ANL/MCS-TM-134

LAPACK Working Note #12

Banded Cholesky Factorization Using Level 3 BLAS

by

Peter Mayes and Giuseppe Radicati

August 1989

MATHEMATICS AND COMPUTER SCIENCE DIVISION

Argonne National Laboratory, with facilities in the states of Illinois and Idaho, is owned by the United States government, and operated by The University of Chicago under the provisions of a contract with the Department of Energy.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439-4801

ANL/MCS-TM-134

LAPACK Working Note #12

Banded Cholesky Factorization Using Level 3 BLAS

by

Peter Mayes [*] *and Giuseppe Radicati* [†]

Mathematics and Computer Science Division

Technical Memorandum No. 134

August 1989

Permanent addresses:
[*] Numerical Algorithms Group Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, England
[†] European Center for Scientific and Engineering Computing, IBM Italia spa, 00147 Roma, via Giorgione 159, Italy

This work was supported in part by IBM Italy and NAG Ltd.

CONTENTS

Abstract....1

1. Introduction....1

2. The LINPACK Code....1

3. Level 2 BLAS Implementations of DPBTRF....2

4. Variants of the Cholesky Factorization....3

5. Block Factorizations of a Banded Matrix....5

6. Timings of the Block Version....7

7. Summary....11

8. Acknowledgements....11

References....12

Code for the Block Factorization....13

Code based on Level 2 BLAS....20

Banded Cholesky Factorization using Level 3 BLAS

Peter Mayes and Giuseppe Radicati

Abstract

This note describes a block implementation of the Cholesky factorization of a symmetric positive definite banded matrix. It is based on Level 3 BLAS and designed to perform well on a machine with a hierarchy of memories. The performance on the IBM 3090/VF of the block implementation is shown to be much better than implementations based on Level 2 BLAS.

1 Introduction

This note describes a series of experiments performed on the IBM 3090/VF with routines for computing the Cholesky factorization of a symmetric positive definite banded matrix. We have compared:

1. the LINPACK routine DPBFA [1],
2. the routine DPBF from IBM's ESSL Release 3 Library [2],
3. several implementations of the proposed LAPACK routine DPBTRF [3,4] based on calls to Level 2 BLAS [5,6],
4. block implementations of DPBTRF based on Level 3 BLAS [7,8].

For problems where the band width is large, the performance of block algorithms for this factorization can be significantly higher than routines based on Level 2 BLAS, and comes close to the speed of routines provided by the manufacturer. This approach can be extended to other factorizations of banded matrices, and also to the Cholesky factorization of a variable band width (or skyline) matrix.

2 The LINPACK Code

The LINPACK routine for computing the Cholesky factorization of a positive definite banded matrix is DPBFA. DPBFA computes the U^TU factorization of the

matrix A, whose upper triangle is supplied in banded storage mode. In this storage scheme, columns of the original matrix A are stored in columns in banded storage, and diagonals of A are stored in rows. If the half band width is K, then the leading diagonal of A is stored in row K+1 in banded storage, as shown in the example below, where N=7 and K=2:

$$\begin{pmatrix} 11 & 12 & 13 & 0 & 0 & 0 & 0 \\ 12 & 22 & 23 & 24 & 0 & 0 & 0 \\ 13 & 23 & 33 & 34 & 35 & 0 & 0 \\ 0 & 24 & 34 & 44 & 45 & 46 & 0 \\ 0 & 0 & 35 & 45 & 55 & 56 & 57 \\ 0 & 0 & 0 & 46 & 56 & 66 & 67 \\ 0 & 0 & 0 & 0 & 57 & 67 & 77 \end{pmatrix}$$

is stored as

$$\begin{pmatrix} & & 13 & 24 & 35 & 46 & 57 \\ & 12 & 23 & 34 & 45 & 56 & 67 \\ 11 & 22 & 33 & 44 & 55 & 66 & 77 \end{pmatrix}.$$

The LINPACK routine is implemented using the Level 1 BLAS routine DDOT, using columns of the matrix A. Since these are stored in columns in banded storage, the elements of the matrix are accessed with unit stride, which is highly desirable on the 3090/VF. The performance is limited only by the speed of DDOT, implemented in IBM's ESSL Library. For very large band widths, the peak speed of DPBFA is about 25 megaflops. For sufficiently large matrices, the speed of DPBFA depends only on the band width, and not on the size of the matrix.

3 Level 2 BLAS Implementations of DPBTRF

The proposed LAPACK routine DPBTRF extends the functionality of the LINPACK routine DPBFA by working not only with the U^TU factorization of a matrix A whose upper triangle is supplied (`UPLO='U'`), but also with the LL^T factorization of a matrix A whose lower triangle is supplied (`UPLO='L'`). For this second case, the matrix A is stored in a banded storage mode analogous to the `UPLO='U'` case, but this time the leading diagonal is stored in the first row in banded storage, as shown below:

$$\begin{pmatrix} 11 & 21 & 31 & 0 & 0 & 0 & 0 \\ 21 & 22 & 32 & 42 & 0 & 0 & 0 \\ 31 & 32 & 33 & 43 & 53 & 0 & 0 \\ 0 & 42 & 43 & 44 & 54 & 64 & 0 \\ 0 & 0 & 53 & 54 & 55 & 65 & 75 \\ 0 & 0 & 0 & 64 & 65 & 66 & 76 \\ 0 & 0 & 0 & 0 & 75 & 76 & 77 \end{pmatrix}$$

is stored as

$$\begin{pmatrix} 11 & 22 & 33 & 44 & 55 & 66 & 77 \\ 21 & 32 & 43 & 54 & 65 & 76 & \\ 31 & 42 & 53 & 64 & 75 & & \end{pmatrix}.$$

Initial implementations of DPBTRF were based on Level 2 BLAS. For example, the code to compute the case `UPLO='U'` was based on the routine DTRSV. Because DTRSV is not included in ESSL Release 3, and no tuned version was available to us, in the timing experiments we describe in the next section we have used DTRSM, which is available in ESSL Release 3, called with only one right hand side.

The code to compute the case `UPLO='L'` was based on the routine DSYR. We also implemented a version based on DTRSV, and although this was faster for small band widths, it has the drawback that the right hand side vector is accessed with stride equal to the half band width. This means that on the 3090/VF this variant is slower than the variant calling DSYR for large band widths.

There appears to be an inherent problem with the `UPLO='L'` case on the 3090/VF, if the factorization is to be based on Level 2 BLAS. Although the desire for accessing vectors with unit stride wherever possible leads us to prefer the version calling DSYR, we do not believe rank-1 updates can run any faster than a DAXPY operation on the 3090/VF. Thus a version based on Level 2 BLAS is likely to perform at Level 1 BLAS speeds, even with highly tuned Level 2 BLAS.

The results for the Level 2 BLAS implementation of DPBTRF are included in Figures 2 and 3, along with the LINPACK routine, the ESSL routine, and the block variants to be described below. For `UPLO='U'`, the Level 2 BLAS implementation calling DTRSV is significantly faster than the LINPACK code, but still falls far short of the ESSL code DPBF. For `UPLO='L'`, the Level 2 BLAS code based on DSYR is actually slower than the LINPACK code.

4 Variants of the Cholesky Factorization

Many algorithms in numerical linear algebra can be expressed in several different, but mathematically equivalent, forms. For example, Dongarra et al. [10] have described different variants of matrix multiplication and LU factorization, emphasizing those variants which use column-wise access, and are suitable for implementation in Fortran. If we consider the LL^T (`UPLO='L'`) Cholesky factorization, then we may derive three block variants by partitioning the matrices A and L as follows:

$$\begin{pmatrix} A_{11} & A_{21}^T & A_{31}^T \\ A_{21} & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T & L_{31}^T \\ 0 & L_{22}^T & L_{32}^T \\ 0 & 0 & L_{33}^T \end{pmatrix}$$

$$= \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T & L_{11}L_{31}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T & L_{21}L_{31}^T + L_{22}L_{32}^T \\ L_{31}L_{11}^T & L_{31}L_{21}^T + L_{32}L_{22}^T & L_{31}L_{31}^T + L_{32}L_{32}^T + L_{33}L_{33}^T \end{pmatrix}$$

1. "Top-looking" or "I" variant: the matrix L is generated one block row at a time, using the triangle already calculated above the current block row. The steps involved and corresponding BLAS calls are:

   (a) Calculate the current block row: $L_{21} \leftarrow A_{21}(L_{11}^T)^{-1}$
   `CALL DTRSM('Right','Lower','Transpose',...)`

   (b) Update the diagonal block: $A_{22} \leftarrow A_{22} - L_{21}L_{21}^T$
   `CALL DSYRK('Lower','No transpose',...)`

   (c) Factorize the current diagonal block: $A_{22} \rightarrow L_{22}L_{22}^T$
   `CALL DPOTF2(...)`

2. "Left-looking" or "J" variant: the matrix L is generated one block column at a time, using the block already calculated to the left of the current block column. The steps involved and corresponding BLAS calls are:

   (a) Update the current diagonal block: $A_{22} \leftarrow A_{22} - L_{21}L_{21}^T$
   `CALL DSYRK('Lower','No transpose',...)`

   (b) Factorize the current diagonal block: $A_{22} \rightarrow L_{22}L_{22}^T$
   `CALL DPOTF2(....)`

   (c) Update the subdiagonal block column: $A_{32} \leftarrow A_{32} - L_{31}L_{21}^T$
   `CALL DGEMM('No transpose','Transpose',...)`

   (d) Compute the subdiagonal block column: $L_{32} \leftarrow A_{32}(L_{22}^T)^{-1}$
   `CALL DTRSM('Right','Lower','Transpose',...)`

3. "Right-looking" or "K" variant: the current block column is computed, and immediately used to update the trailing submatrix to the right. The steps involved and corresponding BLAS calls are:

   (a) Factorize the current diagonal block: $A_{22} \rightarrow L_{22}L_{22}^T$
   `CALL DPOTF2(...)`

   (b) Compute the subdiagonal block column: $L_{32} = A_{32}(L_{22}^T)^{-1}$
   `CALL DTRSM('Right','Lower','Transpose',...)`

   (c) Update the trailing submatrix: $A_{33} \leftarrow A_{33} - L_{32}L_{32}^T$
   `CALL DSYRK('Lower','No transpose',...)`

In each case we need to factorize the current diagonal block, a symmetric positive definite matrix, of which only the lower triangle is stored. There are two routines in LAPACK to form the Cholesky factorization of a symmetric positive definite matrix: a blocked version DPOTRF, and an unblocked version DPOTF2. Since recursion is not permitted in Fortran 77, DPOTRF must call DPOTF2 to factorize the diagonal block.

5 Block Factorizations of a Banded Matrix

If we consider a block right-looking (or K-variant) Cholesky factorization of a full matrix, which on the 3090/VF is the the fastest version for a full matrix, the bulk of the work is performed in the update of the trailing submatrix using DSYRK (see [11,12]). If, however, we consider a banded matrix, as shown in figure 1, then the size of the trailing submatrix which must be updated using DSYRK as well as the size of the triangular system which must be solved using DTRSM, is limited by the size of the band width. Note that the dimensions of the submatrices, shown as `IB`, `I2` and `I3` on figure 1, are called by the same names in the Fortran listing given in the appendix.

As will be seen from figure 1, some operations will be wasted in a block version, because the block row (A_{12}, A_{13}) extends beyond the edge of the band, so that part of the RHS matrix is zero. However, in the case where the half band width is significantly larger than a typical block size (say hundreds compared with tens), the improved performance of the block factorization may make the wasted arithmetic worthwhile.

In the implementation we have considered, we need to use one auxiliary array the size of the diagonal block to store the part of the matrix, A_{13}, which lies astride the band front. This is accessed as a square matrix by the Level 3 BLAS, and part of it lies outside the banded storage array. Note that we always ensure internally that the block size is no bigger than the half band width. Figure 1 shows the way that we have implemented the calculation of the block row and the update of the trailing submatrix for the `UPLO='U'` case.

Initially the upper triangle of the work array is set to zero—this only needs to be done once. The computation then proceeds in the following steps:

1. the diagonal block A_{11} is factorized using DPOTF2,
2. A_{12} is calculated using DTRSM,
3. A_{22} is updated using DSYRK,
4. If the block row only extends as far as the edge of the band (rather than to the edge of the matrix, which happens when the diagonal block nears the end of the matrix), then A_{13} is copied into the work array,
5. DTRSM is called to calculate the remainder of the block row overwriting A_{13},
6. A_{23} and A_{33} are updated using DGEMM and DSYRK respectively.
7. the contents of the work array are copied back into the main array in packed storage.

To summarize, the updates

$$\begin{pmatrix} A_{22} & A_{23} \\ & A_{33} \end{pmatrix} \leftarrow \begin{pmatrix} A_{22} - A_{12}^T A_{12} & A_{23} - A_{12}^T A_{13} \\ & A_{33} - A_{13}^T A_{13} \end{pmatrix} \begin{pmatrix} DSYRK & DGEMM \\ & DSYRK \end{pmatrix}$$

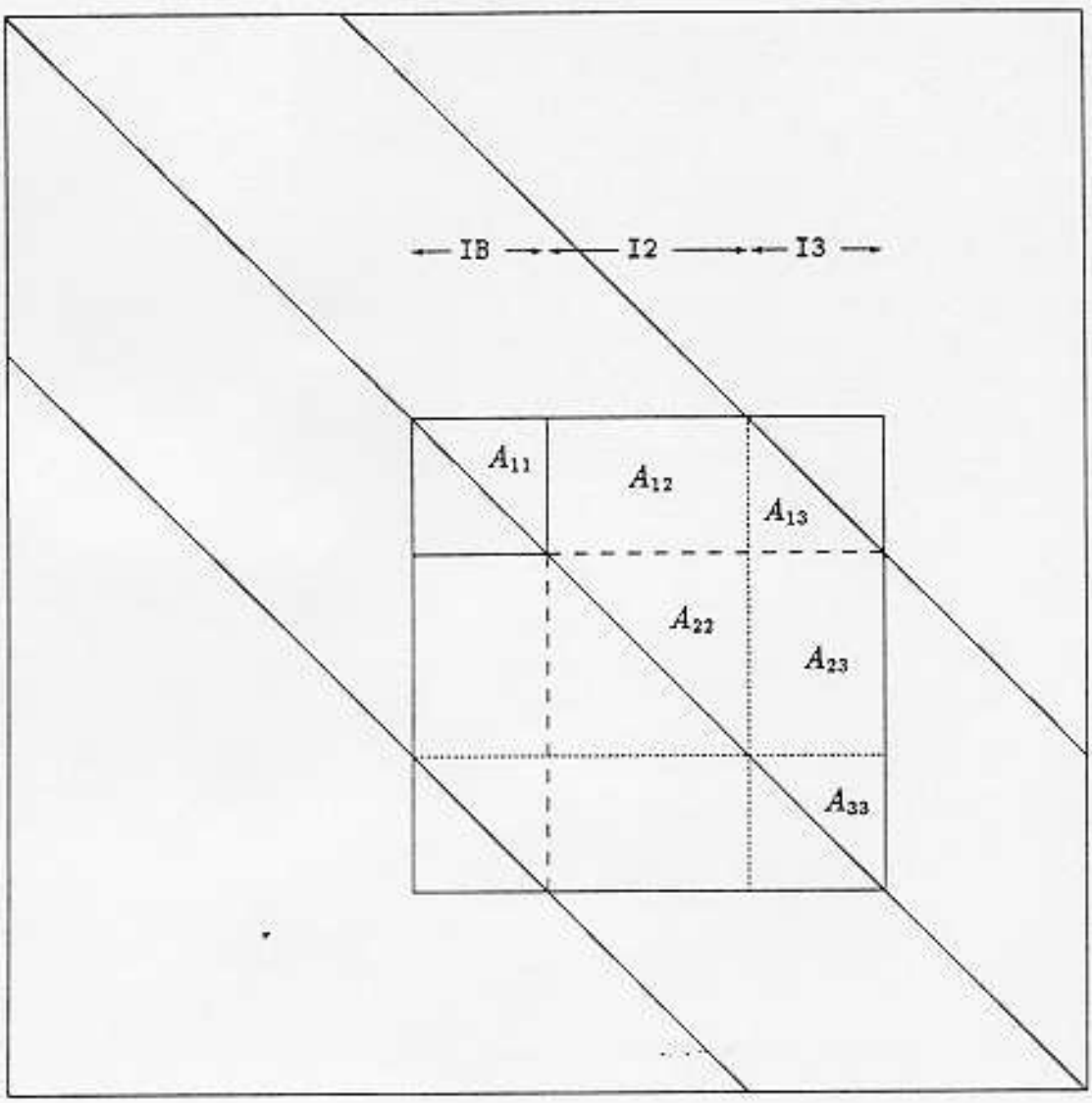

Figure 1: Block banded Cholesky - implementation details

are computed using the Level 3 BLAS calls shown. The factorization of the diagonal block is performed by the unblocked LAPACK routine DPOTF2. We can use DPOTF2 and the Level 3 BLAS on full sub-matrices stored in banded storage by passing the matrices to the lower level subroutines with a leading dimension of `LDA-1`.

The overhead of the block implementation includes setting the upper triangle of the work array to zero (done only once), copying the triangular matrix A_{13} into the work array, performing the operations with zero within DTRSM and DSYRK, and copying A_{13} back into the main array.

6 Timings of the Block Version

The timings described were obtained on the IBM 3090-600E/VF at ECSEC using one processor. We used the MVS/XA operating system, VS Fortran Version 2.3, and ESSL Release 3. We have timed the LINPACK routine DPBFA, the ESSL routine DPBF, the Level 2 BLAS implementation of DPBTRF, and the block implementation described in section 5. All speeds reported are in megaflops. Fluctuations of 5% in measured speeds are typical on a loaded system, such as the one we used. The Level 3 BLAS routines used are DGEMM, DTRSM and DSYRK, together with the Level 2 BLAS routine DGEMV. DTRSM, DGEMM and DGEMV are available in ESSL Release 3. We have developed a tuned Fortran version of DSYRK to improve its performance.

Figures 2 and 3 show our results for `UPLO='U'` and `UPLO='L'` respectively, and are labelled "Block". The graphs show the speed of the block implementation on a matrix of size 2,000 with a block size of 32, and illustrate the effect of varying the half band width. In both graphs the results from the LINPACK and ESSL routines are the same, as only one variant is available in each case, corresponding to the `UPLO='U'` case.

The ESSL routine, although documented as a U^TDU factorization, in fact uses the LAPACK `UPLO='L'` storage scheme. The elements of D are therefore stored in the first row in packed storage mode, and the elements of U differ from those of the corresponding LAPACK matrix by the square roots of the elements of D.

For `UPLO='U'` the speed of the block implementation increases steadily and smoothly, apparently reaching a peak of over 60 megaflops for large band widths and matrix size. By comparison, the ESSL routine DPPF for Cholesky factorization of a full matrix has a speed of around 64 megaflops for matrices of order around 500.

For `UPLO='L'` the performance of the block implementation is less regular but the envelope of the performance curve is very similar to the `UPLO='U'` case. This could be attributed to the less than optimal cache management of our Fortran version of DSYRK. It is likely that the usage of cache and the performance on small matrices could be significantly improved by using an assembler version of DSYRK, although we don't believe that a hand coded version of DSYRK would have a very much

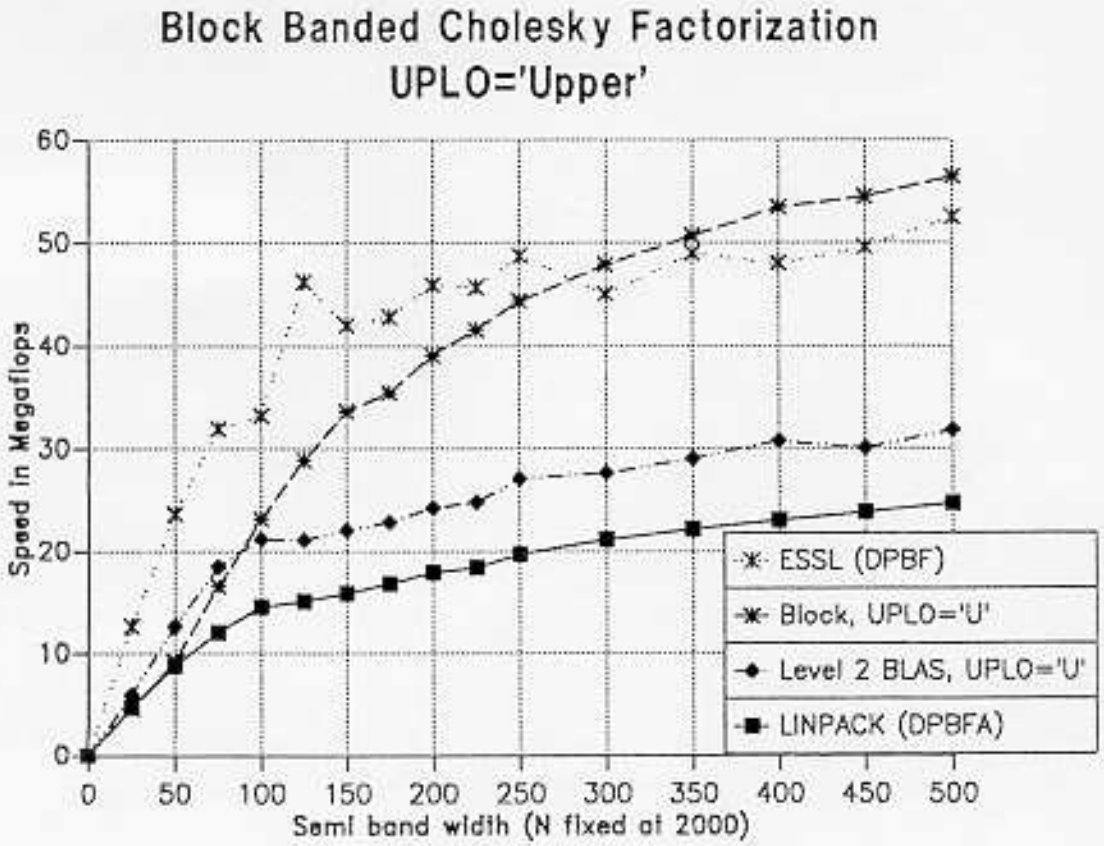

Figure 2: Timings for `UPLO='U'`

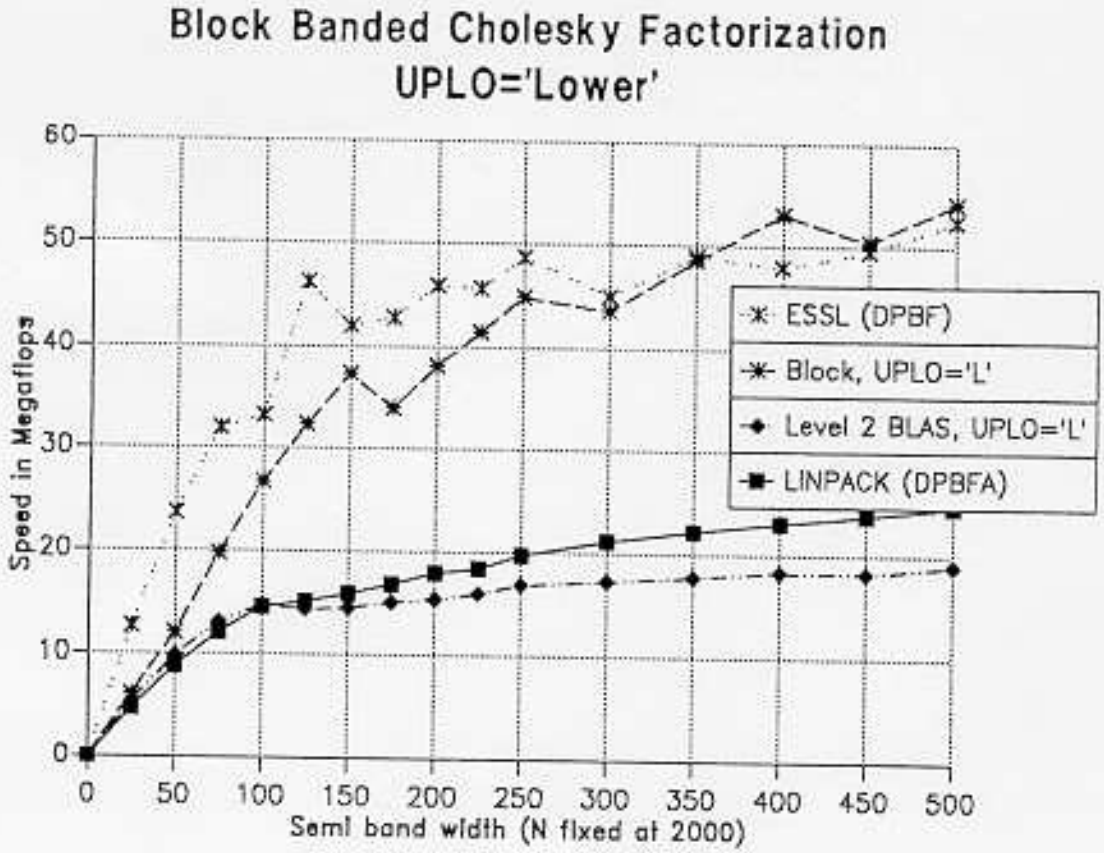

Figure 3: Timings for `UPLO='L'`

M(=N)	DGEMM (Fortran)	DGEMM (ESSL)	DSYRK (Fortran)
32 (LDA=512)	21.7	33.6	13.8
32 (LDA=530)	33.6	36.4	19.6
128 (LDA=512)	57.5	65.8	37.8
128 (LDA=530)	63.3	69.1	48.0
512 (LDA=512)	58.3	68.7	55.5
512 (LDA=530)	63.5	70.7	59.5

Table 1: Comparison of Fortran and ESSL BLAS

higher peak performance than our Fortran version.

In an attempt to see what improvement we might expect from a tuned version of DSYRK, we performed an experiment with DGEMM, which is available in ESSL. We took a Fortran version of this routine, and produced an improved Fortran version using a similar restructuring as used for DSYRK. Although DGEMM updates a full matrix, whereas DSYRK only updates one triangle of a symmetric matrix, the DO-loop whose limit stops at the leading diagonal is an outer one, and the vector loop is the same in both cases. Thus although DSYRK performs a significant amount of its operations with short vectors, the performance gain is likely to be comparable to the gain in DGEMM.

Table 1 shows the speed of the two versions of DGEMM, and for comparison, the speed of our improved DSYRK. It will be seen that the biggest differences between Fortran and ESSL versions of DGEMM occur when the leading dimension, LDA=512, is unfavorable for the 3090/VF, and when the matrices are small. These are the two situations when the block factorizations fall farthest behind ESSL, and which we expect a carefully tuned DSYRK to improve. Whilst not wishing to attach too great an importance to this simple experiment, it is certainly true that the results we have obtained are not the best that can be hoped for.

One curious feature of the ESSL routine DPBF is that it is the only version which is significantly affected by the order of the matrix N, for fixed half band width K. For example, for K=150, as N increased from 200 to 2000, the ESSL routine decreased in speed from 49 megaflops to 41 megaflops, whereas the block implementation for `UPLO='L'` increased from 36 megaflops to 37 megaflops with a block size of 32.

We have stated that the results for the block implementation are for a block size of 32. Although this is not always optimal, the performance is never much greater than the performance with this block size, and it would seem to be a good value at which to fix the block size. For the full matrix case, we found that the optimal block size was 128, although the performance was not very significantly worse for smaller values such as 32 and 64. However, in the banded case, the proportion of wasted operations increases and the performance drops off rapidly for block sizes larger than 32.

To our knowledge, the ESSL implementation of DTRSM does not contain any logic for checking for zeroes in the matrix of right hand sides. Because this check is not done in the innermost DO-loop, we believe that this could be done without

degrading the performance of the routine when the right hand side is mostly full. This could change the performance characteristics of the block implementation, favoring a larger block size as in the full case. The same could also be done for DSYRK.

The fact that operations are wasted at present means that for small band width the versions based on Level 2 BLAS are faster, as the benefits of the block formulation have yet to appear. The question of wasted operations arises with other block algorithms, such as the QR factorization. Although it would be possible to switch from a Level 2 BLAS formulation to a Level 3 BLAS version depending on the size of the half band width, this adds another dimension to the process of timing and implementing LAPACK software on different machines. Since this comes into the realm of "policy decisions" to be made by the LAPACK principal investigators, we prefer to raise the question while leaving it unanswered.

It should be noted that all the timings reported here have been obtained on an IBM 3090-E/VF. The more recent "S" model has a faster clock period, a larger cache, and better memory management. On this machine, the conclusion that the block implementation is superior to the Level 2 BLAS implementation is still valid. However, the performance of the ESSL routine DPBF relative to the block implementation is better. For example, for a matrix of order 2000 and half band width 500 with `UPLO='U'`, DPBFA performs at 34 megaflops, the Level 2 BLAS implementation of DPBTRF performs at 52 megaflops, the block version performs at 66 megaflops and DPBF at 72 megaflops.

7 Summary

We have investigated the use of a block formulation of the Cholesky factorization of a symmetric positive definite banded matrix. For small band widths we have shown that the performance of the block implementations is not much worse than formulations based on Level 2 BLAS and is likely to improve when a tuned version of DSYRK becomes available, and if DTRSM (and DSYRK) are modified to take advantage of sparsity. For larger band widths the block implementation runs at nearly twice the speed of routines based on Level 2 BLAS.

The block implementation described here has been included in the first batch of LAPACK software distributed to test sites. Preliminary results suggest that the conclusions of this report also apply to other machines with a hierarchical memory structure.

8 Acknowledgements

The authors would like to thank Jeremy Du Croz of the Numerical Algorithms Group Ltd and Edward Anderson of Argonne National Laboratory for their helpful suggestions. The first author would like to thank IBM for enabling him to visit ECSEC in Rome.

References

[1] Dongarra, J., Bunch, J., Moler, J., and Stewart, G. LINPACK Users' Guide. SIAM Philadelphia, 1979

[2] Engineering and Scientific Subroutine Library. Guide and Reference. SC23-0184-3

[3] Demmel, J., Dongarra, J.J., Du Croz, J.J., Greenbaum, A., Hammarling, S.J. and Sorensen, D., Prospectus for the Development of a Linear Algebra Library for High-Performance Computers. Technical Memorandum No. 97, MCS Division, Argonne National Laboratory, September 1987.

[4] Bischof, C., Demmel, J., Dongarra, J.J., Du Croz, J.J., Greenbaum, A., Hammarling, S.J. and Sorensen, D., LAPACK Working Note #5 Provisional Contents. ANL-88-38, Argonne National Laboratory, September 1988.

[5] Dongarra, J.J., Du Croz, J.J., Hammarling, S.J. and Hanson, R., An Extended Set of Fortran Basic Linear Algebra Subprograms. ACM Trans. Math. Software, 14(1), pp.1-17, March 1988.

[6] Dongarra, J.J., Du Croz, J.J., Hammarling, S.J. and Hanson, R., Algorithm 656: An Extended Set of Fortran Basic Linear Algebra Subprograms. ACM Trans. Math. Software, 14(1), pp.18-32, March 1988.

[7] Dongarra, J.J., Du Croz, J.J., Duff, I.S. and Hammarling, S.J., A Set of Level 3 Basic Linear Algebra Subprograms. Technical Memorandum No 88 (revision 1), MCS Division, Argonne National Laboratory, May 1988.

[8] Dongarra, J.J., Du Croz, J.J., Duff, I.S. and Hammarling, S.J., A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs. Technical Memorandum No. 122, MCS Division, Argonne National Laboratory, May 1988.

[9] NAG Mark 13 Fortran Library Manual. The Numerical Algorithms Group Ltd, Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK 1988.

[10] Dongarra, J.J., Gustavson, F. and Karp, A. Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine. SIAM Review, 26(1), pp. 91-112, 1984

[11] Mayes, P.J.D., Block Factorization Algorithms on the IBM 3090/VF. NAG Technical Report TR7/88, NAG Ltd, Oxford, 1988.

[12] Mayes, P.J.D. and Radicati, G. Block Factorization Algorithms on the IBM 3090/VF. in Proceedings of the 1989 International Conference on Supercomputing, Crete, Greece. pp. 263-270.

A Code for the Block Factorization

```
      SUBROUTINE DPBTRF( UPLO, N, K, A, LDA, INFO )
*
*  -- LAPACK routine --
*     Argonne National Lab, Courant Institute, and N.A.G. Ltd.
*     April 1, 1989
*
*     .. Scalar Arguments ..
      CHARACTER          UPLO
      INTEGER            INFO, K, LDA, N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * )
*     ..
*
*  Purpose
*  =======
*
*     DPBTRF computes the Cholesky factorization of a symmetric positive
*     definite band matrix A.
*     This is a blocked version which uses a right-looking Cholesky
*     variant.
*
*     Contributed by Peter Mayes and Giuseppe Radicati, IBM ECSEC, Rome
*     March 23, 1989
*
*  Arguments
*  =========
*
*  UPLO    - CHARACTER*1
*            On entry, UPLO specifies whether the upper or lower
*            triangular part of the symmetric band matrix A
*            is stored:
*               UPLO = 'U' or 'u'   The upper triangle of A is stored.
*               UPLO = 'L' or 'l'   The lower triangle of A is stored.
*            Unchanged on exit.
*
*  N       - INTEGER
*            On entry, N specifies the number of rows of the matrix
*            A .  N must be at least  zero.
*            Unchanged on exit.
*
*  K       - INTEGER
*            On entry, K  specifies the number of super-diagonals of the
*            matrix A if UPLO = 'U', or the number of sub-diagonals of
*            the matrix A if UPLO = 'L'.  K  must be at least zero.
```

```
*           Unchanged on exit.
*
*  A      - DOUBLE PRECISION array of dimension( LDA, N )
*           On entry, A contains the upper or lower triangle of the
*           matrix to be factored, stored in band format.
*           On exit, A is overwritten by the Cholesky factorization.
*           The factorization can be written either as
*              A = L*L'   where L is a lower triangular matrix or
*              A = U'*U   where U is an upper triangular matrix.
*
*  LDA    - INTEGER
*           On entry, LDA specifies the first dimension of A as
*           declared in the calling (sub) program.
*           LDA must be at least ( K  + 1 ).
*           Unchanged on exit.
*
*  INFO   - INTEGER
*           On exit, a value of 0 indicates a normal return.
*           A positive value K indicates that the leading minor of
*           order K is not positive definite, which is an error
*           condition that causes the subroutine to end.
*           A negative value, say -K, indicates the K-th argument has an
*           illegal value.
*
*     .. Parameters ..
      DOUBLE PRECISION   ONE
      INTEGER            MAXNB, LDWORK
      PARAMETER          ( ONE = 1.0D+0, MAXNB = 32, LDWORK = MAXNB+1 )
*     ..
*     .. Local Scalars ..
      INTEGER            I, I2, I3, IB, II, J, JJ, NB
*     ..
*     .. Local Arrays ..
      DOUBLE PRECISION   WORK( LDWORK, MAXNB )
*     ..
*     .. External Functions ..
      LOGICAL            LSAME
      EXTERNAL           LSAME
*     ..
*     .. External Subroutines ..
      EXTERNAL           ENVIR, DGEMM, DPBTF2, DPOTF2, DSYRK, DTRSM,
     $                   XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN
*     ..
*     .. Executable Statements ..
```

```
*
*     Test the input parameters.
*
      INFO = 0
      IF( ( .NOT.LSAME( UPLO, 'U' ) ) .AND.
     $    ( .NOT.LSAME( UPLO, 'L' ) ) ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( K.LT.0 .OR. K.GT.MAX( N-1, 0 ) ) THEN
         INFO = -3
      ELSE IF( LDA.LT.K+1 ) THEN
         INFO = -5
      END IF
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DPBTRF', -INFO )
         RETURN
      END IF
*
*     Quick return if possible.
*
      IF( N.EQ.0 )
     $   RETURN
*
*     Determine the block size for this environment
*
      CALL ENVIR( 'B', NB )
*
*     The block size must not exceed the semi-bandwidth K, and must not
*     exceed the limit set by the size of the local array WORK.
*
      NB = MIN( NB, K, MAXNB )
*
      IF( NB.LE.1 ) THEN
*
*        Use unblocked code
*
         CALL DPBTF2( UPLO, N, K, A, LDA, INFO )
      ELSE
*
*        Use blocked code
*
         IF( LSAME( UPLO, 'U' ) ) THEN
*
*           Compute the Cholesky factorization of a symmetric band
*           matrix, given the upper triangle of the matrix in band
*           storage.
```

```
*
*          Zero the upper triangle of the work array.
*
           DO 20 J = 1, NB
              DO 10 I = 1, J - 1
                 WORK( I, J ) = 0.0D0
   10         CONTINUE
   20      CONTINUE
*
*          Process the band matrix one diagonal block at a time
*
           DO 70 I = 1, N, NB
              IB = MIN( NB, N-I+1 )
*
*             Factorize the diagonal block
*
              CALL DPOTF2( UPLO, IB, A( K+1, I ), LDA-1, II )
              IF( II.NE.0 ) THEN
                 INFO = I + II - 1
                 GO TO 150
              END IF
              IF( I+IB.LE.N ) THEN
*
*                Update the relevant part of the trailing submatrix.
*                If A11 denotes the diagonal block which has just been
*                factorized, then we need to update the remaining
*                blocks in the diagram:
*
*                   A11   A12   A13
*                         A22   A23
*                               A33
*
*                The numbers of rows and columns in the partitioning
*                are IB, I2, I3 respectively. The blocks A12, A22 and
*                A23 are empty if IB = K. The upper triangle of A13
*                lies outside the band.
*
                 I2 = MIN( K-IB, N-I-IB+1 )
                 I3 = MIN( IB, N-I-K+1 )
*
                 IF( I2.GT.0 ) THEN
*
*                   Update A12
*
                    CALL DTRSM( 'Left', 'Upper', 'Transpose',
     $                          'Non-unit', IB, I2, ONE, A( K+1, I ),
     $                          LDA-1, A( K+1-IB, I+IB ), LDA-1 )
```

```
*
*                    Update A22
*
                     CALL DSYRK( 'Upper', 'Transpose', I2, IB, -ONE,
     $                            A( K+1-IB, I+IB ), LDA-1, ONE,
     $                            A( K+1, I+IB ), LDA-1 )
                  END IF
*
                  IF( I3.GT.0 ) THEN
*
*                    Copy the lower triangle of A13 into the work array.
*
                     DO 40 JJ = 1, I3
                        DO 30 II = JJ, IB
                           WORK( II, JJ ) = A( II-JJ+1, JJ+I+K-1 )
   30                   CONTINUE
   40                CONTINUE
*
*                    Update A13 (in the work array).
*
                     CALL DTRSM( 'Left', 'Upper', 'Transpose',
     $                           'Non-unit', IB, I3, ONE, A( K+1, I ),
     $                           LDA-1, WORK, LDWORK )
*
*                    Update A23
*
                     IF( I2.GT.0 )
     $                  CALL DGEMM( 'Transpose', 'No Transpose', I2, I3,
     $                              IB, -ONE, A( K+1-IB, I+IB ), LDA-1,
     $                              WORK, LDWORK, ONE, A( 1+IB, I+K ),
     $                              LDA-1 )
*
*                    Update A33
*
                     CALL DSYRK( 'Upper', 'Transpose', I3, IB, -ONE,
     $                           WORK, LDWORK, ONE, A( K+1, I+K ),
     $                           LDA-1 )
*
*                    Copy the lower triangle of A13 back into place.
*
                     DO 60 JJ = 1, I3
                        DO 50 II = JJ, IB
                           A( II-JJ+1, JJ+I+K-1 ) = WORK( II, JJ )
   50                   CONTINUE
   60                CONTINUE
                  END IF
               END IF
```

```
   70       CONTINUE
         ELSE
*
*           Compute the Cholesky factorization of a symmetric band
*           matrix, given the lower triangle of the matrix in band
*           storage.
*
*           Zero the lower triangle of the work array.
*
            DO 90 J = 1, NB
               DO 80 I = J + 1, NB
                  WORK( I, J ) = 0.0D0
   80          CONTINUE
   90       CONTINUE
*
*           Process the band matrix one diagonal block at a time
*
            DO 140 I = 1, N, NB
               IB = MIN( NB, N-I+1 )
*
*              Factorize the diagonal block
*
               CALL DPOTF2( UPLO, IB, A( 1, I ), LDA-1, II )
               IF( II.NE.0 ) THEN
                  INFO = I + II - 1
                  GO TO 150
               END IF
               IF( I+IB.LE.N ) THEN
*
*                 Update the relevant part of the trailing submatrix.
*                 If A11 denotes the diagonal block which has just been
*                 factorized, then we need to update the remaining
*                 blocks in the diagram:
*
*                    A11
*                    A21   A22
*                    A31   A32   A33
*
*                 The numbers of rows and columns in the partitioning
*                 are IB, I2, I3 respectively. The blocks A21, A22 and
*                 A32 are empty if IB = K. The lower triangle of A31
*                 lies outside the band.
*
                  I2 = MIN( K-IB, N-I-IB+1 )
                  I3 = MIN( IB, N-I-K+1 )
*
                  IF( I2.GT.0 ) THEN
```

```
*
*                    Update A21
*
                     CALL DTRSM( 'Right', 'Lower', 'Transpose',
     $                            'Non-unit', I2, IB, ONE, A( 1, I ),
     $                            LDA-1, A( 1+IB, I ), LDA-1 )
*
*                    Update A22
*
                     CALL DSYRK( 'Lower', 'No Transpose', I2, IB, -ONE,
     $                            A( 1+IB, I ), LDA-1, ONE, A( 1, I+IB ),
     $                            LDA-1 )
                  END IF
*
                  IF( I3.GT.0 ) THEN
*
*                    Copy the upper triangle of A31 into the work array.
*
                     DO 110 JJ = 1, IB
                        DO 100 II = 1, MIN( JJ, I3 )
                           WORK( II, JJ ) = A( K+1-JJ+II, JJ+I-1 )
  100                   CONTINUE
  110                CONTINUE
*
*                    Update A31 (in the work array).
*
                     CALL DTRSM( 'Right', 'Lower', 'Transpose',
     $                            'Non-unit', I3, IB, ONE, A( 1, I ),
     $                            LDA-1, WORK, LDWORK )
*
*                    Update A32
*
                     IF( I2.GT.0 )
     $                  CALL DGEMM( 'No transpose', 'Transpose', I3, I2,
     $                              IB, -ONE, WORK, LDWORK,
     $                              A( 1+IB, I ), LDA-1, ONE,
     $                              A( 1+K-IB, I+IB ), LDA-1 )
*
*                    Update A33
*
                     CALL DSYRK( 'Lower', 'No Transpose', I3, IB, -ONE,
     $                            WORK, LDWORK, ONE, A( 1, I+K ), LDA-1 )
*
*                    Copy the upper triangle of A31 back into place.
*
                     DO 130 JJ = 1, IB
                        DO 120 II = 1, MIN( JJ, I3 )
```

```
                        A( K+1-JJ+II, JJ+I-1 ) = WORK( II, JJ )
  120                   CONTINUE
  130                CONTINUE
                  END IF
               END IF
  140       CONTINUE
         END IF
      END IF
      RETURN
*
  150 CONTINUE
      RETURN
*
*     End of DPBTRF
*
      END
```

B Code based on Level 2 BLAS

```
      SUBROUTINE DPBTF2( UPLO, N, KD, A, LDA, INFO )
*
*  -- LAPACK routine --
*     Argonne National Lab, Courant Institute, and N.A.G. Ltd.
*     April 1, 1989
*
*     .. Scalar Arguments ..
      CHARACTER          UPLO
      INTEGER            INFO, KD, LDA, N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * )
*     ..
*
*  Purpose
*  =======
*
*     DPBTF2 computes the Cholesky factorization of a symmetric positive
*     definite band matrix A.
*     If UPLO = 'U' or 'u', the factor U is computed from
*           A = U' * U
*     If UPLO = 'L' or 'l', the factor L is computed from
*           A = L * L'
*     This is the Level 2 BLAS version of the algorithm.
*
*  Arguments
*  =========
```

```
*
*  UPLO   - CHARACTER*1
*           On entry, UPLO specifies whether the upper or lower
*           triangular part of the symmetric band matrix A
*           is stored:
*              UPLO = 'U' or 'u'   The upper triangle of A is stored.
*              UPLO = 'L' or 'l'   The lower triangle of A is stored.
*           Unchanged on exit.
*
*  N      - INTEGER
*           On entry, N specifies the number of rows of the matrix
*           A .  N must be at least  zero.
*           Unchanged on exit.
*
*  KD     - INTEGER
*           On entry, KD specifies the number of super-diagonals of the
*           matrix A if UPLO = 'U', or the number of sub-diagonals of
*           the matrix A if UPLO = 'L'.  KD must be at least zero.
*           Unchanged on exit.
*
*  A      - DOUBLE PRECISION array of dimension( LDA, N )
*           On entry, A specifies the array which contains the matrix
*           being factored.
*
*           Before entry with UPLO = 'U' or 'u', the leading ( KD + 1 )
*           by N part of the array A must contain the upper triangular
*           band part of the symmetric matrix, supplied column by
*           column, with the leading diagonal of the matrix in row
*           ( KD + 1 ) of the array, the first super-diagonal starting
*           at position 2 in row KD, and so on. The top left KD by KD
*           triangle of the array A is not referenced.
*
*           Before entry with UPLO = 'L' or 'l', the leading ( KD + 1 )
*           by N part of the array A must contain the lower triangular
*           band part of the symmetric matrix, supplied column by
*           column, with the leading diagonal of the matrix in row 1 of
*           the array, the first sub-diagonal starting at position 1 in
*           row 2, and so on. The bottom right KD by KD triangle of the
*           array A is not referenced.
*
*           On exit, the array A is overwritten by the Cholesky
*           factorization. The factorization can be written either as
*                A = L * L'  where L is a lower triangular matrix
*           or   A = U' * U  where U is an upper triangular matrix.
*
*  LDA    - INTEGER
*           On entry, LDA specifies the first dimension of A as
```

```
*          declared in the calling (sub) program.
*          LDA must be at least ( KD + 1 ).
*          Unchanged on exit.
*
*  INFO    - INTEGER
*          On exit, a value of 0 indicates a normal return.
*          A positive value K indicates that the leading minor of
*          order K is not positive definite, which is an error
*          condition that causes the subroutine to end.
*          A negative value, say -K, indicates the K-th argument has an
*          illegal value.
*
*     .. Parameters ..
      DOUBLE PRECISION   ONE, ZERO
      PARAMETER          ( ONE = 1.0D+0, ZERO = 0.0D+0 )
*     ..
*     .. Local Scalars ..
      INTEGER            J, KJ, KLD, KN, KROW
      DOUBLE PRECISION   S
*     ..
*     .. External Functions ..
      LOGICAL            LSAME
      DOUBLE PRECISION   DDOT
      EXTERNAL           LSAME, DDOT
*     ..
*     .. External Subroutines ..
      EXTERNAL           DSCAL, DSYR, DTRSV, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX, MIN, SQRT
*     ..
*     .. Executable Statements ..
*
*     Test the input parameters.
*
      INFO = 0
      IF( ( .NOT.LSAME( UPLO, 'U' ) ) .AND.
     $    ( .NOT.LSAME( UPLO, 'L' ) ) ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( KD.LT.0 .OR. KD.GT.MAX( N-1, 0 ) ) THEN
         INFO = -3
      ELSE IF( LDA.LT.KD+1 ) THEN
         INFO = -5
      END IF
      IF( INFO.NE.0 ) THEN
```

```
         CALL XERBLA( 'DPBTF2', -INFO )
         RETURN
      END IF
*
*     Quick return if possible.
*
      IF( N.EQ.0 )
     $   RETURN
*
      KLD = MAX( 1, LDA-1 )
*
      IF( LSAME( UPLO, 'U' ) ) THEN
*
*        Compute the Cholesky factorization of the symmetric band matrix
*        stored in the upper part of the array.
*
         DO 10 J = 1, N
            KJ = MAX( J-KD, 1 )
            KN = J - KJ
            KROW = KD + 1 - KN
            CALL DTRSV( 'Upper', 'Transpose', 'Non-unit', KN,
     $                  A( KD+1, KJ ), KLD, A( KROW, J ), 1 )
            S = A( KD+1, J ) - DDOT( KN, A( KROW, J ), 1, A( KROW, J ),
     $          1 )
*
*           Exit if the matrix is not positive definite.
*
            IF( S.LE.ZERO )
     $         GO TO 30
            A( KD+1, J ) = SQRT( S )
   10    CONTINUE
      ELSE
*
*        Compute the Cholesky factorization of the symmetric band matrix
*        stored in the lower part of the array.
*
         DO 20 J = 1, N - 1
            IF( A( 1, J ).LE.ZERO )
     $         GO TO 30
            A( 1, J ) = SQRT( A( 1, J ) )
            KN = MIN( KD, N-J )
            CALL DSCAL( KN, ONE / A( 1, J ), A( 2, J ), 1 )
            CALL DSYR( 'Lower', KN, -ONE, A( 2, J ), 1, A( 1, J+1 ),
     $                 KLD )
   20    CONTINUE
         J = N
         IF( A( 1, N ).LE.ZERO )
```

```
     $       GO TO 30
             A( 1, N ) = SQRT( A( 1, N ) )
          END IF
          RETURN
*
   30 CONTINUE
          INFO = J
          RETURN
*
*     End of DPBTF2
*
          END
```

LAPACK Working Notes

#1 - James Demmel, Jack J. Dongarra, Jeremy DuCroz, Anne Greenbaum, Sven Hammarling, and Danny Sorensen, "Prospectus for the Development of a Linear Algebra Library for High-Performance Computers," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 97, September 1987.

#2 - Jack J. Dongarra, Sven J. Hammarling, and Danny Sorensen, "Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 99, September 1987.

#3 - James Demmel and W. Kahan, "Computing Small Singular Values of Bidiagonal Matrices with Guaranteed High Relative Accuracy," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 110, February 1988.

#4 - James Demmel, Jeremy DuCroz, Sven Hammarling, and Dan Sorensen, "Guidelines for the Design of Symmetric Eigenroutines, SVD, and Iterative Refinement and Condition Estimation for Linear Systems," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 111, March 1988.

#5 - Chris Bischof, James Demmel, Jack Dongarra, Jeremy DuCroz, Anne Greenbaum, Sven Hammarling, and Danny Sorensen, "Provisional Contents," Argonne National Laboratory Report ANL-88-38, September 1988.

#6 - O. Brewer, J. Dongarra, and D. Sorensen, "Tools to Aid in the Analysis of Memory Access Patterns for Fortran Programs," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 120, June 1988.

#7 - Jesse Barlow and James Demmel, "Computing Accurate Eigensystems of Scaled Diagonally Dominant Matrices," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 126, December 1988.

#8 - Z. Bai and J. Demmel, "On a Block Implementation of Hessenberg Multishift QR Iteration," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 127, January 1989.

#9 - J. Demmel and A. McKenney, "A Test Matrix Generation Suite," Argonne National Laboratory, Mathematics and Computer Science Division, preprint MCS-P69-0389, March 1989.

#10 - Edward Anderson and Jack Dongarra, "Installing and Testing the Initial Release of LAPACK - Unix and Non-Unix Versions," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 130, May 1989.

#11 - Percy Deift, James Demmel, Luen-Chau Li, and Carlos Tomei, "The Bidiagonal Singular Value Decomposition and Hamiltonian Mechanics," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 133, August 1989.

#12 - Peter Mayes and Giuseppe Radicati, "Banded Cholesky Factorization using Level 3 BLAS," Argonne National Laboratory, Mathematics and Computer Science Division, Technical Memorandum No. 134, August 1989.