情報処理学会論文誌　コンピューティングシステム　Vol. 2　No. 2　98–109 (July 2009)

正方行列向け特異値分解のCUDAによる高速化

深谷　猛[†1]　山本有作[†1]
畝山　多加志[†2]　中村佳正[†3]

本論文ではGPGPU向けの統合開発環境CUDAを用いた，正方行列の特異値分解の高速化について報告する．正方行列の特異値分解では，計算対象の行列を二重対角行列に変換してから特異値分解を行い，その後逆変換を行うことで，もとの行列の特異値分解を得る．本論文ではCUDAのBLASライブラリ（CUBLAS）の中の高性能なSGEMM（行列乗算ルーチン）を効率的に利用することで，比較的少ないコストで大幅な高速化を行うことを目指し，演算の大部分がBLASによって行われる二重対角化と逆変換部分をGPUを用いて高速化した．実装にあたっては，行列乗算を中心に二重対角化が可能なBischofの手法がGPU向けに適していることを簡単な性能予測を通して確認し，この手法を採用した．また，各計算ステップにおけるCPUとGPUとの仕事の適切な分担や計算のオーバラップについても考慮した．GPUとしてNVIDIAのGeForce8800 GTXを用いた性能評価の結果，CPU（Intel Core2 Duo 1.86 GHz 2コア使用）のみで計算する場合と比べて，5,120次元の正方行列の特異値分解の計算が約4倍高速化できることを確認した．

Acceleration of the Singular Value Decomposition Algorithm for Square Matrices Using CUDA

Takeshi Fukaya,[†1] Yusaku Yamamoto,[†1]
Takashi Uneyama[†2] and Yoshimasa Nakamura[†3]

In this paper, we report the result of acceleration of computing the singular value decomposition (SVD) for a square matrix using CUDA, which is an integrated development environment for GPGPU. Computing of the SVD for a square matrix consists of the following three parts: bidiagonalization of the input matrix, the SVD of the bidiagonal matrix, and inverse transformation. Among them, we accelerate the first and the third step using GPU. This is because it is easy to use the CUBLAS, the BLAS library provided in CUDA, in these two steps. Through simple performance prediction, we assessed that the Bischof's method, in which bidiagonalization can be computed with matrix multiplications, is effective for computation using GPU. Therefore we implemented the algorithm for the SVD based on such method. When computing the SVD of a $5{,}120 \times 5{,}120$ matrix, we obtained about four times speedup using a GPU over using only a CPU (Intel Core2 Duo, 1.86 GHz, using 2 cores).

1. はじめに

近年，GPU（Graphics Processing Unit）の進化は著しく，たとえば，NVIDIA社のGeForce8800 GTXは単精度演算で汎用CPUの十数倍の500 GFLOPS程度を達成する．この演算能力をグラフィックス演算に限らず一般の高性能計算に活用するGPGPU（General Purpose GPU）が注目されている[1)]．さらに，2006年にNVIDIA社からGPGPUのための統合開発環境CUDA[2)]が発表された．CUDAでは標準のC言語の簡単な拡張によりNVIDIA社のGPUを汎用計算に使えるようになっている．また，チューニング済みのBLASライブラリなども提供されている[3)]ため，GPUを用いた行列計算のプログラミングが非常に容易になった[4)]．

筆者らは，以前から行列計算を汎用CPUとGPUのような浮動小数点演算に長けたコプロセッサ（あるいはアクセラレータ）の併用により高速化することに興味を持ち，研究を行ってきた[5)]．本論文では正方行列の特異値分解計算をCUDAを用いて高速化し，その性能評価と今後に向けた課題の報告を行う．

任意のn次実正方行列Aは，n次直交行列U，Vとn次対角行列$\Sigma = \mathrm{diag}(\sigma_1, \cdots, \sigma_n)$（ただし，$\sigma_1 \geq \cdots \geq \sigma_n \geq 0$）を用いて，

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

と書ける[6)]．これをAの特異値分解と呼ぶ．特異値分解は，情報検索や画像処理など様々な分野で応用される重要な行列計算の一種であり，計算の高速化が望まれている．

一般的に正方行列の特異値分解は，まず計算対象の行列を二重対角行列に変換（二重対角化）し，次に二重対角行列の特異値分解を行い，最後に二重対角化の逆変換を行う，という手

†1 名古屋大学大学院工学研究科計算理工学専攻
Department of Computational Science and Engineering, Graduate School of Engineering, Nagoya University
†2 京都大学化学研究所
Institute for Chemical Research, Kyoto University
†3 京都大学大学院情報学研究科
Graduate School of Infomatics, Kyoto University

順で計算される．その際，計算時間の大部分が二重対角化とその逆変換の計算で占められるので，これらの部分の高速化が最優先となる．また，二重対角化とその逆変換の計算は行列ベクトル積などの基本行列演算が中心なため，CUDA で提供されている GPU 向けにチューニングされた BLAS（CUBLAS）の利用が可能となっている．そこで，本論文では二重対角化とその逆変換部分に絞って，CUDA の CUBLAS の利用を中心とした高速化を行うことにする．

CUBLAS では，行列乗算（SGEMM）などの Level-3 BLAS の性能は非常に高いが，一方で行列ベクトル積（SGEMV）などの Level-2 BLAS の性能が著しく劣っている．この特徴をふまえて，本論文では二重対角化とその逆変換の計算に，演算量は増加するができるだけ Level-3 BLAS により計算を行うアルゴリズムを採用し，より GPU の性能を引き出すようにして実装を行った．また，現状では GPU の高い演算性能は単精度に限られているので，本論文でも GPU を使用する部分の計算は単精度で行う形をとった．そのため，計算結果の精度についても評価を行った．

本論文の構成は以下のとおりである．まず，2 章で GPGPU 向けの統合開発環境である CUDA についての簡単な説明を行う．次に，3 章で正方行列の特異値分解の計算と CUDA による高速化手法について述べる．4 章で性能評価の結果を報告し，5 章で考察を行う．6 章では関連研究について触れ，最後に 7 章でまとめを行う．

2. GPGPU 向け環境 CUDA

本章では GPU 向けの統合開発環境である CUDA の説明を簡単に行う．

2.1 CUDA の概要

CUDA（Compute Unified Device Architecture）は NVIDIA 社が提供する C 言語向けの統合開発環境で，nvcc コンパイラ，BLAS[7] や FFT などのライブラリ，シミュレータなどから構成される．

2.2 GPU の利用方法

CUDA 環境で GPU を使って行列計算を行う方法としては，CUDA の BLAS（CUBLAS）を使用する方法と，プログラム全体を移植して nvcc コンパイラを使用する方法の 2 通りが考えられる．それぞれの特徴を以下で簡単に述べる．

2.2.1 CUBLAS

CUBLAS では通常の BLAS ルーチンの一部に加えて，グラフィックボード上のメモリの確保・開放，メインメモリとグラフィックボード上のメモリ間のデータ転送用ルーチンが提供されている．CUBLAS を使う場合，ユーザは，あらかじめ確保したボード上のメモリに計算に必要なデータを転送し，次に計算を GPU を使って行い，最後に計算結果をボード上のメモリからメインメモリに転送する形でプログラミングをすることになる．なお，ボード上のメモリはプログラムの終了まで保持されるので，繰り返し計算を行う場合には，データ転送を必要最小限に抑えることで，より高い性能を得ることが可能である．

表 1　CUBLAS の性能（GFLOPS）
Table 1　Performance of CUBLAS (in GFLOPS).

n	SGEMV	SGEMM
256	0.65	89.2
512	1.56	114
1,024	3.43	120
2,048	7.20	117
4,096	12.7	116

2.2.2 nvcc コンパイラ

nvcc コンパイラは拡張された C 言語のソースコードから CPU と GPU で実行可能なバイナリファイルを生成する．したがって，従来のプログラムを GPU を使用するように拡張することで，複雑な計算にも GPU の使用が可能となる．しかし，一方で GPU の性能を十分に引き出すためには様々な工夫が必要となるため，従来の線形計算プログラムを GPU の性能を引き出せる形で拡張することは容易ではない．

2.3 本研究で用いる GPU・CUBLAS

本研究では，GPU として NVIDIA 社の GeForce8800 GTX を搭載したグラフィックボードを用いる．GeForce8800 GTX は，各々が 8 個のプロセッサ，8,192 個のレジスタ，16 KB の共有メモリを持つ SIMD 型並列計算ユニット 16 個を 1 チップ上に搭載し，1.35 GHz のクロックで動作する．CUDA を使った場合，利用可能なピーク性能は単精度演算で 345.6 GFLOPS である．倍精度演算は行えない．

また，本研究では CUBLAS ver. 1.0 を用いる．この中の SGEMV と SGEMM の性能を**表 1** に示す．ここで，SGEMV では n 次正方行列と n 次元ベクトルの積を，SGEMM では n 次正方行列どうしの積をそれぞれ計算している．なお，両者とも GPU 上での計算時間のみから性能を計算しており，メインメモリとグラフィックボード上のメモリとの間のデータ転送時間は考慮していない．

3. 正方行列向け特異値分解アルゴリズムと CUDA による高速化手法

本章では，はじめに正方行列の特異値分解の計算方法の説明を行い，その後に CUDA に

よる高速化の方法を検討する．

3.1　正方行列向けの特異値分解の計算方法

正方行列の特異値分解の計算は，一般的に以下の手順で行われる[8]．

(a)　二重対角化：$A = U_1 B V_1^T$ と A を直交行列 U_1，V_1 と下二重対角行列 B の積に分解する．

(b)　二重対角行列の特異値分解：B の特異値分解，$B = U_2 \Sigma V_2^T$ を求める．

(c)　逆変換：$U = U_1 U_2$，$V = V_1 V_2$ をそれぞれ計算する．これにより，$A = U \Sigma V^T$ が A の特異値分解となる．

ここで，上記のステップ中の行列 U_1，V_1 は必ずしも陽的に保持する必要はなく，これらの行列と他の行列の積が計算できればよい．実際の計算では，複数の鏡像変換（ハウスホルダ変換）の積として陰的に保持することが一般的である．また，(b) の二重対角行列の特異値分解アルゴリズムとしては QR 法，分割統治法，MR^3 アルゴリズム，I-SVD など数多くのアルゴリズムが存在している．

各ステップの演算量は，(a) と (c) がそれぞれ $O(n^3)$，(b) はアルゴリズムによって異なり $O(n^2)$〜$O(n^3)$ となる．汎用 CPU（表 7 の環境で 2 コア使用）で実際に計算を行った際の，各ステップの計算時間の占める割合を**図 1** に示す．

3.2　CUDA による高速化の方針

CUDA を用いる方法として，CUBLAS を使用する方法と，プログラムを GPU を使う形に拡張して nvcc コンパイラを使用する方法の 2 通りがあるが，すでに GPU 向けにチューニングされた CUBLAS（特に高性能な SGEMM）の使用を中心にすることで，少ないコストで高い効果が得られると考えられる．また，BLAS の利用を中心に高速化を行えば，GPU 以外の環境でも BLAS ライブラリが提供されてさえいれば，今回と同様の手法を適用することが可能となる．したがって，以下ではできるだけ CUBLAS を利用する方針で各ステップにおける GPU の使用方法を検討する．

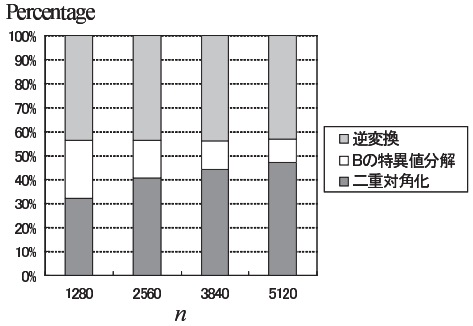


図 1　特異値分解計算における各ステップの実行時間の割合
Fig. 1　Percentage of computation time for each step in computing the SVD.

3.2.1　(a) 二重対角化

従来法による二重対角化の計算は，大部分が行列ベクトル積などの Level-2 BLAS を用いて行われるので，CUBLAS の使用が可能である．また，行列乗算を用いて二重対角化をする方法も存在し，これを用いれば CUBLAS の高性能な SGEMM の適用も可能である．また，図 1 から分かるようにこの部分の計算時間は全体に占める割合が非常に大きいので，高速化の必要性が高い．

3.2.2　(b) 二重対角行列の特異値分解

この部分は全実行時間に占める割合が最も小さいため，GPU を利用しても全体の高速化への効果は小さい．また，BLAS のみで書けないアルゴリズムが多いため，高速化のコストは最も高い．さらに重要な点として，(a)，(c) はほとんどすべての計算が直交変換で単精度化しても安定性が損なわれることはないが，本部分は連立一次方程式の求解（MR^3，I-SVD），非線形方程式の求解（分割統治法）が必要であり，丸め誤差の影響を受けやすい．

3.2.3　(c) 二重対角化の逆変換

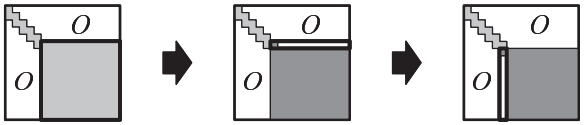
この部分の計算は，従来法のままでもすべての計算が Level-2 BLAS により実行可能である．さらに，ブロック化を行うことで行列乗算を中心に計算を行うことも可能である．図 1 から，計算時間の全体に占める割合は (a) に次いで大きく，特に大きなコストを払うことなく CUBLAS が使用可能になっている．

3.2.4　方針のまとめ

以上の議論から，CUBLAS の利用を中心にして，できるだけコストを抑えて高速化を行うには，(a) と (c) の部分には GPU を使って高速化を行い，(b) の部分は精度の面も考慮して，GPU による高速化は行わず CPU のみで計算することが適当であると判断した．したがって，以下では (a) と (c) の部分を GPU を用いてどのように計算するかを検討する．

3.3　二重対角化手法の選択

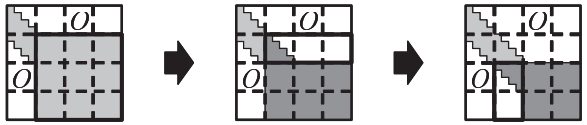
前章で述べたように，本論文の高速化の対象は二重対角化とその逆変換である．従来の二重対角化手法では，すでに述べたとおり計算の中心が Level-2 BLAS となっている．このままでも CUBLAS の適用は可能であるが，CUBLAS では Level-3 BLAS の SGEMM の性能が非常に高いので，これの使用が望まれる．一方で，Level-3 BLAS を中心にした二



（右から鏡像変換を作用）（左から鏡像変換を作用）

図 2　鏡像変換による二重対角化

Fig. 2　Bidiagonalization using Householder transformations.



（右からブロック鏡像変換を作用）（左からブロック鏡像変換を作用）

図 3　ブロック鏡像変換による下三角帯行列化

Fig. 3　Reduction to a lower triangular band matrix using blocked Householder transformations.

重対角化の手法が Bischof らによって提案されている．ただし，この手法では，逆変換のコストが増加するという問題が生じる．本章では，両者の手法の計算の特徴を簡単に説明し，GPU 上での性能を予測することで，適した手法の選択を行う．

3.3.1　従来の二重対角化手法

従来の二重対角化は，式 (1) の形で表される鏡像変換（ハウスホルダ変換）を計算し，A の左右から順番に作用させることで行われる．

$$H = I - t\boldsymbol{y}\boldsymbol{y}^T. \tag{1}$$

ここで，t はスカラー，$\boldsymbol{y}$ はベクトルである．この計算の様子を**図 2** に示す．

鏡像変換を A に作用させる計算

$$\left(I - t\boldsymbol{y}\boldsymbol{y}^T\right) A \to A,$$

は次の 2 つの Level-2 BLAS の演算により行われる．

- 行列ベクトル積（SGEMV）：$\boldsymbol{y}^T A \to \boldsymbol{w}^T$.
- 行列のランク 1 更新（SGER）：$A - t\boldsymbol{y}\boldsymbol{w}^T \to A$.

この部分に CUBLAS を使うことが可能だが，表 1 にあるように Level-2 BLAS では GPU の性能を十分に生かせていない．また，これらの演算をブロック化することで，最大半分まで Level-3 BLAS を使うことが Dongarra らの提案により可能になっている[9] が，残りの半分は依然として Level-2 BLAS のままである．

3.3.2　Bischof の手法による二重対角化

行列乗算（Level-3 BLAS）を使って二重対角化を行う手法が Bischof らによって提案されている[10),11)]．これは，以下のように二重対角化を 2 段階に分けて行う手法である．

(a-1)　下三角帯行列化：$A = U_{11} C V_{11}^T$ と A を直交行列 U_{11}，V_{11} と半帯幅が L_B の下三角帯行列 C の積に分解する．

(a-2)　村田法（下三角帯行列の二重対角化）：$C = U_{12} B V_{12}^T$ と C を直交行列 U_{12}，V_{12} と下二重対角行列 B の積に分解する．

ただし，二重対角化を 2 段階に分けたため，その逆変換の計算も以下のように 2 段階分必要になる．

(c-1)　（村田法の）逆変換：$U' = U_{12} U_2$，$V' = V_{12} V_2$ をそれぞれ計算する．

(c-2)　（帯行列化の）逆変換：$U = U_{11} U'$，$V = V_{11} V'$ をそれぞれ計算する．これにより，$A = U \Sigma V^T$ が A の特異値分解となる．

各ステップの演算量は，$n \gg L_B$ の場合，(a-1) の演算量が $(8/3)n^3$，(a-2) の演算量が $8n^2 L_B$ となり，二重対角化の計算の大部分は (a-1) が占めることになる．また，(c-1) と (c-2) の演算量はともに $4n^3$ である．

この手法で二重対角化の大部分を占める (a-1) の部分の計算は，式 (2) の形で表されるブロック鏡像変換を計算し，A の左右から順番に作用させることで行われる．

$$Q = I - YTY^T. \tag{2}$$

ここで，Y は一般に長方行列，T は三角行列となる．この計算の様子を**図 3** に示す．

ブロック鏡像変換を A に作用させる計算

$$\left(I - YTY^T\right) A \to A,$$

は，次の 3 つ Level-3 BLAS の演算により行われる．

- 一般行列どうしの乗算（SGEMM）：$Y^T A \to W$.
- 一般行列と三角行列の乗算（STRMM）：$TW \to W$.
- 一般行列どうしの乗算（SGEMM）：$A - YW \to A$.

このように二重対角化を 2 段階にすることで，その計算の大部分を占める (a-1) が行列乗算中心で計算可能になる点が，この手法のメリットである．

一方，デメリットとして，逆変換も 2 段階になってしまい，従来法の逆変換の 2 倍の計算が必要となってしまうという点がある．ただし，逆変換はすべて行列乗算を用いて計算できる．

表 2　予測に用いる CUBLAS の性能（GFLOPS）
Table 2　Performance data of CUBLAS for estimation (in GFLOPS).

n	SGEMV	SGEMM
1,280	2.44	82.3
2,560	3.25	89.3
3,840	3.25	89.2
5,120	3.54	95.2

この Bischof の手法を用いて汎用のシングルコア CPU 上で特異値分解を行うと，二重対角化は従来法に比べて高速化されるが，逆変換までを含めると従来法よりも遅くなってしまうという結果が報告されている[12]．ただし，並列計算機など Level-2 BLAS と Level-3 BLAS の性能の差が大きくなるほど有効な手法となりうると期待されている．

3.3.3　GPU 上での 2 つの二重対角手法の性能予測

CUBLAS では Level-2 BLAS の SGEMV と Level-3 BLAS の SGEMM の性能差が非常に大きいので，Bischof の手法の適用を検討する価値は高い．よって，以下で GPU を用いて従来法で二重対角化を行った場合と Bischof の手法を用いて二重対角化を行った場合の簡単な性能予測を行い，適切な手法を選択する．

GPU 上での性能予測を行うために，まず CUBLAS の SGEMV と SGEMM の性能を測定する．ただし，従来法による二重対角化と Bischof の手法における下三角帯行列化の演算は，それぞれ図 2，図 3 に示したような計算によって行われ，図から分かるように，両者とも計算が進むにつれて計算対象の行列サイズが次第に小さくなる．このことをふまえて，ある n から次第にサイズを小さくしながら繰り返し関数を実行し，合計の実行時間と演算量から性能を求める．この結果を**表 2** に示す．

また，両手法の各ステップの演算の種類と演算量は**表 3** のとおりである．ただし，従来法の (a) の計算はブロック化によりその半分の演算を行列乗算により実行可能であるので，そのようにした．

表 2 と表 3 の値を用いて，$n = 1{,}280$，2,560，3,840，5,120 の場合の実行時間を予測する．ただし，(a-2) の村田法の時間の予測は困難なため，ここでは実際に CPU 上で計算した際の実行時間を代用する．この結果を**表 4** に示す．非常に粗い性能予測ではあるが，この結果から，GPU では Bischof の手法によりできるだけ高性能な CUBLAS の SGEMM を利用することが有効であることが確認できる．また，行列サイズが大きくなるほど効果が大きくなることが期待できる．よって，本研究では Bischof の手法を用いて二重対角化を行うこととする．

表 3　従来法と Bischof の手法の各計算ステップの演算の種類と演算量
Table 3　Computational cost and BLAS level of each step in the conventional method and the Bischof's method.

演算の種類	従来法		Bischof	
SGEMV	(a) の半分	$\frac{4}{3}n^3$	-	
SGEMM	(a) の半分	$\frac{4}{3}n^3$	(a-1)	$\frac{8}{3}n^3$
	(c)	$4n^3$	(c-1)	$4n^3$
			(c-2)	$4n^3$
other	-		(a-2)	$8n^2L_B$

表 4　実行時間の予測（秒）
Table 4　Estimated computation time (in sec.).

n	従来法			Bischof（$L_B = 64$）		
	(a)	(c)	(a) + (c)	(a)	(c)	(a) + (c)
1,280	1.18	0.10	1.28	1.25	0.20	1.45
2,560	7.13	0.75	7.88	5.56	1.50	7.06
3,840	24.1	2.54	26.6	13.4	5.08	18.5
5,120	52.4	5.64	58.1	25.2	11.3	36.5

3.4　具体的な実装方法

本節では，前章の議論に基づき，具体的な実装方法について

- CPU と GPU との仕事の分担と両者の間でのデータ通信の必要性．
- GPU の使用方法（CUBLAS を使うのか，プログラムを拡張して nvcc コンパイラを使うのか）．

の 2 点を中心に説明する．

3.4.1　(a-1) 下三角帯行列化

下三角帯行列化の計算は，ブロック鏡像変換の計算と計算されたブロック鏡像変換を A に作用させる計算の 2 つからなる．ブロック鏡像変換の計算は平方根の計算を含み，また行列ベクトル積が計算の中心となる．したがって，この部分は CPU で計算を行う．一方，ブロック鏡像変換の作用は行列乗算のみで計算可能で CUBLAS の SGEMM が適用できるので GPU で計算を行う．この場合，データ転送量を最小化するには，行列 A は GPU のメモリ上に置くことが必要となる．その結果，

(1)　GPU から CPU へブロック鏡像変換の計算に必要な A の一部分を転送する．
(2)　CPU でブロック鏡像変換を計算する．
(3)　CPU から GPU へ計算されたブロック鏡像変換を転送する．

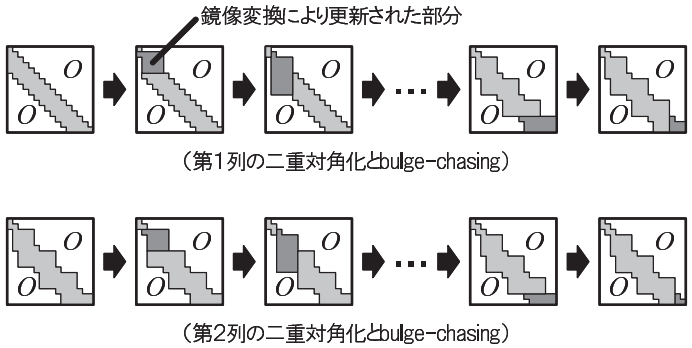


図 4　村田法による二重対角化
Fig. 4　Bidiagonalization using Murata's method.

(4)　GPU でブロック鏡像変換を作用させて A を更新する．((1) へ戻る)
というように，CPU と GPU との双方向の逐次的なデータ転送を含む形での実装が必要になる．

3.4.2　(a-2) 村田法

村田法では，下三角帯行列の左右からサイズの小さい鏡像変換を繰り返し作用させることで下二重対角化を行う[13]．具体的には，**図 4** のような bulge-chasing と呼ばれる演算により，第 1 列から順に二重対角化を行う．一般的に帯行列の構造を利用してメモリに格納するため，既存の BLAS ルーチンの利用は不可能である．また，BLAS ルーチンが利用できる形で格納したとしても，サイズの小さい鏡像変換を繰り返すため，CUBLAS の SGEMM の利用は困難である．

一方で，図 4 から分かるように鏡像変換の作用する部分が局所的であるため，パイプライン型の並列化（たとえば，第 1 列に対する bulge-chasing が十分進んだら，第 2 列に対する bulge-chasing を開始できる）が可能であることが知られている[14]．したがって，この方式を用いて GPU 上で並列計算を行うようにプログラムを拡張し，nvcc コンパイラを利用する．その際に，16 個の計算ユニット単位でパイプライン型の並列化を行い，ユニット内の 8 個のプロセッサで鏡像変換の作用（Level-2 BLAS 相当の演算）を並列実行する形で実装する．

3.4.3　(b) 二重対角行列の特異値分解

二重対角行列の特異値分解計算は，本論文では GPU を使わずに行う方針なので，既存のアルゴリズムを用いることにする．特に今回は，高速・高精度なアルゴリズムとして最近注目されている I-SVD[15] を用いることにする．なお，この部分の計算のみ倍精度で行う．

表 5　村田法の逆変換の実装方法ごとの実行時間（秒）
Table 5　Computation time for inverse transformation of Murata's method (in sec.) on each implementation.

n	GPU のみ	CPU と GPU
1,280	0.18	0.18
2,560	1.15	0.79
3,840	3.56	2.55
5,120	7.99	5.81

3.4.4　(c-1) 村田法の逆変換

村田法の逆変換は，(a-2) で用いた鏡像変換を順番に (b) で得られた U と V に作用させることで計算される．行列 U，V は GPU のメモリ上に置いて計算を進める．ここで，個々の鏡像変換を単独で作用させる場合は行列ベクトル積による演算となるため，複数個の鏡像変換をあらかじめ行列に合成してから，行列乗算により作用させることにする．したがって，CPU で複数個の鏡像変換を合成し，結果を GPU に転送して，GPU で CUBLAS の SGEMM を用いて合成された結果を作用させる形をとる．なお，鏡像変換は (a-2) の結果からすべて事前に分かっているので，CPU は GPU の計算と同期することなく計算をすることができ，CPU と GPU の計算のオーバラップが可能である．予備実験として，CPU と GPU で計算をオーバラップさせる場合とすべて GPU で計算を行う場合の計算時間を測定した結果，**表 5** のようになった．この結果より，オーバラップの有効性が確認できたので，この形で実装を行う．

3.4.5　(c-2) 下三角帯行列化の逆変換

下三角帯行列化の逆変換は，(a-1) で用いたブロック鏡像変換を (c-1) で更新された U と V に作用させることで計算される．ブロック鏡像変換の作用は (a-1) の部分でも述べたように行列乗算のみで計算可能であり，そのまま CUBLAS の SGEMM が適用できる．したがって，行列 U，V を GPU のメモリ上に置き，GPU で SGEMM により計算する形で実装する．

3.4.6　実装方法のまとめ

上で説明した各ステップの実装方法について，CPU と GPU の仕事の分担方法を中心として，**表 6** にまとめた．なお，CPU と GPU との間でのデータ転送はステップ間では当然必要になるが，それ以外に (a-1) では定期的な双方向の通信があり，また (c-1) では CPU から GPU への 1 方向の転送が定期的に必要となる．

データ型については，GPU を使用する (a-1)，(a-2)，(c-1)，(c-2) は単精度で計算を行い，一方で (b) は倍精度で行うようにする．

表 6　CPU と GPU の仕事の分担
Table 6　Assigned tasks for CPU and for GPU.

手順	CPU	通信	GPU
(a-1)	ブロック鏡像変換の計算	↔	ブロック鏡像変換の作用 （CUBLAS の SGEMM）
(a-2)	——		鏡像変換の計算と作用 （nvcc コンパイラ）
(b)	既存のルーチン （I-SVD）		——
(c-1)	鏡像変換の合成	→	合成結果の作用 （CUBLAS の SGEMM）
(c-2)	——		ブロック鏡像変換の作用 （CUBLAS の SGEMM）

表 7　性能評価に使用した計算機環境
Table 7　Computational environments used in the performance evaluation.

項目	条件
CPU	Intel Core2 Duo 1.86 GHz
メモリ	2 GB
OS	openSUSE Linux 10.2（64 ビット版）
BLAS	Intel Math Kernel Library ver. 8.1
LAPACK	Intel Math Kernel Library ver. 8.1
コンパイラ	gcc ver. 4.1.2（オプション -O3）
	g77 ver. 3.3.5（オプション -O3）
	Intel Fortran コンパイラ ver. 9.1（オプション -O3）
GPU	NVIDIA GeForce8800 GTX
GPU メモリ	768 MB
CUBLAS	CUBLAS ver. 1.0
CUDA コンパイラ	nvcc ver. 1.0

4. 性能評価

4.1 評価条件

前章までの内容に基づいて，CUDA を用いて正方行列の特異値分解を計算するプログラムを作成し，性能評価を行った．なお，二重対角行列の特異値分解には，I-SVD を使用した．また，二重対角行列の特異値分解の計算のみを倍精度で行い，それ以外の部分は GPU の使用にかかわらず，すべて単精度で行った．性能評価に使用した計算機環境は**表 7** のとおりである．

性能評価では $[-0.5, 0.5]$ の一様乱数を要素に持つ $n \times n$ の正方行列のすべての特異値とすべての特異ベクトルの計算を行い，全体および各部分の計算時間の測定をした．評価に用いた行列のサイズは $n = 1{,}280$，2,560，3,840，5,120 の 4 種類，Bischof のアルゴリズムにおける半帯幅 L_B は $L_B = 32$，64，128 の 3 種類とした．比較対象としては，CPU 上での従来法（Intel Math Kernel Library の LAPACK）と Bischof の手法を用いた．

表 8　$n = 1{,}280$ のときの実行時間（秒）
Table 8　Computation time (in sec.) for $n = 1{,}280$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 32$)	Bischof ($L_B = 32$)
(a-1)	-	0.46	0.15
(a-2)	-	0.34	0.92
(a) の合計	2.07	0.80	1.07
(b)	1.56	1.57	1.56
(c-1)	-	5.84	0.30
(c-2)	-	0.60	0.12
(c) の合計	2.24	6.44	0.42
合計	5.87	8.81	3.05
(b) 以外の合計	4.31	8.01	1.49

表 9　$n = 2{,}560$ のときの実行時間（秒）
Table 9　Computation time (in sec.) for $n = 2{,}560$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 32$)	Bischof ($L_B = 32$)
(a-1)	-	3.37	0.89
(a-2)	-	1.36	1.49
(a) の合計	15.4	4.73	2.38
(b)	6.39	6.34	6.31
(c-1)	-	41.7	1.89
(c-2)	-	4.87	0.84
(c) の合計	13.4	46.6	2.73
合計	35.2	57.6	11.4
(b) 以外の合計	28.8	51.3	5.11

4.2 評価結果

$n = 1{,}280$，2,560，3,840，5,120 の場合における実行時間は，それぞれ**表 8**，**表 9**，**表 10**，**表 11** に示したとおりである．ここで (a) などの記号は 3 章で説明した各ステップに対応す

表 10　$n = 3{,}840$ のときの実行時間（秒）
Table 10　Computation time (in sec.) for $n = 3{,}840$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 64$)	Bischof ($L_B = 64$)
(a-1)	-	9.86	2.79
(a-2)	-	6.35	3.32
(a) の合計	48.7	16.2	6.11
(b)	14.1	14.1	14.0
(c-1)	-	101	6.16
(c-2)	-	14.6	3.18
(c) の合計	38.0	116	9.34
合計	101	146	27.8
(b) 以外の合計	86.7	132	13.8

表 11　$n = 5{,}120$ のときの実行時間（秒）
Table 11　Computation time (in sec.) for $n = 5{,}120$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 64$)	Bischof ($L_B = 64$)
(a-1)	-	22.8	5.70
(a-2)	-	11.6	5.25
(a) の合計	115	34.4	11.0
(b)	25.3	25.1	25.1
(c-1)	-	223	11.6
(c-2)	-	33.2	5.61
(c) の合計	80.8	256	17.2
合計	221	316	53.3
(b) 以外の合計	196	291	28.2

る．また，Bischof の手法における L_B の値は，GPU 使用時に最も速かった場合を載せた．$n = 5{,}120$ の正方行列のすべての特異値と特異ベクトルを求める場合，CPU のみを使って従来法により計算する場合と比べて，GPU を併用して Bischof の手法により計算すると，計算全体で約 4 倍の高速化が確認できる．また，特に今回高速化の対象とした部分のみの計算時間で比べると，約 7 倍の高速化が確認できる．

また，実際の計算により得られた結果の精度は，**表 12**，**表 13**，**表 14**，**表 15** のようになった．ここで，残差は $||A - U\Sigma V^T||_F$，U の直交性は $||U^T U - I||_F$，V の直交性は $||V^T V - I||_F$ の値となっている．

表 12　$n = 1{,}280$ のときの計算精度
Table 12　Accuracy of computation for $n = 1{,}280$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 32$)	Bischof ($L_B = 32$)
残差	6.07E-04	1.04E-03	1.32E-03
U の直交性	3.65E-05	9.58E-05	1.14E-04
V の直交性	3.55E-05	9.59E-05	1.16E-04

表 13　$n = 2{,}560$ のときの計算精度
Table 13　Accuracy of computation for $n = 2{,}560$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 32$)	Bischof ($L_B = 32$)
残差	1.63E-03	2.83E-03	3.87E-03
U の直交性	6.52E-05	1.86E-04	2.29E-04
V の直交性	6.47E-05	1.86E-04	2.30E-04

表 14　$n = 3{,}840$ のときの計算精度
Table 14　Accuracy of computation for $n = 3{,}840$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 64$)	Bischof ($L_B = 64$)
残差	2.89E-03	4.62E-03	9.41E-03
U の直交性	9.11E-05	2.43E-04	3.04E-04
V の直交性	9.15E-05	2.46E-04	3.07E-04

表 15　$n = 5{,}120$ のときの計算精度
Table 15　Accuracy of computation for $n = 5{,}120$.

計算機	CPU（2 コア）		CPU と GPU
手法	従来法	Bischof ($L_B = 64$)	Bischof ($L_B = 64$)
残差	4.36E-03	6.99E-03	9.41E-03
U の直交性	1.20E-04	3.19E-04	4.00E-04
V の直交性	1.18E-04	3.22E-04	4.03E-04

最後に Bischof の手法において L_B による各部分の計算時間の違いを**表 16** に示す．なお，これは GPU を併用した場合の結果であり，計算時間が L_B に依存しない (b) の二重対角行

表 16　$n = 5{,}120$ における L_B ごとの計算時間（秒）
Table 16　Computation time (in sec.) with each L_B for $n = 5{,}120$.

L_B	32	64	128
(a-1)	5.67	5.70	7.28
(a-2)	3.43	5.25	10.20
(c-1)	13.7	11.6	11.2
(c-2)	6.24	5.61	5.39
(b) 以外の全体	29.0	28.2	34.0

表 17　Bischof の手法の各ステップの GPU による高速化率
Table 17　Speedup by GPU on each step of Bischof's method.

n	1,024	2,560	3,840	5,120
(a-1)	2.64	3.32	3.54	4.00
(a-2)	0.57	1.33	1.91	2.20
(c-1)	13.73	21.04	19.76	19.18
(c-2)	4.98	5.62	5.72	5.92
(b) 以外の全体	3.65	8.15	9.57	10.31

列の特異値分解の計算時間は省いている．

5. 考　　察

性能評価で得られた結果を基にして，以下のような考察を行った．

5.1 Bischof の手法の各ステップごとの GPU による高速化率

$n = 5{,}120$，$L_B = 64$ の場合について，Bischof の手法の各ステップごとの GPU の使用による高速化率（CPU 2 コア使用時との比）を**表 17** にまとめた．この表より，以下の考察を得た．

- (c-1) の村田法の逆変換の計算では，まず CPU で CUBLAS の性能が期待できるサイズの行列にまとめ，それを CUBLAS を用いて GPU 上で作用させ，さらに両者の計算をオーバラップさせたので，GPU の使用による高速化率がとても大きくなったと考えられる．
- (a-1) の下三角帯行列化と (c-2) の下三角帯行列化の逆変換の計算では，CUBLAS の使用が中心となっているので，一定の高速化が得られた．ただし，(a-1) では CPU によるブロック鏡像変換の計算があり，CPU と GPU との間での双方向のデータ転送があるため，GPU の使用による高速化率は (c-2) に比べて劣っている．
- (a-2) の村田法の計算では，本論文で唯一，プログラムを GPU 向けに拡張して nvcc コンパイラを使用した部分である．しかし，アルゴリズムが Level-2 BLAS 相当の演算からなるため，あまり GPU の使用による高速化がなされていない．

5.2 半帯幅 L_B の依存性

表 16 で示された Bischof の手法の各ステップの計算時間の L_B による変化について，以下のような考察を行った．

- (a-2) の村田法は演算量が $O(L_B n^2)$ であるので，この結果は自然であるといえる．
- (c-1) の村田法の逆変換と (c-2) の下三角帯行列化の逆変換では演算量は L_B によらず一定である．ただし，L_B によって CUBLAS を使う際の行列サイズが変わるので，L_B が大きくなるほど CUBLAS の性能が高くなり，時間が短縮されていると思われる．
- (a-1) の下三角帯行列化では，(c-1) や (c-2) と同様に L_B が大きくなると CUBLAS の性能が高くなり，GPU でブロック鏡像変換を作用させる時間は短縮されていると予想できるが，一方で，L_B が大きくなると CPU でブロック鏡像変換を計算する時間が増加し，前者による計算時間の短縮よりも後者による計算時間の増加の方が大きくなり，結果として (a-1) 全体の計算時間が L_B とともに増加していると考えられる．

5.3 計 算 精 度

計算結果の精度に関する簡単な考察を行う．表 12，表 13，表 14，表 15 より，CPU のみで従来法により計算した場合に比べて，GPU を併用して Bischof の手法により計算した場合の方が必ず精度が悪化していることが確認できる．この原因として，以下の 2 点が考えられる．

- Bischof の手法を用いると，従来法に比べて演算回数が増加するため，丸め誤差の影響が大きくなり，精度が悪化する．
- GPU の浮動小数点演算は IEEE の規格に完全に準拠していないため，GPU を使わない場合に比べて計算精度が悪化する．

4 つの表から，前者による精度悪化の方が後者による精度悪化に比べて大きいことが確認できる．ただし，精度の悪化は両方合わせてもたかだか 2 倍から 4 倍程度であり，実用面での問題は小さいと考えられる．

5.4 性能予測による二重対角化手法の選択結果

本論文では，簡単な性能予測をもとにして，Bischof の手法を選択した．この点について簡単に考察する．

GPU 上における Bischof の手法の予測時間と実際の実行時間を**表 18** にまとめた．なお，表には今回予測に関係した計算ステップの時間のみを個別に載せている．この結果について

表 18　予測時間と実際の計算時間の比較（秒）
Table 18　Comparison between estimated time and execution time (in sec.).

n	1,280		2,560		3,840		5,210	
	予測	実際	予測	実際	予測	実際	予測	実際
(a-1)	0.07	0.15	0.50	0.89	1.69	2.79	3.76	5.70
(c-1)	0.10	0.30	0.75	1.89	2.54	5.11	5.64	11.6
(c-2)	0.10	0.12	0.75	0.84	2.54	2.56	5.64	5.61

以下のような考察を行った．

- (a-1) の下三角帯行列化では，ブロック鏡像変換を CPU 上で作成するため，メインメモリとグラフィックボードとの間の双方向のデータ転送が繰り返し行われる．この通信時間の分だけ，実際の時間が予測時間より多くなったと考えられる．また，行列サイズが大きくなるほど，全体の計算に対して通信量の占める割合が減るため，予測精度が向上しているのは自然な結果である．
- (c-1) の村田法の逆変換では，(a-1) や (c-2) とは異なったパターンで計算が進むので，必ずしも表 2 に示した性能が期待できるとは限らない．さらに，計算中の行列サイズが比較的小さいため，その性能は表 2 より劣ることが予想できる．よって，実際の時間は予測時間より大きくなっている．
- (c-2) の下三角帯行列化の逆変換では，データ通信がなく，表 2 に示した性能の測定と同様の演算パターンで計算が進むため，非常に良い精度で予測ができている．

以上の考察から，今回の予測では，GPU 上での Bischof の手法の性能の上限を粗く見積もったといえる．これは，従来法に関しても同様だといえる．今，表 4 の従来法の予測時間と表 8，表 9，表 10，表 11 の GPU 上での Bischof の手法の実際の時間を比べると，$n = 1{,}280$ の場合を除いて，後者の方が小さくなっている．これは，GPU 上での従来法の性能の上限を，Bischof の手法の実際の性能が超えていることを示している．以上の点から，Bischof の手法の選択は妥当であったといえる．

5.5　さらなる高速化に向けて

GPU を併用した正方行列の特異値分解計算をより高速化するための具体的な方法として，以下の 2 つをあげることができる．

- 本論文では，Bischof の手法の下三角帯行列化の中のブロック鏡像変換の計算には CPU のみを使用しているが，この部分は長方行列の QR 分解計算と同様の計算になるので，ブロック化を行い行列乗算を用いて計算する方法が適用できる．したがって，ブロック鏡像変換の計算にも CPU と GPU を併用することでさらなる高速化が可能である．
- 本研究の実装方法をまとめた表 6 から分かるように，現状の実装方法では CPU または GPU が遊んでいる状態がある．特に，今回 GPU を併用して高速化をした結果，従来はあまり目立たなかった二重対角行列の特異値分解の計算時間の全体に占める割合が全体の半分程度と非常に大きくなっている．もちろん，二重対角行列の特異値分解計算を GPU を用いて高速化することも可能であるが，一方で CPU が二重対角行列の特異値分解計算を行っているときに，GPU で逆変換の計算の準備としてサイズの小さい行列の合成などを行い，その後の計算を高速に行うことで，全体の計算時間を短縮することが可能である．

6. 関 連 研 究

長方行列の特異値分解を GPU を使用して高速に計算する研究が高木らによって報告されている[16]．長方行列の特異値分解計算では計算時間の大部分を QR 分解とその逆変換が占めるため，これらの部分を中心に GPU を用いることで全体の高速化を行っているが，二重対角化については GPU による高速化を行っていない．また，基本的に行列を CPU 側メモリに格納している．

一方，我々は Bischof の手法を用いることで，二重対角化部分についても GPU による高速化を可能にした．また，行列を基本的に GPU 側のメモリに置くことで転送のオーバヘッドを削減している．

7. お わ り に

本研究では，CUDA 上で正方行列の特異値分解を GPU を併用して計算することによる高速化を目指した．特に高性能な CUBLAS の SGEMM の利用を高速化の中心として，検討を行った結果，Bischof の手法を用いて特異値分解を計算するのが GPU 向けに有効であると判断した．また，実装をする際に各計算ステップにおいて，CPU と GPU との異なる性能特性を生かせるような仕事の分担や，CPU と GPU での計算のオーバラップなどを考慮して，実装を行った．その結果，性能評価において，正方行列の特異値分解を CPU（2 コア）のみで行う場合と比べて，$n = 5{,}120$ の場合で最大約 4 倍の高速化が確認できた．また，今回高速化の対象とした部分に限ると約 7 倍の高速化であった．

今後の課題として以下のことがあげられる．

- CPU と GPU のより効率的な仕事の分担やオーバラップを行い，性能評価を行う．

- 適切な半帯幅 L_B を選択するために，性能予測モデルを用いた自動チューニング方法を提案する．

謝辞　本論文を丁寧にお読みくださり，有益な助言をくださった査読者の方々に感謝いたします．また，日頃からご指導いただいている名古屋大学大学院工学研究科の張紹良教授と張研究室の皆様，特異値分解に関してご議論いただいている京都大学大学院情報学研究科の中村研究室の皆様に感謝いたします．なお，本研究は科学研究費補助金基盤研究（A）（課題番号 20246027），科学研究費補助金特定領域研究「i-explosion」，科学研究費補助金基盤研究（C）（課題番号 18560058），および名古屋大学 21 世紀 COE プログラム「計算科学フロンティア」の補助を受けている．

参 考 文 献

1) http://www.gpgpu.org/
2) NVIDIA CUDA.
http://www.nvidia.com/object/cuda_home.html
3) CUDA Programming Guide 1.1.
http://www.nvidia.com/object/cuda_develop.html
4) http://www.nvidia.com/object/cuda_showcase.html
5) 深谷　猛，山本有作，畝山多加志，堀　玄，梅野　健：長方行列向け特異値分解の浮動小数点コプロセッサによる高速化，情報処理学会論文誌：コンピューティングシステム，Vol.48, No.SIG 8 (ACS 18) (2007).
6) Demmel, J.W.: *Applied Numerical Linear Algebra*, SIAM, Philadelphia (1997).
7) Dongarra, J.J., Du Croz, J., Hammarling, S. and Duff, I.: A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Software*, Vol.16, pp.1–17 (1990).
8) Golub, G. and Van Loan, C.: *Matrix Computations, 3rd edition*, Johns Hopkins University Press (1996).
9) Dongarra, J.J., Hammarling, S.J. and Sorensen, D.C.: Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations, *Journal of Computational and Applied Mathematics*, Vol.27, pp.215–227 (1989).
10) Bischof, C., Lang, B. and Sun, X.: A Framework for Symmetric Band Reduction, *ACM Trans. Math. Software*, Vol.26, No.4, pp.581–601 (2000).
11) Bischof, C. and Lang, B.: Software for Successive Band Reduction, *ACM Trans. Math. Software*, Vol.26, No.4, pp.602–616 (2000).
12) 山本有作：Level-3 BLAS に基づく特異値分解アルゴリズムの SMP 上での性能，日本応用数理学会 2006 年度年会講演予稿集，pp.326–327 (2006).
13) Murata, K. and Horikoshi, K.: A New Method for the Tridiagonalization of the Symmetric Band Matrix, *Inf. Proc. Japan*, Vol.15, pp.108–112 (1975).
14) Lang, B.: A Parallel Algorithm for Reduction Symmetric Banded Matrices to Tridiagonal Form, *SIAM J. SCI. COMPUT.*, Vol.14, No.6, pp.1320–1338 (1993).
15) 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, pp.461–481 (2005).
16) 高木優一，鈴木智博：長方行列向け特異値分解における CUDA による GPU 援用計算，日本応用数理学会 2008 年度年会講演予稿集，pp.129–130 (2008).

(平成 20 年 10 月 3 日受付)
(平成 20 年 11 月 24 日採録)

深谷　猛（学生会員）

1983 年生．2007 年名古屋大学工学部物理工学科（応用物理学コース）卒業．2009 年同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，同大学院工学研究科計算理工学専攻（博士後期課程）在学中．アクセラレータを用いた行列計算や行列計算アルゴリズムの自動チューニング手法の研究に従事．

山本　有作（正会員）

1966 年生．1990 年東京大学工学部計数工学科（数理工学コース）卒業．1992 年同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．2003 年名古屋大学大学院工学研究科計算理工学専攻助手．現在，同准教授．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．高性能計算とその応用に興味を持つ．博士（工学）．SIAM，INFORMS，日本応用数理学会各会員．

畝山多加志

1980 年生．2003 年名古屋大学工学部物理工学科卒業．2005 年同大学大学院工学研究科計算理工学専攻修士課程修了．2008 年京都大学大学院理学研究科物理学・宇宙物理学専攻修了．現在，京都大学化学研究所特定助教．高分子物理学の理論の開発およびシミュレーションに従事．高速計算アルゴリズムの物理シミュレーションへの応用に興味を持つ．博士（理学）．日本物理学会，高分子学会，日本レオロジー学会各会員．

中村　佳正（正会員）

1955 年生．1983 年京都大学大学院工学研究科博士課程修了．工学博士．岐阜大学助教授，同志社大学工学部教授，大阪大学大学院基礎工学研究科教授等を経て，2001 年より京都大学大学院情報学研究科教授．応用可積分系，計算数学の研究に従事．日本応用数理学会，日本数学会，SIAM 等各会員．