A BLAS-3 Version of the QR Factorization with Column Pivoting*

Gregorio Quintana-Ortí † Xiaobai Sun ‡
Christian H. Bischof §

PRISM Working Note # 26
LAPACK Working Note # 114
Univ. of Tennessee at Knoxville Computer Sci. Dept. CS-96-334
August, 1996

ABSTRACT. The QR factorization with column pivoting (QRP), originally suggested by Golub and Businger in 1965, is a popular approach to computing rank-revealing factorizations. Using BLAS Level 1, it was implemented in LINPACK, and, using BLAS Level 2, in LAPACK. While the BLAS Level 2 version delivers, in general, superior performance, it may result in worse performance for large matrix sizes due to cache effects. We introduce a modification of the QRP algorithm which allows the use of BLAS Level 3 kernels while maintaining the numerical behavior of the LINPACK and LAPACK implementations. Experimental comparisons of this approach with the LINPACK and LAPACK implementations on IBM RS/6000, SGI R8000, and DEC Alpha platforms show considerable performance improvements.

Key words. QR Factorization, Column Pivoting, Rank Revealing Factorization, Block Algorithm

---

*All authors were partially supported by the Advanced Research Projects Agency, under contract DM28E04120 and P-95006. Quintana also received support from the European ESPRIT Project 9072–GEPPCOM. Sun also received support from National Science Foundation agent ASC-ASC-9005933. Bischof also received supported from the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

†Departamento de Informática, Universidad Jaime I, Campus Penyeta Roja, 12071 Castellón, Spain, `gquintan@inf.uji.es`. This work was partially performed while the author was visiting the Departments of Computer Science at Duke University and the University of Tennessee.

‡Department of Computer Science, Duke University, D107, Levine Science Research Center, Durham, NC 27708-0129, `xiaobai@cs.duke.edu`. This work was partially performed while the author was visiting the Department of Computer Science at the University of Tennessee.

§Mathematics and Computer Science Division Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439-4844, `bischof@mcs.anl.gov`.

1 Introduction

For any matrix A, there exists a so-called rank-revealing QR factorization (RRQRF)

$$AP = QR = (Q_1\, Q_2) \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{1}$$

where P is a permutation matrix, R_{11} is upper triangular, and R_{22} is numerically negligible [22]. The order r of R_{11} then reveals the numerical rank of A, the first r columns of Q form an orthonormal basis for the range space of A, and the first r columns of AP are the largest independent set of columns of A. This information is needed, for example, in geodesy [17], computer-aided design [19], nonlinear least-squares problems [25], the solution of integral equations [15], and in the calculation of splines [18]. Other applications arise in beam-forming [8], spectral estimation [23], regularization [21,29], and eigenproblems [3].

Algorithms for the reliable computation of rank-revealing factorizations have recently received considerable attention (see, for example [6, 7, 10, 11, 20, 26, 27]). However, the most common approach to computing such a RRQRF is the column pivoting procedure suggested by Businger and Golub [9]. This QR factorization with column pivoting (QRP) may fail to reveal the numerical rank correctly, but it is widely used due to its simplicity and practical reliability. Thus, it is also a very useful preprocessing step for the more reliable (and more expensive) RRQRF algorithms.

The LINPACK [13] subroutine `xQRDC` and the LAPACK [1, 2] subroutine `xGEQPF` both implement the Businger/Golub scheme using Level 1 and 2 BLAS [24, 14], respectively. As a rule of thumb, Level 2 BLAS perform better than Level 1 BLAS, and Level 3 BLAS [12] using matrix-matrix kernels perform better still. However, on cache-based architectures, this rule of thumb must be used with caution as a Level 1 BLAS based implementation may exhibit better cache data locality than a Level 2 BLAS implementation.

This paper introduces a variant of the QR factorization with column pivoting which allows the use Level 3 BLAS kernels, thus increasing cache data locality while enabling the use of the most efficient BLAS kernels. The paper is structured as follows: In Section 2 we describe the basic QRP algorithm and the pertinent features of the Level 1 and 2 BLAS implementation. A block algorithm for implementing QRP, allowing for the use of Level 3 BLAS kernels while maintaining the behavior of the QRP algorithm, is presented in Section 3. Experimental results on IBM RS/6000, SGI R8000, and DEC Alpha/AXP platforms are presented in Section 4. Lastly, we summarize our work and outline potential avenues of further improvement.

2 The Traditional QR Factorization with Column Pivoting

The basic scheme for the QR factorization with column pivoting as proposed by Businger and Golub [9] can be described as shown in Figure 2, using the MATLAB notation. We assume that the reader is familiar with Householder transformations and their application in the context of a QR factorization (see,

Setup:

Permutation vector: perm(j) = j, $\quad j = 1 : n$

Column norm vector: colnorms(j) = $\|Ae_j\|_2^2$, $\quad j = 1 : n$

Reduction Steps:

For $j = 1 : n$

1. **Pivoting:** Choose p such that colnorms(p) = max(colnorms($j : n$))
   If (colnorms(p) == 0) STOP
   If ($j \neq p$) then % interchange
   perm($[j, p]$) = perm($[p, j]$), $A(:, [j, p]) = A(:, [p, j])$
   colnorms($[j, p]$) = colnorms($[p, j]$)
   Endif
2. **Reduction:** Determine a Householder matrix H_j such that
   $H_j\, A(j : m, j) = \pm \|A(j : m, j)\|_2\, e_1$.
3. **Matrix Update:**
   $A(j : m, j+1 : n) = H_j\, A(j : m, j+1 : n)$
4. **Norm Downdate:**
   colnorms($j+1 : n$) = colnorms($j+1 : n$) $-$ $A(j, j+1 : n)$.^2

Endfor

Figure 1: The Traditional Algorithm for the QR Factorization with Column Pivoting

for example, [16, pp. 195-197,211-212]). The notation e_1 is used to denote the first canonical unit vector $(1, 0, \ldots, 0)^T$ of appropriate length.

The LINPACK routine **xQRDC** and the LAPACK routine **xGEQPF** substantially differ only in the implementation of the matrix update. Since a Householder matrix H is a rank-1 modification of the identity,

$$H = I - \tau v v^T$$

its application requires the computation

$$HA = A - \tau v v^T A.$$

xQRDC is column oriented in the sense that the matrix update is done column by column. For each column j, $j = 1 : n$, it uses **xDOT**, a BLAS Level-1 kernel, to compute $v^T A e_j$ and then updates Ae_j using a **xAXPY** call. In contrast, LAPACK's **xGEQPF** is matrix-vector oriented — it first computes the row vector $v^T A$ using the BLAS Level-2 routine **xGEMV** for a matrix-vector product, then applies a rank-1 update with the BLAS Level-2 routine **xGER**.

Hence, while **xQRDC** fetches and touches each column of A only once, **xGEQPF** has to fetch and touch twice, both for the matrix-vector multiply and the rank-1 update. If the cache is big enough that the second fetch is from the cache and not from memory, this does not matter, but otherwise the BLAS Level

2 style of implementation requires roughly twice the number of main memory accesses. Even though most assembler implementations of BLAS 2 kernels would exploit architectural features and the known and regular data access pattern of a BLAS 2 kernel, the memory access penalty may outweigh these factors. The experimental results for the DEC Alpha platform in Section 4 illustrate this point.

3 A Block Algorithm for the QR Factorization with Column Pivoting

We describe in this section a new variant of the QRP algorithm that can employ BLAS Level 3 kernels. What seems to have kept the QRP procedure in Figure 2 from using BLAS Level 3 is the norm downdate scheme (step 4.) — at every step we must downdate *all* column norms before we can select the next pivot column among the remaining ones. The formula for the norm downdate we used in Figure 2 is obviously not numerically reliable, and G. W. Stewart developed a robust scheme for LINPACK which is also adopted in LAPACK. This scheme monitors the accuracy of the downdate and recomputes the column norms only when serious cancellation occurs. The norm downdate scheme has at least two noticeable features: 1) it makes the computation of column norms affordable and hence make the column pivoting scheme practical, and 2) it governs the numerical aspects of the QRP procedure. For example, it ensures that the diagonal elements of the upper triangular matrix R be arranged in non-increasing order. This property is important, for instance, for graded matrices.

Given the practical reliability of the QR factorization with column pivoting, our goal then is to design a block algorithm that maintains the same norm downdating and pivoting scheme, and hence computes the same numerical factorization. Consulting the algorithm in the previous section, we notice that in order to downdate the column norms after the jth step we only need to know the updated jth row. This allows us to choose the next pivot column p, say. Next, to determine the next Householder transformation, it is sufficient to apply the previous Householder transformation only to the pth column. The update of elements in other rows and columns can be delayed. This analysis underpins our block algorithm: for every consecutive nb steps, we update in each step only one row and one column, leaving the rest to be updated at the end of the nb steps with a block update, namely, a rank-nb update. If this scheme can be carried out successfully for $nb > 1$, the number of memory accesses of A can be reduced by around 50% compared to the BLAS Level 2 version, while opening up the possibility of using the typically very efficient BLAS Level 3 kernels.

Let us now consider the details of the block update. Assume we use the so-called compact WY form [28]

$$Q = I - YTY^T$$

to represent the product Q of nb Householder matrices H_i. Y is lower trapezoidal with nb columns and T is upper triangular of order nb. At first glance, the block

QP3Step (m, n, nb, $rowk$, A)

Setup:

$\text{perm}(j) = j, \text{colnorms}(j) = ||Ae_j||_2^2, \quad j = 1:n$

$F(1{:}\,n, 1{:}\,\text{nb}) = 0$

Reduction Steps:

For $j = 1 : nb$

0. $k = rowk + j - 1$ % current row index
1. **Pivoting:** Choose p such that $\text{colnorms}(p) = \max(\text{colnorms}(j : n))$
   If ($\text{colnorms}(p) == 0$) STOP
   If ($j \neq p$) then % interchange
   $\text{perm}([j,p]) = \text{perm}([p,j])$, $A(:,[j,p]) = A(:,[p,j])$
   $\text{colnorms}([j,p]) = \text{colnorms}([p,j])$, $F([j,p],:) = F([p,j],:)$

   end
2. **Update of pivot column:**
   $A(k:m,j) -= A(k:m,1:j-1) * F(1:j-1,j)$
3. **Reduction:** Generate $H_j = I - tau(j)Y(j)Y(j)^T$ such that
   $H_j A(k:m,j) = \pm||(||_2 A(k:m,j)e_1$.
4. **Incremental Computation of** F**:**
   $F(j+1:n,j) = tau(j)A(j:m,j+1:n)^T Y(j:m,j)$.
   $F(1:n,j) -= tau(j)F(1:n,1:j-1)Y(j:m,1:j-1)^T Y(j:m,j)$.
5. **Update of pivot row:**
   $A(k,j+1:n) -= A(k,1:j) * F(j+1:n,1:j)^T$
6. **Norm downdate :**
   $\text{colnorms}(j+1{:}\,n) = \text{colnorms}(j+1:n) - A(k,j+1:n)\ .2.$

End For

Block update:

$A(k+1:m, \text{nb}+1:n) -= A(k+1:m,1:\text{nb}) * F(\text{nb}+1:n,1:\text{nb})^T$

Figure 2: Algorithm for Reduction of $A(rowk:m,1:nb)$ and Update of $A(rowk:m,1:n)$

update

$$A(nb+1:m,nb+1:n) -= Y(nb+1:n,1:nb)TY^{\mathrm{T}}A(:,nb+1:n),$$

(where $\alpha -= \beta$ is shorthand for $\alpha = \alpha - \beta$) requires the values of the first nb rows of A *before the nb column reductions*. Yet, owing to the need for the norm downdating, these rows must have been updated by the time that we want to compute the block update.

We solve this problem by computing and saving

$$F^{\mathrm{T}} = TY^{\mathrm{T}}A(:,1:n)$$

adaptively, row by row, along with the generation of Y and T and the update of $A(1:nb:, nb+1:n)$. Notice that since Y^{T} is upper trapezoidal and T upper

```
Algorithm    QP3 ( m, n, idealnb, A, ··· )
    Initialize vectors perm and colnorms and set j = 1
    While j ≤ n
        nb = min(idealnb, n−j+1)
        QP3S ( m, n−j+1, j, nb, A(:, j : n), ··· );
        j = j + nb
    End While
End Algorithm
```

Figure 3: Block QR Factorization with Column Pivoting

triangular, the computation of the first row of F^{T} accesses all rows of A and it is needed for updating the first row of A. The computation of the second row of F^{T} accesses all rows of A but the first one (which is already updated by now) and the second row of F^{T} is needed for updating the second row of A, and so on. With careful programming, the use of F causes no increase in the workspace requirement for a block update. We summarize our discussion so far in the algorithm for one step of a block reduction shown in Figure 2.

The incremental update procedure for the auxiliary array F does not only resolve the coherence problem in the block update, it also makes the update of the pivot row and the pivot column easy. The block update can be carried out by a call to the BLAS Level kernel `xGEMM` and we touch A only once in the update of F. Thus, using `QP3Step`, the QRP factorization can be computed block by block.

We mentioned earlier that we aim to arrive at the same factorization as LINPACK and LAPACK by implementing the same norm downdate scheme and pivoting scheme. In particular, this means that if severe cancellation takes place in a norm downdate, the norm of the remaining column is computed from scratch. Thus, in the block scheme, we must update the column in question with all previously generated Householder transformations, even if we have not accumulated *nb* of them yet. If this happens, we shortcircuit the block accumulation and update *all* columns with the Householder transformations already generated. The actual number of reduced columns may be less than the given block size *nb*, but it is at least 1, not worse that the unblocked algorithm. So, unless we experience the (rare) case of frequent occurrences of catastrophic cancellation, we should still be able to perform a significant number of block updates. With this modification, the subroutine parameter *nb* is *both an input and an output parameter.* The overall block QRP algorithm is then shown in Figure 3.

4 Experimental Results

We report in this section experimental results comparing the double precision codes `DQRDC` from LINPACK, `DGEQPF` from LAPACK, and our block algorithm

DGEQP3. The tests were carried out on an IBM RS/6000-370, SGI R8000, and DEC Alpha 3000 Model 600. In each case, we employed the vendor-supplied BLAS in the ESSL, SGIMATH, and DXML libraries, respectively. We generated 18 different matrix types to test the algorithms, with various singular value distributions and numerical rank ranging from 4 to full rank. The matrix collection was constructed to exercise column pivoting, and thus we expect that the need for norm downdating might be if anything more pronounced than in what might be experienced in practice. Thus, we expect this collection to be representative of the pivoting behavior that could be expected in practice. Single, double, complex, and double complex code for xGEQP3 as well as the test and timing drivers used in these experiments are accessible via anonymous ftp from ftp.super.org in pub/prism/qp3.tar.gz.

We present results on matrices of size 150, 250, 500, and 1000, using a block size (*idealnb* in Figure 3) of 1, 5, 8, 12, 16, and 24. Figures 4 through 6 show the Mflop performance, averaged over the 18 matrix types, of the IBM, DEC, and SGI platforms versus block size. In all cases, the dotted line denotes the performance of DQRDC, the solid one that of DGEQPF, and the dashed one that of DGEQP3.

On the IBM, the BLAS hierarchy is intact, so to speak, in that performance increases with the BLAS Level employed. The overall performance of the machine also increases with matrix size, and so does the relative performance gain of DGEQP3 over DGEQPF: from 16 % for matrices of size 150 to 40 % for matrices of size 1000.

The DEC Alpha presents quite a different picture. First of all, the LINPACK code *always* outperforms the LAPACK code. Second, the overall performance of the machine drops substantially for matrix size 1000. However, the relative gain of DGEQP3 over DGEQPF is monotonically increasing: from 11 % for matrices of size 150 to 53% for matrices of size 1000.

The SGI presents a different picture still. Of the machines tested, it has by far the largest data cache memory: 4 MB. In contrast, the IBM and DEC platforms have only a 32KB data cache. Thus, matrices up to order 500 fit in cache, but matrices of order 1000 do not. Therefore, for matrices of size 500 or less we observe limited benefits from the better inherent data locality of the BLAS 3 implementation. However, the transition from BLAS1 to BLAS2 makes a big difference. Nonetheless, for $n = 500$, DGEQP3 outperforms DGEQPF by about 25% and achieves a performance of almost 125 Mflops. For $n = 1000$, overall performance degrades, but the relative advantage of DGEQP3 improves to about 38%.

We also note that on all three machines, the performance of DGEQP3 is rather robust with respect to variations in the block size, and, except for small matrices on the SGI, always superior to that of both the LINPACK and LAPACK implementations. Thus, while not being able to completely shield the user from machine peculiarities, DGEQP3 does significantly better in this respect than the other two implementations.

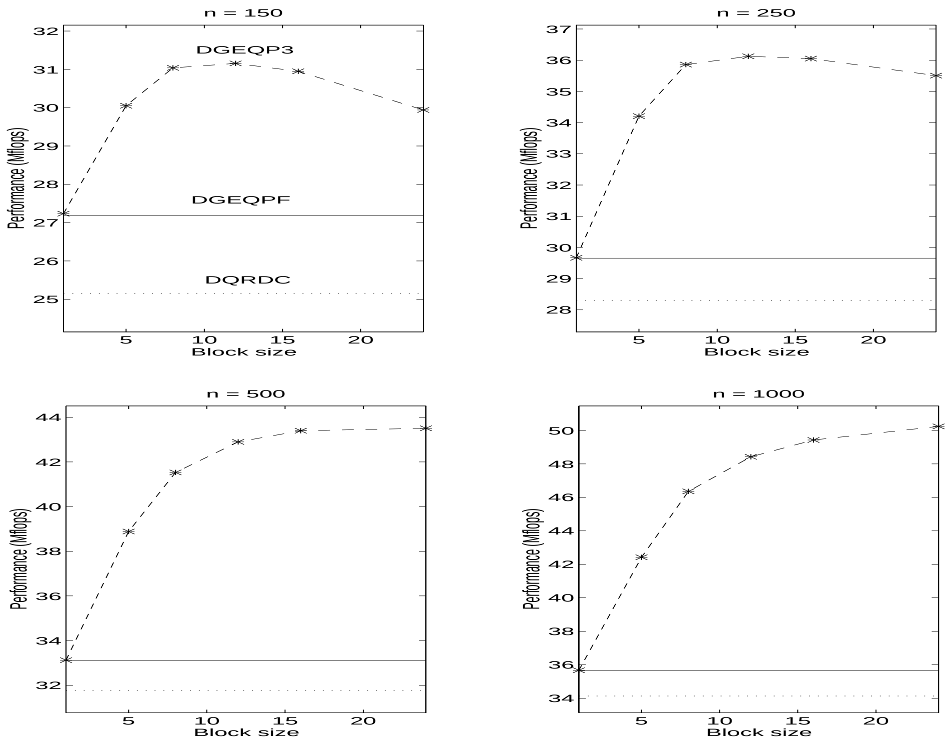

Figure 4: Average Performance (in Mflops) versus Block Size on an IBM RS/6000-370

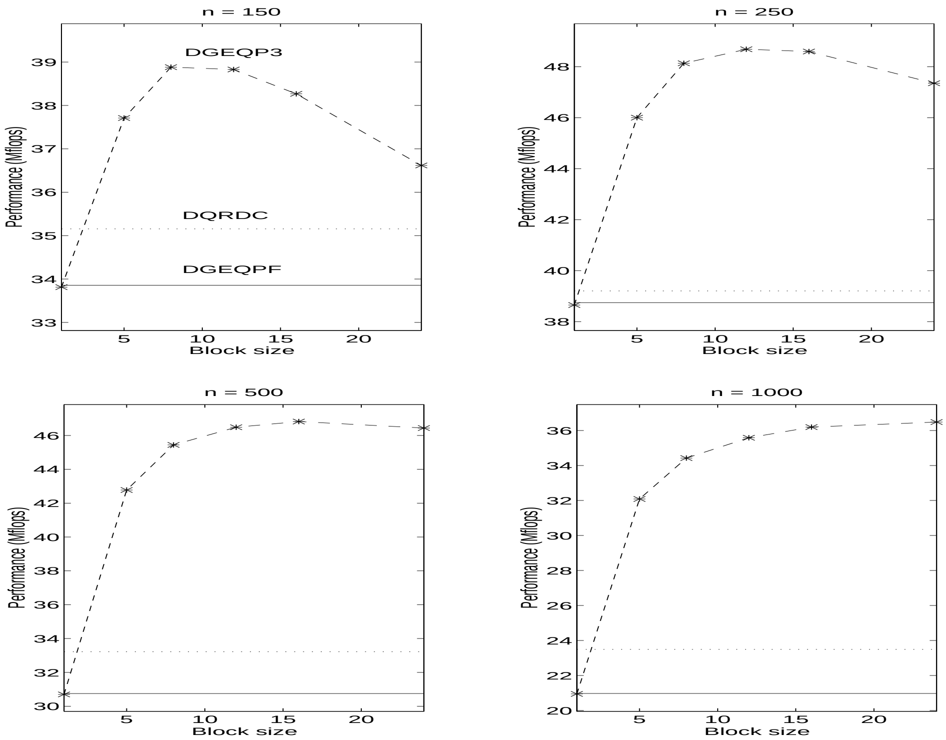

Figure 5: Average Performance (in Mflops) versus Block Size on an DEC Alpha 3000 Model 600.

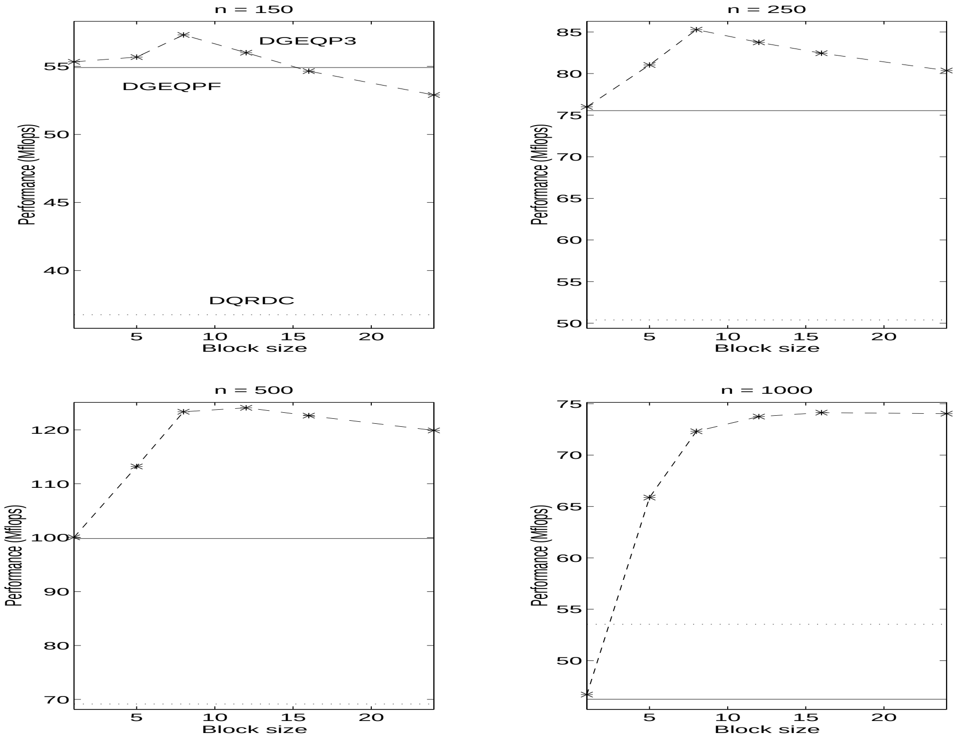

Figure 6: Average Performance (in Mflops) versus Block Size on an SGI R8000.

5 Concluding Remarks

We developed a new block variant of the QR factorization with column pivoting which allows the use of Level 3 BLAS. While maintaining the numerical behavior of the LINPACK and LAPACK implementations, it consistently outperforms them on IBM RS/6000, DEC Alpha, and SGI R8000 workstation platforms. Thus, it does a good job of insulating the user from the particulars of a particular machine, in particular its cache behavior. In contrast, the LINPACK code actually outperforms the LAPACK code on the DEC Alpha platform.

In order to achieve even better performance, we believe it necessary to either modify the norm downdating scheme or to relax the global pivoting criterion. In our tests we observed cases where columns were involved quite a few times in "catastrophic" cancellation scenaria, prompting the recomputation of their norm. How to relax the downdating criterion causing dramatic change in numerical properties of the QR factorization with column pivoting is an open question.

A different approach is to avoid the need for a global pivot search through the introduction of a "pivot window" [4,5]. The resulting algorithms have even higher data locality, but the rank-revealing properties of the resulting orthogonal factorization deteriorate. Thus, such an approach is unlikely to be reliable unless coupled with a post-processing step that tests, and, if necessary, improves the rank-revealing nature of the factorization. How the overall algorithm would perform is unclear at this point.

Acknowledgements

Quintana and Sun thank Jack Dongarra for providing a friendly and productivity-enhancing environment during their visit to the University of Tennessee.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philadelphia, 1992.

[2] ——, *LAPACK User's Guide Release 2.0*, SIAM, Philadelphia, 1994.

[3] Z. Bai and J. Demmel, *Design of a parallel nonsymmetric eigenroutine toolbox, Part I*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, R. F. S. *et al*, ed., SIAM, 1993, pp. 391–398.

[4] C. H. Bischof, *A block QR factorization algorithm using restricted pivoting*, in Proceedings SUPERCOMPUTING '89, Baltimore, Md., 1989, ACM Press, pp. 248–256.

[5] ——, *A parallel QR factorization algorithm with controlled local pivoting*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 36–57.

[6] C. H. Bischof and P. C. Hansen, *Structure-preserving and rank-revealing QR factorizations*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 1332–1350.

[7] C. H. Bischof and P. C. Hansen, *A block algorithm for computing rank-revealing QR factorizations*, Numerical Algorithms, 2 (1992), pp. 371–392.

[8] C. H. Bischof and G. M. Shroff, *On updating signal subspaces*, IEEE Transactions on Signal Processing, 40 (1992), pp. 96–105.

[9] P. A. Businger and G. H. Golub, *Linear least squares solution by Householder transformation*, Numerische Mathematik, 7 (1965), pp. 269–276.

[10] T. F. Chan, *Rank revealing QR factorizations*, Linear Algebra and Its Applications, 88/89 (1987), pp. 67–82.

[11] S. Chandrasekaran and I. Ipsen, *On rank-revealing qr factorizations*, SIAM Journal on Matrix Analysis and Applications, 15 (1994).

[12] J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[13] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM Press, Philadelphia, 1979.

[14] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. J. Hanson, *An extended set of Fortran basic linear algebra subprograms*, ACM Transactions on Mathematical Software, 14 (1988), pp. 1–17.

[15] L. Eldén and R. Schreiber, *An application of systolic arrays to linear discrete ill-posed problems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 892–903.

[16] G. H. Golub and C. F. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 2nd ed., 1989.

[17] G. H. Golub, P. Manneback, and P. L. Toint, *A comparison between some direct and iterative methods for certain large scale geodetic least-squares problem*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 799–816.

[18] T. A. Grandine, *An iterative method for computing multivariate C^1 piecewise polynomial interpolants*, Computer Aided Geometric Design, 4 (1987), pp. 307–319.

[19] ——, *Rank deficient interpolation and optimal design: An example*, Tech. Report SCA–TR–113, Boeing Computer Services, Engineering and Scientific Services Division, February 1989.

[20] M. Gu and S. Eisenstat, *An efficient algorithm for computing a strong rank-revealing factorization*, Tech. Report YALEU/DCS/RR-967, Yale University, Department of Computer Science, 1994.

[21] P. C. Hansen, *Truncated SVD solutions to discrete ill-posed problems with ill-determined numerical rank*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 503 – 518.

[22] Y. P. Hong and C.-T. Pan, *The rank revealing QR decomposition and SVD*, Mathematics of Computation, 58 (1992), pp. 213–232.

[23] S. F. Hsieh, K. J. R. Liu, and K. Yao, *Comparisons of truncated QR and SVD methods for AR spectral estimations*, in SVD and Signal Processing II, R. J. Vaccaro, ed., Elsevier Science Publishers, 1991, pp. 403–418.

[24] C. L. Lawson, R. J. Hanson, R. J. Kincaid, and F. T. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Transactions on Mathematical Software, 5 (1979), pp. 308–323.

[25] J. Moré, *The Levenberg-Marquardt algorithm: Implementation and theory*, in Proceedings of the Dundee Conference on Numerical Analysis, G. A. Watson, ed., Berlin, 1978, Springer Verlag.

[26] C.-T. Pan and P. T. P. Tang, *Bounds on singular values revealed by QR factorizaton*, Tech. Report MCS-P332-1092, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[27] D. Pierce and J. G. Lewis, *Sparse rank revealing QR factorization*, Tech. Report MEA-TR-193, Boeing Computer Services, Engineering and Scientific Services Division, 1992.

[28] R. Schreiber and C. Van Loan, *A storage efficient WY representation for products of Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 53–57.

[29] B. Waldén, *Using a Fast Signal Processor to Solve the Inverse Kinematic Problem with Special Emphasis on the Singularity Problem*, PhD thesis, Linköping University, Dept. of Mathematics, 1991.