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　19–29 (July 2009)

マルチコアCPUシステムおよび小規模SMP並列システム上でのTall Skinny型QR分解法の実験

村　上　　弘[†1]

Tall Skinny 型 Algorithm による QR 分解をマルチコア型 CPU のシステムおよびマルチコア型 CPU からなる小規模な SMP システムの上で，並列化に OpenMP を用いて行った計算の実験例を示す．

An Experiment of Tall Skinny Type QR-factorization Algorithm on a Multicore CPU System and a Small SMP System

Hiroshi Murakami[†1]

We show some experiments of Tall Skinny type Algorithm for QR-factorization on a multicore CPU system and a small shared memory parallel multicore CPU system using OpenMP for parallelization.

1. はじめに

ハウスホルダ型の直交変換は，行列数値計算において安定で精度の高い計算を実現するうえで重要である．

今回は，大規模な線形方程式や固有値問題の密行列に対する各種ブロック化算法の構成要素として，あるいは疎行列に対する反復解法中の複数ベクトルの再直交化操作として重要度の高い「きわめて細長い行列（tall skinny matrix）」の直交変換を用いた QR 分解を扱うことにする（行列があまり細長くない場合にも，行列の列分割を用いて「きわめて細長い行列」の処理を基礎に計算を組み立てられる）．

マルチコア CPU システムおよび小規模 SMP 並列システム上で，Tall Skinny 型算法による QR 分解（あるいは別名 AllReduce 算法）[6),17),18)] に基づいた処理を，キャッシュのサイズを考慮してブロックを分散して，各ブロックの処理を並列に処理することで，列数が 100 以下の細長い行列に対して，実験に用いたシステムの上限性能の約 1/4 程度を得たことを報告する．

以下では直交変換は行列に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，ブロック内で蓄積された複数の鏡映変換の作用を行列積の形に変えて適用するさらに効率的な WY 表現などの「ブロック変換化」[1),8),14),21)] があり，よく知られている．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）を行う方法については，通常の鏡映軸ベクトルを決定し，それを用いて各列ベクトルを鏡映変換する操作をブロック分割に沿って並列化する方法は簡単であるが，同期の頻度が高い．また，通常の鏡映変換ではなくて，行分割に対応する各ブロック内部およびブロック相互間の 2 段階の鏡映変換の合成による直交変換を用いて，毎回 1 列ずつ順番に，対角線より下側を消去（上三角化）する方法もすでに知られている[4),5),7),20)] が，これも同期の頻度が高い．

今回実験例を報告する Tall Skinny 行列に対する QR 分解の方法[6),17),18)] は，毎回 1 列ずつ上三角化のための消去を行う方法と比べて，同期の頻度を大幅に低減させることができる（粒度の大きい並列化）ので，並列処理への適合度が高い．

以前から演算の対象となるデータのプロセッサとメモリ間あるいはプロセッサ相互のデータ転送能力が実効性能上の隘路になりがちであった．この困難を解消/低減するために，線形計算である LU 分解や QR 分解などに対しては大規模行列に対するブロック化算法が開発されてきた．近年，回路製作上の困難と発熱の除去が困難であることにより，演算コア単体での性能向上は停滞傾向を示し，そのためマルチプロセッサ化やプロセッサ内部のマルチコア化による性能上限の向上が追求されているため，最近はブロック化算法の重要度が増しており，この方面の算法の研究は大変さかんであるように思われる[2),3),6),9)–13),15)–19),22)]．

†1 首都大学東京 数理情報科学専攻
Department of Mathematics and Information Sciences, Tokyo Metropolitan University

2. 実験に用いた T-S 型 QR 算法の詳細

$m\times n$ 行列 A は縦長（$m\gg n$）であるとする．

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 1 段階，第 2 段階の 2 段階に分けて処理する（**図 1**）．

第 1 段階：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．ただし $m_i\geq n$ 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，$H_iA_i = U_i$ で，U_i は先頭の n 行以外はすべて 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 2 段階：直交変換を用いて再び B を QR 分解する：$B = VR$．ただし 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 2 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 2 段階の各変換を（単位行列から始めて）逆にたどることで陽的に構成できる（**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 とおく．

（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 1 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（元の行列 A よりも行数の少ない）$(sn)\times n$ 行列 B を作成し，次に第 2 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：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 よりも十分に小さい範囲に制約される．B の行数は s に比例するから，分割数 s を大きくとって並列性も高めるもしくは小行列 A_i のサイズをおさえる場合には，B の QR 分解についても T-S 型 QR 分解を適用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が，今回の実験ではそのような例は示していない．

3. 実　験　例

一般的な $m\times n$ 行列 A を通常のハウスホルダ変換により QR 分解し，陽な形の Q を得る

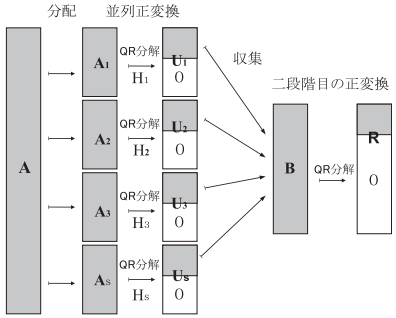

図 1　上三角化（正変換）の過程
Fig. 1　Process of upper triangulation (Forward transformation).

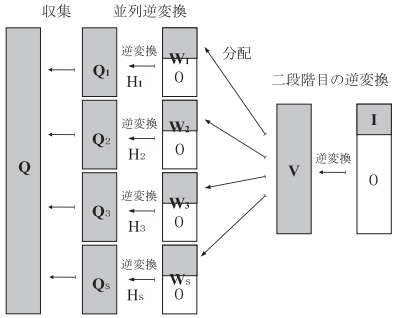

図 2　Q の陽的構成（逆変換）の過程
Fig. 2　Process of explicit formation of Q (Backward transformation).

には，約 $4mn^2 - 4n^3/3$ 回の FP 演算が必要である[14]．そこで FP 演算量を $4mn^2 - 4n^3/3$ と逆に定義し，計算の経過時間で割った値を「実効演算速度」とした．以下のすべての実験例で用いた行列 A の要素は範囲 $(-1, 1)$ の一様乱数である．ただし経過時間は，あらかじめ行分割ブロック化された A を与えて，陽的な Q が同じ分割の形で求めるまでの時間とした．

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（2.4 GHz，4 コア，2 ダイ，各コア専用の命令とデータ用の L1 キャッシュがそれぞれ 32 K バイト，ダイ上の 2 個のコアで共有された 4 MB の L2 キャッシュが 2 組で合計 8 MB，SSSE3 命令，65 nm，FSB 1066 MHz）．コア 1 個の演算性能上限（倍精度）は 4×2.4 Gflops=9.6 Gflops で，CPU パッケージあたりでは 4×9.6=38.4 Gflops．主記憶容量は 4 G バイト（DDR2，PC6400，1 G バイトモジュール ×4 本）．使用したコンパイラは intel Fortran ver.10.1 for Linux で，コンパイルオプションを`-xT -fast -openmp` とし，並列化にはコンパイラの OpenMP の機能だけを利用した．

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 個から 4 個まで用いて並列処理した場合の実効演算速度の Gflops 値のグラフ（**図 3**）を示す（この例は性能が特にばらつくので，4 回測定した中で最良の値をプロットしている）．

各ブロック小行列 A_i は最大の n=100 の場合でも，1000×100 で 0.8 M バイトなので，L2 キャッシュ全体 8 M バイトを最大 4 個のコアで等分した容量 2 M バイトよりも小さく，各小行列の QR 分解による三角化や逆変換の演算のほとんどは L2 キャッシュ内で行える．中間の行列 B のサイズは最大の n=100 の場合でも 1200×100 で，0.96 M バイトであり，L2 キャッシュに十分収まる．

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 個から 4 個まで用いて並列処理した場合の実効演算速度の Gflops 値のグラフ（**図 4**）を示す（4 回計測の最良値をプロットした）．

各ブロックの小行列 A_i のサイズは例 1 の場合と同じで，各小行列の QR 分解による三角化や逆変換の計算はほとんど L2 キャッシュ内で行える．中間の行列 B のサイズは最大の n=100 の場合でも 3600×100 で 2.88 M バイトであり，ダイ 1 つ分の L2 キャッシュ 4 M バイトに収まる．

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 個あるコアをそれぞれ 1 個から 4 個まで用いて並列処理した場合の実効演算速度の Gflops 値のグラフ（**図 5**）を示す（4 回計測の最良値をプロットした）．

各ブロックの小行列は最大の n=100 の場合に，2000×100 で 1.6 M バイトなので，L2

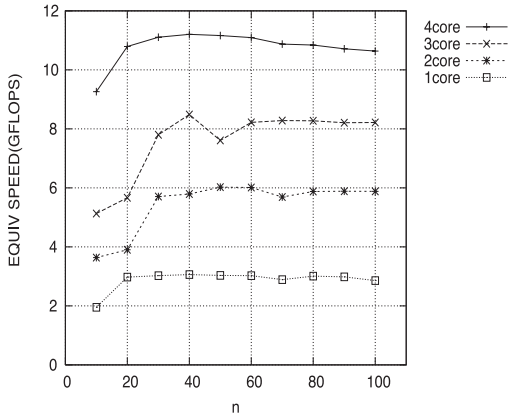

図 3　例 1：実効演算速度（Gflops），m=12000，縦ブロック長 1000（12 分割），倍精度演算
Fig. 3　Example 1: Effective speed (DP Gflops), m=12000, vertical block size=1000 (12 capitation).

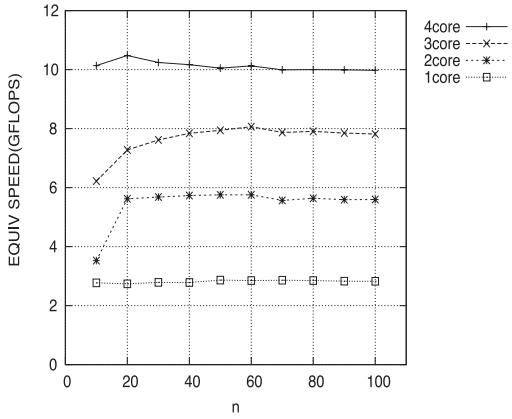

図 4　例 2：実効演算速度（Gflops），m=36000，縦ブロック長 1000（36 分割），倍精度演算
Fig. 4　Example 2: Effective speed (DP Gflops), m=36000, vertical block size=1000 (36 capitation).

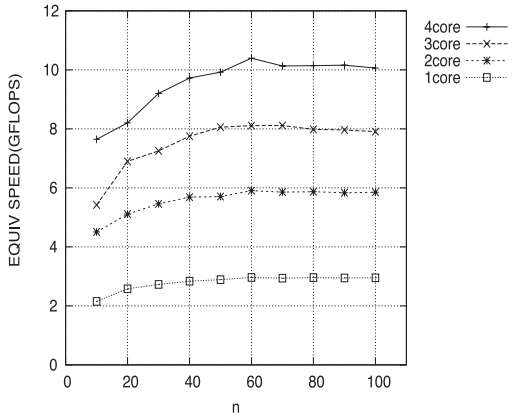

図 5　例 3：実効演算速度（Gflops），m=120000，縦ブロック長 2000（60 分割），倍精度演算
Fig. 5　Example 3: Effective speed (DP Gflops), m=120000, vertical block size=2000 (60 capitation).

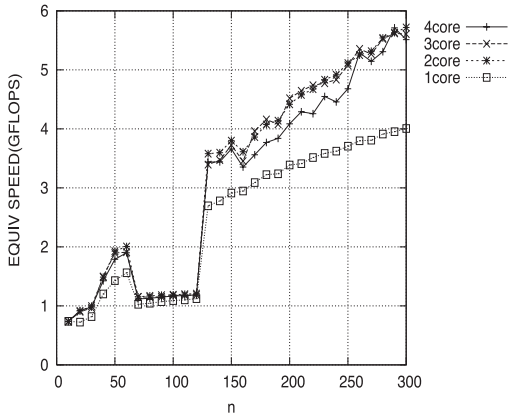

図 6　例 3 との比較：LAPACK（intel MKL），実効演算速度（Gflops），m=120000，倍精度演算
Fig. 6　Comparison to Example 3: Effective speed (DP Gflops) of LAPACK (intel MKL), m=120000.

キャッシュを最大 4 個のコアで等分したサイズの 2 M バイトよりも小さく，各小行列の QR 分解による三角化や逆変換の演算のほとんどは L2 キャッシュ内で行える．中間の行列 B のサイズは最大の n=100 の場合でも 6000×100 で 4.8 M バイトであり，ダイ 2 個分の L2 キャッシュ8 M バイト以内に収まる．例 1〜3 のグラフがどれも似たような傾向を示しているのは，小行列も，中間の行列 B も L2 キャッシュに収まる程度のサイズだからである．

ここで比較のために，intel MKL version 10.1 が提供する LAPACK ルーチンの `DGEQRF` と `DORGQR` の組合せを（行列 A を行分割せずに）直接適用して，行数 m=120000 で列数 n を 10 から 10 刻みで 300 までについて，コア数を 1 から 4 まで変えて QR 分解を行い，Q を陽な形で求める実効速度の Gflops 値をプロットした（**図 6**）．これを見て気がつくことは，n の値が小さい場合の性能が悪く，特にグラフの n が 70 から 120 の範囲では性能が特徴的に著しく劣化してすべてのコア数で 1.1 Gflops 程度であること，n が 130 以上の場合には性能はコア数が 1 から 2 になっても 1.5 倍弱程度以下の増加で，コアの数が 3 の場合は 2 コアと同等だが 4 コアではむしろ劣化していることである．

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 個から 4 個まで用いて並列処理した場合の実効演算速度の Gflops 値のグラフ（**図 7**）を示す（4 回計測をした値にはばらつきがほとんどない）．

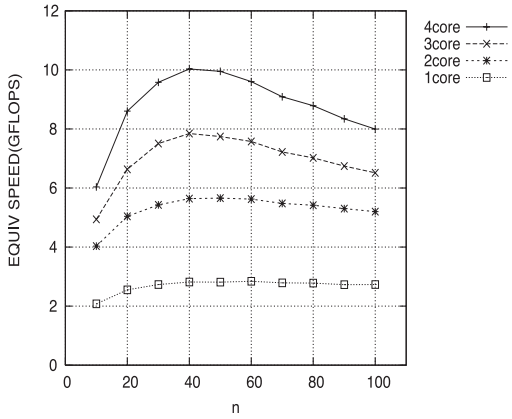

図 7　例 4：実効演算速度（Gflops），m=1200000，縦ブロック長 2000（600 分割），倍精度演算
Fig. 7　Example 4: Effective speed (DP Gflops), m=1200000, vertical block size=2000 (600 capitation).

各ブロックの小行列のサイズは例 3 の場合と同じであり，各小行列の QR 分解による三角

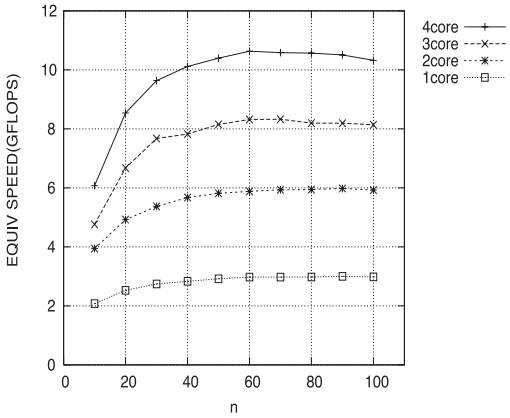

図 8　例 5：実効演算速度（Gflops），m=1200000，縦ブロック長 2000，倍精度演算（改善版）
Fig. 8　Example 5: Effective speed (DP Gflops), improved version, m=1200000, vertical block size=2000 (600 capitation).

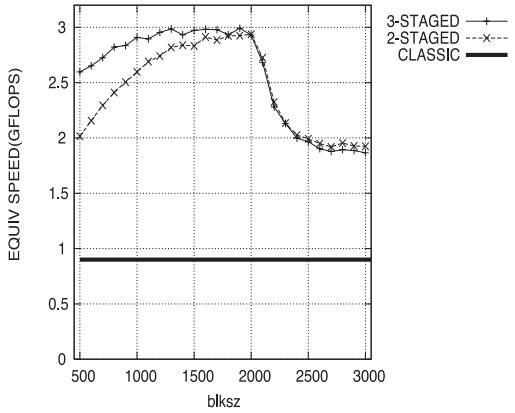

図 9　例 6：実効演算速度（Gflops），m=100000，n=100，Core2 Quad Q6600（2.4 GHz），シングルコア，倍精度演算
Fig. 9　Example 6: Effective speed (DP Gflops), m=100000, n=100, by single core of intel Core2 Quad Q6600 (2.4GHz).

化や逆変換の演算のほとんどは L2 キャッシュ内で行える．中間の行列 B は，n=40 のときに 24000×40 で 7.68 M バイトとなりほとんど L2 キャッシュの全容量 8 M バイトに等しく，最大の n=100 の場合には 60000×100 で 48 M バイトで，L2 キャッシュの容量を大幅に超える．このため B を行分割を行わない方法で QR 分解を行うと L2 とメモリ間の記憶の入れ換えが多発する．このことが n=40 を超えたあたりで性能が劣化している原因であろう．次の例 5 のように，B の QR 分解には行分割ブロック化を適用するのが良い状況にある．

例 5：行列 A の寸法やブロック行分割の数 s は例 4 の場合と同じであるが，算法の 2 段階目の中間の行列 B にも T-S 型 QR 分解を再び適用して，指定された個数のコアで並列処理を行っている点が異なる．実効演算速度の Gflops 値のグラフ（**図 8**）を示す（4 回計測をした値にはばらつきがほとんどない）．先の例 4 の場合に比べて全体的な性能の向上が得られており，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：コンパイル時に OpenMP 並列化の指示をせず，4 個ある CPU コアのうち 1 個だけを用いて倍精度計算を行い，計算法の比較をしてみた（**図 9**）．行列 A のサイズを m=100000，n=100 とした．鏡映変換を逐次適用する標準的なハウスホルダ QR 分解法（CLASSIC）を素直に Fortran90 でコーディングしたものを用いて，A を直交化し Q を陽に求める計算の実効性能は 0.900 Gflops であった．同じ行列を（コアの並列性を用いずに）2 段階法（2-STAGED）でブロック長 2000（ブロック数 50 個）で行分割した場合の実効性能は 2.94 Gflops であった．さらに中間の B の QR 分解にも行分割ブロック化を適用した場合（3-STAGED）の実効性能は 2.95 Gflops であった（ブロック長 `blksz` が 2000 のとき，各小行列 A_i のサイズは 1.6 M バイトで，コアが乗っているダイ 1 個分の L2 キャッシュの容量 4 M バイトに比べて小さい．また中間の行列 B のサイズは 4 M バイトで，ダイ上の L2 キャッシュにちょうど収まる程度の大きさである）．並列処理ではなくても，行分割ブロック化の採用で小行列のブロックがキャッシュ内に収まり，記憶参照の局所性の向上で計算性能が良くなることを示している．計算法としては BLAS2 であるが，コア 1 個の演算性能上限である 9.6 Gflops の 31%程度の比較的高い性能が得られた．ブロック長 `blksz` を 2000 よりも小さくするとブロックの個数が増して中間の行列 B の行数が増えるので，B の QR 分解に行分割ブロック化を再度施す方が有利である傾向も読み取れる（たとえば `blksz` が 500 では，行分割を再度施した場合の性能は 2.6 Gflops 程度に対して，施さない場合の性能は 2.0 Gflops 程度である）．

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，2.3 GHz），L2：512 K バイト/コア，L3：2 M バイト/CPU，演算性能上限（倍精度）：9.2 Gflops/コア．主記憶容量は 32 G バイト．コンパイラは intel Fortran version 10.1，コンパイルオプションを`-axT -O3 -ipo`

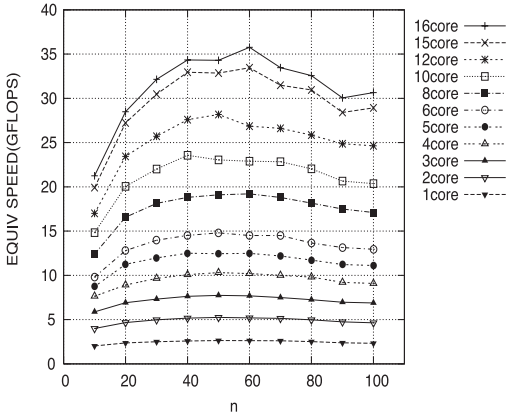

図 10　例 7：実効演算速度（Gflops），m=240000，縦ブロック長 1000（240 分割），2 段階目の QR 分解にも縦分割ブロック化を適用，倍精度演算
Fig. 10　Example 7: Effective speed (DP Gflops), m=240000, vertical block size=1000 (240 capitation). The QR-decomposition in the second stage is also blocked.

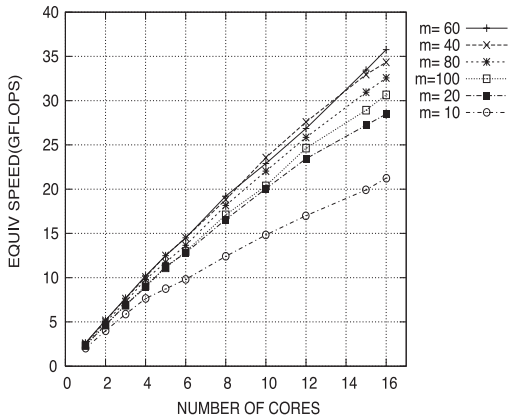

図 11　例 7：実効演算速度（Gflops），m=240000，縦ブロック長 1000（240 分割），2 段階目の QR 分解にも縦分割ブロック化を適用，倍精度演算
Fig. 11　Example 7: Effective speed (DP Gflops), m=240000, vertical block size=1000 (240 capitation). Vertical blocking is also applied to the second stage QR-decomposition.

`-openmp` とし，並列化にはコンパイラの持つ OpenMP の機能だけを使用した．プログラムは intel Core2Duo Q6600 のものとまったく同じで，Fortran90（+OpenMP 指示行）で記述し，ブロック内の最内側の QR 分解には，逐次的に鏡映変換を施す標準的なハウスホルダ QR 法を用いている．行列 A の要素も区間 $(-1, 1)$ の一様乱数で与えた．

例 7：行列 A の行数 m=240000 を行分割で縦ブロック長 1000 により，s=240 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，240 個のブロックの処理を 4CPU 内の 16 個あるコアのうち（240 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，16 個を用いた 11 通りの場合それぞれについて，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 値のグラフ（**図 10**，**図 11**）を示す．各コア数の場合に対して，最も高い演算性能を示した n における演算性能の値の，使用コア数での演算性能上限に対する割合はそれぞれ，1 コア：28.6%，2 コア：28.5%，3 コア：28.0%，4 コア：28.0%，5 コア：27.2%，6 コア：26.8%，8 コア：26.1%，10 コア：25.7%，12 コア：25.5%，15 コア：24.2%，16 コア：24.3%となった．この実験例では中間の B の QR 分解に対して行分割ブロック化を再度適用している（中間の B のサイズは最大の n=100 の場合に，19.2 M バイトで，4CPU の場合でも L2+L3 キャッシュの全容量を超えている）．各コアに専属する L2 キャッシュの容量は 512 K バイトで，コアの L2 に小行列全体がちょうど収まるのは小行列のブロック長が 1000 で 1 語は 8 バイトなので $n = 65$ が上限となる．これが n が 60 を超えるときに性能の下がる原因の 1 つとなっているようである．CPU 内の 4 個のコアで共有された 2 M バイトの L3 キャッシュを均等に各コアで使う場合の容量は 512 K バイト/コアなので，L2+L3 の合計は各コアあたり 1 M バイトある．だから小行列の最大 n=100 の場合の容量 0.8 M バイトでも L3 の範囲までであれば収まるが，L2 へのアクセスは L3 よりも高速であり，n が 60 を超えたあたりから L2–L3 間でのキャッシュラインの入れ換えが発生して速度低下を生じていると思われる．

注記：この例の表やグラフの計測データは，計測を数度繰り返して得られた測定値の中から比較的良い値を採用したものである．特に最大の 16 コアの場合に低い性能値が得られる場合が多いが，これはノード内のシステム管理やノード間通信のために背後で動いている（一般ユーザからは不可視の）プロセスの影響であろうと思われる．性能を安定にするには，各スレッドへの物理 CPU コアや主記憶の物理ページの割当てを制御するのがよいが，今回の実験はその点に関しては特に何も行っていない．

4. 今後の予定

今回の実験では最内側のブロック内の「QR 分解」には鏡映変換を逐次適用する最も単純

な方法を用いて，トータルの性能としてシステムの演算性能上限の 1/4 程度を得ている．行列を適切なブロック分割で小行列に分けて，CPU パッケージ内部のキャッシュに各計算コアが小行列をほぼ格納できるならば，CPU パッケージ外部のメモリや，CPU 間の通信路に対するアクセス量や頻度を低減させることができるので，計算中の記憶参照はほぼ CPU パッケージ内部のキャッシュの速度で行えるようになる．ただし，鏡映変換の逐次適用による QR 分解は Level-2 BLAS で，CPU 内部の最外側のキャッシュを走査するので記憶参照の局所性は十分に高いものではない．各計算コア内部の 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，LAPACK の QR 分解のルーチンでも採用されている列方向ブロック化の技法[1),8),21)]：

- 鏡映変換を複数蓄積，各列に 1 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

を併用すればよいであろう．これについて，現在はまだ予備実験の段階であるが，小行列の QR 分解や逆変換の部分を intel MKL ライブラリ ver.10.1 の LAPACK のルーチンに置き換えた場合には，列数 n=100 以下では単純なハウスホルダ法による Fortran90 コードに比べて性能が半減した．現状の LAPACK の QR 分解関係のルーチンの実装は，列数が十分大きい場合の性能を特に重視しているようである．

今回は，QR 分解の 2 段階への分割例を主に説明したが，それぞれ第 1 段階目の A_i の QR 分解，第 2 段階目の B の QR 分解にも方法を再度適用した多段階化/多階層化は，計算機システムがマルチコア，SMP，通信ネットワークのようなデータ交換の階層構造を持つ場合に，算法もそれと適合させた形で階層的に構成するのがよいと思われるので，今後は MPI も用いて今回よりも多くの計算コアを用いたある程度の規模の並列化も実験したいと考えている．

参 考 文 献

1) Bischof, C. and Van Loan, C.: The WY representation for products of Householder matrices, *SIAM J. Sci. Stat. Comput.*, Vol.8, No.1, pp.2–13 (1987).
2) Buttari, A., Langou, J., Kurzak, J. and Dongarra, J.: Parallel tiled QR factorization for multicore architectures, Tech. Report UT-CS-07-598, Univ. Tennessee (2007). LAPACK Working Note #190
3) Buttari, A., Langou, J., Kurzak, J. and Dongarra, J.: A class of parallel tiled linear algebra algorithms for multicore architectures, Tech. Report UT-CS-07-600, Univ. Tennessee (2007). Also, LAPACK Working Note #191
4) Chu, E. and George, A.: QR factorization of a dense matrix on a hypercube multiprocessor, *SIAM J. Sci. Stat. Comput.*, Vol.11, No.5, pp.990–1028 (1990).
5) 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*, Vol.7, No.2, pp.173–189 (1988).
6) Demmel, J., Grigori, L., Hoemmen, M.F. and Langou, J.: Communication-optimal parallel and sequential QR and LU factorizations, Tech. Report UCB/EECS-2008-89, Univ. California at Berkeley (Aug. 2008). Also, LAPACK Working Note #204
7) Dongarra, J.J., Sameh, A.H. and Sorensen, D.C.: Implementation of some concurrent algorithms for matrix factorization, *Parallel Computing*, Vol.3, pp.25–34 (1986).
8) Dongarra, J.J., Hammarling, S.J. and Sorensen, D.C.: Block reduction of matrices to condensed forms for eigenvalue computations, ANL/MCS-TM-99 (1987). Also, LAPACK Working Note #2
9) Elmroth, E. and Gustavson, F.: New serial and parallel recursive QR factorization algorithms for SMP systems, *PARA '98: Proc. 4th international workshop on applied parallel computing, large scale scientific and industrial problems*, Lec. Notes in Comput. Sci., Vol.1541, pp.120–128, Springer-Verlag, New York (1998).
10) Elmroth, E. and Gustavson, F.: Applying recursion to serial and parallel QR factorization leads to better performance, *IBM J. of Research and Development*, Vol.44, pp.605–624 (2000).
11) Elmroth, E. and Gustavson, F.G.: High-performance library software for QR factorization, *PARA '00: Proc. 5th international workshop on applied parallel computing, new paradigms for HPC in industry and academia*, Lec. Notes in Comput. Sci., Vol.1947, pp.53–63, Springer-Verlag, New York (2001).
12) Elmroth, E. and Gustavson, F.G.: A faster and simpler recursive algorithm for the LAPACK routine DGELS, *BIT*, Vol.41, pp.936–949 (2001).
13) Elmroth, E., Gustavson, F., Jonsson, I. and Kågström, B.: Recursive blocked algorithms and hybrid data structures for dense matrix library software, *SIAM Review*, Vol.46, pp.3–45 (2004).
14) Golub, G.H. and Van Loan, C.F.: *Matrix Computations*, *3rd ed.*, The John Hopkins Univ. Press, Baltimore London (1996).
15) Bunter, B.C. and van de Geijn, R.A.: Parallel out-of-core computation and updating the QR factorization, *ACM, Trans. Mathematical Software* (*TOMS*), Vol.31, No.1, pp.60–78 (2005).
16) Kurzak, J. and Dongarra, J.: QR factorization for the CELL processor, Tech. Report UT-CS-08-616, Univ. Tennessee (2008). Also, LAPACK Working Note #201
17) Langou, J.: Allreduce Householder factorizations, Poster Sessions (Jul. 11th, 2007). *2007 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industorial Applications*, Météopole Toulouse,

France (2007). http://www.precond07.enseeiht.fr/preprogram.html. Proceeding paper is available from: J. Langou: Allreduce Householder factorizations, ENSEEIHT-IRIT RT/AP0/07/10, pp.103–106. (also CERFACS TR/PA/07/71) http://www.precond07.enseeiht.fr/proceedings.pdf

18) Langou, J.: All Reduce Algorithms: Application to Householder QR Factorization, *Sparse Days at CERFACS Meeting*, Oct. 12th, 2007, Toulouse, France (2007). http://www.cerfacs.fr/algor/PastWorkshops/SparseDays2007/schedule.html. Slide is available from: http://www.cerfacs.fr/algor/PastWorkshops/SparseDays2007/Slides/langou.pdf

19) Ltaief, H., Kurzak, J. and Dongarra, J.: Parallel block Hessenberg reduction using algorithms-by-tiles for multicore architectures revisited, Tech. Report UT-CS-08-624, Univ. Tennessee (2008). Also, LAPACK Working Note #208

20) Pothen, A. and Raghavan, P.: Distributed orthogonal factorization: Givens and Householder algorithms, *SIAM J. Sci. Stat. Comput.*, Vol.10, No.6, pp.1113–1134 (1989).

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

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

付　　録

A.1　プログラムリストの例

以下に，本論文中の実験で用いた QR 分解を行分割により 2 段階で行う場合の Fortran90 サブルーチン `QRFC_BLK` のソースコードリスト例を示す．リストには OpenMP 並列化の指示行も含まれている（紙数の制約から行列 B の QR 分解を再度行分割する場合のサブルーチンの例は省略する）．

先頭の`#define` で始まる行で，定義しているマクロ名 `MY_RK` の値は使用する実数型の種別値で，コンパイラ処理系に依存する．Intel Fortran の 64-bit 倍精度実数型の種別値は 8 である．

論文中の m_i はすべて同じ値 `BLKSZ` とし，s を `NBLK` としている．m は `BLKSZ` の倍数 `BLKSZ`×`NBLK` とした（最後のブロックのサイズに半端を許すような変更を加えることは可能である）．小行列 A_i は `ABLK(:,:,i)` に対応する．

Fortran90 プログラムリスト

```
#define MY_RK 8 ! KIND-VALUE OF REAL-TYPE TO USE
!*********************************************
! TALL SKINNY QR-FACTORIZATION WITHOUT COLUMN
! EXCHANGE: "A=QR" ("Q" IS WRITTEN OVER "A").
!THIS CODE IS FREE TO USE, MODIFY, RE-DISTRIBUTE.
!*********************************************
SUBROUTINE QRFC_BLK(ABLK,BLKSZ,N,NBLK,R,LDR,B)
IMPLICIT NONE
INTEGER,PARAMETER::RK=MY_RK
INTEGER,INTENT(IN)::BLKSZ,N,NBLK
REAL(RK),INTENT(INOUT)::ABLK(BLKSZ,N,NBLK)
REAL(RK),INTENT(OUT)::R(LDR,N) ! R(N,N).
INTEGER,INTENT(IN)::LDR
REAL(RK)::B(N,NBLK,N) ! WORK AREA.
!-----------------------
REAL(RK) PIV(N,NBLK) ! SHARED VAR.
REAL(RK) U(N,N)      ! PRIVATE VAR.
INTEGER IB           ! PRIVATE VAR.
!-----------------------
!$OMP PARALLEL DO PRIVATE (U)
DO IB=1,NBLK
  CALL FWD_BLK(BLKSZ,N,ABLK(1,1,IB),PIV(1,IB),U)
  B(:,IB,:)=U(:,:)
ENDDO
!---
CALL QRFC(NBLK*N,N,B,NBLK*N,R,SIZE(R,1))
!---
!$OMP PARALLEL DO PRIVATE (U)
DO IB=1,NBLK
  U(:,:)=B(:,IB,:)
  CALL BWD_BLK(BLKSZ,N,ABLK(1,1,IB),PIV(1,IB),U)
ENDDO
END SUBROUTINE QRFC_BLK

!*************************************
!  FORWARD TRANSFORM IN THE BLOCK.
!*************************************
SUBROUTINE FWD_BLK(BLKSZ,N,ABLK,PIV,U)
IMPLICIT NONE
INTEGER, PARAMETER::RK=MY_RK
INTEGER,INTENT(IN)::BLKSZ,N
REAL(RK),INTENT(INOUT)::ABLK(BLKSZ,N)
REAL(RK),INTENT(OUT)::PIV(N)
REAL(RK),INTENT(OUT)::U(N,N)
!-----------------------
REAL(RK) V(BLKSZ),S,ALPHA,T
INTEGER K,J
!-----------------------
DO K=1,N
  V(K:)=ABLK(K:,K)
  S=SIGN(SQRT(SUM(V(K:)**2)),V(K))
  V(K)=V(K)+S
  ABLK(K,K)=V(K)
  PIV(K)=-S
  U(:K-1,K)=ABLK(:K-1,K)
  U(K,K)=PIV(K)
  U(K+1:,K)=0.0_RK
  IF(K==N)EXIT
```

```
  IF(PIV(K)==0.0_RK)CYCLE
  ALPHA=1.0_RK/(PIV(K)*V(K))
  DO J=K+1,N
    T=DOT_PRODUCT(V(K:),ABLK(K:,J))*ALPHA
    ABLK(K:,J)=ABLK(K:,J)+V(K:)*T
  ENDDO
ENDDO
END SUBROUTINE FWD_BLK

!****************************************
!  BACKWARD TRANSFORM IN THE BLOCK.
!****************************************
SUBROUTINE BWD_BLK(BLKSZ,N,ABLK,PIV,U)
IMPLICIT NONE
INTEGER, PARAMETER::RK=MY_RK
INTEGER,INTENT(IN)::BLKSZ,N
REAL(RK),INTENT(INOUT)::ABLK(BLKSZ,N)
REAL(RK),INTENT(IN)::PIV(N)
REAL(RK),INTENT(IN)::U(N,N)
!-----------------------
REAL(RK) V(BLKSZ),ALPHA,T
INTEGER K,J
!-----------------------
DO K=N,1,-1
  V(K:)=ABLK(K:,K) ! RECALL THE H-VECTOR.
  ABLK(:N,K)=U(:N,K)
  ABLK(N+1:,K)=0.0_RK
  IF(PIV(K)==0.0_RK)CYCLE
  ALPHA=1.0_RK/(PIV(K)*V(K))
  DO J=K,N
    T=DOT_PRODUCT(V(K:),ABLK(K:,J))*ALPHA
    ABLK(K:,J)=ABLK(K:,J)+V(K:)*T
  ENDDO
ENDDO
END SUBROUTINE BWD_BLK

!****************************************
! HOUSEHOLDER QR-FACTORIZATION WITHOUT
! COLUMN EXCHANGE:A=QR(Q IS WRITTEN OVER A).
!****************************************
SUBROUTINE QRFC(M,N,A,LDA,R,LDR)
IMPLICIT NONE
INTEGER,PARAMETER::RK=MY_RK
INTEGER,INTENT(IN)::M,N ! M >= N.
REAL(RK),INTENT(INOUT)::A(LDA,N) ! A(M,N).
INTEGER,INTENT(IN)::LDA ! LEADING SIZE OF A.
REAL(RK),INTENT(OUT)::R(LDR,N) ! R(N,N).
INTEGER,INTENT(IN)::LDR ! LEADING SIZE OF R.
!-----------------------------------------
REAL(RK) S,PIV(N),ALPHA,T
INTEGER J,K
!-----------------------------------------
IF(M<N)STOP 'QRFC: ERROR M < N.'
!-------------------------------
! FORWARD-TRANSFORM MAKES R AND IMPLICIT Q.
DO K=1,N
  ! MAKE THE HOUSEHOLDER VECTOR.
  S=SUM(A(K:M,K)**2)
  S=SIGN(SQRT(S),A(K,K))
  A(K,K)=A(K,K)+S
  PIV(K)=-S
  !-----------------------
  IF(PIV(K)==0.0_RK)CYCLE
  ALPHA=1.0_RK/(PIV(K)*A(K,K))
!$OMP PARALLEL DO PRIVATE (T)
  DO J=K+1,N
    T=DOT_PRODUCT(A(K:M,K),A(K:M,J))*ALPHA
    A(K:M,J)=A(K:M,J)+A(K:M,K)*T
  ENDDO
ENDDO
!-----------------------
! PACK THE MATRIX "R".
!$OMP PARALLEL DO
DO K=1,N
  R(:K-1,K)=A(:K-1,K)
  R(K,K)=PIV(K)
  R(K+1:N,K)=0.0_RK
ENDDO
!-----------------------
! BACK-TRANSFORM TO MAKE Q EXPLICITLY.
DO K=N,1,-1
  IF(PIV(K)/=0.0_RK)THEN
    ALPHA=1.0_RK/(PIV(K)*A(K,K))
!$OMP PARALLEL DO PRIVATE (T)
    DO J=K+1,N
      T=DOT_PRODUCT(A(K:M,K),A(K:M,J))*ALPHA
      A(K:M,J)=A(K:M,J)+A(K:M,K)*T
    ENDDO
    T=A(K,K)*ALPHA
    A(:K-1,K)=0.0_RK
    A(K,K)=1.0_RK+A(K,K)*T
    A(K+1:M,K)=A(K+1:M,K)*T
  ELSE
    A(:K-1,K)=0.0_RK
    A(K,K)=1.0_RK
    A(K+1:M,K)=0.0_RK
  ENDIF
ENDDO
END SUBROUTINE QRFC
```

A.2　タイル分けをしない方式のプログラム例

T-S 行列をタイル分けをせずに，通常の二次元配列の形式で与えて（多段階で）QR 分解を行う Fortran90 によるプログラム（ルーチン HQRF_BLK2）の例（これも計算カーネルは level-2 BLAS）を以下に示す．このタイル分けをしない方式の方が，利用上は多くの場合に便利であろう．リストは OpenMP 並列化の指示行を含んでいる．タイル分けを行う場合と比べて同等の性能結果が実験で得られた．

計算は簡単のため再帰的に書かれている．再帰の中止条件には調整の余地がある．行分割のブロックの大きさ BLKSZ は行列の寸法 M，N に対し計算機特性を考慮し適切に選んで（チューニングして）与える（M の約数でなくてよい）．システムのメモリ割当てが first touch の場合に，不必要なメモリ移動を抑えて良い性能を並列化で出すには，このルーチンを呼ぶ

前の配列 A の初めの要素設定の処理を，並列に行ブロック分割に適合した形で行えるとよい．

先頭の#define で始まる行で，定義しているマクロ名 MY_RK の値は使用する実数型の種別値で，コンパイラ処理系に依存する．たとえば Intel Fortran，日立 Fortran90 での 64-bit 倍精度実数型の種別値は 8 である．

プログラムリスト

```
#define MY_RK 8
!*****************************************************
!THIS CODE IS FREE TO COPY,USE,MODIFY,RE-DISTRIBUTE.
! TALL SKINNY QR-FACTORIZATION WITHOUT COLUMN
!  EXCHANGE: "A=QR" (Q IS WRITTEN OVER "A").
! (NOTE: THIS CODE DOES NOT USE THE FACT THAT
! THE INTERMEDIATE MATRIX B HAS MANY ZEROS.)
!*****************************************************
RECURSIVE SUBROUTINE HQRF_BLK2(M,N,A,LDA,R,LDR,BLKSZ)
IMPLICIT NONE
INTEGER,PARAMETER::RK=MY_RK ! KIND-VALUE.
INTEGER,INTENT(IN)::M,N ! M >= N.
REAL(RK),INTENT(INOUT)::A(LDA,N) ! FOR A(M,N).
INTEGER,INTENT(IN)::LDA
REAL(RK),INTENT(OUT)::R(LDR,N) ! FOR R(N,N).
INTEGER,INTENT(IN)::LDR
INTEGER,INTENT(IN)::BLKSZ ! THE ROW BLOCK SIZE.
!-----------------------------------------------
REAL(RK),ALLOCATABLE::B(:,:),PIV(:,:)
INTEGER IFLAG
INTEGER MB,LDB
INTEGER NBLK,IB
INTEGER NROW ! NUMBER OF ROWS IN THE BLOCK.
NROW(IB)=MIN(BLKSZ,M-(IB-1)*BLKSZ) ! STATEMENT FUNC.
!-----------------------------------------------
 NBLK=(M+BLKSZ-1)/BLKSZ
 IF(M<4*BLKSZ)THEN ! RECURSION STOP CRITERIA.
  CALL HQRF2(M,N,A,LDA,R,LDR) ! TRADITIONAL H-QR.
  RETURN
 ENDIF
 ! ALLOCATION OF WORK ARRAYS B AND PIV.
 MB=(NBLK-1)*N+MIN(NROW(NBLK),N)
 LDB=MB
 ALLOCATE(B(LDB,N),PIV(N,NBLK),STAT=IFLAG)
  IF(IFLAG/=0)THEN
    STOP 'HQRF_BLK2: FAILED TO ALLOCATE B AND PIV.'
 ENDIF
 !-----
!$OMP PARALLEL DO
 DO IB=1,NBLK
  CALL FWD_BLK2(NROW(IB),N, &
        A(1+(IB-1)*BLKSZ,1),LDA, &
        PIV(1,IB),B(1+(IB-1)*N,1),LDB)
 ENDDO
 !-----
 CALL HQRF_BLK2(MB,N,B,LDB,R,LDR,BLKSZ)
 !-----
!$OMP PARALLEL DO
 DO IB=1,NBLK
  CALL BWD_BLK2(NROW(IB),N, &
        A(1+(IB-1)*BLKSZ,1),LDA, &
        PIV(1,IB),B(1+(IB-1)*N,1),LDB)
 ENDDO
 !-----
 DEALLOCATE(B,PIV)
END SUBROUTINE HQRF_BLK2

 !*****************************************
 ! FORWARD DECOMPOSITION INSIDE THE BLOCK.
 !*****************************************
 SUBROUTINE FWD_BLK2(M,N,A,LDA,PIV,U,LDU)
 IMPLICIT NONE
 INTEGER,PARAMETER::RK=MY_RK ! KIND-VALUE.
 INTEGER,INTENT(IN)::M,N
 REAL(RK),INTENT(INOUT)::A(LDA,N) ! FOR A(M,N).
 INTEGER,INTENT(IN)::LDA
 REAL(RK),INTENT(OUT)::PIV(N)
 REAL(RK),INTENT(OUT)::U(LDU,N) ! FOR U(N,N).
 INTEGER,INTENT(IN)::LDU
 !-----------------------------------------
 REAL(RK) S,ALPHA,T
 INTEGER K,J
 !---------------------------------------
  IF(M<=N)THEN
   ! POSSIBLE FOR THE LAST FRAGMENT BLOCK.
   U(1:M,1:N)=A(1:M,1:N)
   RETURN
  ENDIF
  DO K=1,N
   S=SIGN(SQRT(SUM(A(K:M,K)**2)),A(K,K))
   A(K,K)=A(K,K)+S
   PIV(K)=-S
   U(1:K-1,K)=A(1:K-1,K)
   U(K,K)=PIV(K)
   U(K+1:N,K)=0.0_RK
   IF(K==N)EXIT
   IF(PIV(K)/=0.0_RK)THEN
    ALPHA=1.0_RK/(PIV(K)*A(K,K))
    DO J=K+1,N
     T=DOT_PRODUCT(A(K:M,K),A(K:M,J))*ALPHA
     A(K:M,J)=A(K:M,J)+A(K:M,K)*T
    ENDDO
   ENDIF
  ENDDO
 END SUBROUTINE FWD_BLK2

 !***************************************
 ! BACKWARD TRANSFORMATION INSIDE THE BLOCK.
 !***************************************
 SUBROUTINE BWD_BLK2(M,N,A,LDA,PIV,U,LDU)
 IMPLICIT NONE
 INTEGER,PARAMETER::RK=MY_RK ! KIND-VALUE.
 INTEGER,INTENT(IN)::M,N
 REAL(RK),INTENT(INOUT)::A(LDA,N) ! A(M,N).
 INTEGER,INTENT(IN)::LDA
 REAL(RK),INTENT(IN)::PIV(N)
 REAL(RK),INTENT(IN)::U(LDU,N) ! U(N,N).
 INTEGER,INTENT(IN)::LDU
 !-------------------------------------
 REAL(RK) ALPHA,T
```

```
 INTEGER K,J
 !--------------------------------------
  IF(M<=N)THEN
   ! POSSIBLE FOR THE LAST FRAGMENT BLOCK.
   A(1:M,1:N)=U(1:M,1:N)
   RETURN
  ENDIF
  DO K=N,1,-1
   IF(PIV(K)/=0.0_RK)THEN
    ALPHA=1.0_RK/(PIV(K)*A(K,K))
    DO J=N,K+1,-1
     T=DOT_PRODUCT(A(K:M,K),A(K:M,J))*ALPHA
     A(K:M,J)=A(K:M,J)+A(K:M,K)*T
    ENDDO
    T=DOT_PRODUCT(A(K:N,K),U(K:N,K))*ALPHA
    A(1:K-1,K)=U(1:K-1,K)
    A(K:N,K)=U(K:N,K)+A(K:N,K)*T
    A(N+1:M,K)=A(N+1:M,K)*T
   ELSE
    A(1:N,K)=U(1:N,K)
    A(N+1:M,K)=0.0_RK
   ENDIF
  ENDDO
 END SUBROUTINE BWD_BLK2

!*******************************************
! TRADITIONAL METHOD.
! HOUSEHOLDER QR-FACTORIZATION WITHOUT COLUMN
! EXCHANGE: "A=QR" ("Q" IS WRITTEN OVER "A").
!*******************************************
SUBROUTINE HQRF2(M,N,A,LDA,R,LDR)
IMPLICIT NONE
INTEGER,PARAMETER::RK=MY_RK ! KIND-VALUE.
INTEGER,INTENT(IN)::M,N ! M >= N.
REAL(RK),INTENT(INOUT)::A(LDA,N) ! FOR A(M,N).
INTEGER,INTENT(IN):: LDA ! LEADING SIZE OF A.
REAL(RK),INTENT(OUT)::R(LDR,N) ! FOR R(N,N).
INTEGER,INTENT(IN)::LDR ! LEADING SIZE OF R.
!-------------------------------------------
REAL(RK) PIV(N),S,ALPHA,T
INTEGER J,K
!-----------------------------------
 IF(M<N)PRINT*,'HQRF2: ERROR M < N.'
!$OMP PARALLEL PRIVATE(K,ALPHA)
 !--------------------------------
 ! FORWARD-TRANSFORM TO MAKE R, AND IMPLICITLY Q.
 DO K=1,MIN(N,M)
  !-------------------
  ! DETERMINE THE HOUSEHOLDER VECTOR.
!$OMP WORKSHARE
  S=SUM(A(K:M,K)**2)
  S=SIGN(SQRT(S),A(K,K))
  A(K,K)=A(K,K)+S
  PIV(K)=-S
!$OMP END WORKSHARE
  !-------------------
  IF(PIV(K)/=0.0_RK)THEN
   ALPHA=1.0_RK/(PIV(K) * A(K,K))
!$OMP DO PRIVATE(T)
   DO J=K+1,N
    T=DOT_PRODUCT(A(K:M,K),A(K:M,J))*ALPHA
    A(K:M,J)=A(K:M,J)+A(K:M,K)*T
   ENDDO
  ENDIF
 ENDDO
 !-----------------------
 ! PACK THE MATRIX R.
!$OMP DO
 DO K=1,N
  R(1:K-1,K)=A(1:K-1,K)
  R(K,K)=PIV(K)
  R(K+1:N,K)=0.0_RK
 ENDDO
 !-----------------------
 ! BACK-TRANSFORM TO MAKE Q EXPLICITLY.
 DO K=MIN(N,M),1,-1
  IF(PIV(K)/=0.0_RK)THEN
   ALPHA=1.0_RK/(PIV(K)*A(K,K))
!$OMP DO PRIVATE(T)
   DO J=K+1,N
    T=DOT_PRODUCT(A(K:M,K),A(K:M,J))*ALPHA
    A(K:M,J)=A(K:M,J)+A(K:M,K)*T
   ENDDO
!$OMP WORKSHARE
   A(1:K-1,K)=0.0_RK
   T=A(K,K)*ALPHA
   A(K,K)=1.0_RK+A(K,K)*T
   A(K+1:M,K)=A(K+1:M,K)*T
!$OMP END WORKSHARE
  ELSE
!$OMP WORKSHARE
   A(1:K-1,K)=0.0_RK
   A(K,K)=1.0_RK
   A(K+1:M,K)=0.0_RK
!$OMP END WORKSHARE
  ENDIF
 ENDDO
!$OMP END PARALLEL
END SUBROUTINE HQRF2
```

(平成 20 年 10 月 3 日受付)
(平成 21 年 2 月 24 日採録)

村上　弘（正会員）

首都大学東京・数理情報科学専攻の准教授．研究分野は，数学または科学の問題の数値的あるいは記号的方法による効率の良いあるいは精度の良い解法およびその並列化である．1992 年，北海道大学理学博士号（化学第二学専攻）を取得．