2008－HPC－117 (4)
2008／10／15

ハウスホルダ型直交変換算法の多段階化によるブロック化と並列分散処理

村上 弘

首都大学東京 数理情報科学専攻

ハウスホルダ型の直交変換を用いて非常に細長い行列を上三角化するＱＲ－分解の操作を二段階に分けると，記憶参照の局所性が増すほか分散並列処理に対する適合性も向上できる．この方法を再帰的に用いると多段階化された方法も得られる．小規模なマルチコア型ＣＰＵの共有メモリ型の並列システム上での実験を数例示す．

A Multi-staged Block Algorithm of Householder-type Orthogonal Transformation for Distributed Parallel Processings

Murakami Hiroshi

Department of Mathematics and Information Sciences,
Tokyo Metropolitan University

The QR-factorization using the Householder-type orthogonal transformation of a very skinny and tall matrix into an upper triangular form is split in two-stages to improve the locality of storage references and the degree of adaptability to distributed parallel processings. The recursive use of this two-staged method also gives multi-staged methods. Some experiments on some small-sized shared memory parallel multi-core CPU systems are made.

1 はじめに

ハウスホルダ型の直交変換は行列の数値計算に於いて，直交分解（正規直交化，行列の QR-分解），固有値解析（行列の三重対角化やヘッセンベルグ化），特異値解析（行列の二重対角化）等で安定で精度の高い計算を実現する上で重要である．

「極めて細長い行列」を上三角化する（直交変換に基づいた）QR-分解を考える．QR-分解の全体を二段階に分けて処理することで，記憶参照の局所性や分散並列処理への適合性を向上できる．この二段階の方法を再帰的に適用すると，多段階の方法が得られる．「極めて細長い行列」でない場合は，行列の列分割も施すことで「極めて細長い行列」の処理を基にして計算を組み立てることができる．

記憶階層と経過時間： 階層記憶型計算機システムに於いて経過時間を短縮するには，階層間のデータ移動の量と回数を減らすことが重要である．そのためには計算対象のデータを領域分割して部分領域のデータ量が高速な記憶に収まるようにし，なるべく処理中の部分領域に属さないデータへの参照を減らすように算法を設計する．

分散並列処理システムと経過時間： 階層記憶の場合と同様に，複数の処理要素 (PE) を用いた並列計算を行う場合には経過時間を低減するためには，PE 間のデータ移動の量や回数を減らすことが重要である．処理データ全体を PE に収まる程度の容量に領域分割し，各 PE のデータ参照の大部分が PE 内に収まるように計算する．

以下では直交変換は行列の左側から作用するものとする．

複数のハウスホルダ型変換を適用して行列を上三角化する場合を考える．行列の列分割（ブロック分割）を行なう方法として，複数の鏡映変換をブロック内で蓄積した後で各列に対して蓄積された変換を適用する「蓄積法」や，より効率的な複数のブロック内で蓄積された鏡映変換の作用を行列積の形に変えて適用する WY-表現などの「ブロック変換化」[1, 6, 7, 9] があり，良く知られている．記憶参照の局所性が鏡映変換の単純な逐次適用に比べて向上するので処理が高速になる．

行列の行分割（ブロック分割）を行なう方法については，鏡映軸ベクトルを決定してから各列ベクトルを鏡映変換する操作の並列化法は簡単であるが，

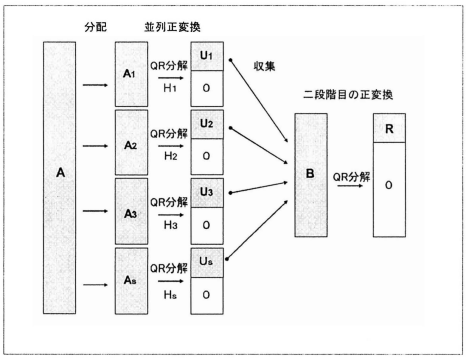

図 1: 上三角化（正変換）の過程.

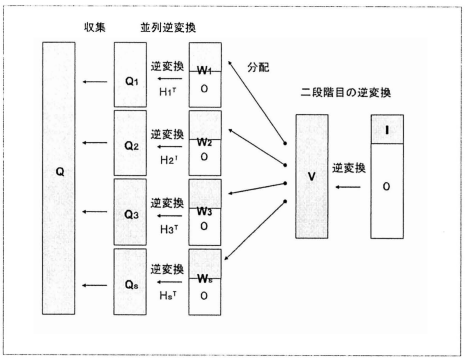

図 2: Q の陽的構成（逆変換）の過程.

データ通信量の割合が多く特に同期の頻度が高い．また，各列ベクトルを単に行分割するのではなくて，遂次の鏡映変換の各々を二段階（多段階）に分けて行なう方法も既に知られているが [3, 4, 5, 8]，これも同期の頻度が高い．今回の，縦長の行列に対する QR-分解過程の全体を二段階（多段階）化する方法は，遂次の鏡映変換ごとに二段階（多段階）で分けて行なう方法に比べると，同期の頻度を大幅に低減できる（粒度の大きい並列化）ので，分散並列処理に対する適合度が高い．

2 算法

$m\times n$ 行列 A は縦長（$m\gg n$）とする．

2.1 上三角化（正変換）.

直交変換を用いた行列の A の上三角化を以下のように，第一段目，第二段階目の 2 段階に分けて処理する（図 1）.

第一段階目： A の行分割による小行列を $A^{(1)}$, $A^{(2)}$, ..., $A^{(s)}$ とする．$A^{(i)}$ は $m_i\times n$ 行列．但し $m_i\geq n$ を仮定．いま，i=1, 2, ..., s に対して独立に，直交変換 $H^{(i)}$ を $A^{(i)}$ の左側から適用して QR-分解して得られる $m_i\times n$ の上三角行列を $U^{(i)}$ とする．つまり，$H^{(i)}A^{(i)}=U^{(i)}$ で，$U^{(i)}$ は先頭の n 行以外は全て 0. さらに，i=1, 2, ..., s に対して各 $U^{(i)}$ の先頭の n 個の行を集めて作った $(sn)\times n$ 行列を B とする．

第二段階目： 直交変換を用いて再び B を QR-分解する：$B=VR$. 但し V は $(sn)\times n$ の列直交な行列で R は $n\times n$ の上三角行列である．A の QR-分解による上三角行列が R となる．

A を上三角化する左側からの直交変換は，次の二種の操作を行の添字の対応を考慮しながら順に適用したものになる：1) 直交変換 $H^{(i)}$, i=1, 2, ..., s. 2) B の上三角化に用いた直交変換.

2.2 Q の陽な構成（逆変換）.

正規直交基底の Q は，正変換を構成した二段階の各変換を（単位行列から始めて）逆にたどると陽的に構成できる（図 2）．B の QR-分解により陽的に V を得た時点から開始するならば，以下のようになる.

1. V の sn 個の行を，「連続する n 個の行の組」s 個に（行列 B を作った際の行の集め方と逆操作になるように）分配する.

2. i=1, 2, ..., s に対し，$m_i\times n$ 行列 $W^{(i)}$ の先頭の n 個の行を i 番目の「連続する n 個の行の組」，後の行を 0 と置く.

3. $H^{(i)}$ の逆を $W^{(i)}$ に作用して $m_i\times n$ 行列 $Q^{(i)}=H^{(i)T}W^{(i)}$ を作る.

4. $Q^{(i)}$, i=1, 2, ..., s を縦に並べた $m\times n$ 行列を Q とする.

上記算法では，第一段階で小行列 $A^{(i)}$ を「直交変換」で完全に QR-分解し，それらの上三角行列を集めて（通常は小さな）$(sn)\times n$ 行列 B を作成し，次に第二段階で B を再び「直交変換」で QR-分解し，最終的な三角化を実現する.

注意： 方法は小行列の行数 m/s と n との比がある程度以上でないと無駄が多いので，s は m/n より十分小さい範囲に制約される．B の行数は s に比例するので，分割数 s を大きくとって並列性も高める場合には，B の QR-分解についても行分割ブロック化の対象にして並列度を高めないと，全体の性能が並列度に比例しなくなり低下する.

m/s を n に比べて十分大きくできない場合には，列分割ブロック化も採用して n が小さい場合に帰着させ，全体の計算を組み立てることが可能である（説明省略）.

表 1: 例 1：実効演算速度，m=12000，縦ブロック長 1000（12 分割），倍精度，Gflops 値．

n	1core	2core	3core	4core
10	1.85	3.20	4.00	4.80
20	2.67	3.37	6.85	6.00
30	2.86	5.53	7.08	9.18
40	3.01	5.60	7.60	9.59
50	2.44	5.42	7.83	9.90
60	2.66	5.60	7.74	9.97
70	2.99	5.60	7.77	10.0
80	2.98	5.47	7.76	10.1
90	2.86	5.20	7.71	10.0
100	2.97	5.54	7.70	10.0

表 2: 例 2：実効演算速度，m=36000，縦ブロック長 1000（36 分割）．倍精度，Gflops 値．

n	1core	2core	3core	4core
10	2.53	4.24	6.00	5.14
20	2.33	4.72	6.94	8.73
30	2.76	5.38	7.28	9.53
40	2.64	5.50	7.50	9.60
50	2.89	5.35	7.64	9.89
60	2.76	5.56	7.63	9.89
70	2.84	5.48	7.65	9.82
80	2.83	5.50	7.61	9.67
90	2.82	5.47	7.56	9.66
100	2.69	5.44	7.51	9.68

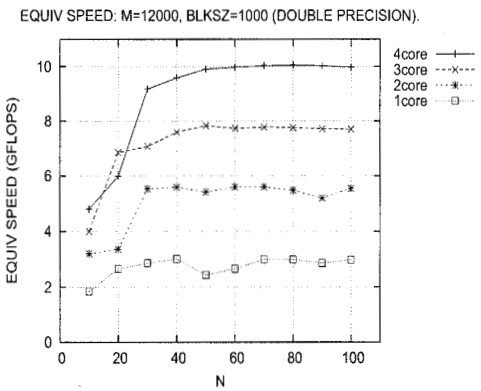

図 3: 例 1：実効演算速度，m=12000，縦ブロック長 1000（12 分割）．

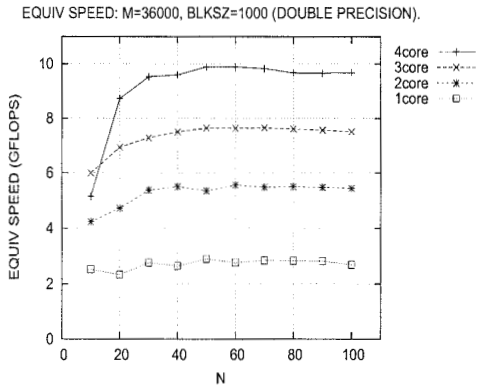

図 4: 例 2：実効演算速度，m=36000，縦ブロック長 1000（36 分割）．

3 実験

「通常」のハウスホルダ変換により $m \times n$ 行列を QR-分解して Q を陽な形で得るまでには，約 $4mn^2 - 4n^3/3$ 回の FP 演算が必要である [7]．そこで，逆に FP 演算量を $4mn^2 - 4n^3/3$ と定義して計算の経過時間で割った値である「実効演算速度」を求めた．

3.1 Intel Core2 Quad のシステム

使用した計算機システムは 1CPU のマルチコア (4 コア) 型 SMP 計算機で，CPU は intel Core2 Quad Q6600 (4 コア，2 ダイ，65nm，2.4GHz，8MB 共有 L2 (4MB×2)，SSE3)．理論ピーク性能 (倍精度) は 4×9.6=38.4Gflops．主記憶容量は 4 Gbytes (DDR-2，1Gbyte モジュール ×4 本)．使用したコンパイラは intel Fortran ver.10.0 for Linux で，コンパイルオプションは: -axT -O3 -openmp -ipo．並列化には OpenMP だけを用いた．

例 1： 行列 A の行数 m=12000 を行分割で縦ブロック長 1000 により，s=12 個の長さの等しいブロックに分けた例である．列数 n は 10 から 10 刻みで 100 までとし，12 個のブロックを CPU 内の 4 個あるコアをそれぞれ 1 個，2 個，3 個，4 個用いて並列処理した場合の実効演算速度の Gflops 値の表 (表 1)，およびグラフ (図 3) を示す．

例 2： 行列 A の行数 m=36000 を行分割で縦ブロック長 1000 により，s=36 個の長さの等しいブロックに分けた例である．列数 n は 10 から 10 刻みで 100 までとし，36 個のブロックを CPU 内の 4 個あるコアをそれぞれ 1 個，2 個，3 個，4 個用いて並列処理した場合の実効演算速度の Gflops 値の表 (表 2)，およびグラフ (図 4) を示す．

例 3： 行列 A の行数 m=120000 を行分割で縦ブロック長 2000 により s=60 個の長さの等しいブロックに分けた例である．列数 n は 10 から 10 刻みで 100 までとし，60 個のブロックを CPU 内の 4 個あるコ

表 3: 例 3：実効演算速度，m=120000，縦ブロック長 2000（60 分割），倍精度，Gflops 値．

n	1core	2core	3core	4core
10	2.07	4.17	5.05	6.49
20	2.27	4.78	6.23	7.90
30	2.70	5.18	7.02	8.94
40	2.80	5.43	7.31	9.41
50	2.78	5.51	7.56	9.40
60	2.88	5.55	7.73	9.72
70	2.90	5.67	7.65	9.65
80	2.91	5.68	7.64	9.72
90	2.89	5.67	7.51	9.54
100	2.87	5.67	7.53	9.31

表 4: 例 4：実効演算速度，m=1200000，縦ブロック長 2000（600 分割），倍精度，Gflops 値．

n	1core	2core	3core	4core
10	1.86	3.56	4.47	5.63
20	2.45	4.71	6.11	8.03
30	2.66	5.23	7.01	9.12
40	2.72	5.42	7.49	9.26
50	2.75	5.43	7.36	9.42
60	2.74	5.31	7.15	9.12
70	2.73	5.32	6.95	8.75
80	2.71	5.20	6.66	8.41
90	2.67	5.12	6.45	8.06
100	2.64	5.10	6.02	7.76

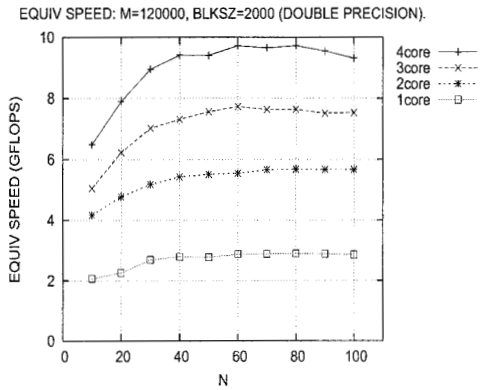

図 5: 例 3：実効演算速度，m=120000，縦ブロック長 2000（60 分割）．

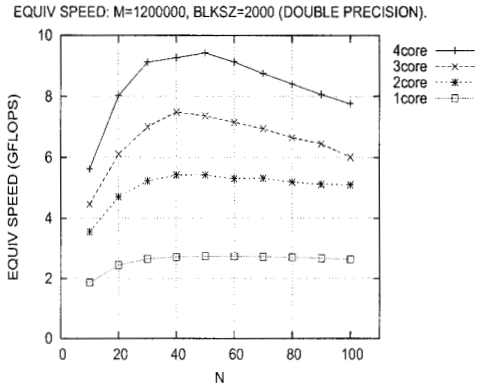

図 6: 例 4：実効演算速度，m=1200000，縦ブロック長 2000（600 分割）．

アをそれぞれ 1 個，2 個，3 個，4 個用いて並列処理した場合の実効演算速度の Gflops 値の表（表 3），およびグラフ（図 5）を示す．

例 4： 行列 A の行数 m=1200000 を行分割で縦ブロック長 2000 により s=600 個の長さの等しいブロックに分けた例である．列数 n は 10 から 10 刻みで 100 までとし，600 個のブロックを CPU 内の 4 個あるコアをそれぞれ 1 個，2 個，3 個，4 個用いて並列処理した場合の実効演算速度の Gflops 値の表（表 4），およびグラフ（図 6）を示す．

例 5： 例 4 の場合と行列 A の寸法やブロック行分割の数 s は同じであるが，算法の二段階目の行列 B の QR-分解に縦分割ブロック化を再び適用し，指定された個数のコアで並列処理を行なっている点が異なる．実効演算速度の Gflops 値の表（表 5），およびグラフ（図 7）を示す．例 4 の場合に比べると全体的な性能の向上が得られていること，特に n が大きいところでの性能低下が改善されている．

例 6： 並列化をコンパイラに指示せず，4 個の CPU コアの 1 個だけを用いた倍精度計算を行ない，計算法の比較をしてみた．m=100000, n=100 の乱数行列 A を（コアの並列性を用いずに）鏡映変換の逐次適用の標準的なハウスホルダ QR-分解法を素直に Fortran90 でコーディングしたものを用いて Q を陽に求める計算を行なったところ，実効性能は 0.895Gflops であった．同じ行列を（コアの並列性を用いずに）二段階法でブロック長 2000 で行分割した場合の実効性能は 2.88Gflops であった．さらに中間の B の QR-分解にも二段階法を適用した場合の実効性能は 3.01Gflops であった．並列処理ではなくても，二段階法の採用により記憶参照の局所性が高まることで性能が良くなることを示している．BLAS3 化を行なっていないが，コア 1 個の理論ピーク性能である 9.6Gflops の 30%程度の比較的高い性能が得られている．

表 5: 例 5：実効演算速度，m=1200000, 縦ブロック長 2000（改善版）．二段階目の QR-分解にも，縦分割ブロック化を適用．倍精度，Gflops 値．

n	1core	2core	3core	4core
10	1.98	3.87	4.96	6.09
20	2.38	4.90	6.51	8.44
30	2.59	5.40	7.49	9.62
40	2.84	5.57	7.99	10.3
50	2.77	5.65	8.02	10.4
60	2.87	5.96	8.35	10.7
70	2.89	5.87	8.28	10.5
80	2.91	5.95	8.30	10.6
90	2.88	5.88	8.19	10.5
100	2.93	5.95	8.18	10.3

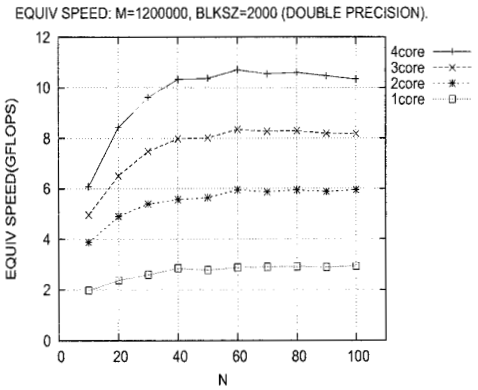

図 7: 例 5：実効演算速度，m=1200000，縦ブロック長 2000（改善版）．

3.2 HA8000 システムの 1 ノード

HA8000（東京大学情報基盤センター T2K システム）の 1 ノードは 4CPU（=16 コア）である．CPU は AMD Opteron 8356（4 コア/CPU, 2.3GHz），L2：512kbyte/コア，L3：2Mbyte/CPU，ピーク演算性能（倍精度）：9.2GFLOPS/コア．主記憶容量は 32Gbytes．コンパイラは intel Fortran version 10.1, コンパイルオプションは：-axT -O3 -openmp -ipo. 並列化には OpenMP だけを使用した．

例 7： 行列 A の行数 m=144000 を行分割で縦ブロック長 1200 により，s=120 個の長さの等しいブロックに分けた例である．列数 n は 10 から 10 刻みで 100 までとし，120 個のブロックの処理を 4CPU 内の 16 個あるコアのうち（120 の約数である）1 個，2 個，3 個，4 個，5 個，6 個，8 個，10 個，12 個，15 個を用いた 10 通りの場合について，OpenMP による並列処理で実測した．得られた実効演算速度の Gflops 値の表（表 6），およびグラフ（図 8, 9）を示す．この例も，二段階目の B の QR-分解に縦分割ブロック化を適用している．すべての最内側の QR-分解には，スカラー計算法では標準的な逐次的に鏡映変換を施すハウスホルダ QR-法を Fortran90 のソースコードで記述したものを用いている．CPU 内の 2M バイトの L2 キャッシュを 4 個のコアが平等に分配して使う場合，つまり 512k バイト/コアであるが，1 語 8 バイトで，小行列のブロック長を 1200 とすると，小行列全体が L2 にちょうど収まるのは $n = 52$ が上限となる．これが n が 50 を越えると性能が下がる原因の一つとなっているようである．

表 6: 例 7：実効演算速度，m=144000, 縦ブロック長 1200（120 分割）．二段階目の QR-分解にも縦分割ブロック化を適用，倍精度，Gflops 値．

n	1core	2core	3core	4core	5core
10	2.01	3.93	5.72	7.07	7.89
20	2.37	4.71	6.89	8.89	10.9
30	2.54	5.03	7.49	9.76	11.9
40	2.63	5.25	7.65	10.2	12.4
50	2.65	5.23	7.78	10.1	12.9
60	2.57	5.12	7.61	9.91	12.0
70	2.49	4.90	7.24	9.44	11.7
80	2.39	4.76	7.02	9.40	11.4
90	2.32	4.57	6.79	8.81	11.1
100	2.28	4.50	6.60	8.65	10.9

n	6core	8core	10core	12core	15core
10	9.35	11.7	14.5	17.2	20.6
20	12.5	15.2	18.6	21.3	25.2
30	13.4	17.5	21.2	24.5	28.8
40	14.5	18.7	22.5	25.9	30.3
50	14.9	19.0	22.2	26.6	31.7
60	14.4	18.7	22.5	26.4	31.4
70	13.5	18.3	22.0	25.8	30.7
80	13.2	17.6	21.3	25.0	29.5
90	12.6	16.1	20.7	24.1	28.9
100	12.6	15.8	20.2	23.1	28.3

今後の予定

今回の実験で用いた最内側「QR-分解」は鏡映変換の逐次適用でトータルの性能は演算ピーク性能の 1/5〜1/4 程度であった．各 PE 内部の FP 演算装置の性能を十分に引き出すには，(特に列数 n が十分に大きいときには) 最内側の QR-分解には，列方向ブロック化の技法である：

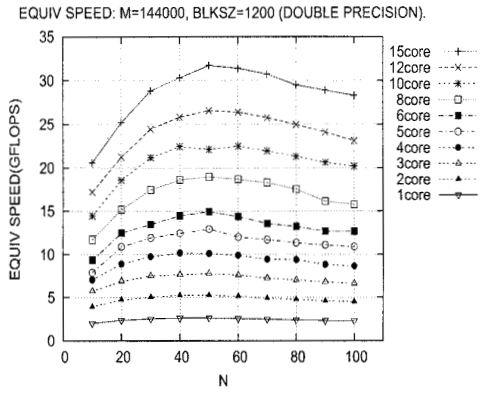

図 8: 例 7：実効演算速度，m=144000，縦ブロック長 1200（120 分割）．

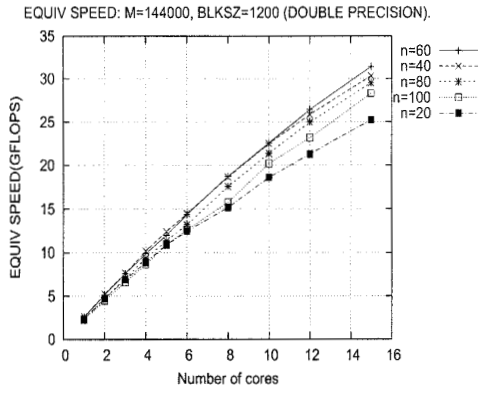

図 9: 例 7：実効演算速度，m=144000，縦ブロック長 1200（120 分割）．

- 鏡映変換を複数蓄積，各列に一度に施す（列ブロック化）．
- 鏡映変換をブロック化，level-3 BLAS で演算密度を高める．

[1, 6, 9] を併用すると良いであろう．

今回の説明は，QR-分解を二段階に分割する例を中心としたが，第一段階目の $A^{(i)}$ の QR-分解，第二段階目の B の QR-分解にもそれぞれ二段階化法の適用が再度できて，多段階化／多階層化が可能である．計算機システムが階層的な通信ネットワーク構造を持つ場合には，それに適合した形で階層的に算法を設計するのが良いであろう．

MPI を用いたより大規模な並列化や，二段階に分割された直交変換を用いる右三角化（階段化）を用いる応用例として，大規模な対称行列のブロック三重対角化などについても今後実験したい．

参考文献

[1] Bischof C. and Van Loan C.: The *WY* representation for products of Householder matrices, *SIAM J. Sci. Stat. Comput.*, **8**, No.1 (1987), 2–13.

[2] Bischof C.H.: QR Factorization Algorithms For Coarse-Grained Distributed Systems, Ph.D. Thesis, *Dep. Comput. Sci. Cornell Univ. Ithaca, NY*, TR 88-939(1988).

[3] Chu E. and George A.: QR Factorization of a Dense Matrix on a Hypercube Multiprocessor, *SIAM J. Sci. Stat. Comput.*, **11**, No.5 (1990), 990–1028.

[4] Chuang H.Y.H., Chen L., and Qian D.: A Size-Independent Systolic Array for Matrix Triangularization and Eigenvalue Computation, *Circuits,Systems,and Signal Processing*, **7**, No.2 (1988), 173–189.

[5] Dongarra J.J., Sameh A.H. and Sorensen D.C.: Implementation of some concurrent algorithms for matrix factorization, *Parallel Computing*, **3** (1986), 25–34.

[6] Dongarra J.J., Hammarling S.J. and Sorensen D.C.: LAPACK Working Note #2; Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations, ANL/MCS-TM-99 (1987).

[7] Golub G.H. and Van Loan C.F.: Matrix Computations, 3rd ed., The John Hopkins University Press, Baltimore London, 1996.

[8] Pothen A. and Raghavan P.: Distributed Orthogonal Factorization: Givens and Householder Algorithms, *SIAM J. Sci. Stat. Comput.*, **10**, No.6 (1989), 1113–1134.

[9] Schreiber R. and Van Loan C.: A storage efficient *WY* representation for products of Householder transformations, *SIAM J. Sci. Stat. Comput.*, **10**, No.1 (1989), 53–57.

[10] Stathopoulos A., and Wu K.: A block orthogonalization procedure with constant synchronization requirements, *SIAM J. Sci. Comput.*, **23**, No.6 (2002), 2165–2182.