BIT
2001, Vol. 41, No. 5, pp. 936–949
0006-3835/01/4105-0936 $16.00

A FASTER AND SIMPLER RECURSIVE ALGORITHM FOR THE LAPACK ROUTINE DGELS *

ERIK ELMROTH[1] and FRED G. GUSTAVSON[2] †

[1] *Department of Computing Science and HPC2N, Umeå University, SE–901 87 Umeå, Sweden. email: elmroth@cs.umu.se*

[2] *IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA. email: gustav@watson.ibm.com*

Abstract.

We present new algorithms for computing the linear least squares solution to overdetermined linear systems and the minimum norm solution to underdetermined linear systems. For both problems, we consider the standard formulation $\min \|AX - B\|_F$ and the transposed formulation $\min \|A^T X - B\|_F$, i.e, four different problems in all. The functionality of our implementation corresponds to that of the LAPACK routine DGELS. The new implementation is significantly faster and simpler. It outperforms the LAPACK DGELS for all matrix sizes tested. The improvement is usually 50–100% and it is as high as 400%. The four different problems of DGELS are essentially reduced to two, by use of explicit transposition of A. By explicit transposition we avoid computing Householder transformations on vectors with large stride. The QR factorization of block columns of A is performed using a recursive level-3 algorithm. By interleaving updates of B with the factorization of A, we reduce the number of floating point operations performed for the linear least squares problem. By avoiding redundant computations in the update of B we reduce the work needed to compute the minimum norm solution. Finally, we outline fully recursive algorithms for the four problems of DGELS as well as for QR factorization.

AMS subject classification: 65F20, 65Y20.

Key words: Overdetermined systems, underdetermined systems, linear least squares, minimum norm solution, recursion, high performance library software.

1 Introduction.

The LAPACK algorithm DGELS [2] solves overdetermined linear systems in the least squares sense and it computes the minimum norm solution to underdetermined systems. We present a new simpler and significantly faster algorithm to perform these computations.

Given an $m \times n$ matrix A with full row or column rank, and matrices X and B, both with *nrhs* columns and appropriate row dimensions, we consider the four problems:

*Received September 2000. Communicated by Gustaf Söderlind.

†This research was conducted using the resources of High Performance Computing Center North (HPC2N). The work is part of an HPC2N–IBM collaboration (SUR grant). Financial support has been provided by the Swedish Research Council for Engineering Sciences under grant TFR 98-604.

1. Linear least squares solution to $\min \|AX - B\|_F$ $(m \geq n)$;
2. Linear least squares solution to $\min \|A^T X - B\|_F$ $(m < n)$;
3. Minimum norm solution to $\min \|A^T X - B\|_F$ $(m \geq n)$;
4. Minimum norm solution to $\min \|AX - B\|_F$ $(m < n)$.

The work is a continuation of our work on recursive and hybrid recursive QR factorization. Previous work [4, 5, 6] by us and others, e.g., [1, 7, 8, 10, 12] has shown that recursion can be successfully used to replace level-2 algorithms (matrix-vector computations) by level-3 algorithms (matrix-matrix computations). The recursion leads to an automatic variable blocking that dynamically adjusts to an arbitrary number of levels of a memory hierarchy.

We use recursion to QR factorize block columns of A, which leads to performance gains. Other, often even larger improvements are achieved by reducing the number of floating point operations performed. Specifically, we save floating point operations in two ways. First we only compute the T matrices of the compact WY representation once. This requires saving the T matrices for Problem 3. The second way pertains to Problem 3 only. We reduce the number of floating point operations in the $B \leftarrow QB$ computations by skipping a sub computation of LAPACK DGELS that updates a submatrix C (DGEMM: $C \leftarrow C + A \cdot B$) where the operand B is a zero matrix. Finally, we demonstrate that the single most important performance issue is to avoid the computations of Householder transformations for vectors stored with large stride, which is typically done in the LQ factorization of A. By doing so, we also show that Problems 2 and 4 map to Problems 1 and 3.

The paper is organized as follows. In Section 2, we present the basic algorithms for Problems 1 and 3. Both algorithms are based on a QR factorization of A. In Section 3, we explain why it is imperative *not* to compute Householder transformations on vectors stored with large stride. This is typically what one does in order to factor $A = LQ$ to solve Problems 2 and 4. We show that it is significantly more efficient to explicitly transpose A and reuse our algorithms from Section 2. Performance comparisons with the LAPACK routine DGELS are presented in Section 4. In Section 5, we demonstrate that Problem 3, viewed as a single computation, can be further optimized. The subcomputations $A = QR$ and $B \leftarrow QB$ of Problem 3 are related in that we can choose various representations for Q. Specifically, we can increase the cost of computing $A = QR$ because the new Q obtained reduces the cost of computing $B \leftarrow QB$. We derive (assuming $m \gg n$) a relation between m, n, and $nrhs$ that tells one when to produce a different QR factorization of A. In Section 6, we outline fully recursive algorithms for QR factorization and the four problems of DGELS. Our conclusions are given in Section 7. Our DGELS algorithm is now part of the IBM ESSL software library [9].

2 Improved algorithms for Problems 1 and 3.

Our new software, named RGELS, solves the four problems listed in the introduction. Algorithm 2.1 for solving Problem 1 is outlined below. It is assumed that A is $m \times n$, X is $n \times nrhs$ and B is $m \times nrhs$.

The *do* loop interleaves a factorization $A = QR$ with the application of Q^T to B, i.e., $B \leftarrow Q^T B$. The QR factorization of each block column of A, consisting of jb columns, is performed by the recursive level-3 routine RGEQR3 described in [4] and further optimized in [5]. The original matrix A is overwritten by R and the Householder vectors $u_i, i = 1, \ldots, n$. The matrix Q is not explicitly formed, but available as the product of n Householder transformations $H_i = I - \tau_i u_i u_i^T$. For the updates of B or the trailing part of A, the compact WY representation $Q = I - YTY^T$ is used [3, 4, 5, 6, 11]. Here T is a matrix formed by calculations using Householder vectors and τ_i values.

ALGORITHM 2.1. *Algorithm for the non-transposed linear least squares problem.*

do $j = 1, n, nb$! nb is the block size
  $jb = \min(n - j + 1, nb)$
  call RGEQR3 $A(j : m, j + jb - 1) = (Y, R, T)$
  if $(j + jb \leq n)$ **then**
    compute $A(j{:}m, j + jb{:}n) \leftarrow (I - YT^TY^T)A(j{:}m, j + jb{:}n)$
  endif
  compute $B \leftarrow (I - YT^TY^T)B$
enddo
! B is now Q^TB and $A = QR$
solve $RX = B$! X overwrites $B(1 : n, 1 : nrhs)$

RGEQR3 returns upper triangular R, lower trapezoidal Y and $jb \times jb$ upper triangular T such that $Q = I - YTY^T$ and $A = QR$. The τ_i-values are stored on the diagonal of T, and for compatibility with LAPACK, in the TAU vector. Immediately after the factorization of each block column, the corresponding orthogonal matrix Q^T is applied to the trailing right hand side of A and the matrix B. Since B is updated before the next block column of A is factorized, we can reuse the T matrix, i.e., we do not have to save or recompute T in order to apply Q^T to B.

The above algorithm has two main advantages over LAPACK DGELS. First, DGELS computes the QR factorization of the whole matrix A before the orthogonal transformations are applied to B. Each block column of A is factorized using the level-2 algorithm DGEQR2. The corresponding T matrices are constructed using the level-2 algorithm DLARFT. When applying the transformations to B, all T matrices are reconstructed by additional calls to the level-2 DLARFT. Hence, our algorithm actually performs less floating point operations than DGELS. Secondly, in RGELS we use the recursive algorithm RGEQR3 that performs both the QR factorization of A and the computations for the triangular T using level-3 operations. This algorithm performs significantly better than the two corresponding LAPACK routines [4, 5].

A brief description of our Algorithm 2.2 for solving Problem 3 is presented below. Here, A is $m \times n$, X is $m \times nrhs$ and input B is $n \times nrhs$. Since B is overwritten by the solution X, the size allocated for B must be at least $m \times nrhs$.

The computations proceed as follows. First, A is factorized $A = QR$ and then the $n \times n$ triangular system $R^TX = B$ with $nrhs$ right hand sides is solved.

Conceptionally, the *minimum norm* solution is obtained by assigning $X(n+1:m, 1:nrhs) = 0$ and applying $X \leftarrow QX$. In LAPACK DGELS, $X(n+1:m, 1:nrhs)$ is explicitly set to zero and the corresponding calculations are performed. In our algorithm, the assignment $X(n+1:m, 1:nrhs) = 0$ is implicit, i.e., we avoid performing computations with elements that are known to be zero. Hence we both save computations and the work needed to assign $X(n+1:m, 1:nrhs) = 0$.

ALGORITHM 2.2. *Algorithm for the transposed minimum norm solution.*

do $j = 1, n, nb$! nb is the block size
  $jb = \min(n - j + 1, nb)$
  call RGEQR3 $A(j:m, j+jb-1) = (Y, R, T)$
  Save the T matrix for later use
  if $(j + jb \le n)$ **then**
    compute $A(j:m, j+jb:n) \leftarrow (I - YT^TY^T)A(j:m, j+jb:n)$
  endif
enddo
solve $R^T X(1:n, 1:nrhs) = B$! X overwrites $B(1:n, 1:nrhs)$
do $j = \lfloor n/nb \rfloor nb + 1, 1, -nb$
  $jb = \min(n - j + 1, nb)$
  Let Y consist of Householder vectors $j : j + jb - 1$
  Retrieve the corresponding T matrix, saved above
  if $(j = \lfloor n/nb \rfloor nb + 1)$ **then**
    compute $B \leftarrow (I - YTY^T)B$! knowing $B(n+1:m, 1:nrhs) = 0$
  else
    compute $B \leftarrow (I - YTY^T)B$! here $B(n+1:m, 1:nrhs) \neq 0$
  endif
enddo

We note that the first application of Q to the right hand side involves less floating point operations than the subsequent applications (see the above paragraph). Now, if n is small but larger than nb, it may therefore be beneficial to use the block size $nb = n$ for such cases. This will increase the amount of work for the QR factorization but reduces the amount of work for the application of Q to the right hand side (see Section 5 for details). Such a modification is included in our software but for clarity omitted in Algorithm 2.2.

3 Transposition avoids computations on vectors with large stride.

Problems 2 and 4 are the transposed versions of Problems 1 and 3. The standard technique to solve Problems 2 and 4, is to compute the LQ factorization of A followed by an update of B and a triangular solve (for Problem 2) or a triangular solve and an update of X (Problem 4). This is what is done in LAPACK DGELS. Here, we show that it is more efficient to explicitly transpose A and then to reuse the algorithms from Section 2.

Consider the LQ factorization of A, stored in column major order. We believe it is impossible to efficiently compute the LQ factorization working on A directly on todays processors, since each Householder transformation needs to be computed

for each row of A. For each element, todays processors will transfer an amount of data that corresponds to the size of one cache line, here called the line size (LS). For a row vector of length n, the processor will transfer $n * LS$ data, of which $n * (LS - 1)$ data is not needed, nor can it subsequently be used efficiently if n is large. Furthermore, this extra $n * (LS - 1)$ data causes cache thrashing.

Our remedy is to perform an explicit transposition of the matrix A, i.e., let $C = A^T$ where C is $n \times m$ ($m < n$). Then Problems 2 and 4 are solved using the algorithms for Problems 1 and 3, i.e., Problem 2 becomes Problem 1:

$$\textit{Least squares solution to } \min \|CX - B\|_F,$$

where X is $m \times nrhs$ and B is $n \times nrhs$ and Problem 4 becomes Problem 3:

$$\textit{Minimum norm solution to } \min \|C^T X - B\|_F,$$

where X is $n \times nrhs$ and B is $m \times nrhs$. Thus, we solve Problems 2 and 4 using $A^T = C = QR$ instead of $A = L\tilde{Q}$. Since $A = C^T = R^T Q^T$, we have $R^T = L$ and $Q^T = \tilde{Q}$.

If one is only interested in the solution, this would be all we have to do. In order to be consistent with LAPACK DGELS we also transpose A on output so that it contains L and the Householder vectors. In all, this procedure gives us significant performance improvements (see Section 4). Furthermore, it basically reduces the code to 2 cases instead of 4.

4 Performance results.

Performance evaluation have been performed on a 160 MHz IBM Power2 processor with a theoretical peak performance of 640 Mflops/s. For all four problems of RGELS we have run tests with the large dimension of A varying from 100 to 2000 with step 100 and the small dimension varying from 50 to 300 with step 50. For each size of A the number of right hand sides ($nrhs$) have been varied from 1 to 251 with step 50. We remark that the new algorithm RGELS outperforms LAPACK DGELS for all cases tested, though it requires too much space to present all the performance results here.

In order to facilitate the performance comparisons (and to keep the number of pages down) we present all results in terms of performance ratios of RGELS to DGELS, e.g., a ratio 1.x shows that RGELS is x% faster than DGELS. It is, however, worth mentioning that maximum performance obtained is over 500 Mflops/s for RGELS, i.e., around 80% of the theoretical peak performance.

Figures 4.1–4.3 show the performance ratios for Problems 1 and 2. The performance ratios for Problems 3 and 4 are presented in Figures 4.4–4.6. For each problem, we present three dual graphs. The first (4.1 and 4.4) is for the important special case $nrhs = 1$ where we show the performance for varying m and n. In the second (4.2 and 4.5) we keep the small dimension of A fixed to 50 and let the larger dimension vary. In the third (4.3 and 4.6) we repeat this with the smaller dimension increased to 150. In each figure, the results are presented for the non-transposed case at the top and the transposed case at the bottom.

RGELS clearly perform better. For very small problems, the gains are marginal since the potential for data reuse in the memory hierarchy is small. For larger problems, the differences become significant. For cases where the LAPACK DGELS shows reasonably good performance, the difference is 20–120%. For more extreme cases, the difference is a factor 2–5, i.e., 100–400%.

The gain in performance obtained by replacing the LQ decomposition (as in LAPACK DGELS) by an explicit transposition of A followed by a QR decomposition is clearly illustrated in the figures. These cases correspond to the top most graphs in Figures 4.1–4.3 and the bottom most graphs in Figures 4.4–4.6. The gain over the LAPACK is roughly a factor of 2 larger in these cases than for the other cases (where both RGELS and DGELS compute the QR decomposition).

We note that for almost all values of m and n for all four problems, the difference is largest for the important special case $nrhs = 1$, i.e., the right hand side (and the solution) of the system consists of a single vector. The explanation for this is that this is where we see the largest relative gain from not recomputing the T matrix

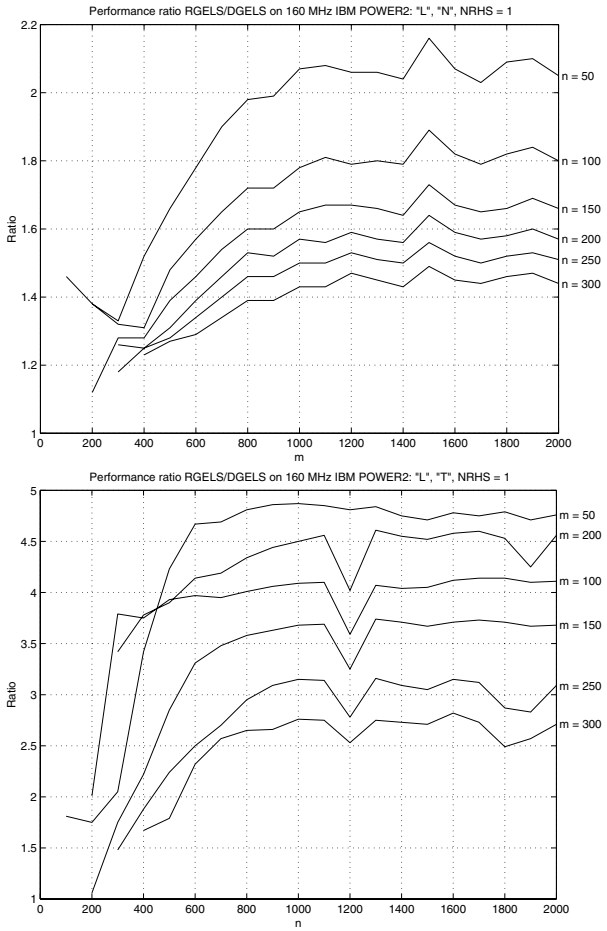

Figure 4.1: Performance ratio RGELS/DGELS for non-transposed and transposed linear least squares problem for $nrhs = 1$.

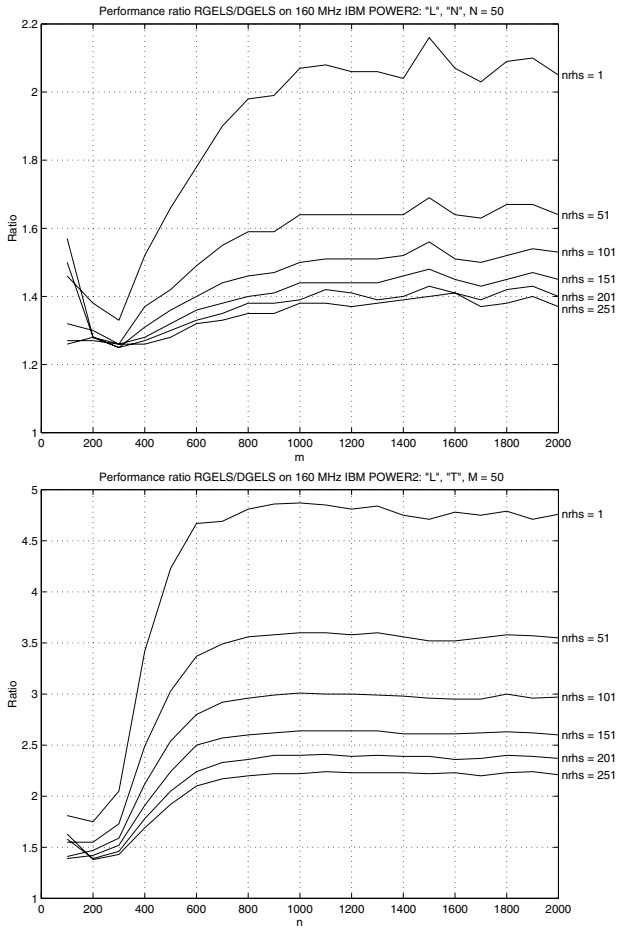

Figure 4.2: Performance ratio RGELS/DGELS for non-transposed and transposed linear least squares problem for 50 unknowns per right hand side.

before applying the orthogonal matrix to the right hand side.

A general trend is that the difference slowly becomes smaller as *nrhs* increases. This follows naturally from the fact that the relative part of the updates of the right hand side increases. At the same time, the relative time for recomputing T matrices and performing redundant operations in the updates becomes smaller.

5 Increasing nb to n for the T computation of Problem 3.

Here we partly quantify the point we made in the last paragraph of Section 2. By increasing the cost of calling RGEQR3 in Algorithm 2.2 we hope to reduce the cost of computing $B \leftarrow QB$ in the last part of Algorithm 2.2. In the analysis below we do an approximate calculation. We assume $m \gg n$.

Let $n = (k+1)nb$. Then in RGEQR3 we have $A = QR$ where $Q = Q_0 Q_1 \cdots Q_k$

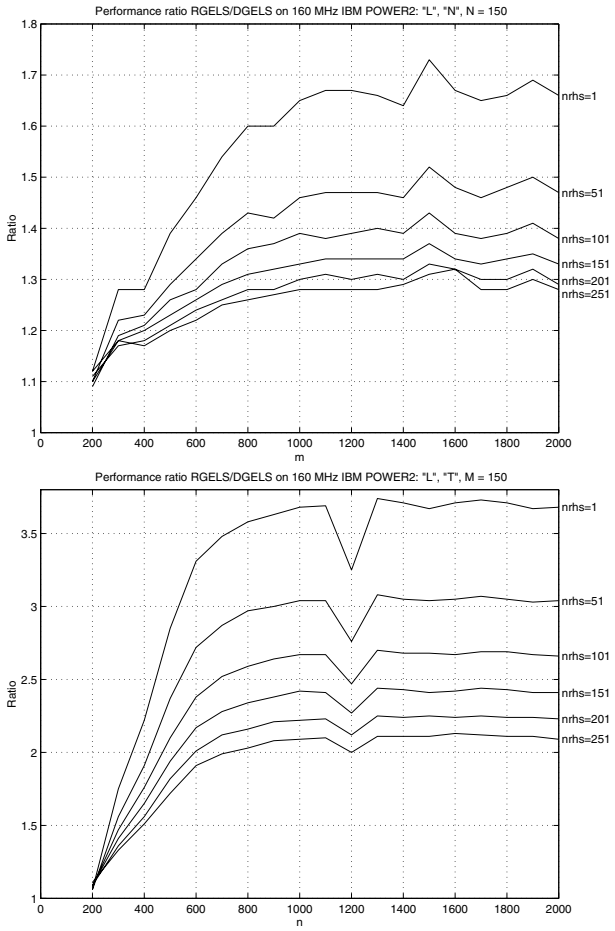

Figure 4.3: Performance ratio RGELS/DGELS for non-transposed and transposed linear least squares problem for 150 unknowns per right hand side.

where $Q_i = I - Y_i T_i Y_i^T$ and each T_i is nb by nb. Secondly, suppose we set $A = QR$ where $nb = n$. Then $Q = I - YTY^T$ and T is n by n. The extra computation would involve computing $0.5k(k+1)nb^2$ additional elements of T. Now the major part of the T computation for $t_{i,j}$ is a dot product of length about m; i.e., (y_i^T, y_j). Thus the approximate additional flop count is $mk(k+1)nb^2$ if we decide to use $nb = n$ in the computation of $A = QR$, i.e., in the second factorization.

In Section 2 we saved some flops by knowing $B(n+1 : m, 1 : nrhs) = 0$. The amount of savings was $2(m-n)nb \cdot nrhs$. However, if $nb = n$, as would be the case for the second QR factorization of A the amount of savings would be $2(m-n)n \cdot nrhs$. The difference is $2(m-n)k \cdot nb \cdot nrhs$ flops and this is approximately the amount of flops that would be saved by increasing nb to n. As mentioned, due to lack of space we are only attempting here to illustrate the trade-off that can occur if one were to increase nb to n in RGEQR3.

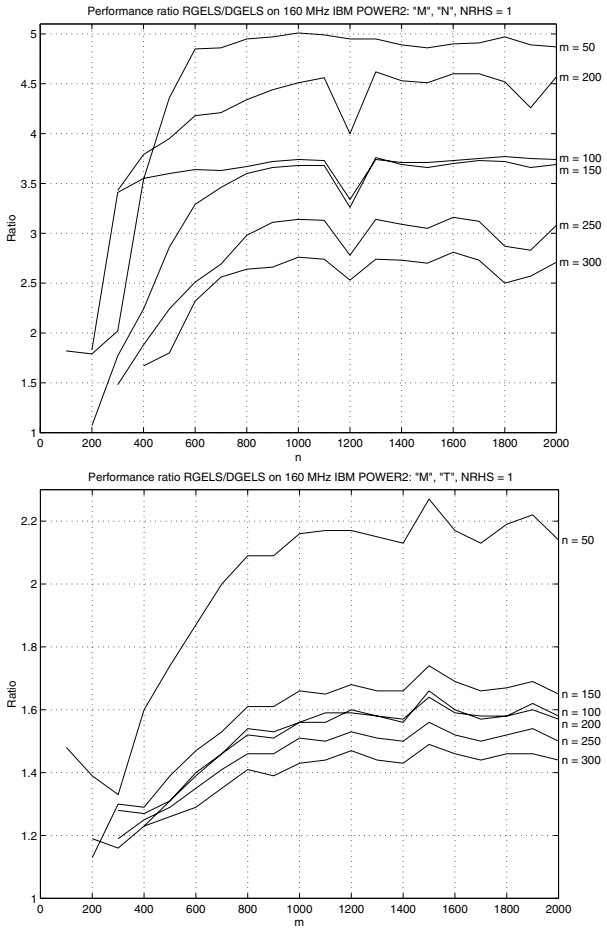

Figure 4.4: Performance ratio RGELS/DGELS for non-transposed and transposed minimum norm solution for $nrhs = 1$.

Now let us set the extra flop cost equal to extra flop count saved. We get the relation $nrhs = 5n(1 - n/m)^{-1}$. Assuming the flop rates for the extra T computation is the same as the extra flops saved in the QB computation the total computation time will be the same. This last assumption appears to be fair because both computations are DGEMM computations, the former being $0.5k(k+1)nb^2$ independent DDOT computations of length m. Also, the saved flops computation is indeed a DGEMM computation. From the above relation we see that if $nrhs$ is more than about $n/2$ then it pays to compute the extra T's.

As a rough breakpoint, we used $n \geq nrhs$ and $m \geq 3n$. This was not the true breakpoint; however, it was safe in the sense the performance for all cases never deteriorated over the straight application of Algorithm 2.2.

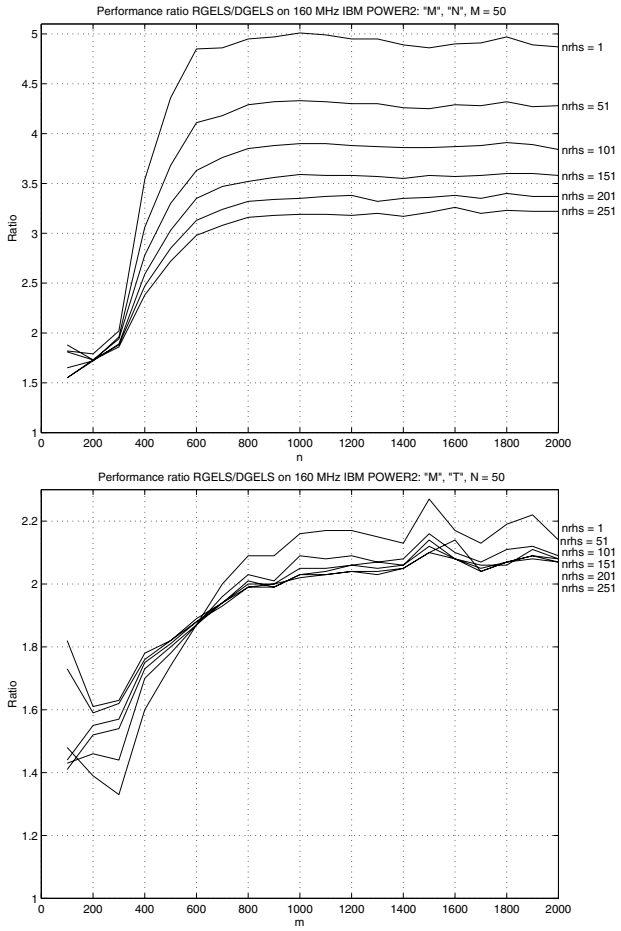

Figure 4.5: Performance ratio RGELS/DGELS for non-transposed and transposed minimum norm solution for 150 unknowns per right hand side.

6 Fully recursive algorithm formulations.

The algorithms presented in Section 2 are based on a standard blocking technique and use of a recursive QR factorization on specific submatrices. Here, we rewrite these recursively and show how to combine the blocking of the hybrid recursive QR routine RGEQRF with the pure recursive RGEQR3.

Given an $m \times n$ matrix A, RGEQRF (presented in [4, 5]) steps over the n dimension with a fixed block size nb. For each step, a block column with nb columns is factorized by a call to the pure recursive RGEQR3, and the remaining part of the matrix is updated with respect to this factorization. RGEQR3 factors a matrix by recursively dividing it into two approximately equal halves. For each recursive step, the first half is recursively factorized, the second half is updated and also recursively factorized. As noted in [4, 5], RGEQR3 cannot be efficiently

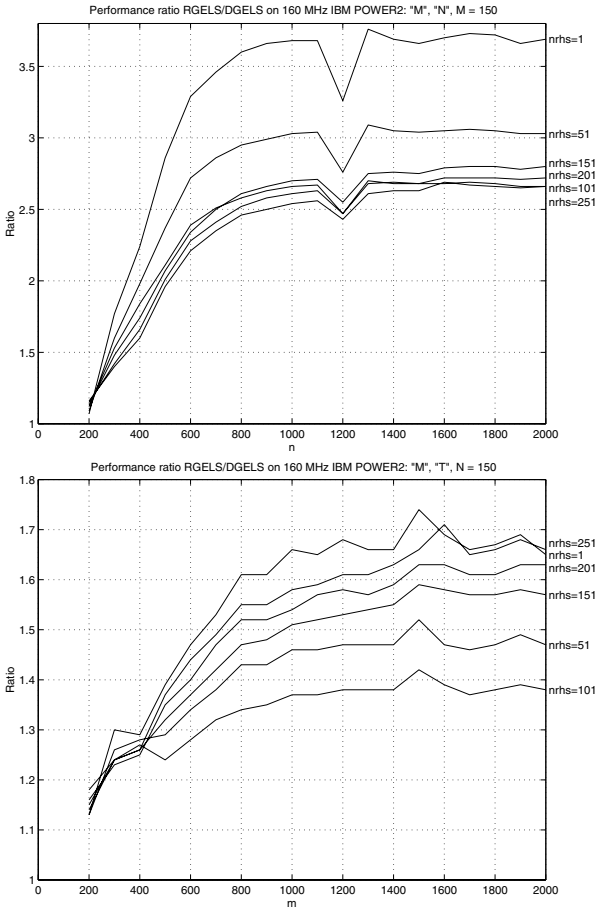

Figure 4.6: Performance ratio RGELS/DGELS for non-transposed and transposed minimum norm solution for 150 unknowns per right hand side.

applied when n is large, as the flop count increases with the increasing block size in the updates. Our remedy was to perform an outer blocking in the routine RGEQRF and use RGEQR3 for subproblems.

However, the outer blocking performed in RGEQRF can be incorporated directly into RGEQR3 by creating a hybrid recursive QR algorithm (HRQR) as sketched in Algorithm 6.1. By dividing A into two sets of columns where the first part has no more than nb columns, the algorithm HRQR also provides the blocking of RGEQRF. Thus, RGEQRF is no longer needed and its excellent performance is obtained by only HRQR. In order to make an efficient implementation of this algorithm, there are a number of details to consider, as described in [4, 5] for the original RGEQR3 (and RGEQRF), including adjustment of the stopping criteria (e.g., to $n = 4$), the ordering of matrix multiplications in updates using $Q = I - YTY^T$, etc.

Similarly, the algorithms for computing the least squares and minimum norm solution can also be rewritten as recursive algorithms, as outlined in Algorithms 6.2 and 6.3. Here, the parameters nb and nbl are used to decide when to stop the recursion.

We remark that the recursive Algorithms 6.2 and 6.3 perform a slightly different blocking than the algorithms presented in Section 2.. In practice, A should be overwritten by R and Y (containing the Householder vectors) in all three algorithms.

ALGORITHM 6.1. Recursive algorithm for computing $A = QR$, where A is $m \times n$ ($m \geq n$). In step 7, T is formed only if $n \leq nb$, otherwise $T = [T_1 \;\; T_2]$ is returned.

$(Y, R, T) = \mathrm{HRQR}(A, nb)$! $Q = I - YTY^T$
If $n = 1$
1. Compute Householder transform $Q = I - \tau uu^T$ such that $QA = (x, 0)^T$
2. Return $Y = u, R = x,$ and $T = \tau$

else
3. Let $A = \begin{bmatrix} A_1 & A_2 \end{bmatrix}$! if $n > nb$ A_1 has nb cols, else A_1 has $\lfloor n/2 \rfloor$ cols
4. $(Y_1, R_{11}, T_1) = \mathrm{HRQR}(A_1, nb)$
5. Update $\begin{bmatrix} R_{12} \\ A_{22} \end{bmatrix} \leftarrow (I - Y_1 T_1^T Y_1^T) A_2$
6. $(Y_2, R_{22}, T_2) = \mathrm{HRQR}(A_{22}, nb)$
7. $Y = \begin{bmatrix} Y_1 & Y_2 \end{bmatrix}$; $R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$; $T = \begin{bmatrix} T_1 & -T_1(Y_1^T Y_2)T_2 \\ 0 & T_2 \end{bmatrix}$

endif

ALGORITHM 6.2. Recursive algorithm for computing the least squares solution to $AX = B$, where A is $m \times n$ ($m \geq n$). B is overwritten by the solution X.

$B = \mathrm{HRLS}(A, B, nb)$
If $n \leq nb$
1. $(Y, R, T) = \mathrm{HRQR}(A, nb)$; $B \leftarrow (I - YT^T Y^T)B$;
   $B(1 : n, 1 : nrhs) \leftarrow R^{-1} B(1 : n, 1 : nrhs)$

else ! here $n > nb$
2. Let $A = \begin{bmatrix} A_1 & A_2 \end{bmatrix}$; $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$! with nb cols in A_1, nb rows in B_1
3. $(Y, R_{11}, T) = \mathrm{HRQR}(A_1, nb)$
4. Set $\begin{bmatrix} R_{12} & B_1 \\ A_{22} & B_2 \end{bmatrix} \leftarrow (I - YT^T Y^T) \begin{bmatrix} A_2 & B \end{bmatrix}$
5. $B_2 = \mathrm{HRLS}(A_{22}, B_2, nb)$
6. Compute $B_1 \leftarrow R_{11}^{-1}(B_1 - R_{12}B_2)$; return $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$

endif

ALGORITHM 6.3. Recursive algorithm for computing the minimum norm solution to $A^T X = B$, where A is $m \times n$ ($m \geq n$). B is overwritten by the solution X. The size allocated for B and X must be $m \times nrhs$.

$B = \mathrm{HRMN}(A, B, nb, nbl)$! $nbl \geq nb$ is assumed
If $n \leq nbl$
1. $(Y, R, T) = \mathrm{HRQR}(A, n)$
2. Let $B \leftarrow R^{-T}B$, $B(1:m, 1:nrhs) \leftarrow (I - YTY^T) \begin{bmatrix} B \\ 0 \end{bmatrix}$

else ! here $n > nbl \geq nb$
3. Let $A = \begin{bmatrix} A_1 & A_2 \end{bmatrix}$; $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$! with nb cols in A_1, nb rows in B_1
4. $(Y, R_{11}, T) = \mathrm{HRQR}(A_1, nb)$
5. Set $B_1 \leftarrow R_{11}^{-T} B_1$; $\begin{bmatrix} R_{12} \\ A_{22} \end{bmatrix} \leftarrow (I - YT^T Y^T) A_2$; and $B_2 \leftarrow B_2 - R_{12}^T B_1$
6. $B_2 = \mathrm{HRMN}(A_{22}, B_2, nb, nbl)$; return $B = (I - YTY^T) \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$

endif

7 Concluding remarks.

This contribution illustrates how excellent results can be obtained when algorithms are matched to the architecture. By relatively small novel changes we have achieved significant performance improvements.

We have mainly used four techniques to improve over LAPACK DGELS. The QR factorization and the generation of the matrix T is performed by a recursive level-3 algorithm that perform better than the corresponding level-2 algorithms used in DGELS. The matrix T is reused instead of being recomputed for the applications of the orthogonal matrix on the right hand side. Redundant computations in the updates of the right hand side is avoided for the underdetermined systems. Finally, for the Problems 2 and 4 where $A = LQ$ is apparently needed, we have explicitly transposed A and used the algorithms for Problems 1 and 3 that are based on QR factorization. The last modification improves the performance by roughly a factor of two and it basically reduces the program to handle two cases instead of four.

Our routine RGELS has the same calling sequence as the corresponding LAPACK routine. For a user, the only difference is that in order to benefit from the better performance of RGELS the size of the workspace provided to the routine must be larger. The new algorithm outperforms LAPACK DGELS for all cases tested. The improvement is usually 50–100% and it is as high as 400%. Finally, we outline pure recursive algorithms for all problems considered.

REFERENCES

1. B. S. Andersen, F. G. Gustavson, and J. Waśniewski, *A recursive formulation of Cholesky factorization of a matrix in packed storage*, ACM Trans. Math. Software, 27:2 (2001), pp. 214–244.
2. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide—Release 3.0*, SIAM, Philadelphia, PA, 1999.
3. C. H. Bischof and C. Van Loan, *The WY representation for products of Householder matrices*, SIAM J. Sci. Stat. Comput., 8:1 (1987), pp. s2–s13.
4. E. Elmroth and F. G. Gustavson, *Applying recursion to serial and parallel QR factorization leads to better performance*, IBM J. Res. Develop., 44:4 (2000), pp. 605–624.
5. E. Elmroth and F. G. Gustavson, *High-performance library software for QR factorization*, in Applied Parallel Computing. New Paradigms for HPC in Industry and Academia, T. Sørvik et al. (eds.), Lecture Notes in Computer Science, Vol. 1947, Springer-Verlag, New York, 2001, pp. 53–63.
6. E. Elmroth and F. G. Gustavson, *New serial and parallel recursive QR factorization algorithms for SMP systems*, in Applied Parallel Computing, Large Scale Scientific and Industrial Problems, B. Kågström et al., eds., Lecture Notes in Computer Science, Vol. 1541, Springer-Verlag, New York, 1998, pp. 120–128.
7. F. G. Gustavson, *Recursion leads to automatic variable blocking for dense linear-algebra algorithms*, IBM J. Res. Develop., 41:6 (1997), pp. 737–755.
8. F. G. Gustavson, and I. Jonsson, *Minimal storage high performance Cholesky factorization via blocking and recursion*, IBM J. Res. Develop., 44:6 (2000), pp. 823–850.
9. IBM Corporation, *Engineering and Scientific Subroutine Library Guide and Reference*, Version 3, Release 3, December 2001. Order No. SA22-7272-04.
10. I. Jonsson and B. Kågström, *Parallel triangular Sylvester-type matrix equation solvers for SMP systems using recursive blocking*, in Applied Parallel Computing. New Paradigms for HPC in Industry and Academia, T. Sørvik et. al. (eds.), Lecture Notes in Computer Science, Vol. 1947, Springer-Verlag, New York, 2001, pp. 64–73.
11. R. Schreiber and C. Van Loan, *A storage efficient WY representation for products of Householder transformations*, SIAM J. Sci. Stat. Comput., 10:1 (1989), pp. 53–57.
12. S. Toledo, *Locality of reference in LU decomposition with partial pivoting*, SIAM J. Matrix. Anal. Appl., 18:4 (1997), pp. 1065–1081.