_c}^(0)]
3:      V^(0) := Q^(0) R^(0)                          ▷ QR 分解
4:      for k = 1, ..., m_c do
5:          T − λ̃_k I := P_k L_k U_k                  ▷ 部分ピボット選択付き LU 分解
6:      end for
7:      i := 0
8:      repeat
9:          i := i + 1
10:         for k = 1, ..., m_c do
11:             Normalize q_k^(i−1)
12:             Solve P_k L_k U_k v_k^(i) = q_k^(i−1)   ▷ 連立方程式の求解
13:         end for
14:         V^(i) := Q^(i) R^(i)                      ▷ QR 分解
15:     until converge
16:     return Q = [Q^(i)]
17: end function
```

Algorithm 2 再直交化付きブロック逆反復法（RBInv）

```
1:  function RBInv(T, r, λ̃_1, ..., λ̃_{m_c})
2:      for j = 1 to m_c/r do
3:          i := 0
4:          Generate V_{j,r}^(0) = [v_{(j−1)r+1}^(0) ... v_jr^(0)]
5:          V_{j,r} := Q_{j,r}^(0) R_{j,r}^(0)        ▷ QR 分解
6:          for k = (j − 1)r + 1, ..., jr do
7:              T − λ̃_k I := P_k L_k U_k              ▷ 部分ピボット選択付き LU 分解
8:          end for
9:          repeat
10:             i := i + 1
11:             for k = (j − 1)r + 1, ..., jr do
12:                 Solve P_k L_k U_k v_k^(i) = q_k^(i−1)   ▷ 連立方程式の求解
13:             end for
14:             V_{j,r}^(i) := V_{j,r}^(i) − Q_{(j−1)r} Q_{(j−1)r}^⊤ V_{j,r}   ▷ 行列乗算 ×2
15:             V_{j,r}^(i) := Q̄_{j,r}^(i) R_{j,r}^(i)   ▷ QR 分解
16:             Q̄_{j,r}^(i) := Q̄_{j,r}^(i) − Q_{(j−1)r} Q_{(j−1)r}^⊤ Q̄_{j,r}^(i)   ▷ 行列乗算 ×2
17:             Q̄_{j,r}^(i) := Q_{j,r}^(i) R_{j,r}^(i)   ▷ QR 分解
18:         until converge
19:         Q_jr = [Q_{(j−1)r}  Q_{j,r}^(i)]  (Q_r = [Q_{1,r}^(i)])
20:     end for
21:     return Q_{m_c} = [q_1 ... q_{m_c}]
22: end function
```

3.3 再直交化付きブロック逆反復法

再直交化付きブロック逆反復法は，同時逆反復法にブロックサイズパラメータ r を導入したアルゴリズムである．あるクラスター内の固有値の数を m_c 個とすると，r は $r \ll m_c$ を想定する．以下では簡単のため，r を m_c の約数とする.

再直交化付きブロック逆反復法では，ある m_c 本の固有ベクトルを r 本ごとのブロックとみなし，ブロック内の全ての固有ベクトルを計算した後，次のブロックについての固有ベクトル計算を行う．以下では，$(j-1)r$ 本の固有ベクトル $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{(j-1)r}$ の計算が終了し，$\boldsymbol{q}_{(j-1)r+1}, \ldots, \boldsymbol{q}_{jr}$ の r 本の

計算を行う場合について示す．ここで，$j = 1, 2, \ldots, m_c/r$ とする．

再直交化付きブロック逆反復法では，任意の $n \times r$ 行列 $V_{j,r}^{(0)}$ を乱数により生成し，QR 分解 $V_{j,r}^{(0)} := Q_{j,r}^{(0)} R_{j,r}^{(0)}$ により正規直交行列 $Q^{(0)}$ を得る．$Q^{(0)}$ を初期行列として次の 3 ステップから成る反復計算を行うことにより，$\lambda_{(j-1)r+1}, \ldots, \lambda_{jr}$ に対応する固有ベクトル $\boldsymbol{q}_{(j-1)r+1}, \ldots, \boldsymbol{q}_{jr}$ が得られる．尚，$V_{j,r}^{(i)} = \left[\boldsymbol{v}_{(j-1)r+1}^{(i)} \quad \cdots \quad \boldsymbol{v}_{jr}^{(i)}\right]$，$Q_{j,r}^{(i)} = \left[\boldsymbol{q}_{(j-1)r+1}^{(i)} \quad \cdots \quad \boldsymbol{q}_{jr}^{(i)}\right]$ とする．

(i) 以下の r 本の連立方程式を解く

$$\left(T - \tilde{\lambda}_k I\right) \boldsymbol{v}_k^{(i)} = \boldsymbol{q}_k^{(i-1)},\ k = (j-1)r + 1, \ldots, jr. \quad (4)$$

(ii) $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{(j-1)r}$ と直交するように $V_{j,r}^{(i)}$ を再直交化する

(iii) (ii) で得られたベクトルを並べた $n \times r$ 行列を QR 分解し，$Q_{j,r}^{(i)}$ とする

ここで，(i) は，式 (4) の連立方程式を r 個の異なる $\tilde{\lambda}_j$ について解くことに他ならない．また，(ii) と (iii) は，まとめて，ブロック再直交化計算という．ブロック再直交化計算のためのアルゴリズムとしては，BCGS（Block Classical Gram-Schmidt）法 [16] が提案されている．BCGS 法は行列乗算を中心とした実装が可能なアルゴリズムであるが，計算誤差の観点から，得られるベクトルの直交性を保証できない．これに対して，BCGS2 法 [2] は，直交性を高めるために BCGS 法を 2 回用いる方法である．本研究では，(ii) と (iii) の演算として BCGS2 法を適用する．

以上をまとめると，再直交化付きブロック逆反復法の擬似コードは Algorithm 2 のようになる．ここで，14 行目から 17 行目は BCGS2 法の演算である．

尚，再直交化付きブロック逆反復法は，$r = m_c$ のときに同時逆反復法，$r = 1$ のときに逆反復法と等価なアルゴリズムである．ここで，同時逆反復法では P_j, L_j, U_j, $\boldsymbol{v}_j^{(i)}$ それぞれについて m_c 要素分メモリに保持する必要があるが，再直交化付きブロック逆反復法では r 要素分で済む．Peters-Wilkinson の判定基準を用いた場合，数千以上のサイズの行列の固有値の大半が一つの固有値クラスターに属してしまうことから，大規模な行列に対しては，$m_c \sim n$ と考えられる．このとき $r \ll m_c$ ならば，再直交化付きブロック逆反復法は，同時逆反復法に比べて非常に少ないメモリ使用量で済む．

3.4 CPU コードの実装

5 章の数値実験で使用するマルチコア CPU 向けの実装コードについて説明する．同時逆反復法を実装したものを **SInv (CPU)**，再直交化付きブロック逆反復法を実装したものを **RBInv (CPU)** とする．SInv (CPU) では Algorithm 1，RBInv (CPU) では Algorithm 2 を各クラスターに適用している．このため，固有ベクトル計算を行う前に，2 分法などで得られた固有値を事前にクラスター分けるルーチンも実装している．また，どちらの実装もほとんどの演算を BLAS や LAPACK ルーチンを適用することで実現でき，以下でその適用方法について述べる．尚，反復の停止条件は，LAPACK の DSTEIN ルーチンと同じ条件を複数ベクトル向けに変更して使用した．

3.4.1 連立方程式の求解

連立方程式の求解の演算については，LAPACK の DLAGTF，DLAGTS を用いることができる．DLAGTF で部分ピボット選択付き LU 分解を行い，そこで得られたベクトル要素を用いて DLAGTS により連立方程式の解を計算する．これらの演算は，同時逆反復法では m_c 本，再直交化付きブロック逆反復法では r 本の連立方程式を独立に求解することができるため，これらのルーチンを各スレッドで並列に処理することで並列計算を行うことができる．本研究では，OpenMP の指示文を用いることで，この並列計算を実装した．

3.4.2 QR 分解および行列乗算

同時逆反復法では $n \times m_c$ の長方行列，再直交化付きブロック逆反復法では $n \times r$ の長方行列に対して QR 分解を行う．[12] では，QR 分解には古典グラム・シュミット（CGS）法をベースとしたアルゴリズムを適用しているが，本研究ではこれらに LAPACK ルーチンの DGEQRF，DORGQR を適用する．これらのルーチンは行列乗算中心の実装が施されており，DGEQEF で Householder 変換に基づく QR 分解を行い，DORGQR により直交行列の計算を行う．また，再直交化付きブロック逆反復法では Algorithm 2 の 14，16 において，それぞれ 2 回ずつ行列乗算を行っている．この演算は，level-3 BLAS の DGEMM が適用できる．

Intel MKL は，DGEMM，DGEQRF，DORGQR の並列ルーチンを提供している．本研究では，Intel MKL を利用することで以上のルーチン内部で演算をスレッド並列化する．

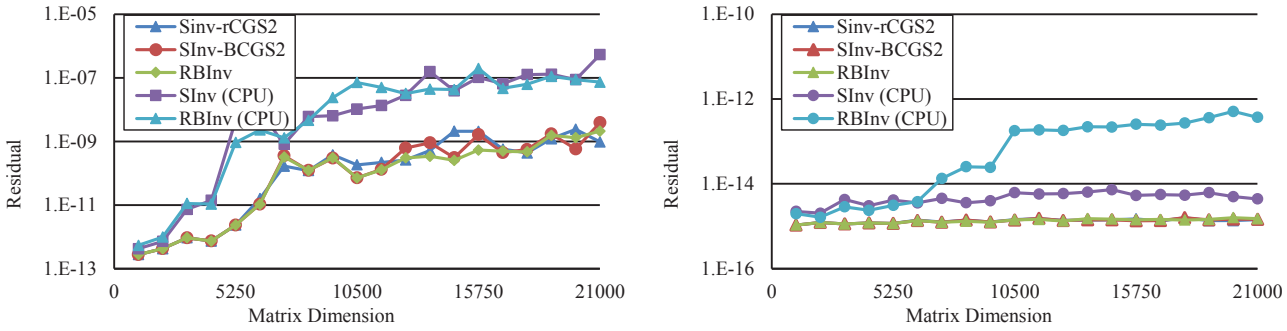
4. ライブラリ利用に基づいた GPU 実装

本章では，同時逆反復法と再直交化付きブロック逆反復法の GPU 実装について述べる．ここでは特に，CULA のルーチンによる実装を行う．

4.1 実装の詳細

3.4 章で説明した CPU コードを基に，CULA が提供するルーチンを代替可能な演算に適用することで，GPU コードを作成した．ここで，同時逆反復法および再直交化付きブロック逆反復法の GPU 向け実装コードを，それぞれ，**SInv (GPU)**，**RBInv (GPU)** とする．

SInv (CPU) を構成するルーチンの中では，QR 分解に用いる DGEQRF，DORGQR の GPU 実装が CULA に用意されている．また，RBInv (CPU) の中では，QR 分解に用いるルーチンだけでなく，行列乗算ルーチンである DGEMM



(a) Cases of T_1 (b) Cases of T_2

図 1: [12] におけるコードと本研究の CPU コードにより計算した固有ベクトルの残差 $\|TQ - QD\|_\infty$

Fig. 1 Comparison of residual $\|TQ - QD\|_\infty$, where Q is computed by each code.

表 1: 各コードで使用する BLAS および LAPACK ルーチン

Table 1 BLAS and LAPACK routines using on each code

	SInv-rCGS2[12]	SInv (CPU)	SInv (GPU)	RBInv[12]	RBInv (CPU)	RBInv (GPU)
連立方程式の求解	DLAGTF	DLAGTF	DLAGTF	DLAGTF	DLAGTF	DLAGTF
	DLAGTS	DLAGTS	DLAGTS	DLAGTS	DLAGTS	DLAGTS
行列乗算	なし	なし	なし	DGEMM	DGEMM	CULA_DGEMM
QR 分解	再帰的 CGS 法 ×2	DGEQRF	CULA_DGEQRF	再帰的 CGS 法	DGEQRF	CULA_DGEQRF
		DORGQR	CULA_DORGQR		DORGQR	CULA_DORGQR

黒字：ルーチン内で並列計算（CPU のみ）
青字：各スレッドに割り当てた並列計算（CPU のみ）
赤字：ルーチン内で並列計算（CPU+GPU）

についても，GPU 実装が CULA に用意されている．これらの GPU 実装では，引数となる行列要素およびベクトル要素をホストメモリに用意することで，CPU と GPU それぞれに自動的に負荷分散を行って並列計算を行う．

一方，両者の CPU コードにおいて連立方程式の求解に用いる DLAGTF および DLAGTS は，CULA において実装ルーチンが提供されていない．しかしながら，CULA で実装可能な他のルーチンに比べて計算量の少ないものであることから，CPU コードと同様の CPU 上でのスレッド並列化に留める．

以上をまとめると，SInv (CPU), SInv (GPU), RBInv (CPU), RBInv (GPU) それぞれのコードにおいて用いた BLAS および LAPACK ルーチンとその並列化法は表 1 のようになる．ここで，**SInv-rCGS2** および **RBInv** は，[12] におけるマルチコア CPU 向けの実装コードである．

4.2 ライブラリを利用することの利点と欠点

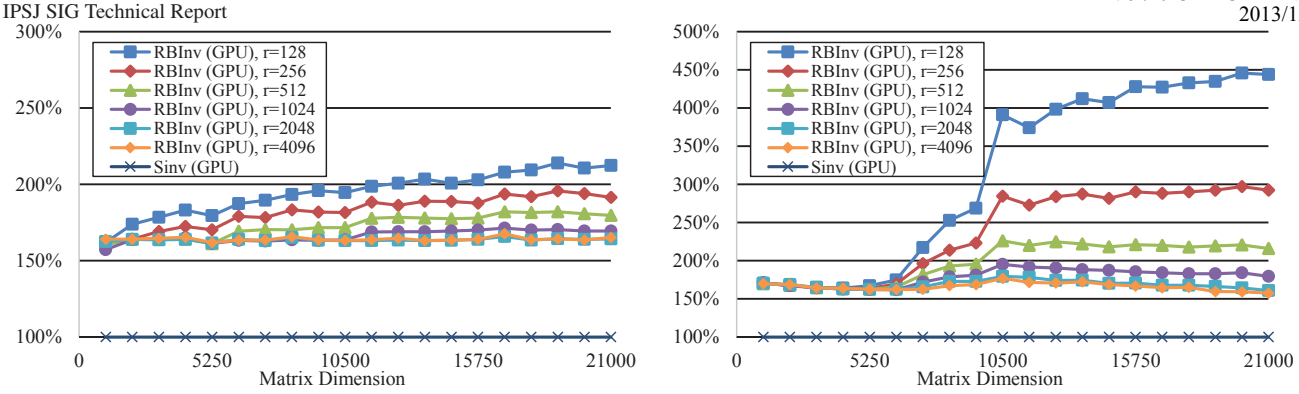
上記のように，LAPACK ルーチンを CULA ルーチンで置き換えるだけでアルゴリズムの GPU 実装が行えることは，ライブラリを利用することの非常に大きな利点である．

一方で，CGS 法ベースの QR 分解は，CULA や LAPACK のルーチンとして実装されていない．しかし，$n \times r$ 長方行列に対して，CGS 法による QR 分解の計算量は約 $2r^2n$ であり，DGEQRF と DORGQR のような Householder 変換ベースの QR 分解の計算量の約半分で済む．このため，BCGS2 法を適用した再直交化付きブロック逆反復法については，CGS 法によって QR 分解を行う方が計算量の点で有利である．

また，図 1 は [12] で用いた同時逆反復法および再直交化付きブロック逆反復法の実装コードと本研究で用いる CPU コードで得られた固有ベクトルの残差 $\|TQ - QD\|_\infty$ を表している．ここで，再直交化付きブロック逆反復法のブロックサイズは $r = 256$ とし，D は対角要素に固有値が並ぶ対角行列である．また，テスト行列には，5 章の評価実験に用いる行列 T_1，T_2 を使用している．図 1 から，[12] で用いた実装コード SInv-rCGS2，SInv-BCGS および RBInv に対して，本研究で用いた CPU コードによって計算した固有ベクトルの精度は悪化してしまうことが分かる．これは，同時逆反復法や再直交化付きブロック逆反復法においては，QR 分解に CGS 法ベースのアルゴリズムを用いた方が高い精度の固有ベクトル計算ができることを示している．

5. 性能評価

マルチコア CPU と GPU で構成される計算機（表 2）において数値実験を行い，提案した GPU 実装コードの性能を評価する．また，性能評価を基に，更なる実装の改良点についても考察する．



(a) Cases of T_1 (b) Cases of T_2

図 2: GPU コードによる固有ベクトル計算に要した時間の比較

Fig. 2 Comparison of elapsed times for computing eigenvectors of target matrices using GPU code.

表 2: 実験環境

Table 2 Specifications of the experimental environment

CPU	Intel core i5 2500 (3.3GHz, 4 core)
	Memory: DDR3-1600 32GB
GPU	NVIDIA GeForce GTX TITAN
	Memory: GDDR5 6GB
Compiler	Intel Fortran Compiler 13.1.3
	Intel C++ Compiler 13.1.3
Options	-O3 -xHOST -ipo -no-prec-div
Software	Intel Math Kernel Library 11.0 update 1
	CUDA Toolkit 5.0
	CULA Dense R17

5.1 実験内容

実対称 3 重対角行列 T_1 および T_2 の全固有ベクトルを計算することにより，再直交化付きブロック逆反復法の GPU コードの性能を評価した．また，比較のため，3.4 章で説明した CPU コードも使用した．

テスト行列には，固有値分布が異なる 2 種類の n 次元行列 T_1，T_2 を用意した．T_1 は，Glued-Wilkinson 行列 [5], [7] である．この行列の固有値は，Peters-Wilkinson の判定基準において，大きさが $n/21$ となるクラスターと $2n/21$ となるクラスターがそれぞれ 7 つ，計 14 のクラスターに分かれることが知られている．更に，それぞれのクラスターに属する固有値が非常に密集しており，条件数の悪い問題として知られている．T_2 は，全要素を $(0, 1)$ の範囲の乱数により生成した実対称 3 重対角行列である．この行列の固有値は，小次元行列では数十から数百のクラスターに分かれるが，大次元行列ではほとんどが一つのクラスターに含まれる．本実験においても，10500 次元以上の行列 T_2 の多くは，9 割以上の固有値が一つのクラスターに属していた．

また，各テスト行列の固有値は，Intel MKL に実装された DSTEBZ ルーチンを利用することにより計算した．DSTEBZ は，実対称 3 重対角行列の固有値を 2 分法によって計算する倍精度演算ルーチンである．

逆反復法において，我々が許容する反復回数は 5 回である．しかし，全ての実験において，いかなる入力行列の場合でも 3 回の反復回数で収束することが確認できている．

5.2 GPU コードの比較

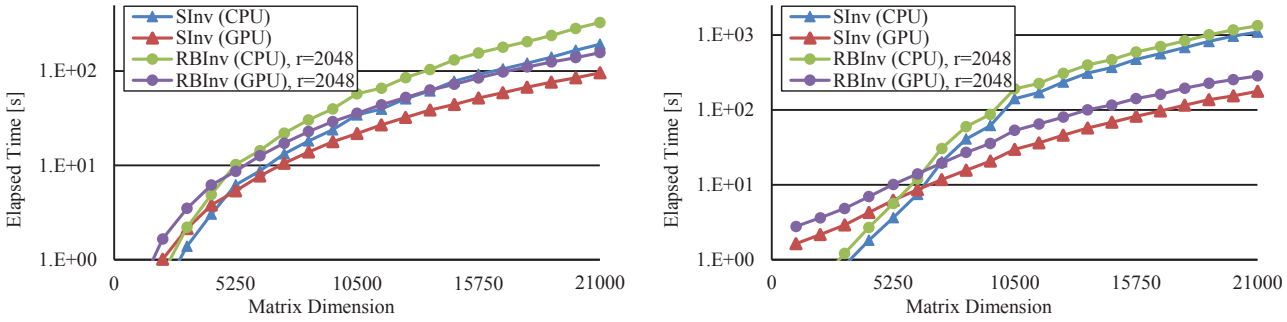
図 2 は，SInv (GPU) と RBInv (GPU) によって各テスト行列の全固有ベクトル計算を行った時の計算時間を比較している．RBInv (GPU) については，r = 128, 256, 512, 1024, 2048, 4096 の場合それぞれについて計算時間を計測し，SInv (GPU) の計算時間を基準（100%）としてグラフにプロットした．SInv (GPU) の計算量は RBInv (GPU) に比べて少ないことから，SInv (GPU) がどの r の値の RBInv (GPU) よりも高速である．

図 2 の比較により，行列サイズが大きくなるにつれ，RBInv (GPU) の r が大きいもの程高速になることが分かる．ある程度小さな行列に対する乗算はプロセッサのパフォーマンスを引き出すことが難しいことから，この結果は妥当である．この傾向は特に T_2 において顕著である．これは，r が小さい場合には行列乗算を呼び出す回数が増えるので，データ転送に伴うオーバーヘッドが計算時間を遅延させてしまうからだと考えられる．T_2 の場合，行列サイズが大きければ，固有値クラスターは行列サイズに近くなるため，この遅延が大きくなる．

このような傾向がある一方，どちらの行列に対しても，r が大きくなるにつれて性能上昇は鈍化しているため，ある程度大きな r においてはこのアルゴリズムにおける限界のパフォーマンスに近い性能が得られるといえる．また，実用上では，ある程度高性能な r を自動で選択できることが望ましい．

5.3 CPU コードと GPU コードの比較

図 3 は，同時逆反復法と再直交化付きブロック逆反復法それぞれの CPU コードと GPU コードによって全固有ベクトル計算を行った時の計算時間を比較している．ここで，再直交化付きブロック逆反復法の各コードではブロックサ



(a) Cases of T_1 (b) Cases of T_2

図 3: CPU コードおよび GPU コードによる固有ベクトル計算に要した時間

Fig. 3 Elapsed times for computing eigenvectors of target matrices using each CPU and GPU code.

イズを $r = 2048$ とした．

これらの図から，いずれのテスト行列に対しても，小さなサイズの行列に対しては CPU コード，大きなサイズの行列に対しては GPU コードの方が高速に計算できることが分かる．小さなサイズの行列においてはアルゴリズム全体の計算量が少なく，GPU による高速化が総計算時間の削減に寄与しにくい．その一方で，CPU と GPU 間でのデータ転送による遅延の影響が大きくなったからだと考えられる．逆に，大きなサイズの行列ではアルゴリズム全体の計算量が比較的大きいため，CPU と GPU 間のデータ転送による遅延の影響が小さくなり，CPU と GPU の協調演算による高速化が総計算時間の削減に大きく寄与したと考えられる．

同時逆反復法は $n = 21000$ のとき，行列 T_1 に対しては 2.0 倍，行列 T_2 に対しては 6.2 倍，GPU コードが高速であった．同様に，再直交化付きブロック逆反復法は，$n = 21000$ のとき，行列 T_1 に対しては 2.1 倍，行列 T_2 に対しては 4.7 倍，GPU コードが高速であった．このように T_1 と T_2 において高速化の度合いに差があるのは，テスト行列の固有値分布が異なることに起因する．固有値クラスターの大きさ m_c に対して，連立方程式の求解に要する計算量は $O(m_c n)$，QR 分解やブロック再直交化計算に要する計算量は $O(m_c^2 n)$ である．このため，T_1 に比べて固有値クラスターのサイズが大きい T_2 では，後者の計算量が非常に支配的になる．本研究で GPU による高速化の対象となったのは，QR 分解やブロック再直交化計算の部分である．これらの計算は，プロセッサにとって性能の出やすい演算である行列乗算の割合が高く，計算量が大きいほど高速化の効果は得やすい．一方で，連立方程式の求解はマルチコア CPU 上でのスレッド並列化しか施していないため，GPU を用いた場合と比較すると，あまり高速化は期待できない．上記のような理由で，行列 T_2 の場合に GPU による高速化が実行時間の削減に大きく影響し，T_1 の場合には影響が小さかったため，T_1 と T_2 において高速化の度合いが異なったのだと考えられる．

ここで，同時逆反復法や再直交化付きブロック逆反復法の GPU 実装によって部分固有対計算を行う場合を考える．このとき，必然的に QR 分解やブロック再直交化計算のルーチンの計算量の割合は小さくなる．上記の考察を踏まえると，現状の実装では，GPU による高速化は T_2 のように大きなものは期待できない．したがって，このような場合にも GPU の性能をより多く引き出せるよう，連立方程式の求解ルーチンにおいても CPU と GPU 両方を用いた並列計算を実装する必要がある．

6. まとめと今後の課題

本論文では，行列乗算を用いた固有ベクトル計算アルゴリズムである同時逆反復法，再直交化付きブロック逆反復法の GPU 実装について検討を行った．Intel MKL や CULA といった既存のライブラリに着目した GPU 実装を行い，性能評価においても CPU 実装に対して高速化が見られることを確認した．しかしながら，性能評価からも提案実装に改善の必要性があることが伺える．特に，本研究において未実装であった，CGS 法をベースとした QR 分解ルーチンや連立方程式の求解ルーチンの GPU 実装について検討を行うべきである．

また，今後の課題として，1 ノードあたりに複数の GPU を搭載した分散並列環境向けに実装したものの性能評価が挙げられる．

謝辞 本研究は科学研究費補助金特別研究員奨励費（課題番号：25・2820），基盤研究（B）（課題番号：24360038）の補助を受けている．

参考文献

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J. W., Dongarra, J., Du Croz, J., Hammarling, S., Greenbaum, A., McKenney, A. and Sorensen, D.: *LAPACK Users' Guide (Third ed.)*, SIAM, Philadelphia, PA, USA (1999).

[2] Barlow, J. L. and Smoktunowicz, A.: Reorthogonalized block classical Gram-Schmidt, *Numer. Math.*, Vol. 123, No. 3, pp.

1–29 (2012).

[3] Bientinesi, P., Igual, F. D., Kressner, D. and Enrique, S. Q.: Reduction to Condensed Forms for Symmetric Eigenvalue Problems on Multi-core Architectures, *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science, Vol. 6067, Springer Berlin Heidelberg, pp. 387–395 (2010).

[4] Chatelin, F. C. and Ahués, M.: *Eigenvalues of Matrices*, SIAM, Philadelphia, PA, USA (2012).

[5] Demmel, J. W., Marques, O. A., Parlett, B. N. and Vömel, C.: Performance and accuracy of LAPACK's symmetric tridiagonal eigensolvers, *SIAM J. Sci. Comput.*, Vol. 30, No. 3, pp. 1508–1526 (2008).

[6] 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).

[7] Dhillon, I. S., Parlett, B. N. and Vömel, C.: Glued matrices and the MRRR algorithm, *SIAM J. Sci. Comput.*, Vol. 27, No. 2, pp. 496–510 (2005).

[8] EM Photonics: CULA, (online), available from ⟨http://www.culatools.com/⟩ (accssesed 2013-11-18).

[9] Intel: Math Kernel Library, (online), available from ⟨http://software.intel.com/en-us/intel-mkl⟩ (accssesed 2013-11-18).

[10] Ipsen, I. C. F.: Computing an Eigenvector with Inverse Iteration, *SIAM Review*, Vol. 39, No. 2, pp. 254–291 (1997).

[11] Ishigami, H., Kimura, K. and Nakamura, Y.: GPU implementation of inverse iteration algorithm for computing eigenvectors, *Proc. of the 22nd Euromicro International Conference on Parallel, distributed and network-based Processing (PDP14)* (accepted).

[12] 石上裕之，木村欣司，中村佳正：再直交化付きブロック逆反復法による固有ベクトルの並列計算，2014 年ハイパフォーマンスコンピューティングと計算科学シンポジウム (accepted).

[13] NVIDIA: CUDA, (online), available from ⟨https://developer.nvidia.com/category/zone/cuda-zone/⟩ (accssesed 2013-11-18).

[14] Parlett, B. N.: *The Symmetric Eigenvalue Problem*, SIAM, Philadelphia, PA, USA (1998).

[15] Peters, G. and Wilkinson, J.: *The calculation of specified eigenvectors by inverse iteration*, Handbook for Automatic Computation, pp. 418–439, Springer-Verlag, Berlin (1971).

[16] Stewart, G.: Block Gram-Schmidt orthogonalization, *SIAM Journal on Scientific Computing*, Vol. 31, No. 1, pp. 761–775 (2008).

[17] Tomov, S., Nath, R. and Dongarra, J.: Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing, *Parallel Computing*, Vol. 36, No. 12, pp. 645 – 654 (2010).

[18] Volkov, V. and Demmel, J. W.: Using GPUs to Accelerate the Bisection Algorithm for Finding Eigenvalues of Symmetric Tridiagonal Matrices, *LAPACK Working Note*, No. 197 (online), available from ⟨http://www.netlib.org/lapack/lawnspdf/lawn197.pdf⟩ (2008).