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

深谷 猛[†] 山本有作[†] 畝山多加志[††]
堀 玄[†††] 梅野 健[†††]

本論文では，ClearSpeed 社の浮動小数点コプロセッサ CSX600 を用いた長方行列の特異値分解の高速化について報告する．長方行列の特異値分解は，入力行列 A の QR 分解 $A = QR$，行列 R の二重対角化，二重対角行列の特異値分解，逆変換，QR 分解の逆変換の 5 つのステップからなる．本研究では，この各部分において level-3 BLAS の DGEMM（行列乗算）を効率的に利用できるようにアルゴリズムをチューニングし，DGEMM の部分を CSX600 で高速に実行する方式を採った．CSX600 を 2 個搭載したボードを用いて本方式を実装し，様々なサイズの長方行列に適用した結果，40000×2000 の行列の場合に，Xeon（3.2GHz）の 2.3 倍の性能が得られた．また，更なる性能向上のための課題を明らかにした．

Acceleration of the Singular Value Decomposition Algorithm for Rectangular Matrices with a Floating-Point Coprocessor

TAKESHI FUKAYA,[†] YUSAKU YAMAMOTO,[†]
TAKASHI UNEYAMA,[††] GEN HORI[†††] and KEN UMENO[†††]

In this paper, we propose an approach for accelerating the singular value decomposition (SVD) of a rectangular matrix with the CSX600 floating point coprocessor. The SVD of rectangular matrices consists of five steps, namely, QR decomposition of the input matrix A, transformation of R into a bidiagonal matrix, SVD of the resulting bidiagonal matrix, back-transformation and back-transformation corresponding to the QR decomposition. In our study, we optimize each step so that most of the computation is done using the level-3 BLAS (DGEMM) and accelerate the execution of DGEMM with the CSX600. We implemented the proposed method using an accelerator board with two CSX600 chips and obtained up to 2.3 times speedup over 3.2GHz Xeon processor when computing the SVD of a 40000×2000 rectangular matrix. Technical issues for further improving the performance are also discussed.

1. はじめに

A を任意の $m \times n$ 実行列（$m \geq n$）とするとき，A は $U^T U = I_n$ を満たす $m \times n$ 行列 U，$V^T V = I_n$ を満たす $n \times n$ 行列 V，対角行列 $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$（$\sigma_1 \geq \cdots \geq \sigma_n \geq 0$）を用いて $A = U\Sigma V^T$ と書ける[9]．ただし，I_n は n 次の単位行列，上付き添字 T は行列の転置を表す．これを A の特異値分解と呼ぶ．

特異値分解は，情報検索[3]，画像処理[21]，最小 2 乗法[28]，電子状態計算[25] など様々な応用で重要な役割を果たす．特に $m \gg n$ である長方行列の特異値分解は多くの問題で利用され，大規模問題の求解への需要も高まりつつある．たとえば filter diagonalization 法による大規模電子状態計算[25] では，数万×数千以上の長方行列の特異値分解を繰り返し行うことが必要である．また，Latent Semantic Indexing 法による情報検索[3] では，検索対象の文書数を行数，キーワード数を列数とする長方行列の特異値・特異ベクトルを計算するため，文書数・キーワード数が増えると行列サイズは巨大になる．

いま，filter diagonalization 法による電子状態計算を例に取り，特異値分解に必要な演算量を見積もってみる．この方法では電子軌道を表現する基底の数を

† 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
†† 京都大学大学院理学研究科物理学・宇宙物理学専攻
Department of Physics, Graduate School of Science, Kyoto University
††† 理化学研究所 次世代移動体通信研
Laboratory for Next Mobile Communications Systems, The Institute of Physical and Chemical Research (RIKEN)

行数，計算する電子軌道の数を列数とする長方行列の特異値分解が必要である．擬ポテンシャルを使う場合，一般に電子軌道数は原子数の数倍必要であるから[25]，たとえば 10^5 個の基底を使って 1000 個の原子の動力学シミュレーションを行う場合，行列サイズは $100,000 \times 5000$ 程度となる．このとき，特異値分解の演算量は約 1.5×10^{13}FLOP となる．これは 1 時間ステップあたりの演算量であるが，時間発展を 10,000 ステップ行うとすると，シミュレーション全体を 1 週間で終えるには，1 回の特異値分解を 1 分以内，すなわち実効速度 250GFLOPS で行う必要がある．

このような大規模計算を高速に行うには，PC クラスタの利用，ベクトル型スーパーコンピュータの利用などいくつかの方法が考えられるが，近年，1 チップに多数の浮動小数点演算器を集積した専用プロセッサが多数開発され，コンパクトさと低消費電力の面から注目を集めている．たとえば GRAPE-DR[16] は 1 チップに 1024 個の演算プロセッサを集積し，最大 512GFLOPS の性能を持つ．また，ClearSpeed 社の CSX600[8] は 1 チップに 96 個のプロセッサを集積し，最大 48GFLOPS の性能を持つ．Cell プロセッサ[7] は，ゲーム等のマルチメディア処理が主目的ではあるが，チップ上に 8 個の専用プロセッサを持ち，最大 256GFLOPS の性能を達成する．これらの中で，CSX600 プロセッサは製品化が 2005 年と比較的早く，科学技術計算で不可欠な BLAS や FFT が，一部のみではあるが，既にライブラリとして提供されているという利点を持つ．CSX600 が 2 個搭載されたボードが PCI-X 用の拡張ボードとして用意されており，ボードの消費電力は約 25W と，汎用プロセッサで同じピーク性能を実現する場合に比べて極めて小さい．

そこで本研究では，CSX600 向けに長方行列の特異値分解分解アルゴリズムをチューニングし，実装を行う．そして，汎用プロセッサ上での実行に比べてどの程度の速度向上が得られるかを評価し，更なる高速化に向けた課題を明らかにする．

本論文の構成は以下の通りである．まず 2 章で，本研究で用いる ClearSpeed 社の CSX600 チップについて，その特徴と基礎的な性能評価の結果を紹介する．3 章では長方行列の特異値分解アルゴリズムの概要を述べ，CSX600 で実行するための性能チューニング法を検討する．4 章で性能評価の結果を報告し，5 章で結果に対する考察を行う．6 章では関連研究について述べる．最後に 7 章でまとめと今後の課題を述べる．

2. CSX600 の特徴と性能

2.1 CSX600 プロセッサ

ClearSpeed 社の CSX600[8] は，数値計算専用に開発された浮動小数点コプロセッサである．チップ上に 1 個の主プロセッサと 96 個の計算専用プロセッサ，キャッシュメモリ，メモリコントローラ等が集積されており，250MHz で動作する．各プロセッサの仕様を表 1 に示す．チップの理論ピーク性能は倍精度演算で 48GFLOPS である．

表 1 CSX600 の仕様
Table 1 Specification of the CSX600.

種別	個数	仕様
主プロセッサ	1	64 ビット 64 バイト レジスタファイル 8K バイト 命令キャッシュ 4K バイト データキャッシュ
計算専用プロセッサ	96	64 ビット 2 演算同時実行可能 128 バイト レジスタファイル 6K バイト SRAM IEEE754 準拠 浮動小数点演算 0.5GFLOPS（倍精度）

2.2 ClearSpeed Advance ボード

CSX600 を PC 上で利用するため，ClearSpeed Advance というボードが製品化されている．本ボードは 2 個の CSX600 チップと最大 4G バイトの DRAM を搭載し，PCI-X バスにより PC と接続される．ボードの消費電力は約 25W である．本研究ではこのボードを使用する．

2.3 ソフトウェア開発環境

CSX600 の開発環境として，コンパイラ，デバッガ，シミュレータ等を含む Software Development Kit が販売されている[8]．コンパイラは C 言語を拡張して並列データ型を定義した C^n という言語であり，MPI と類似の記述形式で，96 個のプロセッサを利用する SPMD 形式のプログラムを書くことができる．

また，ライブラリとして，BLAS（Basic Linear Algebra Subprograms）[12] の機能を提供する CSXL ライブラリ，1 次元と 2 次元の FFT を行う CSFFT ライブラリが提供されている．これらのライブラリはホスト PC のプログラムからコールする形で使うことができ，PC の主記憶から ClearSpeed Advance ボード上へのデータ転送，ボード上での CSX600 による演算，ボードから PC の主記憶へのデータの再転送を行う．ただし，現在の CSXL ライブラリは機能が非常に限

られており，長方行列どうしの乗算を行う DGEMM ルーチンと密行列のピボット選択付き LU 分解を行う DGETRF ルーチンのみが用意されている．

長方行列向け特異値分解を CSX600 を用いて高速化するには，SDK を使ってプログラム全体を移植する方法と，BLAS 部分を CSXL を用いて高速化する方法が考えられる．我々は将来的には前者を目標とするが，現在の SDK はベータ版であり，コンパイラの最適化機能が弱いため移植に多大の工数が掛かることが予想される．そこで本研究では後者の方法を採用する．

2.4 行列乗算の性能

CSXL を用いて長方行列の特異値分解を高速化するに当たっては，次章で述べるように，DGEMM ルーチンを多用する．そこで，本節でその基礎的な性能について報告する．DGEMM では，$M \times K$ 行列 A，$K \times N$ 行列 B，$M \times N$ 行列 C，スカラー α，β に対して $C := \alpha AB + \beta C$ を計算する[12]．ただし，CSXL では，行列サイズが小さいと PCI-X バスによるデータ転送のオーバーヘッドが大きくなって CSX600 による性能向上効果が得られないため，M，N，K すべてが 448 以上でなければならないという制限を課している．そこで，M，N，K のうち 1 個あるいは 2 個を最小値付近の値 450 に固定し，他のサイズを変えた場合の性能を測定した．なお，以下の測定ではすべて，ボード上の CSX600 を 2 個とも用いている．また，性能算出に用いた実行時間には，PC の主記憶と ClearSpeed Advance ボードの間のデータ転送時間が含まれている．

$M = K = 450$ とし，N を 1000 から 6000 まで変えた場合の性能を図 1 に示す．ここで，入力行列 A は非転置とし，B は転置と非転置両方の場合を測定した．また，$K = 450$ とし，$M = N$ としてこれを 1000 から 6000 まで変えた場合の性能を図 2 に示す．この場合も A は非転置とし，B は転置と非転置の両方を測定した．これらの結果から，次のことが分かる．

(i) $M = K = 450$ の場合の性能は 11GFLOPS 程度で飽和し，N を 3000 より大きくしても改善が見られない．

(ii) $K = 450$ の場合は M，N に連れて性能は順調に向上する．$M = N = K$ の場合の性能は最大 50GFLOPS 程度と報告されているが[8]，$K = 450$，$M = N = 6000$ でもその 80〜90%の性能は達成できる．

なお，他の組合せに対しても，3 つのサイズのうち 2 個が 450 の場合は図 1，1 個のみが 450 の場合は図 2 と同様の性能傾向を示した．

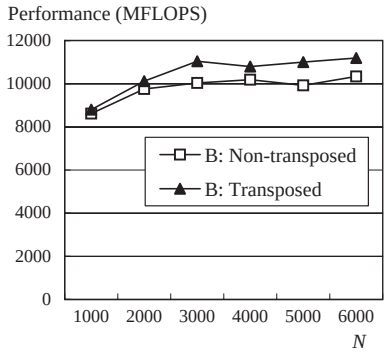

図 1 $M = K = 450$ の場合の CSXL の DGEMM の性能
Fig. 1 Performance of the DGEMM routine in CSXL for $M = K = 450$.

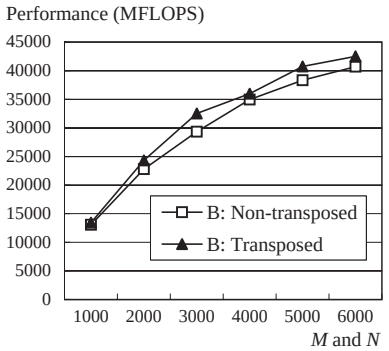

図 2 $K = 450$ の場合の CSXL の DGEMM の性能
Fig. 2 Performance of the DGEMM routine in CSXL for $K = 450$.

以上の結果より，CSXL の DGEMM を活用するには，3 つのサイズすべてが 448 以上であり，かつそのうち 2 つができるだけ大きな値になるような形でコールを行う必要があることが分かる．

次に，CSXL の実行時間におけるデータ転送のオーバーヘッドの割合について検討する．本来ならば，データ転送時間だけを取り出して実測すべきであるが，CSXL はバイナリで提供されており，中にタイマーを挿入できないため，これは困難である．そこで，PCI-X バスの理論性能である 1.066GB/s と，DGEMM の入出力の行列サイズから，データ転送時間を推定した．図 1 と図 2 に対応する DGEMM の全実行時間の実測値（データ転送を含む）とデータ転送時間の推定値を図 3 と図 4 に示す．

図より，データ転送時間は全実行時間の 30%〜75%である．これは PCI-X バスのピーク性能が出たという仮定の下での時間，すなわち下限であるから，実際のデータ転送時間はかなり大きいことがわかる．ただ

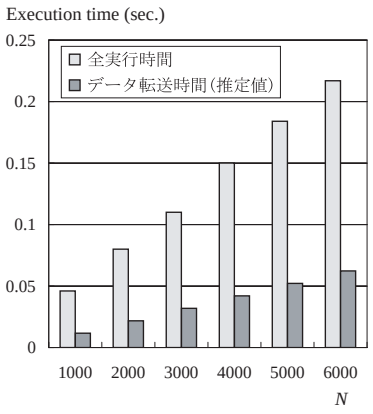

図 3 $M = K = 450$ の場合の CSXL の DGEMM の実行時間
Fig. 3 Execution time of the DGEMM routine in CSXL for $M = K = 450$.

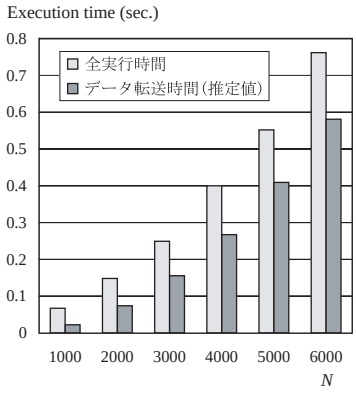

図 4 $K = 450$ の場合の CSXL の DGEMM の実行時間
Fig. 4 Execution time of the DGEMM routine in CSXL for $K = 450$.

し，この時間がそのままオーバーヘッドになるわけではない．なぜなら，全実行時間，データ転送時間の推定値，および CSX600 のピーク性能の 3 つのデータから考えると，CSXL ではデータ転送とボード内での演算のオーバーラップを行っているはずだからである．より詳細にデータ転送のオーバーヘッドを調べるには，SDK を使って自分で CSXL 相当のルーチンを作成する必要があり，今後の課題である．

3. 長方行列の特異値分解アルゴリズムと CSX600 向けチューニング手法

3.1 長方行列の特異値分解アルゴリズム

$m \times n$ の実長方行列 A の特異値分解 $A = U\Sigma V^T$ の計算は，$m \gg n$ の場合，次のような手順で行う[9)15)]．

(a) **QR 分解：** $A = QR$ と QR 分解を行う．ここで，Q は $m \times n$ の列直交行列（$Q^TQ = I_n$ を満たす行列），R は $n \times n$ の上三角行列である．

(b) **二重対角化：** $R = U_1 B V_1^T$ と，R を直交行列 U_1，下二重対角行列 B，直交行列 V_1 の転置の積に分解する．

(c) **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．

(d) **逆変換：** $\bar{U} = U_1 U_2$，$V = V_1 V_2$ を計算する．これにより，$R = \bar{U}\Sigma V^T$ が R の特異値分解となる．

(e) **QR 分解の逆変換：** $U = Q\bar{U}$ を計算する．これにより，$A = U\Sigma V^T$ が A の特異値分解となる．U の各列を A の左特異ベクトル，V の各列を A の右特異ベクトルと呼ぶ．

各ステップの演算量は，ステップ (a)，(e) が $O(mn^2)$，ステップ (b)，(d) が $O(n^3)$，ステップ (c) は使うアルゴリズムによって異なり $O(n^2) \sim O(n^3)$ となる．したがって，$m \gg n$ の場合はステップ (a)，(e) が特に重要である．

以上では，U，Σ，V すべてを計算する場合を考えたが，電子状態計算，画像処理など，左特異ベクトル U のみが必要な応用も多い．この場合，ステップ (d) の逆変換では $\bar{U}$ のみを計算すればよく，計算量が半減できる．また，一部の特異ベクトルのみが必要な応用も多いが，この場合，ステップ (d)，(e) では U，$\bar{U}$，$\bar{V}$ の各列を独立に計算できるので，必要な列のみを計算すればよい．

なお，上記の手順中に出てくる行列 Q，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，およびその逆行列による乗算ができればよい．実際，後述のアルゴリズムでは，Q，U_1，V_1 はそれぞれ複数のハウスホルダー変換の積として陰的に定義される．

3.2 アルゴリズムのチューニング手法

前節で述べた特異値分解の計算を ClearSpeed Advance ボードで高速に実行するには，CSXL の DGEMM ルーチンが効率的に利用できるよう，アルゴリズムを構成する必要がある．そのためには，2.4 節で述べた通り，行列乗算の 3 つのサイズがすべて 448 以上であり，かつそのうち 2 つをできるだけ大きな値にする必要がある．以下，この方針に沿ったアルゴリズムのチューニングについて述べる．

3.2.1 二重対角行列の特異値分解

まず，ステップ (c) の二重対角行列の特異値分解について検討する．この計算に対しては，QR 法[10)]，分割統治法[18)]，MR^3 アルゴリズム[17)]，I-SVD[19)20)] など，数多くのアルゴリズムが存在する．以下，それぞれの長所・短所について検討する．

QR 法は高い安定性を持つアルゴリズムであり，LA-

PACK[2] の DBDSQR として実装され，広く使われている．しかし，全特異ベクトルが必要な場合は常に $O(n^3)$ の演算が必要であり，他の手法に比べて大幅に遅い[9]．また，特異ベクトル計算部分を行列乗算を使うように書き換えることは可能であるが[22]，CSXL の DGEMM が活用できるほど大きなサイズの行列乗算にまとめることは困難である．

分割統治法は LAPACK の DBDSDC として実装され，QR 法より高速な解法として広く使われている．計算量は最悪の場合で $O(n^3)$ であるが，特異値の分布によっては $O(n^2)$ にまで低下する．また，計算の大部分が行列乗算の形で行われるという特徴を持つ．この行列乗算は正方行列どうしの乗算であり，そのサイズは最大 $n/2$ である．したがって n が数千程度と大きければ，CSXL の DGEMM が十分活用できる．

MR^3 アルゴリズムは，$O(n^2)$ の計算量で全特異ベクトルを計算でき，画期的なアルゴリズムとして注目されている．また，I-SVD は可積分系の考え方に基づくアルゴリズムであり，MR^3 アルゴリズムと同等の計算量とより高い精度を実現している．ただし，両者とも計算の大部分は level-1 BLAS（ベクトルとベクトルの内積，和など）であり，行列乗算は使用しない．

本研究では，(i) アルゴリズムが公開のソフトウェアとして実装され，容易に入手できること，(ii) CSXL の DGEMM により加速が期待できること，の 2 点から，分割統治法を採用することにした．コードとしては，LAPACK の DBDSDC を使用する．

3.2.2 二重対角化とその逆変換

ステップ (b) の二重対角化では，大きく分けて，ハウスホルダー法[15] と Bischof の方法[4][5] の 2 つのアルゴリズムが存在する．以下，それぞれの長所・短所について検討する．

ハウスホルダー法は LAPACK 等で採用されている標準的なアルゴリズムであり，ステップ (b) において，ハウスホルダー変換を左右から順次作用させることにより，行列を二重対角化する．ステップ (d) では，(b) で生成したハウスホルダー変換を逆順にかけていくことにより，逆変換を行う．演算量は，二重対角化が約 $(8/3)n^3$，逆変換が $4n^3$（全特異ベクトルが必要な場合）である．このアルゴリズムでは，ブロック化により，(b) の演算の半分を行列乗算を用いて行うことが可能である．また逆変換も，後述する compact-WY representation の利用により，行列乗算を用いて行える．しかし，(b) の演算の残り半分は行列ベクトル積であり，行列データのアクセスでキャッシュを利用できないため，性能が出にくいことが指摘されている[27]．

一方，Bischof の方法は二重対角化を 2 段階で行うアルゴリズムであり，入力の正方行列をまず半帯幅が L_B の下三角帯行列に変換してから，これを村田法[23] により二重対角化する．ここで，L_B は自由に設定可能なパラメータである．$L_B \ll n$ の場合，下三角帯行列化の演算量は $(8/3)n^3$，村田法の演算量は $8n^2L_B$ となる．このアルゴリズムでは，計算の大部分を占める下三角帯行列への変換において，計算をサイズ M，N，K がすべて L_B 以上の行列乗算として行うことができ，キャッシュサイズに合わせて適切に L_B を取ることにより，性能が向上する．また，逆変換もすべて行列乗算を用いて行える．ただし，逆変換でも 2 段階の変換が必要となり，演算量が $8n^3$ と倍増する．

いま，CSXL の DGEMM の利用を前提とすると，ハウスホルダー法でブロックサイズを十分大きく取り，(b) の演算の半分と (d) の逆変換で CSXL を用いる方式が一つの候補として考えられる．しかしこの方法では，将来的に特異値分解プログラム全体を ClearSpeed Advance ボードに移植した場合，性能を十分引き出せない可能性が高い．(b) の行列ベクトル積の演算において，CSX600 チップ上の 96 個のプロセッサが同時にボード上の行列データにアクセスし，データ転送のネックが生じるからである．一方，Bischof のアルゴリズムでは，$O(n^3)$ の演算量を持つすべての部分において，各プロセッサに付随する SRAM をキャッシュとして有効利用でき，ボード上のデータへのアクセスは大きく削減できる．本研究では，将来における性能向上の余地を重視し，Bischof の方法を採用する．

次に，Bischof の方法を用いる場合，CSXL による高速化が可能かどうかを検討する．Bischof の方法において，(b) の演算量の大部分を占める下三角帯行列化を CSXL の DGEMM で実行するには，半帯幅 L_B を 448 以上に取る必要がある．そこで，このときの性能について予備的な評価を行った．評価環境は 4.1 節に述べる通りである．

$n = 3000$ の場合に，$L_B = 50$ として下三角帯行列化と村田法の両方を CPU のみで行ったときの実行時間と，$L_B = 450$ として下三角帯行列化で CSXL の DGEMM を使用したときの実行時間を，それぞれ図 5 の最左列，左から 2 番目の列に示す．CSXL の利用により，下三角帯行列化の時間は 2/3 程度に減少する．しかし，村田法の時間が約 10 倍になり，二重対角化全体の時間は増加している．これは村田法の演算量が L_B に比例することから，止むを得ない結果である．n が大きくなると村田法の実行時間の占める割合は減り，CSXL による高速化効果が支配的になると期

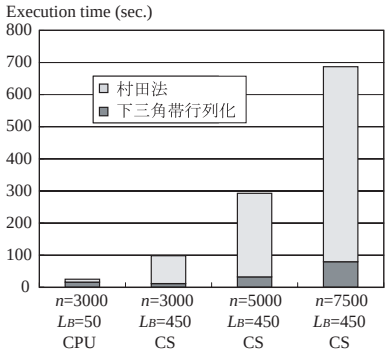

図 5 下三角帯行列化と村田法の実行時間
Fig. 5 Execution time of transformation to a lower band matrix and Murata's method.

待されるが，図 5 に示す通り，$n = 7500$ でも村田法の時間が圧倒的である．

以上の結果より，本研究では，二重対角化および逆変換では CSXL を使用せず，CPU のみで計算を行う方針を採る．ただし，もし村田法の実行効率が改善できれば，二重対角化で CSXL を使う方式も実用化できる能性があり，今後の課題である．

3.2.3 QR 分解とその逆変換

ステップ (a) の QR 分解およびステップ (e) の QR 分解の逆変換は，ともに $O(mn^2)$ の計算量を必要とする部分であり，高速化が特に重要である．QR 分解の手法としては，大きく分けて，グラム・シュミット法に基づくアルゴリズムとハウスホルダー変換に基づくアルゴリズムがある[15]．特異値分解の場合は，元の行列 A の列ベクトルが線形従属に近い場合でも Q の列ベクトルに高い直交性が保証される後者の方法を使うのが普通であり，本研究でも後者を採用する．

ハウスホルダー法に基づく QR 分解を行列乗算を用いて行うため，まず compact-WY representation について説明する．いま，l 個のハウスホルダー変換

$$H_i = I_m - t_i \mathbf{y}_i \mathbf{y}_i^T \quad (i = 1, 2, \ldots, l) \qquad (1)$$

（ただし t_i はスカラー，$\mathbf{y}_i$ は m 次元ベクトル）があるとき，それらの積はある $m \times l$ 行列 Y，$l \times l$ 上三角行列 T によって

$$H_l \cdots H_2 H_1 = I_m - YTY^T \qquad (2)$$

と書ける[24]．これを compact-WY representation と呼ぶ．

Compact-WY representation を用いて QR 分解を行う手法としては，単純なブロック化による方法[15]と再帰的 QR 分解[13]とが知られている．また，この 2 つを組み合わせた手法も考えられる．以下，これら 3 つの手法について，CSXL を利用する場合の長所・短所を検討する．

単純なブロック化による QR 分解をアルゴリズム 1 に示す．

[アルゴリズム 1: 単純なブロック化]
入力行列 A を幅 L_{QR} のブロック p 個に分割し，$A = [A_1|A_2|\cdots|A_p]$ とする．
do $k = 1, p$
⟨1⟩ A_k の第 $(k-1)L_{QR}+1$ 行以降を上三角行列に変換するハウスホルダー変換 $H_k^1, H_k^2, \ldots, H_k^{L_{QR}}$ を求める．
⟨2⟩ compact-WY representation $H_k^L \cdots H_k^2 H_k^1 = I - Y_k T_k Y_k^T$ を求める．
⟨3⟩ $[A_k|A_{k+1}|\cdots|A_p] := (I - Y_k T_k Y_k^T)[A_k|A_{k+1}|\cdots|A_p]$.
end do

アルゴリズム 1 では，行列 A が行列 $\bar{R}$（R の下側に 0 を詰めてできる $m \times n$ 行列）で上書きされる．また，p 個の compact-WY representation $I - Y_k T_k Y_k^T$ $(k = 1, \ldots, p)$ を左から順に掛ける作用が，A を $\bar{R}$ に変換する直交変換に相当する．アルゴリズム 1 の演算量は，$m \gg n$，$L_{QR} \ll n$ のとき，約 $2mn^2$ である．

一方，再帰的 QR 分解をアルゴリズム 2 に示す．

[アルゴリズム 2: 再帰的 QR 分解]
⟨1⟩ 入力行列 A を半分の幅 m' のブロック 2 個に分割し，$A = [A_1|A_2]$ とする．
⟨2⟩ A_1 を上三角行列 $\bar{R}_1 = [R_{11}^T|O]^T$ に変換するハウスホルダー変換をアルゴリズム 2 により求める．その compact-WY representation を $I - Y_1 T_{11} Y_1^T$ とする．
⟨3⟩ $A_2 := (I - Y_1 T_{11} Y_1^T) A_2$.
⟨4⟩ A_2 の最初の m' 行からなる部分行列を R_{12} とする．
⟨5⟩ A_2 の第 $m'+1$ 行以降を上三角行列 $\bar{R}_2 = [R_{22}^T|O]^T$ に変換するハウスホルダー変換をアルゴリズム 2 により求める．その compact-WY representation を $I - Y_2 T_{22} Y_2^T$ とする．
⟨6⟩ $T_{12} = -T_{11}(Y_{11}^T Y_{22}) T_{22}$.
⟨7⟩ $Y = [Y_1|Y_2]$, $R = \begin{bmatrix} R_{11} & R_{12} \\ O & R_{22} \end{bmatrix}$, $T = \begin{bmatrix} T_{11} & T_{12} \\ O & T_{22} \end{bmatrix}$.

このアルゴリズムでは，行列 A を入力とし，上三

角行列 R と，A を $\bar{R}$（R の下側に 0 を詰めてできる $m \times n$ 行列）に変換する直交変換の compact-WY representation $I - YTY^T$ を求める．これは再帰的なアルゴリズムであり，中で自分自身を 2 回呼び出す．演算量は $m \gg n$ のとき約 $3mn^2$ である．

アルゴリズム 1 と 2 を比較すると，アルゴリズム 1 ではステップ ⟨3⟩ については行列乗算を利用できるが，ステップ ⟨1⟩，⟨2⟩ では行列乗算を使えない[24]．一方，アルゴリズム 2 では，すべての演算が行列乗算となる（ただし，行列乗算のサイズは再帰のレベルが深くなるにつれ小さくなる）．ただし，演算量はアルゴリズム 2 がアルゴリズム 1 の 1.5 倍である．

そこで本研究では，両アルゴリズムを組み合わせる方式を採用した．具体的には，L_{QR} を 448 付近の値に固定してアルゴリズム 1 を用いるが，そのステップ ⟨1⟩，⟨2⟩ で幅 L_{QR} の行列 A_k に対して compact-WY representation を求める部分をアルゴリズム 2 により行う．この方式をアルゴリズム 3 に示す．

[アルゴリズム 3: 両手法の組み合わせ]
入力行列 A を幅 L_{QR} のブロック p 個に分割し，$A = [A_1|A_2|\cdots|A_p]$ とする．
do $k = 1, p$
⟨1⟩ A_k の第 $(k-1)L_{QR}+1$ 行以降を上三角行列に変換するハウスホルダー変換をアルゴリズム 2 により求める．
その compact-WY representation を $I - Y_k T_k Y_k^T$ とする．
⟨2⟩ $[A_k|A_{k+1}|\cdots|A_p] := (I - Y_k T_k Y_k^T)[A_k|A_{k+1}|\cdots|A_p]$.
end do

アルゴリズム 3 では，アルゴリズム 2 と同様，演算すべてを行列乗算により行うことが可能である．ステップ ⟨1⟩ では再帰的 QR 分解を用いるため，演算量が 1.5 倍となるが，$L_{QR} \ll n$ ならばこの部分の演算量は全演算量に比べて少ないので[24]，全演算量の増加は小さい．

アルゴリズム 3 では，演算量の大部分を占めるステップ ⟨2⟩ を CSXL の DGEMM により実行できる．ステップ ⟨1⟩ については，扱う行列の幅が L_{QR} 以下となる場合は，CPU により実行する．

QR 分解の逆変換は，ステップ (d) で得られた行列 $\bar{U}$ に，compact-WY representation $I - Y_k T_k Y_k^T$ の転置を逆順に掛けていくことにより行う．この部分の演算量は $4mn^2$ である．

予備評価として，本項で述べた 3 つのアルゴリズムの

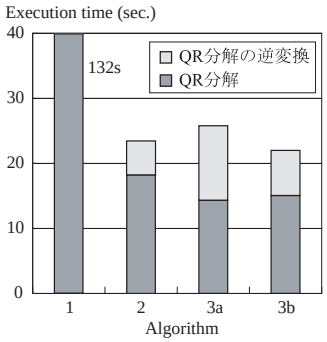

図 6 各アルゴリズムによる QR 分解とその逆変換の実行時間
Fig. 6 Execution times of the three algorithms for QR decomposition.

CSXL 上での性能を比較した結果を図 6 に示す．評価環境は 4.1 節に述べる通りである．図の横軸はアルゴリズムの番号である．アルゴリズム 1 では，$L_{QR} = 500$ とし，ステップ ⟨3⟩ のみを CSXL で実行した．アルゴリズム 2 では，すべてのステップにおいて，3 つのサイズ M,N,K いずれもが 448 以上となる行列乗算すべてを CSXL で実行した．アルゴリズム 3a，3b はそれぞれ $L_{QR} = 500$，1000 としたアルゴリズム 3 である．アルゴリズム 3 では，ステップ ⟨2⟩ すべて，およびステップ ⟨1⟩ 中で行列乗算のサイズが 448 以上となる部分を CSXL で実行した．QR 分解対象の行列サイズは $m = 10000$，$n = 3000$ である．

図より，この例では $L_{QR} = 500$ としたアルゴリズム 3 がもっとも高速に QR 分解を実行できることがわかる．ただし，逆変換も含めて考えると，$L_{QR} = 1000$ としたアルゴリズムがもっとも速い．これは，L_{QR} を大きくすることで compact-WY representation における行列サイズが大きくなり，CSXL の性能が上がるためである．そこで本研究ではアルゴリズム 3 を用い，行列サイズに応じて L_{QR} を調節することにする．

3.3 アルゴリズムのまとめ

以上の検討結果について，(a) ～ (e) の各ステップで用いるアルゴリズム，特異値分解を求める場合の演算量，CSXL の使用の有無を表 2 にまとめる．

3.4 CSX600 での目標性能

以上で述べたアルゴリズムのうち，QR 分解とその逆変換の部分について，CSX600 上での目標性能を設定する．そのため，まずアルゴリズム中で用いる主要な行列乗算について，CSXL の性能を実測する．QR 分解における主要な行列乗算は以下の 2 つである．

(i) Y_k^T と $[A_k|A_{k+1}|\cdots|A_p]$ の乗算．サイズは $M = L_{QR}$，$L_{QR} \le N \le n$，$K \sim m$.
(ii) Y_k と $T_k Y_k^T [A_k|A_{k+1}|\cdots|A_p]$ の乗算．サイズは

表 2 実験に使用した計算機環境
Table 2 Computational environments used in the experiments.

項目	アルゴリズム	演算量	CSXL
(a)	単純なブロック化 + 再帰的 QR 分解	$2mn^2$	○
(b)	Bischof	$(8/3)n^3 + 8n^2L_B$	×
(c)	分割統治法	$O(n^2) \sim O(n^3)$	○
(d)	Bischof	$8n^3$	×
(e)	単純なブロック化	$4mn^2$	○

$M \sim m$, $L_{QR} \leq N \leq n$, $K = L_{QR}$.
QR 分解の逆変換でも，この 2 つのタイプを用いる．ただし，逆変換の場合は常に $N = n$ である．

この 2 つのタイプについて，CSXL の性能の実測結果を表 3 に示す．ただし，(i) については $M = 500$, $K = 10000$, (ii) については $M = 10000$, $K = 500$ とし，N の値を 500 から 3000 まで変えて測定した．

表 3 CSXL の DGEMM の性能（GFLOPS）
Table 3 Performance of the DGEMM routine in CSXL (in GFLOPS).

タイプ	(i)	(ii)
転置指定	A 転置，B 非転置	A，B 非転置
M	500	10000
K	10000	500
$N = 500$	17.10	20.00
$N = 1000$	20.29	29.31
$N = 1500$	23.94	36.20
$N = 2000$	24.18	41.13
$N = 2500$	20.43	44.41
$N = 3000$	21.17	47.55

一方，アルゴリズム 3 においてステップ $\langle 1 \rangle$ は CPU で実行するため，その性能も必要である．これについては，Xeon（3.2GHz）のピーク性能である 6.4GFLOPS を用いる．

以上のデータを基に，アルゴリズム 3 の各ステップに対して演算量を性能値で割って予測実行時間を算出し，それを積み上げることにより，性能予測を行う．ただし，Xeon の性能としてピーク性能を用いており，また，演算量の小さい T_k による乗算の時間を省略していることから，ここで算出する値は性能の上限である．$n = 2000$, $L_{QR} = 500$ とし，$m = 10000$, 20000, 30000 と変えた場合の予測値を表 4 に示す．表中の (a) の行が QR 分解の時間，(e) の行が QR 分解の逆変換の時間を表す．

次章以降の性能評価では，これらの値をチューニングの効果を測る目安として用いる．

表 4 CSXL を使った QR 分解とその逆変換の予測時間（秒）
Table 4 Predicted execution time (in sec.) of QR decomposition using CSXL.

n	2000			
m	10000	20000	30000	40000
(a)	6.66	13.76	20.87	27.97
(e)	4.86	10.11	15.37	20.62

4. 性能評価

4.1 評価条件

前節の検討結果に基づいて長方行列の特異値分解を行うプログラムを作成し，性能評価を行った．言語は FORTRAN である．ただし，二重対角行列の特異値分解の部分のみは LAPACK の DBDSDC を使用した．使用した計算機環境は表 5 の通りである．

表 5 実験に使用した計算機環境
Table 5 Computational environments used in the experiments.

項目	条件
CPU	Xeon 3.2GHz（1CPU のみ使用）
メモリ	4GB（コンパイラの制限より 2GB のみ使用）
OS	Red Hat Enterprise Linux
コンパイラ	g77 3.4.5（オプション -O3）
BLAS	Intel Math Kernel Library Ver. 8.1
コプロセッサ	ClearSpeed Advance ボード × 1（CSX600 チップ 2 個搭載）

評価では，$[0, 1]$ の一様乱数を要素に持つ $m \times n$ 行列の特異値分解を ClearSpeed Advance ボードあり／なしで計算し，3.1 節の 5 個のステップそれぞれの時間を計測した．ボードを使う場合，ボード上の CSX600 は 2 個とも使用した．m は 10000, 20000, 30000, 40000 の 4 通り，n は 1000, 2000, 3000 の 3 通りに変えて測定を行った．ただし，メモリの制限から，$m = 40000$, $n = 3000$ のケースは測定できなかった．Bischof の方法における半帯幅は $L_B = 40$ とし，QR 分解におけるブロック幅は，$n = 3000$ で ClearSpeed ボードを使う場合のみ $L_{QR} = 1000$，それ以外の場合は $L_{QR} = 500$ と設定した．

4.2 評価結果

測定結果を表 6 〜表 9 に示す．表中で「CPU」「CS」と書いた列は，それぞれ ClearSpeed ボードなし，ありのときの実行時間を示す．(a) から (e) の記号は，3.1 節の 5 個のステップに対応する．また,「合計 1」は特異値分解の実行時間,「合計 2」は特異値と左特異ベクトルのみの実行時間を示す.「速度向上 1」「速度向上 2」はそれぞれ特異値分解の計算，特異値と左特異ベ

クトルのみの計算における ClearSpeed ボード使用による速度向上効果を示す．$m = 40000$，$n = 2000$ のとき，特異値分解で 2.3 倍，特異値と左特異ベクトルのみを求める場合で 2.5 倍の性能向上が得られていることが分かる．

表 6 $m = 10000$ のときの実行時間（秒）
Table 6 Computational time (in sec.) for $m = 10000$.

m	10000					
n	1000		2000		3000	
計算機	CPU	CS	CPU	CS	CPU	CS
(a)	5.03	3.91	16.50	8.80	33.75	15.06
(b)	1.62	1.65	8.87	8.77	24.81	24.89
(c)	0.90	0.78	4.95	3.10	14.15	6.94
(d)	2.94	2.96	25.61	25.51	84.44	83.91
(e)	6.93	1.87	26.21	5.84	55.91	6.94
合計 1	17.42	11.16	82.13	52.01	213.05	137.73
合計 2	15.95	9.68	69.33	39.26	170.83	95.77
速度向上 1		1.56		1.58		1.54
速度向上 2		1.65		1.77		1.78

表 7 $m = 20000$ のときの実行時間（秒）
Table 7 Computational time (in sec.) for $m = 20000$.

m	20000					
n	1000		2000		3000	
計算機	CPU	CS	CPU	CS	CPU	CS
(a)	10.19	7.82	33.64	17.97	69.76	30.96
(b)	1.64	1.64	8.79	8.80	25.00	24.89
(c)	0.93	0.79	4.92	3.11	14.00	6.87
(d)	3.06	3.01	25.30	25.34	86.17	83.71
(e)	13.74	3.56	53.79	11.44	117.58	13.30
合計 1	29.56	16.82	126.45	66.65	312.51	159.73
合計 2	28.03	15.32	113.79	53.98	269.42	117.88
速度向上 1		1.76		1.90		1.96
速度向上 2		1.83		2.11		2.29

4.3 精度評価

ClearSpeed Advance ボードを使った場合，使わない場合の両方について，残差 $\| A - U\Sigma V^T \|_F$（$\| \cdot \|_F$ はフロベニウスノルム），左特異ベクトルの直交性 $\| U^TU - I \|_F$，右特異ベクトルの直交性 $\| V^TV - I \|_F$ をそれぞれ図 7，図 8，図 9 に示す．図の横軸は最初の 4 列が $n = 1000$ の場合であり，左から順に $m = 10000$，20000，30000，40000 に対する結果を示す．それ以降も同様である．CSX600 の倍精度演算は IEEE 754 準拠であるため，もともと両者の間に大きな違いはないはずである．グラフより，残差と左特異ベクトルに関しては CSX600 の方がやや誤差が大きい傾向が見られるが，その差は 2 〜 3 倍程度であり，多くの応用では問題にならない範囲だと考えられる．また，右特異ベクトルについてはほとんど差がないが，これは，上三角行列 R の右特異ベクトルがそのまま A の右特異ベクトルとなり，CSX600 を使用している QR 分解の逆変換の影響を受けないためだと考えられる．

表 8 $m = 30000$ のときの実行時間（秒）
Table 8 Computational time (in sec.) for $m = 30000$.

m	30000					
n	1000		2000		3000	
計算機	CPU	CS	CPU	CS	CPU	CS
(a)	15.41	11.86	50.92	27.29	106.03	47.04
(b)	1.63	1.64	8.76	8.75	24.93	24.67
(c)	0.92	0.80	4.91	3.16	14.06	6.89
(d)	2.99	2.98	25.46	25.39	83.97	83.70
(e)	20.77	5.35	81.32	16.81	179.88	18.65
合計 1	41.72	22.63	171.37	81.35	408.87	180.95
合計 2	40.22	21.14	158.64	68.66	366.88	139.10
速度向上 1		1.84		2.11		2.26
速度向上 2		1.90		2.31		2.64

表 9 $m = 40000$ のときの実行時間（秒）
Table 9 Computational time (in sec.) for $m = 40000$.

m	40000			
n	1000		2000	
計算機	CPU	CS	CPU	CS
(a)	20.41	15.67	68.01	36.17
(b)	1.63	1.63	8.75	8.79
(c)	0.92	0.79	4.95	3.10
(d)	2.96	2.95	25.59	25.47
(e)	27.65	6.42	108.91	20.42
合計 1	53.56	27.45	216.22	93.95
合計 2	52.08	25.98	203.42	81.22
速度向上 1		1.95		2.30
速度向上 2		2.00		2.50

5. 考　　察

5.1 各部分の性能分析

特異値分解の 5 つのステップのうち，本研究で CSXL による高速化を行ったのはステップ (a)，(c)，(e) である．このうち，(e) の QR 分解の逆変換は最大 5 倍と加速が良く，$m = 40000$，$n = 2000$ の場合で約 31GFLOPS の性能が得られている．これは，演算 $(I - Y_kT_kY_k)^T\bar{U}$ において，サイズ M，N，K のうち 2 つが大きい行列乗算がほとんどを占めるからだと考えられる．また，表 4 の予測時間と実測時間とを比べると，誤差 17%〜 1%で良く一致している．したがって，この部分についてはチューニングが予想通りの効果を上げていると言える．

(c) の分割統治法は，n が大きくなるにつれ加速が

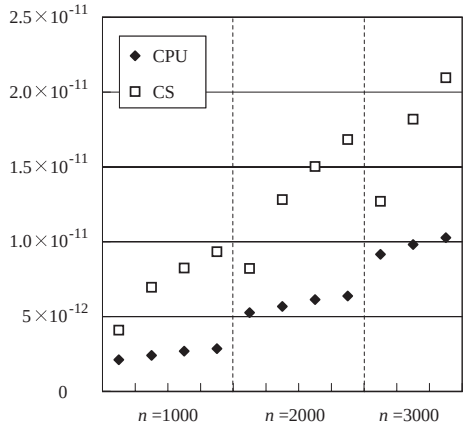

図 7 残差 $\| A - U\Sigma V^T \|_F$
Fig. 7 Residual $\| A - U\Sigma V^T \|_F$.

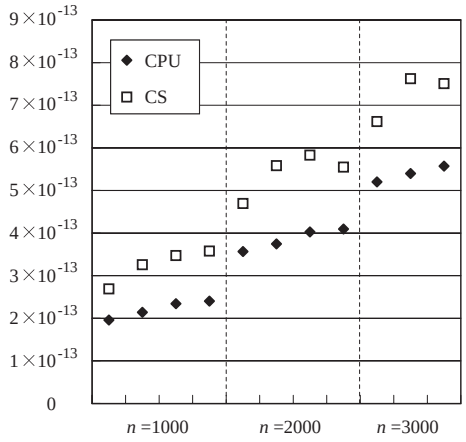

図 8 左特異ベクトルの直交性
Fig. 8 Orthogonality of the left singular vectors.

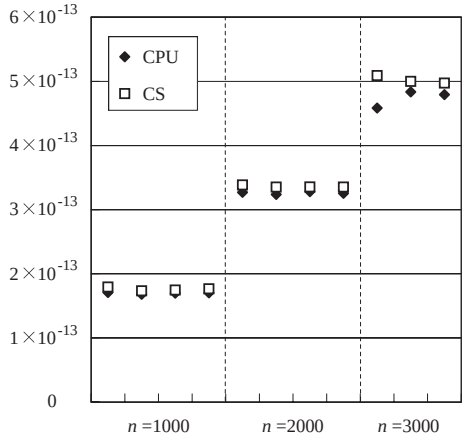

図 9 右特異ベクトルの直交性
Fig. 9 Orthogonality of the right singular vectors.

向上している．これについても，分割統治法ではサイズが最大 $n/2$ の正方行列の乗算が演算の大部分を占めることから，予想通りの結果である．

一方，(a) については，$m = 40000$，$n = 2000$ の場合を見ると，予測時間 27.97 秒に対し，実測時間は 36.17 秒となっている．これは他の m の値についても同じ傾向である．この理由を調べると，CPU で実行するアルゴリズム 3 のステップ ⟨1⟩ の部分について，実測性能が 4.5GFLOPS と，表 4 で仮定したピーク性能の 70%程度になっているのが誤差の主な原因であることがわかった．さらに，仮にピーク性能を出せた場合でも，ステップ ⟨1⟩ は QR 分解の実効時間の 2/3 を占め，QR 分解の加速を制約する原因となることがわかった．しかし，CSXL にサイズ制限がある以上，この部分は CPU で実行せざるを得ず，ここが性能ネックになるのは止むを得ないと考えられる．

一方，CSXL を使用していない (b)，(d) の部分は，$m \gg n$ であれば演算量に占める割合は小さく，実行時間に占める割合も，$m = 40000$，$n = 2000$ を CPU のみで実行した場合は 16%程度である．しかし，CSXL を使った場合には，この割合が 1/3 以上まで増加する．これが ClearSpeed ボードを使ったときの加速率を制約するもう一つの要因となっている．

n が数千以上と十分大きく，かつ $m \gg n$ である場合は，アルゴリズム 3 におけるステップ ⟨1⟩ の演算量の割合が減少し，かつ全体におけるステップ (b)，(d) の演算量の割合が減少する．したがって，以上で考察した性能制約要因の影響は減少する．これにより，本研究で作成したプログラムの加速率はより向上すると期待される．しかし，表 6 〜表 9 で示した程度の m，n でより大きな加速を得るには，ClearSpeed ボードの使用方法の改良が必要となる．

5.2 更なる高速化の可能性

現状のプログラムをより高速化する方法の一つとして，CPU と ClearSpeed ボードの並列処理が挙げられる．たとえばステップ (a) では，アルゴリズム 3 で k に対する ⟨2⟩ と $k+1$ に対する ⟨1⟩ が重ねて実行できる．これにより，加速率向上の制約要因となっている CPU による ⟨1⟩ の処理をある程度隠蔽することが可能となる．また，ステップ (e) でも，行列乗算を CPU とボードが分担して処理することにより，高速化が期待できる．

更に性能を向上させるには，SDK を使い，アルゴリズム 3 の ⟨1⟩ およびステップ (b)，(d) を CSX600 に移植することが考えられる．移植は，現在 FORTRAN で書かれているプログラムを C 言語に変換し，さらに

Cⁿ 言語を用いて並列化することにより，原理的には可能である．現状では，これらの部分の性能は CPU のピーク性能により制約を受けているが，CSX600 の 96 個の計算プロセッサを使った実装を行うことで，高速化できる可能性がある．また，行列全体をボード上に格納できるサイズの問題に対しては，PCI-X バスの転送時間を削減でき，この点での高速化も見込める．これらの手法による高速化の効果を定量的に評価して実装を行うことは，今後の課題である．

6. 関連研究

CSX600 を用いた行列計算については，大規模連立一次方程式を解く LINPACK ベンチマーク[26] の高速化が報告されている[8)14)]．特に，遠藤ら[14] は LINPACK のプログラムを不均一環境向けに最適化することにより，ホスト計算機と CSX600 を併用して性能を向上させる方式を提案している．ただし，LINPACK ベンチマークでは M，N が数千以上と大きい DGEMM が計算のほとんどを占めるため，CSXL の性能を引き出しやすい．これに対して本研究では，より複雑な演算パターンを持つ長方行列の特異値分解において，CSX600 の性能を引き出す方式を検討した．

一方，専用ハードウェアを用いた特異値分解の高速化については，1)，6)， 11) など多くの研究がある．これらのほとんどは，特異値分解のアルゴリズムとしてヤコビ法系統の解法を用いている．しかしヤコビ法は演算パターンが単純で並列性が高い反面，収束が遅く，二重対角化に基づく特異値分解の 5 倍以上の演算量を要する[9)]．本研究では，汎用プロセッサ上で最も効率的なアルゴリズムをベースとし，それを CSX600 によりどれだけ高速化できるかを評価している点に特色がある．

7. おわりに

本研究では，浮動小数点コプロセッサ CSX600 による長方行列の特異値分解の高速化について報告した．特異値分解を構成する 5 つのステップのそれぞれにおいて，行列乗算を効率的に使えるようチューニングを行った結果，40000×2000 の行列の特異値分解に対して Xeon（3.2GHz）の 2.3 倍の性能を得ることができた．また，各ステップの性能について分析を行い，更なる高速化のための課題を述べた．

今後の課題としては，5.2 節に述べた改良を行い，更に高速な特異値分解プログラムを作成すること，二重対角行列の特異値分解の部分を，優れた特性を持つ新しいアルゴリズムである I-SVD[19)20)] に置き換えて評価することなどが挙げられる．また，本論文で述べた行列乗算による高速化は，演算器性能に対するメモリバンド幅の低さをカバーできること，コードを高度に最適化する必要がある部分を行列乗算のみに局限できること，などの利点があり，Cell や GRAPE-DR など他のプロセッサに対しても効果的であると考えられる．これらのプロセッサに対して本手法を適用することも，今後の課題である．

参考文献

1) Ahmedsaid, A., Amira, A. and Bouridane, A.: Improved SVD Systolic Array and Implementation on FPGA, *Proc. IEEE International Conference on Field-Programmable Technology (FPT'03)*, pp. 35–42 (2003).
2) Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S. and Sorensen, D.: *LAPACK User's Guide*, SIAM, Philadelphia, 1992.
3) Berry, M. W. and Browne, M.: *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, SIAM, Philadelphia, 1999.
4) 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).
5) Bischof, C. and Lang, B.: Algorithm 807: The SBR Toolbox - Software for Successive Band Reduction, *ACM Trans.Math.Software*, Vol.26, No. 4, pp. 602–616 (2000).
6) Bodba, C., Danne, K., Ahmadinia, A. and Teich, J.: A New Approach for Reconfigurable Massively Parallel Computers, *Proc. IEEE International Conference on Field-Programmable Technology (FPT'03)*, pp. 391–394 (2003).
7) Cell Broadband Engine Resource Center. http://www.128.ibm.com/developerworks /power/cell/
8) ClearSpeed Technology Inc. http://www.clearspeed.com/
9) Demmel, J. W.: *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
10) Demmel, J. W. and Kahan, W.: Accurate Singular Values of Bidiagonal Matrices, *SIAM J.Sci.Stat.Comput.*, Vol.11, pp.873–912 (1990).
11) Dickson, K., Liu, Z. and McCanny, J. V.: Programmable Processor Design for Givens Rotations Based Applications, *Proc. IEEE Workshop on Sensor Array and Multichannel Signal Processing*, pp. 84–87 (2006).
12) 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).
13) Elmroth, E. and Gustavson, F.: Applying Recursion to Serial and Parallel QR Factorization Leads to Better Performance, *IBM Journal of Research and Development*, Vol.44, p.605 (2000).
14) 遠藤敏夫, 松岡聡, 橋爪信明, 長坂真路, 後藤和成: ヘテロ型スーパーコンピュータ TSUBAME の Linpack による性能評価, 情報処理学会研究報告, 2006-HPC-107, pp. 43–48 (2006).
15) Golub, G.H.and Van Loan, C.F.: *Matrix Computations*, Third ed., Johns Hopkins University Press, Baltimore, 1996.
16) GRAPE-DR Project. http://grape-dr.adm.s.u-tokyo.ac.jp/
17) Grosser, B. and Lang, B.: An $O(n^2)$ Algorithm for the Bidiagonal SVD, *Linear Algebra Appl.*, Vol. 358, pp. 45–70 (2003).
18) Gu, M. and Eisenstat, S. C.: A Divide-and-Conquer Algorithm for the Bidiagonal SVD, *SIAM J. Mat. Anal. Appl.*, Vol. 16, No. 1, pp. 79–92 (1995).
19) 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).
20) Iwasaki, M. and Nakamura, Y.: Accurate Computation of Singular Values in terms of Shifted Integrable Schemes, *Japan J. Indust. Appl. Math.*, Vol. 23, (2006).
21) Kakarala, R. and Ogunbona, P. O.: Signal Analysis Using a Multiresolution Form of the Singular Value Decomposition, *IEEE Trans. Image Processing*, Vol. 10, No. 5, pp. 724–735 (2001).
22) Lang, B.: Using Level 3 BLAS in Rotation-Based Algorithms, *SIAM J. Sci. Comput.*, Vol. 19, pp. 626–634 (1998).
23) 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).
24) Schreiber, R. and Van Loan, C. F.: A Storage-Efficient WY Representation for Products of Householder Transformations, *SIAM J. Sci. Stat. Comput.*, Vol. 10, pp. 53-57 (1989).
25) Toledo, S. and Rabani, E.: Very Large Electronic Structure Calculations using an Out-of-Core Filter-Diagonalization Method, *J. Comput. Phys.* Vol. 180, No. 1, pp. 256–269 (2002).
26) Top500 Supercomputing Sites. http://www.top500.org/
27) 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). http://www.na.cse.nagoya-u.ac.jp/ yamamoto /work.html
28) 柳井晴夫, 竹内啓: 射影行列・一般逆行列・特異値分解, 東京大学出版会, 1983.