情報処理学会論文誌　コンピューティングシステム　Vol.5　No.5　163–176 (Oct. 2012)

倍精度正方行列特異値分解アルゴリズムの GPGPU 上での性能・精度評価

廣田 悠輔[1,a)]　橋本 拓也[2]　山本 有作[1,3]

受付日 2012年3月28日, 採録日 2012年7月5日

概要：Bischof の方法による正方行列の特異値分解アルゴリズムを GPGPU 向けに倍精度で実装した．実装のパラメータ（帯幅）を様々に変化させて実行し，NVIDIA 社の Tesla C2050 を搭載した計算機を含む 2 つの計算機で評価を行った．さらに，LAPACK の CPU への実装（MKL）および GPGPU への実装（CULA，MAGMA）と比較を行った．我々の実装により得られる特異値，特異ベクトルの精度は他の実装と同程度となった．また，Xeon X5680（3.3 GHz，hexa-core）および Tesla C2050 を搭載した計算機で特異値分解を行ったとき，我々の実装は MKL の 3.9 倍，CULA および MAGMA の 1.8 倍高速となった．さらに，実行時間の内訳について詳細な分析を行い，その結果に基づき我々の実装の高速化手法について検討した．

キーワード：特異値分解，GPGPU，性能評価，精度評価，Bischof の方法

Performance and Accuracy of Singular Value Decomposition of Square Matrices in Double Precision Arithmetic on GPGPU

Yusuke Hirota[1,a)]　Takuya Hashimoto[2]　Yusaku Yamamoto[1,3]

Received: March 28, 2012, Accepted: July 5, 2012

Abstract: We develop a GPGPU implementation for singular value decomposition of square matrices based on Bischof's algorithm in double precision arithmetic. The performance and the accuracy of the implementation with various parameter values (band width) is evaluated and compared to LAPACK implementations for CPU (MKL) and for GPGPU (CULA and MAGMA). The accuracy of singular values and singular vectors by our implementation is comparable with that of other implementations. Our implementation executing on a GPGPU (Tesla C2050) is about 3.9 times faster than MKL executing on a CPU (Xeon X5680, 3.3 GHz, hexa-core). Also, our implementation is about 1.8 times faster than CULA and MAGMA executing on the GPGPU. In addition, the breakdown of the execution time is analyzed in detail and some performance improvement methods of our implementation based on the analysis are discussed.

Keywords: singular value decomposition, GPGPU, performance evaluation, accuracy evaluation, Bischof's method

[1] 神戸大学大学院システム情報学研究科計算科学専攻
Department of Computational Science, Graduate School of System Infomatics, Kobe University, Kobe, Hyogo 657–8501, Japan
[2] 神戸大学大学院システム情報学研究科システム科学専攻
Department of Systems Science, Graduate School of System Infomatics, Kobe University, Kobe, Hyogo 657–8501, Japan
[3] 科学技術推進機構，戦略的創造研究推進事業
Japan Science and Technology Agency, CREST
[a)] hirota@stu.kobe-u.ac.jp

1. はじめに

任意の実正方行列 $A \in \mathbb{R}^{N\times N}$ は，

$$A = U\Sigma V^\top$$

と直交行列 $U, V \in \mathbb{R}^{N\times N}$ と対角行列 $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_N) \in \mathbb{R}^{N\times N}$ の積に分解できる．ただし，$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N \geq 0$ である．この分解は特異値分解と呼ばれる．この

© 2012 Information Processing Society of Japan

ときの，U，V の各列ベクトルをそれぞれ左特異ベクトル，右特異ベクトルといい，σ_i（$1 \le i \le N$）を特異値という．特異値分解は画像処理やデータマイニング，電子状態計算などに応用され，その高速化が求められている．

特異値分解を高速に行う方法として，きわめて高い浮動小数点演算能力を持つ General Purpose Graphics Processing Unit（GPGPU）に，特異値分解アルゴリズムを実装するというアプローチが考えられる．そのようなアプローチについての研究として，2009 年の深谷らによる研究がある [1]．深谷らの作成した特異値分解の GPGPU 向け実装では，特異値分解のステップである二重対角化とその逆変換に，Bischof らの提案したアルゴリズム [2], [3] が用いられている．このアルゴリズムを用いた場合，特異値分解の演算の大部分が level-3 BLAS によって実行できる．深谷らは，特異値分解の level-3 BLAS の部分を GPGPU 向けにチューニングされたライブラリで実行することで，CPU を用いた場合と比べてきわめて高速に特異値分解を行えることを示した．しかしながら，深谷らの実装は，当時の GPGPU の制約のため，GPGPU での浮動小数点演算を単精度で行っており，得られる特異値，特異ベクトルの精度は単精度相当のものであった．一方，LAPACK [4] の GPGPU 向けの実装として CULA [5]，MAGMA [6] などがあり，特異値分解を行うルーチンが提供されている．これらの実装では，深谷らの研究と異なり，level-2 BLAS と level-3 BLAS の両方を用いる Dongarra のアルゴリズム [7] が採用されている．

本研究の目的は，Bischof の方法に基づく特異値分解アルゴリズムを GPGPU 向けに倍精度で実装し，最新の GPGPU を搭載したマシンを含む 2 種類のマシンで実行し，性能評価と詳細な性能分析を行うことである．特に，次の 3 点に着目して評価と分析を行う．1 点目は，Bischof のアルゴリズムの精度評価である．Bischof の方法に基づく特異値分解では，そのステップである二重対角化および逆変換がそれぞれ 2 段階に分けて行われる．そのため，これらのステップを 1 段階で行う Dongarra の方法と比べたとき，二重対角化の演算量は若干増加し，逆変換の演算量は Dongarra の方法を用いる場合の 2 倍になる．この計算量の増大が精度に及ぼす影響について，GPGPU 上での Bischof の方法に基づく特異値分解の実装と CPU 上での LAPACK の実装とを比較することで評価を行う．2 点目は，Bischof の方法が性能面で優位となるための条件を明らかにすることである．Bischof の方法は，演算量の大部分を level-3 BLAS により実行できるという利点を持つものの，演算量は上記のとおり Dongarra のアルゴリズムに比べて増大する．そのため，対象とするマシンのアーキテクチャにより，どちらの方法が性能面で有利かが分かれるはずである．本研究では，Dongarra のアルゴリズムに基づく CULA および MAGMA との比較により，GPGPU 上での両手法の優劣について，まず実験的に評価する．次に，level-2 BLAS と level-3 BLAS の性能比に基づいて両手法の優劣を予測する簡単な性能モデルを構築し，実験結果によりその有効性を検証する．本モデルは，新しいアーキテクチャの計算機において，どちらの手法が有利かを事前に予測するために有用であると考えられる．3 点目は，GPGPU 上での実行性能に関する詳細な分析を行い，改良のための今後の研究方向を検討することである．Bischof の方法は level-3 BLAS を使うために性能上有利とされるが，アルゴリズム中で使われている様々な level-3 BLAS について，それぞれどの程度の性能が出ているのか，また，level-3 BLAS 以外に使われる時間はどの程度か，などの点については，これまで十分な分析が行われてこなかった．本研究では，各ステップ，あるいは各 BLAS 単位での詳細な性能分析を行い，これらの点を明らかにする．さらに，その結果に基づき，さらなる性能向上のための方策を提案する．なお，以下では，特に断らない限り浮動小数点演算を倍精度で行うものとする．

本稿の構成は以下のとおりである．2 章では GPGPU を数値計算に用いる利点と，プログラムの実装時の注意点について述べる．3 章では特異値分解の数値計算アルゴリズムについて述べる．4 章では，特異値分解アルゴリズムの GPGPU への実装について説明する．5 章では，4 章で述べた実装の精度，実行時間，性能について評価し，他の実装との比較を行う．また，評価結果について考察を行う．6 章では，5 章の評価結果，考察をもとに，Bischof の方法に基づく特異値分解の GPGPU 向け実装の高速化手法について議論する．7 章で本稿のまとめを述べる．

2. GPGPU による数値計算

General Purpose Graphics Processing Unit（GPGPU）は，Graphics Processing Unit（GPU）を汎用の計算に用いることを可能とした装置である．近年開発された GPGPU はきわめて高い浮動小数点演算性能を持つ．たとえば，現在最速の GPGPU の 1 つである NVIDIA 社の Tesla C2050 の浮動小数点演算性能は，倍精度では 515 GFLOPS，単精度では 1.03 TFLOPS であり，現在主流の CPU と比べて非常に高い．また，NVIDIA 社の提供する CUDA [8] などの開発環境を利用すれば，C 言語などをベースとしたプログラミング言語を用いて GPGPU 向けのプログラム開発を行うことができる．加えて，CUBLAS や MAGMA BLAS などの GPGPU 向けにチューニングされた基本行列演算ライブラリ（Basic Linear Algebra Subprograms; BLAS）が提供されており，CPU 向けの数値計算を行う場合と同様に，簡単に高性能な行列計算を行うことができる環境が整っている．

一方で，数値計算プログラムを GPGPU に実装する場合には，いくつかの GPGPU の特性を考慮する必要がある．

© 2012 Information Processing Society of Japan

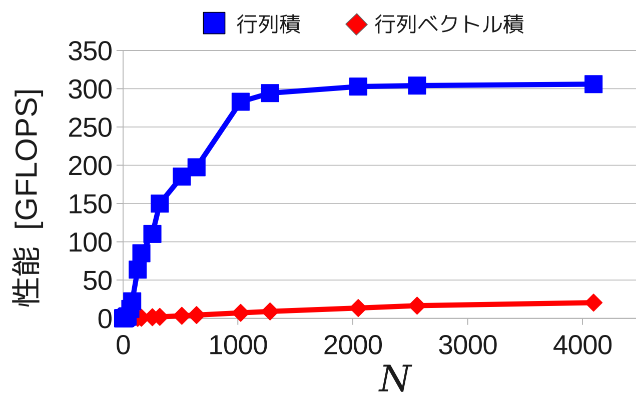

図 1 行列積 $C = \alpha AB + \beta C$，行列ベクトル積 $\boldsymbol{y} = \alpha A\boldsymbol{x} + \beta\boldsymbol{y}$（$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$，$A, B, C \in \mathbb{R}^{N\times N}$）の性能

Fig. 1 Performance of matrix product $C = \alpha AB + \beta C$ and matrix-vector product $\boldsymbol{y} = \alpha A\boldsymbol{x} + \beta\boldsymbol{y}$ ($\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$, $A, B, C \in \mathbb{R}^{N\times N}$).

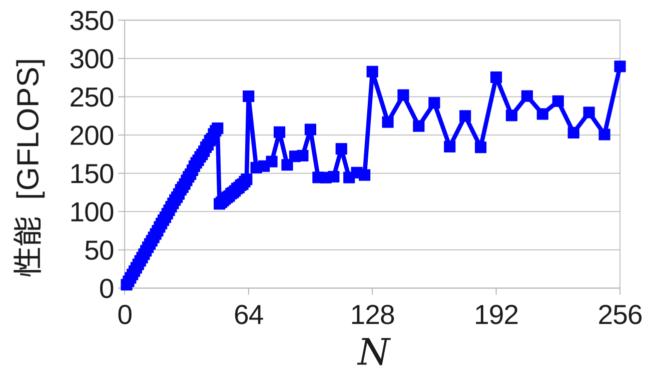

図 2 行列積 $A = A + BC$，$A \in \mathbb{R}^{M\times M}$，$B \in \mathbb{R}^{M\times N}$，$C \in \mathbb{R}^{N\times M}$ の性能（$M = 2560$）．$128 \le N \le 256$ では，8 刻みでプロットしているしている

Fig. 2 Performance of matrix product $A = A + BC$, $A \in \mathbb{R}^{M\times M}$, $B \in \mathbb{R}^{M\times N}$, $C \in \mathbb{R}^{N\times M}$ ($M = 2560$). The performance in $128 \le N \le 256$ are plotted by 8.

第 1 に，メインメモリと GPU ボードのメモリ（GPU メモリ）間のデータ転送速度である．GPGPU は GPU メモリのデータにのみアクセスが可能であるため，GPGPU で計算を行うには，あらかじめデータをメインメモリから GPU メモリに転送する必要がある．しかしながら，メインメモリと GPU メモリ間のデータ転送性能はきわめて低くレイテンシも大きい．このため，頻繁なデータ転送や大量のデータ転送を行う場合，その転送時間がプログラムの性能を阻害するおそれがある．第 2 に，GPU メモリから GPGPU の演算器へのデータ転送性能が，GPGPU がピーク演算性能で動作するのに必要な性能を大きく下回っているというという点である．この特性のため，level-3 BLAS に含まれる行列積などのデータ再利用性の高い演算ルーチンは GPGPU で高い性能を示す一方，行列ベクトル積やランク 1 更新などの level-2 BLAS に含まれる演算ルーチンの性能は低くなる．**図 1** は，Tesla C2050 で CUBLAS により，行列積 $C \leftarrow \alpha AB + \beta C$，行列ベクトル積 $\boldsymbol{y} \leftarrow \alpha A\boldsymbol{x} + \beta\boldsymbol{y}$（$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$，$A, B, C \in \mathbb{R}^{N\times N}$）を行ったときの性能である．行列積ではピーク性能の 60%程度の 300 GFLOPS を達成する一方，行列ベクトル積の性能は 20 GFLOPS 程度となっている．第 3 に，NVIDIA 社の GPGPU では，演算や GPU メモリへのデータアクセスが，一定数（16 や 32 など）ごとにまとめて行われるという点である．このため，計算で扱う行列の次数がその定数の倍数であるか否かにより，性能が大きく変化することがある．たとえば，特異値分解で頻出する行列積 $A \leftarrow A + BC$，$A \in \mathbb{R}^{M\times M}$，$B \in \mathbb{R}^{M\times N}$，$C \in \mathbb{R}^{N\times M}$，$M \gg N$ を CUBLAS を用いて GPGPU によって計算したときの性能は**図 2** のようになる．N の増大によりデータ再利用性が増大し性能が向上する傾向が見られる一方，N が 16 の倍数でない場合には，16 の倍数である場合と比べて性能が低下する傾向があることが分かる．

このように，GPGPU は高い性能を持つ一方で，性能に関する様々な制約がある．したがって，GPGPU 向けに高性能な数値計算プログラムを開発するには，適切なアルゴリズムを選択し，GPGPU の特性を考慮して実装を行う必要がある．

3. 特異値分解のアルゴリズム

一般的に，正方行列 A の特異値分解は以下の 3 段階の手順で行われる．

(1) 二重対角化 $A = U_1 B V_1^\top$
　(U_1，V_1：直交行列，B：下二重対角行列)．

(2) 二重対角行列の特異値分解 $B = U_2 \Sigma V_2^\top$
　(U_2，V_2：各列に B の特異ベクトルが並ぶ直交行列)．

(3) 特異ベクトルの逆変換 $U = U_1 U_2$，$V = V_1 V_2$．

二重対角化に用いた行列 U_1，V_1 は行列を陽に持つ必要はなく，逆変換における行列の積 $U = U_1 U_2$，$V = V_1 V_2$ が計算できればよい．

二重対角行列の特異値分解には，QR 法 [9]，分割統治法 [10], [11]，I-SVD アルゴリズム [12] などが用いられる．二重対角化のステップは Dongarra のアルゴリズム，Bischof らにより提案された 2 段階の二重対角化手法のいずれかがよく用いられる．また，逆変換はこれらの 2 つの二重対角化アルゴリズムに対応するアルゴリズムが用いられる．LAPACK では Dongarra のアルゴリズムが用いられ，深谷らによる実装では，Bischof の方法が用いられている．

Dongarra のアルゴリズムによる二重対角化は，古典的なハウスホルダ変換による二重対角化をブロック化したものである．古典的なハウスホルダ変換による二重対角化では，以下の手順を $i = 1, \ldots, N-1$ について順に繰り返すことで行列 A を下帯行列化する．

(1) 第 i 行の第 $i+1$ 列から第 N 列の要素を消去するハウスホルダ変換 $H_i^{(\mathrm{R})} = I - t_i^{(\mathrm{R})} \boldsymbol{y}_i^{(\mathrm{R})} (\boldsymbol{y}_i^{(\mathrm{R})})^\top \in \mathbb{R}^{N\times N}$ を

© 2012 Information Processing Society of Japan

生成する．

(2) 変換を A に右から作用させ $A \leftarrow A(H_i^{(\mathrm{R})})^\top = A - t_i^{(\mathrm{R})}(A\boldsymbol{y}_i^{(\mathrm{R})})(\boldsymbol{y}_i^{(\mathrm{R})})^\top$ と A を更新する．

(3) 第 i 列の第 $i+2$ 行から第 N 行の要素を消去するハウスホルダ変換 $H_i^{(\mathrm{L})} = I - t_i^{(\mathrm{L})}\boldsymbol{y}_i^{(\mathrm{L})}(\boldsymbol{y}_i^{(\mathrm{L})})^\top \in \mathbb{R}^{N\times N}$ を生成する．ただし，$i = N-1$ の場合には何も行わない．

(4) 変換を A に左から作用させ $A \leftarrow H_i^{(\mathrm{R})}A = A - t_i^{(\mathrm{L})}(\boldsymbol{y}_i^{(\mathrm{L})})[(\boldsymbol{y}_i^{(\mathrm{L})})^\top A]$ と A を更新する．ただし，$i = N-1$ の場合には何も行わない．

演算量のほとんどを (2)，(4) が占め，(2)，(4) あわせた演算量は $8/3N^3$ となる．これらの演算は半分が行列ベクトル積，残り半分がランク 1 更新として行われる．このため，古典的なハウスホルダ変換による二重対角化の実装では level-3 BLAS を用いることができず，高い性能が得られない．Dongarra のアルゴリズムでは，行列の更新の部分で，後続する何個かのハウスホルダ変換の生成に必要な行列の一部分のみを更新し，残りの部分は複数のハウスホルダ変換による更新をまとめて行う．Dongarra のアルゴリズムの演算量は，最高次のオーダのみに着目すると古典的なハウスホルダ変換による二重対角化アルゴリズムと同じ $8/3N^3$ であるが，演算量の半分を行列積として行うことができる．しかしながら，残りの半分は依然として行列ベクトル積によって行う必要がある．

Bischof の方法では，以下のように二重対角化および逆変換をさらに 2 ステップに分けて実行する．

(1) 二重対角化．
  (1-1) A の下帯行列化 $A = U_{1,1}CV_{1,1}^\top$
  ($U_{1,1}$，$V_{1,1}$：直交行列，C：下帯行列，$(C)_{i,j} = 0$ $(i > j+L, i < j)$)．
  (1-2) 下帯行列 C の二重対角化（村田法）$C = U_{1,2}BV_{1,2}^\top$
  ($U_{1,2}$，$V_{1,2}$：直交行列)．

(2) 二重対角行列の特異値分解．

(3) 特異ベクトルの逆変換．
  (2-1) 村田法の逆変換 $U' = U_{1,2}U_2$，$V' = V_{1,2}V_2$．
  (2-2) 下帯行列化の逆変換 $U = U_{1,1}U'$，$V = V_{1,1}V'$．

ただし，L は下帯行列 C の帯幅で，$1 \le L \le N-1$ を満たす任意整数を用いることができ，実装の際のパラメータとなる．

下帯行列化は，以下の手順の繰返しにより，図 **3** に示される順にブロック行，ブロック列ごとに消去することで行われる．

(1) 第 i ブロック行の第 $i+1$ ブロック列から第 N/L ブロック列を消去するブロックハウスホルダ変換 $Q_i^{(\mathrm{R})} = I - Y_i^{(\mathrm{R})}T_i^{(\mathrm{R})}(Y_i^{(\mathrm{R})})^\top \in \mathbb{R}^{N\times N}$ を生成する．

(2) 変換を A に右から作用させ $A \leftarrow A(Q_i^{(\mathrm{R})})^\top = A - AY_i^{(\mathrm{R})}T_i^{(\mathrm{R})}(Y_i^{(\mathrm{R})})^\top$ と A を更新する．

(3) 第 i ブロック列の第 $i+2$ ブロック行から第 N/L ブロック行を消去するブロックハウスホルダ変換 $Q_i^{(\mathrm{L})} = I - Y_i^{(\mathrm{L})}T_i^{(\mathrm{L})}(Y_i^{(\mathrm{L})})^\top \in \mathbb{R}^{N\times N}$ を生成する．ただし，$i = N/L-1$ の場合には何も行わない．

(4) 変換を A に左から作用させ $A \leftarrow Q_i^{(\mathrm{L})}A = A - Y_i^{(\mathrm{L})}T_i^{(\mathrm{L})}(Y_i^{(\mathrm{L})})^\top A$ と A を更新する．ただし，$i = N/L-1$ の場合には何も行わない．

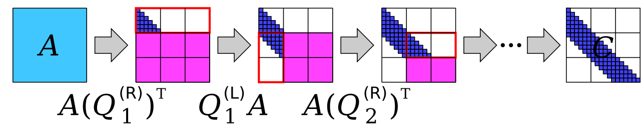

図 **3**　密行列 A から下帯行列 C への変換
Fig. 3　Transformation to lower band matrix C from dense matrix A.

$$Q = I - Y\,T\,Y^\top$$

図 **4**　ブロックハウスホルダ変換 Q
Fig. 4　Block Householder transformation Q.

ただし，$Q_i^{(\mathrm{R})}$，$Q_i^{(\mathrm{L})}$ は図 **4** に示される形の変換であり，$T_i^{(\mathrm{R})}$，$T_i^{(\mathrm{L})}$ は $L\times L$ 三角行列，$Y_i^{(\mathrm{R})}$，$Y_i^{(\mathrm{L})}$ は $N\times L$ 行列となる．ブロックハウスホルダ変換で用いられる行列の組 $(Y_i^{(\mathrm{L/R})}, T_i^{(\mathrm{L/R})})$ をブロックリフレクタと呼ぶ．Bischof の方法の二重対角化は，演算量のほとんどを (2)，(4) が占め，合わせて $8/3N^3$ となり，演算のほとんどを行列積として行うことができる．

下帯行列 C の二重対角化は村田法 [13] により行うことができる．村田法では，ハウスホルダ変換を繰り返し適用することにより，図 **5** に示される手順で二重対角化を行う．このときのハウスホルダ変換 $H_{i,j}^{(\mathrm{L/R})} = I - t_{i,j}^{(\mathrm{L/R})}\boldsymbol{y}_{i,j}^{(\mathrm{L/R})}(\boldsymbol{y}_{i,j}^{(\mathrm{L/R})})^\top$ のリフレクタ $\boldsymbol{y}_{i,j}^{(\mathrm{L/R})}$ は図 5 に示されるように，ベクトル中の連続する L 要素のみが非ゼロ要素となる．村田法の演算量は $8N^2L$ となり，ほとんどの演算がサイズの小さな行列ベクトル積，ランク 1 更新として行われる．また，第 i 列消去の $H_{i,j}^{(\mathrm{R})}$ 以降の更新と，第 $i+1$ 列消去の $H_{i,j-2}^{(\mathrm{R})}$ 以前の更新は，更新される範囲が重複しない．このため，第 i 列消去の更新が十分に進めば，第 $i+1$ 列消去の更新を開始することができる．さらに，第 $i+2$ 列以降も同様で，先行する列の消去がある程度進めば，消去を開始することができる．この原理を用いて，複数列の消去を，パイプライン並列実行することが可能である．

村田法の逆変換は，村田法で用いたハウスホルダ変換を順に U_2，V_2 に作用させることで行うことができる．しかしながら，単独のハウスホルダ変換 $H_{i,j}^{(\mathrm{L/R})} = I - t_{i,j}^{(\mathrm{L/R})}\boldsymbol{y}_{i,j}^{(\mathrm{L/R})}(\boldsymbol{y}_{i,j}^{(\mathrm{L/R})})^\top$ を作用させた場合，演算は行列ベクトル積とランク 1 更新によって行われ，実装を行うときに level-3 BLAS を用いることができない．そこで，村

© 2012 Information Processing Society of Japan

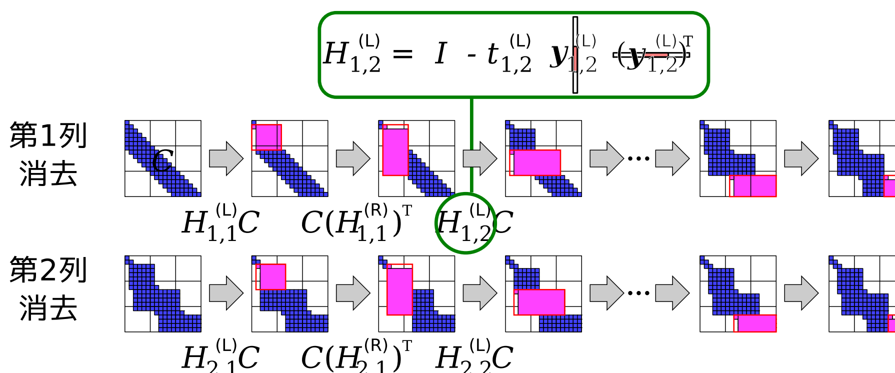

図 5　村田法による下帯行列 C から二重対角行列 B への変換
Fig. 5　Bidiagonalization of lower band matrix C by Murata's method.

ハウスホルダリフレクタをL本合成

$$Q = I - Y\,T\,Y^{T}$$

図 6　村田法の逆変換で用いられる合成された変換
Fig. 6　Combined matrix for inverse transformation of Murata's method.

田法のハウスホルダ変換を L 個合成 [14] して**図 6** に示される形の行列を作成し，合成された行列 $I - YTY^{\top}$ を U_2，V_2 に作用させるブロックアルゴリズムが用いられる．このアルゴリズムを用いるとき，演算量 $4N^3$ のほとんどを行列積として実行することが可能である．

下帯行列化の逆変換では，生成したブロックハウスホルダ変換 $Q^{(R)}_i$（$i = 1, 2, \ldots, N/L - 1$），$Q^{(L)}_i$（$i = 1, 2, \ldots, N/L - 2$）を，$U = Q^{(L)}_{N/L-2} \cdots Q^{(L)}_2 Q^{(L)}_1 U'$，$V = Q^{(R)}_{N/L-1} \cdots Q^{(R)}_2 Q^{(R)}_1 V'$ と順に U'，V' に作用させることで行われる．演算量は $4N^3$ となり，ほとんどを行列積として実行することが可能である．

Bischof の方法を用いたときの二重対角化の演算量は，Dongarra のアルゴリズムを用いた場合と比べて $8N^2L$ だけ多い．また，Bischof の方法を用いたときの逆変換の演算量は $8N^3$ であり，Dongarra のアルゴリズムを用いたときの逆変換の演算量の 2 倍となる．しかしながら，$L \ll N$ であれば $8N^2L$ は全体の演算量と比べて非常に少ない．また，Bischof の方法では，下帯行列化および 2 段階の逆変換のほとんどの演算を行列積として行うことができる．本研究では，実装時に演算の大部分を level-3 BLAS で実行することができる Bischof の方法に基づく特異値分解アルゴリズムを採用し，GPGPU 向けの実装を作成する．

4.　GPGPU への実装

Bischof の方法による特異値分解アルゴリズムの GPGPU への実装について述べる．我々の開発する実装は，二重対角化および逆変換を GPGPU を用いて行い，二重対角行列の特異値分解については外部ライブラリを用いて CPU で行う．以下では，二重対角化および逆変換の GPGPU への実装方法について説明する．

4.1　下帯行列化

あらかじめ，A を GPU メモリに転送しておき，以下の手順を繰り返すことで下二重対角化を行う．

(1) ブロックリフレクタの作成に必要な A の一部分をメインメモリに転送する．
(2) CPU でブロックリフレクタを生成する．
(3) ブロックリフレクタを GPU メモリに転送する．
(4) GPGPU で level-3 BLAS を使用して A の更新を行う．

演算量のほとんどは (4) で level-3 BLAS によって実行される．しかしながら，(1) から (4) の手順には逐次性があるためオーバラッピングして実行することができない．このため，(1) から (3) の実行時間が下帯行列化全体の性能に影響を及ぼす可能性がある．

4.2　村田法

下帯行列を疎構造を利用して GPU メモリに格納し，GPGPU のみを使用して二重対角化を行う．演算のほとんどは行列ベクトル積，ランク 1 更新として行うことができるが，疎構造を利用してデータを格納するため，BLAS ルーチンは使用できない．そのため，GPGPU 向けの自作ルーチンを実装する．また，その際，演算の並列性を利用しパイプライン並列化を行う．

4.3　村田法の逆変換

3 章で述べたアルゴリズムを GPGPU 向けに素朴に実装すると，以下のような手順になる．

(1) 合成する L 個のハウスホルダ変換のスカラファクタ $t^{(L/R)}_{i,j}$ およびリフレクタ $\boldsymbol{y}^{(L/R)}_{i,j}$ を，GPU メモリからメインメモリに転送する．
(2) CPU でハウスホルダ変換の合成 $Q = I - YTY^{\top}$ を生成する．

© 2012 Information Processing Society of Japan

表 1　MKL，CULA，MAGMA で使用される演算ルーチン
Table 1　Subroutines used in MKL, CULA and MAGMA.

実装名	二重対角化	二重対角行列の特異値分解	逆変換
MKL	DGEBRD	DBDSLV	DORMBR
CULA	CULA_DGEBRD	DBDSLV	MKL の DORMBR
MAGMA	MAGMAF_DGEBRD	DBDSLV	MKL の DORMBR

(3) Y，T を GPU メモリに転送する．
(4) GPU で合成された行列を U'，V' に作用させる．

しかしながら，(4) を level-3 BLAS の三角行列積ルーチンを用いて行うと，演算の途中で行列データのコピーが必要となる．これは $Q = I - YTT^\top$ を U'，V' に作用させるには $C \leftarrow AB + C$ の形のルーチンが必要となるが，BLAS の三角行列積には $B \leftarrow AB$ の形のルーチンしか用意されていないためである．このため，上記の実装を用いる場合には高い性能が得られない可能性がある．そこで，これを改良した以下の手順が考えられる．

(1) 合成する L 個のハウスホルダ変換のスカラファクタおよびリフレクタを，GPU メモリからメインメモリに転送する．
(2) CPU でハウスホルダ変換の合成 $Q = I - YTY^\top$ を生成する．
(3) CPU で $W \leftarrow YTY^\top$ を陽に計算する．
(4) W を GPU メモリに転送する．
(5) GPGPU の行列積ルーチンにより，$I - W$ を U'，V' に作用させる．

この方法では，W が Y，T の持っていた疎構造を失うため，演算量がもとの方法の 2 倍となるという問題があるが，W の GPGPU への転送以外では行列データのコピーが不要となる．また，(1) から (5) の手順は繰り返し行われるが，(5) と，次の反復における (1) から (4) の手順はオーバラッピングして行うことができ，CPU における計算時間は全体の実行時間には影響を与えない．

我々は，予備評価でより高速であった後者の実装を用いる．

4.4　下帯行列化の逆変換

下帯行列化で GPU メモリに転送されたブロックハウスホルダ変換を，GPGPU で level-3 BLAS を使用して順に U'，V' に作用させることにより逆変換を行う．この際，CPU では計算を行わない．

5.　性能評価

本章では，以下の 5 つの特異値分解の実装の精度および性能について評価を行う．

- LAPACK の CPU 向け実装 Intel Math Kernel Library [15]．以下，MKL と呼ぶ．
- Bischof の方法の CPU 向け実装．以下, Bischof（CPU）と呼ぶ．
- 前章で述べた，Bischof の方法の GPU 向け実装．以下，Bischof（GPU）と呼ぶ．
- LAPACK の GPGPU 向け実装 CULA．以下 CULA と呼ぶ．
- LAPACK の GPGPU 向け実装 MAGMA．以下 MAGMA と呼ぶ．

ただし，いずれの実装でも，二重対角行列の特異値分解には，京都大学中村研究室提供の I-SVD ライブラリ [16] の高速な特異値分解ルーチン（DBDSLV）を使用する．また，二重対角化および逆変換は，MKL，CULA，MAGMA では表 1 に示すルーチンを使用し，Bischof（CPU），Bischof（GPU）では自作のルーチンを使用した．Bischof（CPU）および Bischof（GPU）の帯幅 L は，24 から 128 の特異値分解対象の行列の次数の約数となるすべての値について評価を行う．CULA および MAGMA では逆変換に MKL のルーチンが使用されているが，これは，CULA および MAGMA に MKL の DORMBR に相当する逆変換ルーチンが実装されていないためである．CULA には二重対角化の際に用いたハウスホルダ変換から U_1 を計算するルーチン（CULA_DORGBR）が実装されているため，二重対角化ルーチンの計算終了直後に U_1 を計算し，CULA の QR 法ルーチン（CULA_DBDSQR）内部で $U = U_1U_2$ を計算することが可能である．しかしながら，DBDSLV にはそのような機能が備わっていない．また，予備実験から CULA_DBDSQR と CULA_DORGBR を用いる方法は，表 1 に示した方法より実行時間が長くなることが分かっている．そこで，本性能評価では CULA，MAGMA について実行時間が最短となる表 1 に示した方法を用いる．本性能評価では，これらの実装を表 2，表 3 に示される計算機で，精度，実行時間およびその内訳，性能（FLOPS 値）について調査した．テスト行列は各要素が $[-0.5, 0.5]$ の一様乱数である 2560×2560，5120×5120 および 7680×7680 行列を用いた．

5.1　精度評価

テスト行列として 7680×7680 行列を使用したときの, U, V の直交性 $\|I - UU^\top\|_\mathrm{F}$, $\|I - VV^\top\|_\mathrm{F}$, 残差 $\|I - U\Sigma V^\top\|_\mathrm{F}$, 特異値の誤差 $\max_i \|\sigma_i - \sigma_i^\mathrm{DC}\|$ の 4 つの項目の評価結果を

© 2012 Information Processing Society of Japan

表 2　Tesla C1060 マシンの計算機環境
Table 2　Computational environment of Tesla C1060 machine.

CPU	Intel Xeon E5502（2 コア，1.86 GHz，計 15 GFLOPS）
メインメモリ容量	6 GB
コンパイラ（CPU）	Intel C++/Fortran Compiler 12.0 update 2
コンパイルオプション（CPU）	`-openmp -O3`
BLAS（CPU）	Intel Math Kernel Library 10.3 update 8
GPU	Tesla C1060（78 GFLOPS）
GPU メモリ容量	4 GB
CUDA バージョン	3.2
BLAS（GPGPU）	CUBLAS 3.2
コンパイルオプション（nvcc）	`-O3 -arch=sm_13 -Xcompiler -fno-strict-aliasing`
比較ライブラリ	CULA R11a, MAGMA 1.0
OS	CentOS 5.5

表 3　Tesla C2050 マシンの計算機環境
Table 3　Computational environment of Tesla C2050 machine.

CPU	Intel Xeon E5680（6 コア，3.33 GHz，計 80 GFLOPS）
メインメモリ容量	12 GB
コンパイラ（CPU）	Intel C++/Fortran Compiler 12.0 update 1
コンパイルオプション（CPU）	`-openmp -O3`
BLAS（CPU）	Intel Math Kernel Library 10.3 update 1
GPU	Tesla C2050（515 GFLOPS）
GPU メモリ容量	3 GB
CUDA バージョン	4.0
BLAS（GPGPU）	CUBLAS 4.0
コンパイルオプション（nvcc）	`-O3 -arch=sm_20 -Xcompiler -fno-strict-aliasing`
比較ライブラリ	CULA R13, MAGMA 1.1
OS	CentOS 5.5

図 7，**図 8** に示す．ただし，誤差の評価に用いた σ_i^{DC} は，実装 MKL の DBDSLV を分割統治法ルーチン DBDSDC に置き換えて得られる特異値である．いずれの評価項目でも，最大値は最小値の 2 倍以下となっている．また，帯幅 L を変化させた場合にも同様の結果となった．2560×2560，5120×5120 のテスト行列を使用した場合でも，同様の結果が得られた．これらの結果から，我々の GPGPU 向けの倍精度の実装 Bischof（GPU）は，Intel による LAPACK の実装や，CULA，MAGMA などの他の GPGPU 向け実装と同程度の精度が得られることが確認された．

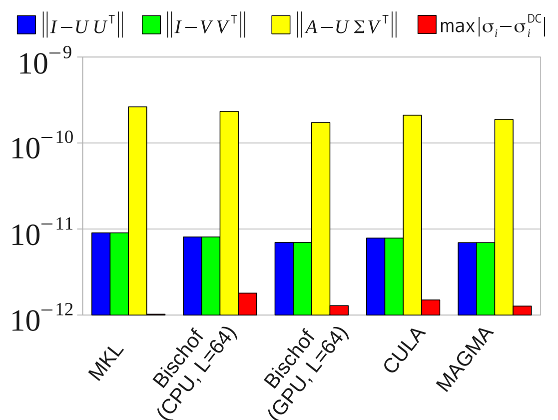

図 7　Tesla C1060 マシンでの精度評価結果（7680 × 7680 行列）
Fig. 7　Accuracy evaluation result of Tesla C1060 machine (7680-by-7680 matrix).

5.2　実行時間および性能

5.2.1　評価結果

2560×2560 行列を使用したときの実行時間を**図 9**，**図 10** に，5120×5120 行列を使用したときの実行時間を**図 11**，**図 12** に，7680×7680 行列を使用したときの実行時間を

© 2012 Information Processing Society of Japan

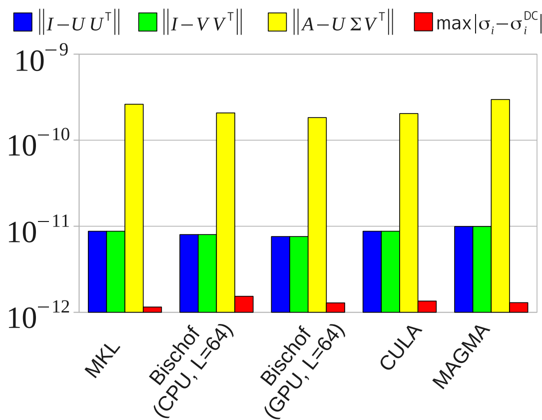

図 8　Tesla C2050 マシンでの精度評価結果（7680 × 7680 行列）
Fig. 8　Accuracy evaluation result of Tesla C2050 machine (7680-by-7680 matrix).

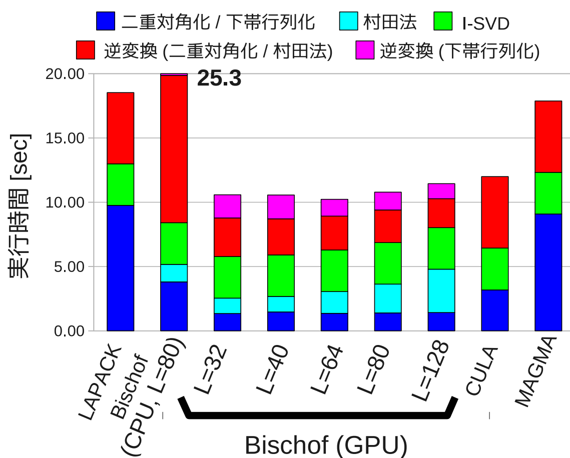

図 9　Tesla C1060 マシンでの実行時間（2560 × 2560 行列）
Fig. 9　Execution time on Tesla C1060 machine (2560-by-2560 matrix).

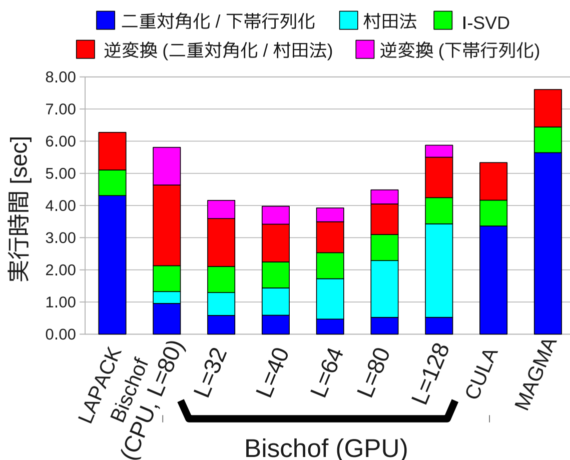

図 10　Tesla C2050 マシンでの実行時間（2560 × 2560 行列）
Fig. 10　Execution time on Tesla C2050 machine (2560-by-2560 matrix).

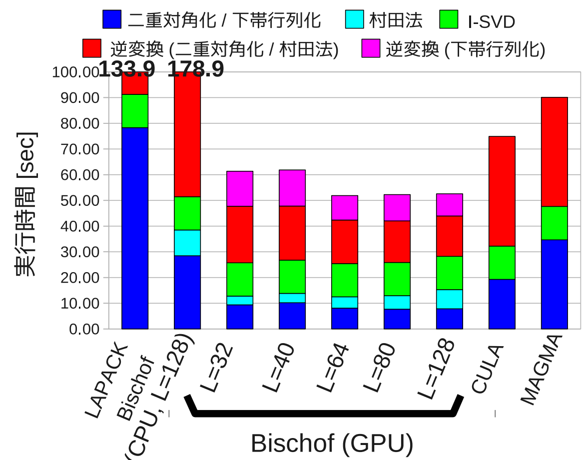

図 11　Tesla C1060 マシンでの実行時間（5120 × 5120 行列）
Fig. 11　Execution time on Tesla C1060 machine (5120-by-5120 matrix).

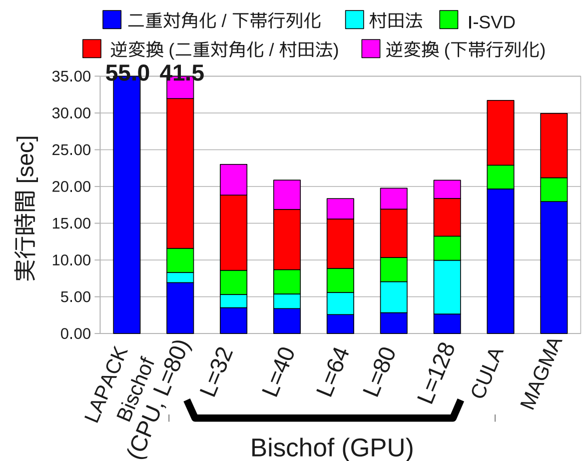

図 12　Tesla C2050 マシンでの実行時間（5120 × 5120 行列）
Fig. 12　Execution time on Tesla C2050 machine (5120-by-5120 matrix).

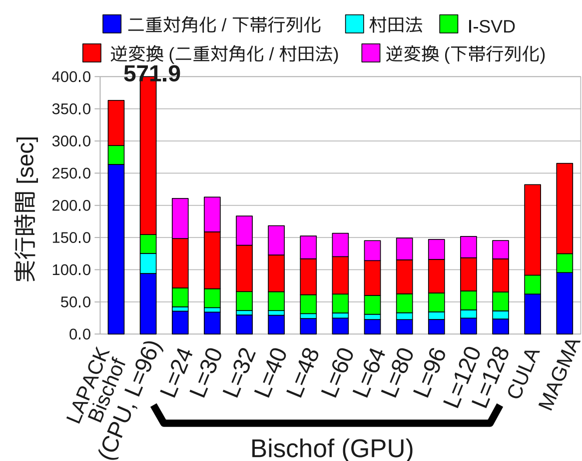

図 13　Tesla C1060 マシンでの実行時間（7680 × 7680 行列）
Fig. 13　Execution time on Tesla C1060 machine (7680-by-7680 matrix).

図 13，**図 14** に示す．ただし，MKL，Bischof（CPU）では，CPU の全コアを使用している．また，Bischof（CPU）については，実行時間が最短となった帯幅 L での結果のみを掲載している．また，7680 × 7680 行列を使用したときの下帯行列化の実行時間の内訳を**図 15**，**図 16** に示す．特異値分解全体の実行時間は，Tesla C2050 マシンで

© 2012 Information Processing Society of Japan

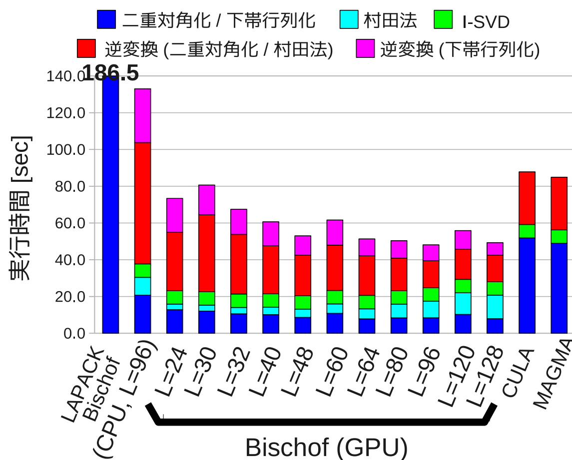

図 14　Tesla C2050 マシンでの実行時間（7680 × 7680 行列）

Fig. 14 Execution time on Tesla C2050 machine (7680-by-7680 matrix).

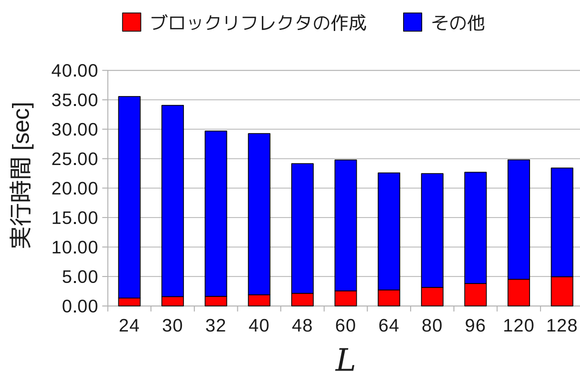

図 15　Tesla C1060 マシンでの下帯行列化の実行時間の内訳（7680 × 7680 行列）

Fig. 15 Execution time breakdown of the matrix reduction to band on Tesla C1060 machine (7680-by-7680 matrix).

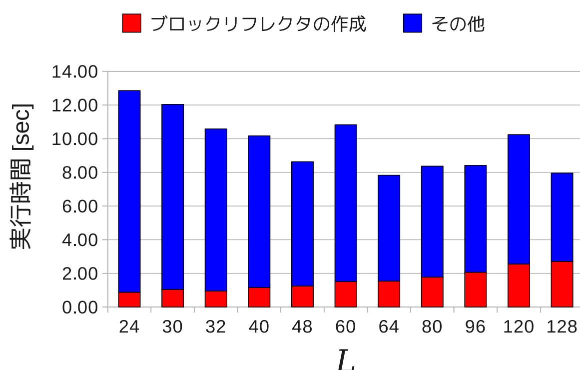

図 16　Tesla C2050 マシンでの下帯行列化の実行時間の内訳（7680 × 7680 行列）

Fig. 16 Execution time breakdown of the matrix reduction to band on Tesla C2050 machine (7680-by-7680 matrix).

2560×2560 行列を使用した場合を除き，Bischof（GPU）が L の値によらず最速となっている．また，Tesla C2050 マシンで 2560×2560 行列を使用した場合でも，最適な帯幅 L を使用した場合は Bischof（GPU）が最速である．次数が最大である 7680×7680 行列の場合，最適な帯幅 L を

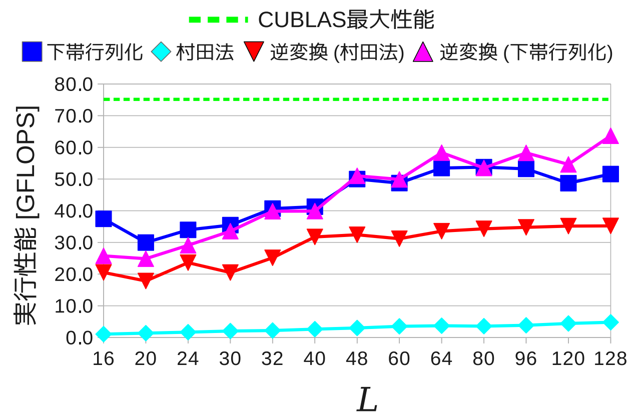

図 17　Tesla C1060 マシンでの実行性能（7680 × 7680 行列）

Fig. 17 Performance on Tesla C1060 machine (7680-by-7680 matrix).

使用した場合は Bischof（GPU）は，Tesla C1060 マシンでは，MKL，CULA，MAGMA に対してそれぞれ 2.5 倍，1.6 倍，1.8 倍高速である．また，Tesla C2050 マシンの場合では，MKL の 3.9 倍，CULA および MAGMA の 1.8 倍高速であり，さらに高速化率が高くなっている．Bischof（GPU）は他の実装に比べ，実行時間中に逆変換の占める割合が高いが，特異値のみが必要である場合には逆変換が不要であり，高速化率はさらに高くなる．これらの結果から，Bischof の方法の GPGPU 向け実装は，LAPACK の CPU 向け実装や GPGPU 向け実装と比べて高速であることが確認できる．

実行時間の内訳について見てみると，7680×7680 行列の場合，実行時間はいずれの計算機環境でも村田法の逆変換が最長となっている．また，Tesla C2050 マシンでは，最適な帯幅 $L = 96$ では村田法の実行時間の割合が 19% と村田法の逆変換に続いて多く，実行時間が $L = 96$ の場合に次いで短い $L = 128$ の場合には村田法の占める割合はさらに増大する．次数が小さい行列では村田法の実行時間の占める割合はさらに大きい．下帯行列化の実行時間の内訳を見ると，帯幅 L が大きいときにブロックリフレクタの生成の実行時間が長く，Tesla C1060 マシンでは最大 21%，Tesla C2050 マシンでは最大 33%を占めていることが分かる．紙面の都合上グラフを載せていないが，2560×2560 行列，5120×5120 行列の場合のブロックリフレクタの生成の実行時間の割合は，7680×7680 行列の場合よりもさらに大きくなっている．

次に，Bischof（GPU）の性能について述べる．7680×7680 行列を使用したときの性能を**図 17**，**図 18** に示す．ただし，性能は (本来必要な計算量)/(実行時間) によって求めている．また，図中の CUBLAS の最大性能とは，行列の次数が十分大きい正方行列積 $C \leftarrow AB + C$, $A, B, C \in \mathbb{R}^{4096 \times 4096}$ を計算したときの性能を示している．Tesla C1060 マシン使用時には，最適帯幅 $L = 64$ で，下帯行列化がピーク性

© 2012 Information Processing Society of Japan

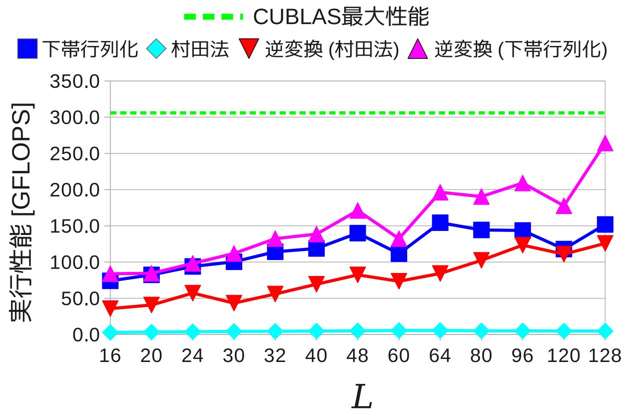

図 18　Tesla C2050 マシンでの実行性能（7680 × 7680 行列）

Fig. 18　Performance on Tesla C2050 machine (7680-by-7680 matrix).

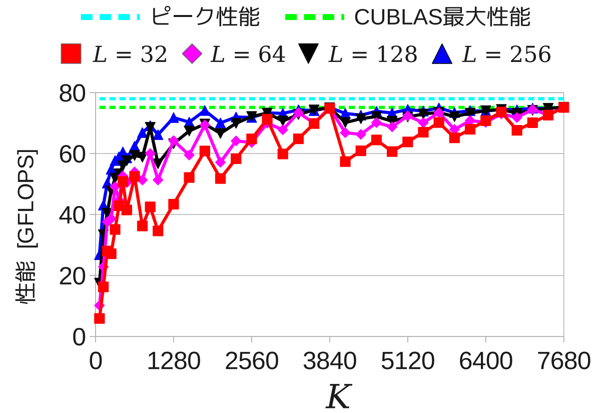

図 19　Tesla C1060 マシンでの $C \leftarrow AB + C$，$A \in \mathbb{R}^{K\times K}$，$B \in \mathbb{R}^{K\times L}$，$C \in \mathbb{R}^{K\times L}$ 型行列積の性能

Fig. 19　Performance of matrix multiplication $C \leftarrow AB+C$, $A \in \mathbb{R}^{K\times K}$, $B \in \mathbb{R}^{K\times L}$, $C \in \mathbb{R}^{K\times L}$ on Tesla C1060 machine.

能の 69%，村田法の逆変換が 43%，下帯行列化の逆変換が 75%と，下帯行列化とその逆変換は高い性能を示している．Tesla C2050 マシン使用時には，最適帯幅 $L = 96$ で，下帯行列化がピーク性能の 30%，村田法の逆変換が 16%，下帯行列化の逆変換が 38%となっている．いずれのマシンでも村田法の逆変換のピーク性能比が低くなっているが，4.3 節に述べた理由で，本来必要な量の 2 倍の計算を行っていることに注意しなければならない．一方，村田法はほとんどの演算が level-2 BLAS によって行われるため，L の増大に従って若干の性能向上が見られるものの，いずれの計算機環境でも他の演算と比べて実行性能はきわめて低くなっている．

5.2.2　考察

本項では，前項での評価結果について考察を行う．

Bischof の方法の実装の性能　Tesla C2050 マシン使用時のピーク性能比は Tesla C1060 マシンを使用時に比べて低い値となっている．これは，図 1 に示されるように，Tesla C2050 マシンでは CUBLAS の性能がピーク性能の 60%程度であることが原因の 1 つであると考えられる．したがって，CUBLAS の性能が改善された際には性能が向上すると予想される．

次に，下帯行列化および下帯行列化の逆変換の 2 つのステップの性能について考える．下帯行列化，下帯行列化の逆変換の性能は，図 17，図 18 に示されるように CUBLAS のピーク性能と比べても低い値となっている．これらのステップでは，演算の半分ずつが $C \leftarrow AB+C$, $A \in \mathbb{R}^{K\times K}$, $B \in \mathbb{R}^{K\times L}$，$C \in \mathbb{R}^{K\times L}$ 型の行列積（ブロック行列ベクトル積）と，$C \leftarrow AB + C$，$A \in \mathbb{R}^{K\times L}$，$B \in \mathbb{R}^{L\times K}$，$C \in \mathbb{R}^{K\times K}$ 型の行列積（ランク L 更新）として行われる．ただし K は，下帯行列化では $N - L, N - 2L, \ldots, L$ と減少していき，下帯行列化の逆変換では $L, 2L, \ldots, N$ と増大していく．このような形の行列積を CUBLAS によって実行したときの性能は Tesla C1060 マシンでは**図 19**，

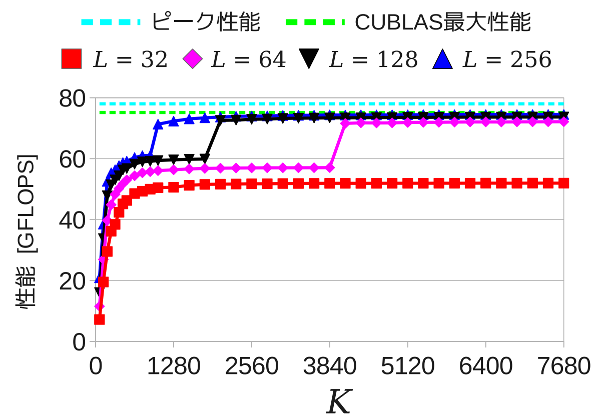

図 20　Tesla C1060 マシンでの $C \leftarrow AB + C$，$A \in \mathbb{R}^{K\times L}$，$B \in \mathbb{R}^{L\times K}$，$C \in \mathbb{R}^{K\times K}$ 型行列積の性能

Fig. 20　Performance of matrix multiplication $C \leftarrow AB+C$, $A \in \mathbb{R}^{K\times K}$, $B \in \mathbb{R}^{L\times K}$, $C \in \mathbb{R}^{K\times K}$ on Tesla C1060 machine.

図 20，Tesla C2050 マシンでは**図 21**，**図 22** に示されるものになる．K の値が小さいとき，行列積の性能が CUBLAS の最大性能と比べて低くなっている．特に，L が小さい場合にはその傾向が顕著である．Tesla C2050 マシンでは，村田法の実行時間（図 14）およびブロックリフレクタの実行時間（図 16）の関係から，$L = 96$ が最適となっているが，このとき，K が 5000 以下の領域では，特にランク L 更新の実行性能が低い（図 22）．これが，下帯行列化，下帯行列化の逆変換の性能が CUBLAS の最大性能と比べて低い原因の 1 つと考えられる．

Bischof の方法が Dongarra のアルゴリズムと比べて高速になる条件　MAGMA および CULA に GPGPU 向け逆変換ルーチンが実装された場合に，これらの Dongarra のアルゴリズムに基づく実装と Bischof の方法の実装のどちらが高速であるかについて考える．Tesla C2050 マシンでは，いずれのサイズの行列でも，CULA，MAGMA

© 2012 Information Processing Society of Japan

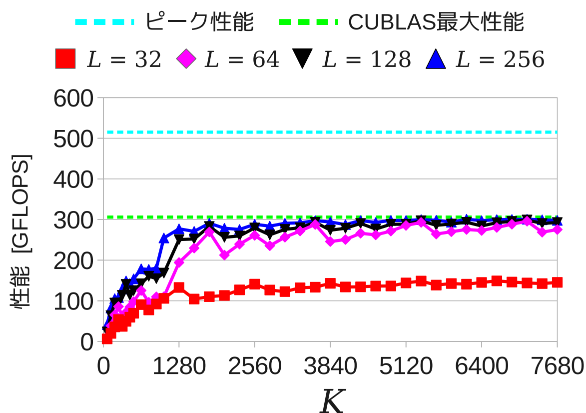

図 21　Tesla C2050 マシンでの $C \leftarrow AB + C$，$A \in \mathbb{R}^{K\times K}$，$B \in \mathbb{R}^{K\times L}$，$C \in \mathbb{R}^{K\times L}$ 型行列積の性能

Fig. 21　Performance of matrix multiplication $C \leftarrow AB+C$, $A \in \mathbb{R}^{K\times K}$, $B \in \mathbb{R}^{K\times L}$, $C \in \mathbb{R}^{K\times L}$ on Tesla C2050 machine.

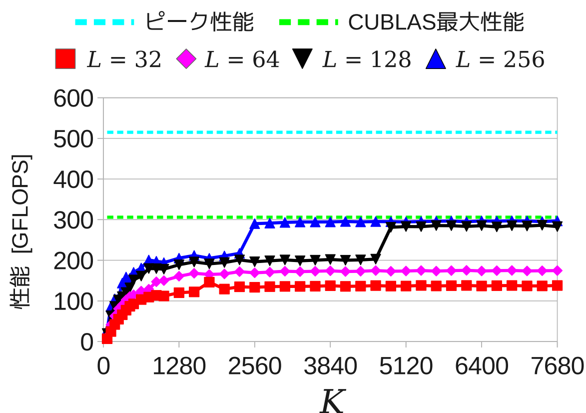

図 22　Tesla C2050 マシンでの $C \leftarrow AB + C$，$A \in \mathbb{R}^{K\times L}$，$B \in \mathbb{R}^{L\times K}$，$C \in \mathbb{R}^{K\times K}$ 型行列積の性能

Fig. 22　Performance of matrix multiplication $C \leftarrow AB+C$, $A \in \mathbb{R}^{K\times K}$, $B \in \mathbb{R}^{L\times K}$, $C \in \mathbb{R}^{K\times K}$ on Tesla C2050 machine.

の二重対角化と二重対角行列の特異値分解の実行時間の合計が最適帯幅の Bischof（GPU）の実行時間を超えている．このため，Tesla C2050 マシンを使用する場合，将来的に CULA，MAGMA に GPGPU 向け逆変換ルーチンが実装されて逆変換が高速化されたとしても，Bischof の方法の GPGPU への実装が速度面で優位であるといえる．しかしながら，Tesla C1060 では CULA，MAGMA が Bischof（GPU）より高速になる可能性が残る．また，実験で用いた計算機環境以外では，いずれの実装が高速であるかは明らかではない．そこで，非常にシンプルな実行時間モデルを立てて，CUBLAS の実行性能から 2 つの実装の実行時間の優劣を予測する．

Bischof の方法は，下帯行列化，村田法，二重対角行列の特異値分解，村田法の逆変換，下帯行列化の逆変換の 5 ステップで構成され，Dongarra のアルゴリズムは，二重対角化，二重対角行列の特異値分解，逆変換の 3 ステップで構成される．このうち，二重対角行列の特異値分解は，いずれの実装を用いた場合でも実行時間は同じであり，除外して考えても実行時間の優劣を予測するうえで影響はない．さらに，単純化のために Bischof の方法から演算量が小さく実行時間の予測が難しい村田法を除外し，下帯行列化，村田法の逆変換，下帯行列化の逆変換の推定実行時間 T_{band}，T_{murata}，T_{bband} の合計により Bischof の方法の実行時間を予測する．Dongarra のアルゴリズムについては，二重対角化，逆変換の推定実行時間 T_{bi}，T_{bbi} の合計により実行時間を予測する．

下帯行列化，村田法の逆変換，下帯行列化の逆変換，二重対角化および二重対角化の逆変換の各ステップに現れる演算は**表 4** に示される 4 種類に分けられ，各ステップで**表 5**，**表 6** 示される演算量が必要となる．したがって，表 4 に示した BLAS の性能が一定であると考え，その値を P_{GEMM1}，P_{GEMM2}，P_{GEMM3}，P_{GEMV} とおくと，

$$T_{\mathrm{band}} = \frac{4/3N^3}{P_{\mathrm{GEMM1}}} + \frac{4/3N^3}{P_{\mathrm{GEMM2}}},$$
$$T_{\mathrm{murata}} = \frac{4N^3}{P_{\mathrm{GEMM3}}},$$
$$T_{\mathrm{bband}} = \frac{2N^3}{P_{\mathrm{GEMM1}}} + \frac{2N^3}{P_{\mathrm{GEMM2}}},$$
$$T_{\mathrm{bi}} = \frac{4/3N^3}{P_{\mathrm{GEMV}}} + \frac{4/3N^3}{P_{\mathrm{GEMM2}}},$$
$$T_{\mathrm{bbi}} = \frac{2N^3}{P_{\mathrm{GEMM1}}} + \frac{2N^3}{P_{\mathrm{GEMM2}}}$$

と実行時間を推定できる．したがって不等式

$$\left(\frac{4/3N^3}{P_{\mathrm{GEMM1}}} + \frac{4/3N^3}{P_{\mathrm{GEMM2}}}\right) + \frac{4N^3}{P_{\mathrm{GEMM3}}} + \left(\frac{2N^3}{P_{\mathrm{GEMM1}}} + \frac{2N^3}{P_{\mathrm{GEMM2}}}\right) < \left(\frac{4/3N^3}{P_{\mathrm{GEMV}}} + \frac{4/3N^3}{P_{\mathrm{GEMM2}}}\right) + \left(\frac{2N^3}{P_{\mathrm{GEMM1}}} + \frac{2N^3}{P_{\mathrm{GEMM2}}}\right)$$

が成り立つ場合には，Bischof の方法の実装が高速であると考えられる．この不等式を整理し，P_{GEMV} に関する式に変形すると

$$P_{\mathrm{GEMV}} < \frac{P_{\mathrm{GEMM1}}P_{\mathrm{GEMM3}}}{3P_{\mathrm{GEMM1}} + P_{\mathrm{GEMM3}}} \tag{1}$$

となる．したがって，P_{GEMM1}，P_{GEMM3}，P_{GEMV} の値が決められれば，2 つの実装の実行時間の優劣が予測できる．ただし，このモデルでは演算の種類ごとに 1 つの固定値を実行性能として使用していることや，村田法の実行時間を無視しているために予測の精度が低くなるおそれがあることに注意しなければならない．

性能評価で使用した Tesla C1060 マシン，Tesla C2050 マシンで，7680 × 7680 行列の特異値分解を $L = 64$ の Bischof の方法の実装と Dongarra のアルゴリズムの実装で行う場合を考える．P_{GEMM1}，P_{GEMM3}，P_{GEMV} の値として $K = N/2 = 3840$ での CUBLAS の測定結果

© 2012 Information Processing Society of Japan

表 4　Bischof の方法および Dongarra のアルゴリズムで使用される BLAS
Table 4　BLAS used in he Bischof's algorithm and Dongarra's one.

名称	記号	演算
ブロック行列ベクトル積	GEMM1	$C \leftarrow AB + C$, $A \in \mathbb{R}^{K \times K}$, $B \in \mathbb{R}^{K \times L}$, $C \in \mathbb{R}^{K \times L}$
ランク L 更新	GEMM2	$C \leftarrow AB + C$, $A \in \mathbb{R}^{K \times L}$, $B \in \mathbb{R}^{L \times K}$, $C \in \mathbb{R}^{K \times K}$
横長行列積	GEMM3	$C \leftarrow AB + C$, $A \in \mathbb{R}^{K \times K}$, $B \in \mathbb{R}^{K \times L}$, $C \in \mathbb{R}^{K \times L}$
行列ベクトル積	GEMV	$\boldsymbol{c} \leftarrow A\boldsymbol{b} + \boldsymbol{c}$, $A \in \mathbb{R}^{K \times K}$, $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^{K}$

表 5　Bischof の方法における演算パターンと演算量
Table 5　Computational pattern and number of floating point operations in Bischof's algorithm.

ステップ	演算パターン	演算量
下帯行列化	GEMM1	$4/3N^3$
	GEMM2	$4/3N^3$
村田法の逆変換	GEMM3	$4N^3$
下帯行列化の逆変換	GEMM1	$2N^3$
	GEMM2	$2N^3$

表 6　Dongarra のアルゴリズムにおける演算パターンと演算量
Table 6　Computational pattern and number of floating point operations in Dongarra's algorithm.

ステップ	演算パターン	演算量
二重対角化	GEMV	$4/3N^3$
	GEMM2	$4/3N^3$
二重対角化の逆変換	GEMM1	$2N^3$
	GEMM2	$2N^3$

を用いる．ただし，P_{GEMM3} の値は村田法の逆変換の実行時間 T_{murata} の推定のみに使用され，村田法の逆変換で本来必要な量の 2 倍の計算が行われため，CUBLAS の測定値の半分の値を P_{GEMM3} として用いる．このとき，Tesla C1060 マシンでは $P_{\mathrm{GEMM1}} = 75.0$ [GFLOPS]，$P_{\mathrm{GEMM3}} = 35.3$ [GFLOPS]，$P_{\mathrm{GEMV}} = 20.7$ [GFLOPS] であり，式 (1) の右辺の値は 10.2 GFLOPS となる．したがって，(左辺) $>$ (右辺) であるから Dongarra のアルゴリズムの実装が高速であると予測される．一方，Tesla C2050 マシンでは $P_{\mathrm{GEMM1}} = 245.8$ [GFLOPS]，$P_{\mathrm{GEMM3}} = 82.7$ [GFLOPS]，$P_{\mathrm{GEMV}} = 20.3$ [GFLOPS] であり，式 (1) の右辺の値は 24.8 GFLOPS となる．したがって，(左辺) $<$ (右辺) であるから Bischof の方法の実装が高速であると予想できる．これは，実際の結果と矛盾しない．式 (1) の条件は，新しい計算機環境において Bischof の方法と Dongarra のアルゴリズムのどちらが優位かを見積もるための指針として，有用であると考えられる．

帯幅の実行性能への影響　帯幅の実行時間への影響について議論する．実行時間が最短となる帯幅 L について見てみると，Tesla C2050 マシンで 7680×7680 行列を使用した場合には $L = 96$，それ以外では $L = 64$ であった．また，行列のサイズ，実験機によらず，$64 \leq L \leq 96$ の範囲にある場合の実行時間は，最適帯幅での実行時間 1.15 倍以下と短くなっている．L の増大による下帯行列化，村田法の逆変換，下帯行列化の逆変換の性能向上と，村田法の演算量の増大による実行時間の増大はトレードオフの関係にあり，その結果として $64 \leq L \leq 96$ の範囲の帯幅での実行時間が短くなったと考えられる．また，Tesla C2050 マシンで 7680×7680 行列を使用した場合，$L = 60, 120$ で実行時間が長くなっているが，これは，性能についての議論で述べたように，L が 16 の倍数でない場合の性能低下の影響であると考えられる．

6.　高速化手法についての検討

本章では，前章の評価結果を基に，Bischof の方法の GPGPU 向け実装の高速化手法について検討する．

6.1　ブロックリフクレクタ作成部の並列化

現在は下帯行列化のための方法として，シングルスレッドで実行されているブロックリフレクタ生成のための QR 分解をマルチスレッドで実行することが考えられる．マルチスレッド向けの QR 分解アルゴリズムとしては，TSQR アルゴリズム [17] が有効とされている．TSQR アルゴリズムには生成されたブロックハウスホルダ変換を作用させる際の行列積のサイズが小さくなるという問題があるが，TSQR アルゴリズムで生成される複数のブロックリフレクタを合成し，従来と同サイズの 1 つのブロックリフレクタを生成する手法 [18] が提案されている．この方法を用いる場合，ブロックリフレクタ合成のための新たな計算が必要となるが，ブロックハウスホルダ変換を作用させる際の行列積のサイズはそのままに，マルチスレッド化による QR 分解の高速化が期待できる．しかしながら，この方法により作成されるブロックハウスホルダ変換の精度については，まだ解析が行われれいないため，この点について先に検討を行う必要がある．

© 2012 Information Processing Society of Japan

6.2　下帯行列化に現れるランク L 更新の多段化

次に，下帯行列化に現れるランク L 更新の多段化があげられる．下帯行列化では，K のサイズが L から N の範囲で変動する $K \times K$ 行列のランク L 更新が現れるが，図 20，図 22 より，小さな L では実行性能が低くなっている．実行性能を向上する方法として帯幅 L を大きくとることが考えられるが，その場合，村田法の計算量が増大するという問題がある．この問題に対する解決策として，Wu のアルゴリズム [19], [20] と呼ばれる Dongarra のアルゴリズムと Bischof の方法を組み合わせたものが提案されている．このアルゴリズムでは，下帯行列 C の帯幅を L としたままで，下帯行列化に現れるランク L 更新が i 個まとめてランク iL 更新として実行されるため，村田法の実行時間を増大させずに下帯行列化を高速化することが期待できる．

6.3　下帯行列化の逆変換のブロック化

3 番目に，下帯行列化の逆変換のブロック化があげられる．下帯行列化の逆変換で $U = Q^{(\mathrm{L})}_{N/L-2} \cdots Q^{(\mathrm{L})}_2 Q^{(\mathrm{L})}_1 U'$，$V = Q^{(\mathrm{R})}_{N/L-1} \cdots Q^{(\mathrm{R})}_2 Q^{(\mathrm{R})}_1 V'$ を計算するときに，ブロックハウスホルダ変換を 1 つずつ作用させるのではなく i 個まとめて作用させることを考える．i 個のブロックハウスホルダ変換の積 $Q' = Q^{(\mathrm{L})}_{j+i-1} Q^{(\mathrm{L})}_{j+i-2} \cdots Q^{(\mathrm{L})}_j$ は，$Q' = I - Y'T'(Y')^\top$ の形で表すことができる．ただし，$Y' = [T^{(\mathrm{L})}_{j+i-1} T^{(\mathrm{L})}_{j+i-2} \cdots T^{(\mathrm{L})}_j] \in \mathbb{R}^{N \times iL}$ であり，T' は $iL \times iL$ の下三角行列である．このような合成を行ってから特異ベクトルの逆変換を行う場合，T' を求める余分な計算が必要になるが，下帯行列化の逆変換に現れる行列積の次数が大きくなり，性能向上が期待できる．

6.4　逆変換における CPU と GPGPU の並列実行

4 番目に，特異値分解の逆変換ステップの CPU と GPGPU での並列実行があげられる．本研究で用いた実装では演算のほとんどを GPGPU のみで行っており，CPU の演算能力が無駄になっている．実行時間の半分以上を占める 2 段階の逆変換は，逆変換する複数の特異ベクトルを各 CPU と GPGPU に分配することでそれぞれ独立に逆変換を実行することが可能であるため，並列実行による高速化は比較的容易であると考えられる．村田法の逆変換は全実行時間に占める割合が大きいため，並列実行による高速化ができれば，全実行時間に対する短縮の効果は大きい．

6.5　村田法の逆変換のための高速な三角行列積ルーチンの実装

最後に，$C \leftarrow AB + C$ 型の高速な三角行列積ルーチンの作成による村田法の逆変換の高速化が考えられる．$C \leftarrow AB + C$ 型の三角行列積ルーチンが利用可能になれば，現在の実装で長方行列用の行列積ルーチンを用いて実行している村田法の逆変換の計算量は現在の半分になるため，村田法の逆変換を大幅に高速化できると考えられる．

7.　まとめ

Bischof の特異値分解アルゴリズムを GPGPU 向けに倍精度で実装し，様々な帯幅 L で精度および性能を評価した．また，LAPACK の CPU 向け実装 Intel MKL や，GPGPU 向け実装 CULA，MAGMA との比較を行った．その結果，以下の結論が得られた．

- Bischof の方法の GPGPU 向け実装により得られる特異値，特異ベクトルの精度は，他の実装と同程度である．
- Bischof の方法の GPGPU 向け実装は他の実装と比べて高速である．Tesla C2050 マシンで，7680×7680 行列の特異値分解を行う場合，最適帯幅の Bischof の方法の GPGPU 向け実装は，Intel MKL の 3.9 倍，CULA および MAGMA の 1.8 倍高速となった．また，Tesla C2050 を用いる場合，CULA，MAGMA に逆変換を GPGPU で行うルーチンが実装されたとしても，Bischof の方法の GPGPU 向け実装がより高速である．
- Bischof の方法の GPGPU 向け実装は，数値実験に用いた環境では，帯幅が 64 から 96 の範囲にあるとき高速である．また，Tesla C2050 を使用する場合，帯幅が 16 の倍数でない場合には性能が低下する．
- 実行時間に占める村田法およびその逆変換の実行時間が長い．

また，以下の点について考察を行った．

- Bischof の方法の GPGPU 向け実装の実行性能
- CULA，MAGMA に逆変換ルーチンが実装された場合の，これらの実装と Bischof の方法の GPGPU 向け実装の実行時間に関する優劣
- 帯幅の実行性能への影響

さらに，これらの評価結果，考察をもとに Bischof の方法の GPGPU 向け実装の性能改善手法についても検討した．

今後の課題として，性能改善手法としてあげた内容の実装およびその評価があげられる．

謝辞　原稿を丁寧にお読みくださり，多くの的確なご指摘をくださった査読者の方々に心より感謝いたします．また，本研究を進めるにあたり日頃からお世話になっている神戸大学の谷口隆晴講師，深谷猛特命助教に感謝いたします．本研究は科学研究費補助金および独立行政法人科学技術振興機構（JST），戦略的創造研究推進事業（CREST）「ポストペタスケールに対応した階層モデルによる超並列固有値解析エンジンの開発」の助成を受けた．

© 2012 Information Processing Society of Japan

参考文献

[1] 深谷　猛，山本有作，畝山多加志，中村佳正：正方行列向け特異値分解の CUDA による高速化，情報処理学会論文誌コンピューティングシステム（ACS），Vol.2, No.2, pp.98–109 (2009).

[2] Bischof, C.H., Lang, B. and Sun, X.: Algorithm 807: The SBR Toolbox—software for successive band reduction, *ACM Trans. Mathematical Software* (*TOMS*), Vol.26, No.4, pp.602–616 (2000).

[3] Bischof, C.H., Lang, B. and Sun, X.: A framework for symmetric band reduction, *ACM Trans. Mathematical Software* (*TOMS*), Vol.26, No.4, pp.581–601 (2000).

[4] LAPACK (online), available from ⟨http://www.netlib.org/lapack/⟩ (accessed 2012-03-25).

[5] EM Photonics: CULA (online), available from ⟨http://www.culatools.com/⟩ (accessed 2012-03-25).

[6] MAGMA (online), available from ⟨http://icl.cs.utk.edu/magma/⟩ (accessed 2012-03-25).

[7] Dongarra, J.J., Sorensen, D.C. and Hammarling, S.J.: Block reduction of matrices to condensed forms for eigenvalue computations, *Journal of Computational and Applied Mathematics*, Vol.27, No.1, pp.215–227 (1989).

[8] NVIDIA: CUDA (online), available from ⟨http://developer.nvidia.com/category/zone/cuda-zone/⟩(accessed 2012-03-25).

[9] Golub, G. and Van Loan, C.: *Matrix Computation, 3rd edition*, Johns Hopkins University Press (1996).

[10] Jessup, E.R. and Sorensen, D.C.: A parallel algorithm for computing the singular value decomposition of a matrix, *Siam Journal on Matrix Analysis and Applications*, Vol.15, No.2, pp.530–548 (1994).

[11] Gu, M. and Eisenstat, S.C.: A Divide-and-Conquer Algorithm for the Bidiagonal SVD, *Siam Journal on Matrix Analysis and Applications*, Vol.16, No.1, pp.79–92 (1995).

[12] 岩崎雅史，阪野真也，中村佳正：実対称 3 重対角行列の高精度ツイスト分解とその特異値分解への応用，日本応用数理学会論文誌，Vol.15, No.3, pp.461–481 (2005).

[13] Murata, K. and Horikoshi, K.: A new method for the tridiagonalization of the symmetric band matrix, *Information Processing in Japan*, Vol.15, pp.108–112 (1975).

[14] Schreiber, R.S. and Van Loan, C.: A Storage-Efficient WY Representation for Products of Householder Transformations, *SIAM Journal on Scientific and Statistical Computing*, Vol.10, No.1, pp.53–57 (1989).

[15] Intel: Math Kernel Library (online), available from ⟨http://www.intel.com/software/products/mkl/⟩ (accessed 2012-03-25).

[16] I-SVD Library (online), available from ⟨http://www-is.amp.i.kyoto-u.ac.jp/lab/isvd/⟩ (accessed 2012-03-25).

[17] Langou, J.: All Reduce Algorithms: Application to Householder QR Factorization, *Proc. 2007 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applicatipns*, pp.103–106 (2007).

[18] 山本有作：TSQR アルゴリズムで生成される Compact WY 表現の合成について，日本応用数理学会 2011 年度年会講演予稿集，pp.95–96 (2011).

[19] Wu, Y.-J.J., Alpatov, P.A., Bischof, C. and Van De Geijn, R.A.: A parallel implementation of symmetric band reduction using PLAPACK, *Proc. Scalable Parallel Library Conference, Mississippi State University*, pp.1–16 (1996).

[20] Gansterer, W.N., Kvasnicka, D.F., Schwarz, K. and Ueberhuber, C.W.: Blocking techniques in symmetric eigensolvers, *Proc. 9th SIAM Conference on Parallel Processing for Scientific Computing* (1999).

廣田 悠輔　（学生会員）

1985 年生．2008 年東京工科大学コンピュータサイエンス学部卒業．2010 年名古屋大学大学院工学研究科計算理工学専攻（博士前期課程）修了．現在，神戸大学大学院システム情報学研究科計算科学専攻計算科学インテンシブコース（博士後期課程）在学中．信号処理に現れる線型計算の高速化技術の研究に従事．

橋本 拓也

1989 年生．2012 年神戸大学工学部情報知能工学科卒業．現在，同大学院システム情報学研究科システム科学専攻（博士前期課程）在学中．GPGPU を用いた量子画像認識に関する研究に従事．

山本 有作　（正会員）

1992 年東京大学大学院工学系研究科物理工学専攻修士課程修了．同年（株）日立製作所中央研究所入所．2001 年から 2002 年コロンビア大学ビジネススクール客員研究員．2003 年名古屋大学大学院工学研究科助手．同年名古屋大学博士（工学）．同講師，助教授，准教授を経て，現在，神戸大学大学院システム情報学研究科教授．並列計算機向け行列計算アルゴリズムおよび金融工学向け高速計算アルゴリズムの研究開発に従事．日本応用数理学会，SIAM，INFORMS 各会員．

ⓒ 2012 Information Processing Society of Japan