Appendix B - Calling Sequences for all the Level 3 BLAS.

name	options					dim			scalar	matrix		matrix		scalar	matrix	
_GEMM (			TRANSA,	TRANSB,		M,	N,	K,	ALPHA,	A,	LDA,	B,	LDB,	BETA,	C,	LDC)
_SYMM (	SIDE,	UPLO,				M,	N,		ALPHA,	A,	LDA,	B,	LDB,	BETA,	C,	LDC)
_HEMM (	SIDE,	UPLO,				M,	N,		ALPHA,	A,	LDA,	B,	LDB,	BETA,	C,	LDC)
_SYRK (		UPLO,	TRANS ,				N,	K,	ALPHA,	A,	LDA,			BETA,	C,	LDC)
_HERK (		UPLO,	TRANS ,				N,	K,	ALPHA,	A,	LDA,			BETA,	C,	LDC)
_SYR2K(		UPLO,	TRANS ,				N,	K,	ALPHA,	A,	LDA,	B,	LDB,	BETA,	C,	LDC)
_HER2K(		UPLO,	TRANS ,				N,	K,	ALPHA,	A,	LDA,	B,	LDB,	BETA,	C,	LDC)
_TRMM (	SIDE,	UPLO,	TRANSA,		DIAG,	M,	N,		ALPHA,	A,	LDA,	B,	LDB)			
_TRSM (	SIDE,	UPLO,	TRANSA,		DIAG,	M,	N,		ALPHA,	A,	LDA,	B,	LDB			

```
      SUBROUTINE DLLT(N,A,LDA,INFO)
*
*     Computes an L*L**T factorization of a symmetric positive-definite
*     matrix A.
*     Unblocked version, calling Level 2 and Level 1 BLAS.
*
      INTEGER            INFO, LDA, N
      DOUBLE PRECISION A(LDA,*)
      INTEGER            J
      DOUBLE PRECISION DDOT
      EXTERNAL           DDOT
      EXTERNAL           DGEMV, DSCAL
*
      INFO = 0
      DO 10 J = 1, N
*
*        Update a(j,j).
*
         A(J,J) = A(J,J) - DDOT(J-1,A(J,1),LDA,A(J,1),LDA)
*
*        Compute l(j,j) and test for non-positive-definiteness.
*
         IF (A(J,J).LE.0.0D0) GO TO 20
         A(J,J) = SQRT(A(J,J))
*
         IF (J.LT.N) THEN
*
*           Update elements j+1:n of j-th column.
*
            CALL DGEMV('No transpose',N-J,J-1,-1.0D0,A(J+1,1),LDA,
     $                 A(J,1),LDA,1.0D0,A(J+1,J),1)
*
*           Compute elements j+1:n of j-th column of L.
*
            CALL DSCAL(N-J,1.0D0/A(J,J),A(J+1,J),1)
         END IF
   10 CONTINUE
      RETURN
*
   20 INFO = J
      RETURN
      END
```

Appendix A - Blocked Cholesky Factorization

```
      SUBROUTINE DLLTB(N,A,LDA,INFO)
*
*     Computes an L*L**T factorization of a symmetric positive-definite
*     matrix A.
*     Blocked version, calling Level 3 BLAS.
*
      INTEGER          INFO, LDA, N
      DOUBLE PRECISION A(LDA,*)
      INTEGER          J, JB
      INTEGER          NB
      PARAMETER        (NB=64)
      EXTERNAL         DGEMM, DLLT, DSYRK, DTRSM
*
      INFO = 0
      DO 10 J = 1, N, NB
         JB = MIN(NB,N-J+1)
*
*        Update diagonal block.
*
         CALL DSYRK('Lower','No transpose',JB,J-1,-1.0D0,A(J,1),LDA,
     $              1.0D0,A(J,J),LDA)
*
*        Factorize diagonal block and test for
*        non-positive-definiteness.
*
         CALL DLLT(JB,A(J,J),LDA,INFO)
         IF (INFO.NE.0) GO TO 20
*
         IF (J+JB.LE.N) THEN
*
*           Update subdiagonal block.
*
            CALL DGEMM('No transpose','Transpose',N-J-JB+1,JB,J-1,
     $                 -1.0D0,A(J+JB,1),LDA,A(J,1),LDA,1.0D0,A(J+JB,J),
     $                 LDA)
*
*           Compute subdiagonal block of L.
*
            CALL DTRSM('Right','Lower','Transpose','Non-unit',N-J-JB+1,
     $                 JB,1.0D0,A(J,J),LDA,A(J+JB,J),LDA)
         END IF
   10 CONTINUE
      RETURN
*
   20 INFO = INFO + J - 1
      RETURN
      END
```

C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, ‘‘Basic Linear Algebra Subprograms for Fortran Usage,’’ *ACM Trans. Math. Software* **5**, pp. 308-323 (1979a).

C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, ‘‘Algorithm 539: Basic Linear Algebra Subprograms for Fortran Usage,’’ *ACM Trans. Math. Software* **5**, pp. 324-325 (1979b).

A. C. McKellar and E. G. Coffman, Jr., ‘‘Organizing Matrices and Matrix Operations for Paged Memory Systems,’’ *CACM* **12,3**, pp. 153-165 (1969).

Y. Robert and P. Sguazzero, ‘‘The LU Decomposition Algorithm and Its Efficient Fortran Implementation on the IBM 3090 Vector Multiprocessor,’’ IBM ECSEC Report ICE-0006 (1987).

R. Schreiber, ‘‘Module Design Specification (Version 1.0),’’ SAXPY Computer Corporation, 255 San Geronimo Way, Sunnyvale, Calif. 94086 (1986).

R. Schreiber and B. Parlett, ‘‘Block Reflectors: Theory and Computation,’’ *SIAM J. Numer. Anal.* **25, 1**, pp. 189-205 (February 1988).

J. J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, ‘‘An Extended Set of Fortran Basic Linear Algebra Subprograms,’’ *ACM Trans. Math. Software* **14, 1**, pp. 1-17 (March 1988a).

J. J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, ‘‘An Extended Set of Fortran Basic Linear Algebra Subprograms: Model Implementation and Test Programs,’’ *ACM Trans. Math. Software* **14, 1**, pp. 18-32 (March 1988b).

J. J. Dongarra, J. DuCroz, I. S. Duff, and S. Hammarling, ‘‘A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs,’’ Argonne National Laboratory Report, ANL-MCS-P88-2 (August 1988).

J. J. Dongarra and I. S. Duff, ‘‘Advanced Architecture Computers,’’ Argonne National Laboratory Report, ANL-MCS-TM-57 (Revision 1) (January, 1987).

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

J. J. Dongarra, S. Hammarling, and D.C. Sorensen, ‘‘Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations,’’ Argonne National Laboratory Report, ANL-MCS-TM-99 (September 1987).

J. J. Dongarra and T. Hewitt, ‘‘Implementing Dense Linear Algebra Algorithms Using Multitasking on the CRAY X-MP-4,’’ *SIAM J. Sci. Stat. Comp.* **7, 1**, pp. 347-350 (January, 1986).

J. J. Dongarra and D.C. Sorensen, ‘‘Linear Algebra on High-Performance Computers,’’ pp. 113-136 in *Proceedings Parallel Computing 85*, ed. U. Schendel, North Holland (1986).

J. DuCroz, S. Nugent, J. Reid, and D. Taylor, ‘‘Solving Large Full Sets of Linear Equations in a Paged Virtual Store,’’ *ACM Trans. Math. Software* **7, 4**, pp. 527-536 (1981).

I. S. Duff, ‘‘Full Matrix Techniques in Sparse Gaussian Elimination,’’ *Numerical Analysis Proceedings, Dundee 1981, Lecture Notes in Mathematics 912*, pp. 71-84, Springer-Verlag (1981).

K. Gallivan, W. Jalby, and U. Meier, ‘‘The Use of BLAS3 in Linear Algebra on a Parallel Processor with a Hierarchical Memory,’’ *SIAM J. Sci. Stat. Comput.* **8, 6**, pp. 1079-1084 (November 1987).

A. George and H. Rashwan, ‘‘Auxiliary Storage Methods for Solving Finite Element Systems,’’ *SIAM J. Sci. Stat. Comput.* **6, 4**, pp. 882-910 (October 1985).

IBM, ‘‘Engineering and Scientific Subroutine Library,’’ Program Number: 5668-863 (1986).

D. W. Barron and H. P. F. Swinnerton-Dyer, ‘‘Solution of Simultaneous Linear Equations Using a Magnetic-Tape Store,’’ *Computer J.* **3**, pp. 28-33 (1960).

M. Berry, K. Gallivan, W. Harrod, W. Jalby, S. Lo, U. Meier, B. Philippe, and A. Sameh, ‘‘Parallel Algorithms on the CEDAR System,’’ *CSRD Report No. 581* (1986).

C. Bischof and C. Van Loan, ‘‘The WY Representation for Products of Householder Matrices,’’ *SIAM J. Sci. Stat. Comp.* **8, 1**, pp. s2-s13 (January 1987).

O. E. Bronlund and T. Johnsen, ‘‘QR-factorization of Partitioned Matrices,’’ *Comput. Methods Appl. Mech. Engrg.* **3**, pp. 153-172 (1974).

I. Bucher and T. Jordan, ‘‘Linear Algebra Programs for Use on a Vector Computer with a Secondary Solid State Storage Device,’’ pp. 546-550 in *Advances in Computer Methods for Partical Differential Equations*, ed. R. Vichnevetsky and R. Stepleman, IMACS (1984).

D. A. Calahan, ‘‘Block-Oriented Local-Memory-Based Linear Equation Solution on the CRAY-2: Uniprocessor Algorithms,’’ *Proceedings International Conference on Parallel Processing*, IEEE Computer Society Press (August 1986).

P. Carnevali, G. Radicati di Brozolo, Y. Robert, and P. Sguazzero, ‘‘Efficient Fortran Implementation of the Gaussian Elimination and Householder Reduction Algorithms on the IBM 3090 Vector Multiprocessor,’’ IBM ECSEC Report ICE-0012 (1987).

B. Chartres, ‘‘Adaption of the Jacobi and Givens Methods for a Computer with Magnetic Tape Backup Store,’’ *University of Sydney Technical Report No. 8* (1960).

A. K. Dave and I. S. Duff, ‘‘Sparse Matrix Calculations on the CRAY-2,’’ *Parallel Computing* **5**, pp. 55-64 (July 1987).

J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen, ‘‘Prospectus for the Development of a Linear Algebra Library for High-Performance Computers,’’ Argonne National Laboratory Report, ANL-MCS-TM-97 (September 1987).

G. Dietrich, ‘‘A New Formulation of the Hypermatrix Householder QR-decomposition,’’ *Comput. Methods Appl. Mech. Engrg.* **9**, pp. 273-280 (1976).

D. Dodson and J. Lewis, ‘‘Issues Relating to Extension of the Basic Linear Algebra Subprograms,’’ *ACM SIGNUM Newsletter* **20, 1**, pp. 2-18 (1985).

J. J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *LINPACK Users’ Guide*, SIAM Pub. (1979).

$$A'_{22} \rightarrow L_{22}L_{22}^T$$

3. update the subdiagonal block:

$$A'_{32} \leftarrow A_{32} - L_{31}L_{21}^T$$

4. compute the subdiagonal block of L:

$$L_{32} \leftarrow A'_{32}(L_{22}^T)^{-1}$$

To express the complete algorithm, we adopt a notation in which the matrix is partitioned into blocks A_{ij} of size $nb \times nb$ with $p = \lceil n/nb \rceil$ and use $A_{j,1:j-1}$, for example, to denote the block row $[A_{j,1}, A_{j,2}, \cdots, A_{j,j-1}]$. The algorithm is

```
for $j = 1$ to $p$
    $A_{jj} \leftarrow A_{jj} - A_{j,1:j-1}A_{j,1:j-1}^T$                          (_SYRK)
    factorize $A_{jj}$                                        (unblocked algorithm)
    $A_{j+1:p,j} \leftarrow A_{j+1:p,j} - A_{j+1:p,1:j-1}A_{j,1:j-1}^T$                 (_GEMM)
    $A_{j+1:p,j} \leftarrow A_{j+1:p,j}(A_{jj}^T)^{-1}$                          (_TRSM)
```

In Appendix A we give the Fortran code for a block Cholesky factorization routine DLLTB, calling Level 3 BLAS routines; and also a lower level routine DLLT which is called by DLLTB and calls Level 1 and 2 BLAS routines. A separate lower level routine is needed since current standard Fortran forbids recursion. The structure of DLLTB has been kept as similar as possible to that of DLLT. The call-tree is

```
                              DLLTB

DSYRK              DLLT                DGEMM     DTRSM

          DDOT     DGEMV     DSCAL
```

Acknowledgments

Draft proposals for the Level 3 BLAS were discussed initially at a workshop at Argonne National Laboratory, on January 26-27, 1987, and subsequently at various meetings. We thank all the participants in those discussions for their comments and encouragement. We thank John Lewis in particular for urging us to provide for complex symmetric matrices.

There are other ways to organize the computation: for example, it is equally possible to compute a block of consecutive rows at each stage. The analysis of (Dongarra, Gustavson and Karp, 1984) can easily be extended to block algorithms. We have chosen an organization that works by columns rather than rows, and that involves fewest memory references.

Also, we have implemented the algorithms in such a way that submatrices passed to the Level 3 BLAS routines are kept as large as possible (once the block size has been fixed): this gives greatest scope for achieving efficiency within the Level 3 BLAS. Alternatively one might explicitly partition the matrix into, say, square blocks of size $nb \times nb$: this would require many more calls to the Level 3 BLAS routines, but might allow a more precise control of memory or of parallelism.

We assume that we are given a positive-definite symmetric matrix A whose lower triangle is stored in the lower triangle of a two-dimensional array, and we wish to compute L overwriting the given elements of A.

We can partition the matrices so that

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

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

Here L_{22} and L_{32} constitute the current block of columns of L to be computed, and we assume that L_{11}, L_{21}, and L_{31} constitute the blocks, if any, that have already been computed. Note that the blocks in the above partitioning are not all of equal size: the off-diagonal blocks are, in general, rectangular.

Equating blocks, we have

$$A_{22} = L_{21}L_{21}^T + L_{22}L_{22}^T$$

$$A_{32} = L_{31}L_{21}^T + L_{32}L_{22}^T$$

so that

$$L_{22}L_{22}^T = A_{22} - L_{21}L_{21}^T$$

$$L_{32} = (A_{32} - L_{31}L_{21}^T)(L_{22}^T)^{-1}$$

Thus the computation of one block-column of the result involves the following operations:

1. update the diagonal block:

$$A_{22}' \leftarrow A_{22} - L_{21}L_{21}^T$$

2. compute the Cholesky factorization of the diagonal block:

type. This conforms to the conventions already established for the Level 1 and Level 2 BLAS, and also other software such as Linpack. However the fact that single and double precision versions of a BLAS routine have different names can be an obstacle to portability because the actual precision of an S routine, say, may differ considerably between machines. For example, SGEMM on a Cray 2 will use arithmetic with similar precision to DGEMM on an IBM 3090. The ideal solution would be to use generic names, not only for single and double precision versions, but also for real and complex versions. This option is not available in a standard Fortran 77 environment. However for implementations in other environments or programming languages, which do permit generic names, we propose that the first character of the Fortran 77 names should simply be omitted, giving the generic names: GEMM, SYMM, SYRK, SYR2K, HEMM, HERK, HER2K, TRMM, TRSM.

9. Applications

The primary intended application of the Level 3 BLAS is in implementing algorithms of numerical linear algebra in terms of operations on submatrices (or blocks). There is a long history of block algorithms: a few references are (Barron and Swinnerton-Dyer, 1960), (Chartres, 1960), (McKellar and Coffman, 1969), (Bronlund and Johnsen, 1974), (Dietrich, 1976), (DuCroz *et al.*, 1981), (Calahan, 1986), and (Dave and Duff, 1987). Both the NAG and the IMSL (Edition 9) libraries include such algorithms (F01BTF and F01BXF in NAG; LEQIF and LEQOF in IMSL). The earlier work was usually concerned with submatrices being transferred between the main memory and disk or tape. Similar concerns motivated work designed to exploit common page-swapping algorithms in virtual memory machines. Indeed the techniques are similar wherever there exists any hierarchy of data storage (in terms of access speed). Additionally, full blocks, and hence the multiplication of full matrices, might appear as a subproblem when handling large sparse systems of equations (for example, (Duff, 1981), (George and Rashwan, 1985), (Dave and Duff, 1987)).

More recently several workers have demonstrated the effectiveness of block algorithms on a variety of modern computer architectures, with vector-processing or parallel-processing capabilities, on which potentially high performance can easily be degraded by excessive transfer of data between different levels of memory (vector registers, cache, local memory, main memory or solid-state disks) (Dongarra and Hewitt, 1986), (Berry, Jalby and Meier, 1986), (IBM, 1986), (Robert and Sguazzero, 1987), (Bischof and Van Loan, 1987), (Bucher and Jordan, 1984), (Schreiber, 1986), (Calahan, 1986), (Dongarra and Sorensen, 1986), (Gallivan *et al.*, 1987), (Dongarra, Hammarling, and Sorensen, 1987), (Carnevali *et al.*, 1987), and (Schreiber and Parlett, 1988). See (Demmel *et al.*, 1987) for a proposal to develop a new linear algebra library, using block algorithms wherever possible and calling Level 3 BLAS.

Here we illustrate how the Level 3 BLAS routines can be used to implement a simple algorithm of numerical linear algebra, namely, Cholesky factorization.

The strategy is to compute at each stage a block of consecutive columns of the result. The size of the block is a parameter, *nb*, that may be varied to suit the size of the problem and the architecture of the machine. (For transportable software, we shall need some means of determining the blocksize within the routine, but we set that issue aside here.)

In almost every case, where appropriate, we include operations involving a matrix and its transpose (the only exceptions are the _SYMM and _HEMM routines). We could ask the user to transpose the input matrix but feel that this would be an imposition, particularly if the BLAS routine is being called from deep within the user's code. It would also increase the amount of data movement, whereas one of the aims of our proposal is to assist the development of software that minimizes data movement.

It could also be argued that algorithms can be rewritten to require only one of the patterns of access for symmetric, Hermitian or triangular matrices (i.e., upper or lower triangle), but we do not feel that the BLAS should be dictating this to the user.

We do not provide routines for operations involving trapezoidal matrices; all our triangular matrices are square. This is consistent with the Level 2 BLAS. It would be possible to extend the routines for triangular matrices, so that they could handle trapezoidal matrices, at the cost of introducing extra arguments. On the other hand, a trapezoidal matrix can always be partitioned into a triangular matrix and a rectangular matrix.

We have not included specialized routines to take advantage of packed storage schemes for symmetric, Hermitian, or triangular matrices, nor of compact storage schemes for banded matrices, because such storage schemes do not seem to lend themselves to partitioning into blocks and hence are not likely to be so useful in the type of application we are aiming at. Also packed storage is required much less with large memory machines available today, and we wish to keep the set of routines as small as possible.

We also have not specified a set of extended-precision routines analogous to the ES and EC routines in the Level 2 BLAS, since this would require a two-dimensional array in extended precision.

As with the Level 2 BLAS no check has been included for singularity, or near singularity, in the routines for solving triangular equations. The requirements for such a test depend on the application and so we felt that this should not be included, but should instead be performed outside the triangular solver.

We have tried to adhere to the convention of, and maintain consistency with, the Level 2 BLAS; however, we have deliberately departed from this approach in a few cases. The input-output matrix C in the matrix-multiply routines is the analogue of the vector y in the matrix-vector product routines. But here C always has the same dimensions, whereas y was either of length m or n depending on context. In the rank-k update routines we have included a parameter β which was not present in the Level 2 rank update routines. Here we felt that the parameter β is useful in applications, and since the matrix multiply routines can also be viewed as rank-k update routines, we have consistency between the MM, RK and R2K routines.

We have also added a parameter α to the routines involving triangular matrices. This was not felt to be needed in the corresponding Level 2 BLAS, because there would be little additional cost in a separate operation to scale the result vector by α. However in the Level 3 BLAS where there is a whole matrix to be scaled, it is advantageous to incorporate the scaling within _TRMM or _TRSM.

Additionally, we have provided for complex symmetric, as well as complex Hermitian, matrices since they occur sufficiently often in applications.

In our proposed naming scheme, the first character (S, D, C or Z) indicates the relevant Fortran data

f) Solution of triangular systems of equations:

_TRSM (SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)

Operation (B is always $m \times n$, A is triangular):

	TRANSA =‘N’	TRANSA =‘T’	TRANSA =‘C’
SIDE =‘L’	$B \leftarrow \alpha A^{-1} B$ A is triangular $m \times m$	$B \leftarrow \alpha A^{-T} B$ A is triangular $m \times m$	$B \leftarrow \alpha A^{-H} B$ A is triangular $m \times m$
SIDE =‘R’	$B \leftarrow \alpha B A^{-1}$ A is triangular $n \times n$	$B \leftarrow \alpha B A^{-T}$ A is triangular $n \times n$	$B \leftarrow \alpha B A^{-H}$ A is triangular $n \times n$

(In the real case the values ‘T’ and ‘C’ have the same meaning.)

7. Numerical Stability

Although it is intended that the Level 3 BLAS be implemented as efficiently as possible, it is essential that efficiency should not be achieved at the cost of sacrificing numerical stability. This is particularly important since the Level 3 BLAS are intended to be used as building blocks in higher-level algorithms of linear algebra. If the Level 3 BLAS themselves were unstable, they would destroy the stability of the algorithms that call them.

8. Rationale

In the design of all levels of BLAS, one of the main concerns is to keep both the calling sequences simple and the range of options limited, while at the same time maintaining sufficient functionality. This clearly implies a compromise, and a good decision is vital if the BLAS are to be accepted as a useful standard. In this section we discuss the reasoning behind some of the decisions we have made.

The three basic matrix-matrix operations were chosen because they occur in a wide range of linear algebra applications: this is consistent with the criteria used for the Level 1 and Level 2 BLAS. We have again aimed at a reasonable compromise between a much larger number of routines each performing only one type of operation (e.g., $B \leftarrow L^T B$), and a smaller number of routines with a more complicated set of options. There are in fact, in each precision, 6 real routines performing altogether 48 different combinations of options, and 9 complex routines performing altogether 81 different combinations of options. The number of routines is much smaller than in the Level 2 BLAS.

The routines that we have specified are not intended as high-level matrix algebra routines, but rather as building blocks for the construction of such routines.

d) Rank-$2k$ updates of a real or complex symmetric or complex Hermitian matrix:

_SYR2K (UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
_HER2K (UPLO, TRANS, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

Operation (C is always $n \times n$)

For _SYR2K routines (C is symmetric)

TRANS ='N'	TRANS ='T'
$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$	$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$
A and B are $n \times k$	A and B are $k \times n$

For _HER2K routines (C is Hermitian)

TRANS ='N'	TRANS ='C'
$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C$	$C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C$
A and B are $n \times k$	A and B are $k \times n$

(In the real case the values 'T' and 'C' have the same meaning. In the complex case TRANS = 'C' is not allowed in _SYR2K, and TRANS = 'T' is not allowed in _HER2K.)

e) Triangular matrix-matrix products:

_TRMM (SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB)

Operation (B is always $m \times n$, A is triangular):

	TRANSA ='N'	TRANSA ='T'	TRANSA ='C'
SIDE ='L'	$B \leftarrow \alpha AB$ A is triangular $m \times m$	$B \leftarrow \alpha A^T B$ A is triangular $m \times m$	$B \leftarrow \alpha A^H B$ A is triangular $m \times m$
SIDE ='R'	$B \leftarrow \alpha BA$ A is triangular $n \times n$	$B \leftarrow \alpha BA^T$ A is triangular $n \times n$	$B \leftarrow \alpha BA^H$ A is triangular $n \times n$

(In the real case the values 'T' and 'C' have the same meaning.)

b) Matrix-matrix products where one matrix is real or complex symmetric or complex Hermitian:

_SYMM (SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
_HEMM (SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

Operation (C is always $m \times n$, A is symmetric for the _SYMM routines, and Hermitian for the _HEMM routines):

SIDE =‘L’	SIDE =‘R’
$C \leftarrow \alpha AB + \beta C$	$C \leftarrow \alpha BA + \beta C$
A is $m \times m$	B is $m \times n$
B is $m \times n$	A is $n \times n$

c) Rank-k updates of a real or complex symmetric or complex Hermitian matrix:

_SYRK (UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)
_HERK (UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC)

Operation (C is always $n \times n$):

For _SYRK routines (C is symmetric)

TRANS =‘N’	TRANS =‘T’
$C \leftarrow \alpha AA^T + \beta C$	$C \leftarrow \alpha A^T A + \beta C$
A is $n \times k$	A is $k \times n$

For _HERK routines (C is Hermitian)

TRANS =‘N’	TRANS =‘C’
$C \leftarrow \alpha AA^H + \beta C$	$C \leftarrow \alpha A^H A + \beta C$
A is $n \times k$	A is $k \times n$

(In the real case the values ‘T’ and ‘C’ have the same meaning. In the complex case TRANS = ‘C’ is not allowed in _SYRK, and TRANS = ‘T’ is not allowed in _HERK.)

```
      COMPLEX  ALPHA
      REAL     BETA
```

For routines whose first letter is a Z:

```
   COMPLEX*16  ALPHA, BETA   or    DOUBLE COMPLEX  ALPHA, BETA
   COMPLEX*16  A(LDA,*)            DOUBLE COMPLEX  A(LDA,*)
   COMPLEX*16  B(LDB,*)            DOUBLE COMPLEX  B(LDB,*)
   COMPLEX*16  C(LDC,*)            DOUBLE COMPLEX  C(LDC,*)
```

except that for ZHERK the scalars α and β are real, so the first declaration above is replaced by

```
      DOUBLE PRECISION  ALPHA, BETA
```

and for ZHER2K α is complex and β is real, so this is replaced by

```
      COMPLEX*16       ALPHA
      DOUBLE PRECISION BETA
```

a) General matrix-matrix products:

_GEMM (TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

Operation (C is always $m \times n$):

	TRANSA =‘N’	TRANSA =‘T’	TRANSA = ‘C’
TRANSB =‘N’	$C \leftarrow \alpha AB + \beta C$ A is $m \times k$, B is $k \times n$	$C \leftarrow \alpha A^T B + \beta C$ A is $k \times m$, B is $k \times n$	$C \leftarrow \alpha A^H B + \beta C$ A is $k \times m$, B is $k \times n$
TRANSB =‘T’	$C \leftarrow \alpha AB^T + \beta C$ A is $m \times k$, B is $n \times k$	$C \leftarrow \alpha A^T B^T + \beta C$ A is $k \times m$, B is $n \times k$	$C \leftarrow \alpha A^H B^T + \beta C$ A is $k \times m$, B is $n \times k$
TRANSB = ‘C’	$C \leftarrow \alpha AB^H + \beta C$ A is $m \times k$, B is $n \times k$	$C \leftarrow \alpha A^T B^H + \beta C$ A is $k \times m$, B is $n \times k$	$C \leftarrow \alpha A^H B^H + \beta C$ A is $k \times m$, B is $n \times k$

(In the real case the values ‘T’ and ‘C’ have the same meaning.)

5. Storage Conventions

All matrices are stored conventionally in a two-dimensional array with matrix-element $a_{i,j}$ stored in array-element A(I,J); there is no provision for packed storage for symmetric, Hermitian or triangular matrices.

For symmetric and Hermitian matrices, only the upper triangle (UPLO=‘U’) or the lower triangle (UPLO=‘L’) is stored.

For triangular matrices, the argument UPLO serves to define whether the matrix is upper (UPLO=‘U’) or lower (UPLO=‘L’) triangular.

For a Hermitian matrix the imaginary parts of the diagonal elements are of course zero, and thus the imaginary parts of the corresponding Fortran array elements need not be set, but are assumed to be zero. In the HERK and HER2K routines these imaginary parts will be set to zero on return, except when $\beta = 1$ and α or $k = 0$, in which case the routines exit immediately.

6. Specification of the Level 3 BLAS

Type and dimension for variables occurring in the subroutine specifications are as follows:

```
INTEGER     M, N, K, LDA, LDB, LDC
CHARACTER*1 SIDE, UPLO, TRANSA, TRANSB, TRANS, DIAG
```

For routines whose first letter is an S:

```
REAL  ALPHA, BETA
REAL  A(LDA,*), B(LDB,*), C(LDC,*)
```

For routines whose first letter is a D:

```
DOUBLE PRECISION  ALPHA, BETA
DOUBLE PRECISION  A(LDA,*), B(LDB,*), C(LDC,*)
```

For routines whose first letter is a C:

```
COMPLEX  ALPHA, BETA
COMPLEX  A(LDA,*), B(LDB,*), C(LDC,*)
```

except that for CHERK the scalars α and β are real, so the first declaration above is replaced by

```
REAL  ALPHA, BETA
```

and for CHER2K α is complex and β is real, so this is replaced by

Value	Meaning
‘U’	Unit triangular
‘N’	Non-unit triangular

When DIAG is supplied as ‘U’, the diagonal elements are not referenced.

Thus, UPLO and DIAG have the same values and meanings as for the Level 2 BLAS; TRANSA, TRANSB and TRANS have the same values and meanings as TRANS in the Level 2 BLAS, where TRANSA and TRANSB apply to the matrices A or B, respectively.

We recommend that the equivalent lower-case characters be accepted with the same meaning, although, because they are not included in the standard Fortran character set, their use may not be supported on all systems. See Section 7 of (Dongarra *et al.*, 1988a) for further discussion.

The sizes of the matrices are determined by the arguments M, N, and K. It is permissible to call the routines with M or N = 0, in which case the routines exit immediately without referencing their matrix arguments. If M and N > 0, but K = 0, the operation reduces to $C \leftarrow \beta C$ (this applies to the GEMM, SYRK, HERK, SYR2K, and HER2K routines). The input-output matrix (B for the TR routines, C otherwise) is always $m \times n$ if rectangular, and $n \times n$ if square.

The description of the matrix consists of the array name (A, B or C) followed by the leading dimension of the array as declared in the calling (sub)program (LDA, LDB or LDC).

The scalars always have the dummy argument names ALPHA and BETA.

The following values of arguments are invalid:

Any value of the character arguments SIDE, TRANSA, TRANSB, TRANS,
UPLO, or DIAG, whose meaning is not specified.
M < 0
N < 0
K < 0
LDA < the number of rows in the matrix A.
LDB < the number of rows in the matrix B.
LDC < the number of rows in the matrix C.

If a routine is called with an invalid value for any of its arguments, then it must report the fact and terminate execution of the routine. In the model implementation (see Appendix B), each routine, on detecting an error, calls a common error handling routine XERBLA, passing to it the name of the routine and the number of the first argument which is in error. Specialized implementations may call system-specific exception-handling and diagnostic facilities.

4. Argument Conventions

We follow a convention for the argument lists similar to that for the Level 2 BLAS, with the necessary adaptations. The order of arguments is as follows:

a) Arguments specifying options
b) Arguments defining the sizes of the matrices
c) Input scalar
d) Description of input matrices
e) Input scalar (associated with input-output matrix)
f) Description of the input-output matrix

Note that not every category is present in each of the routines.

The arguments that specify options are character arguments with the names SIDE, TRANSA, TRANSB, TRANS, UPLO and DIAG.

SIDE is used by the routines as follows:

Value	Meaning
‘L’	Multiply general matrix by symmetric, Hermitian or triangular matrix on the left.
‘R’	Multiply general matrix by symmetric, Hermitian or triangular matrix on the right.

TRANSA, TRANSB and TRANS are used by the routines as follows:

Value	Meaning
‘N’	Operate with the matrix.
‘T’	Operate with the transpose of the matrix.
‘C’	Operate with the conjugate transpose of the matrix.

In the real case the values ‘T’ and ‘C’ have the same meaning.

UPLO is used by the Hermitian, symmetric, and triangular matrix routines to specify whether the upper or lower triangle is being referenced as follows:

Value	Meaning
‘U’	Upper triangle
‘L’	Lower triangle

DIAG is used by the triangular matrix routines to specify whether the matrix is unit triangular, as follows:

GE	All matrices are general rectangular
HE	One of the matrices is Hermitian
SY	One of the matrices is symmetric
TR	One of the matrices is triangular

The fourth and fifth, and in one case sixth, characters in the name denote the type of operation, as follows:

MM	Matrix-matrix product
RK	Rank-k update of a symmetric or Hermitian matrix
R2K	Rank-$2k$ update of a symmetric or Hermitian matrix
SM	Solve a system of linear equations for a matrix of right-hand sides

The permitted combinations are indicated in Table 3.1. In the first column, under *complex*, the initial C may be replaced by Z. In the second column, under *real*, the initial S may be replaced by D. See Appendix B for the full subroutine calling sequences.

The collection of routines can be thought of as being divided into four separate parts: *real*, *double precision*, *complex*, and *complex*16*. The routines can be written in ANSI standard Fortran 77, with the exception of the routines that use COMPLEX*16 variables. These routines are included for completeness and for their usefulness on those systems that support this data type; but because they do not conform to the Fortran standard, they may not be available on all machines.

The combinations provided are:

Table 3.1

Complex	Real	MM	RK	R2K	SM
CGE	SGE	*			
CSY	SSY	*	*	*	
CHE		*	*	*	
CTR	STR	*			*

However, note that rank-k updates of general matrices are provided by the GEMM routines.

c) Multiplying a matrix by a triangular matrix:

$$B \leftarrow \alpha TB$$

$$B \leftarrow \alpha T^T B$$

$$B \leftarrow \alpha BT$$

$$B \leftarrow \alpha BT^T$$

d) Solving triangular systems of equations with multiple right-hand sides:

$$B \leftarrow \alpha T^{-1} B$$

$$B \leftarrow \alpha T^{-T} B$$

$$B \leftarrow \alpha BT^{-1}$$

$$B \leftarrow \alpha BT^{-T}$$

Here α and β are scalars, A, B and C are rectangular matrices (in some cases square and symmetric), and T is an upper or lower triangular matrix (and non-singular in (d)).

Analogous operations are proposed in complex arithmetic: conjugate transposition is specified as well as simple transposition, and additional operations in (b) provide for updates of a Hermitian matrix as follows:

$$C \leftarrow \alpha AA^H + \beta C$$

$$C \leftarrow \alpha A^H A + \beta C$$

with α and β real, and

$$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C$$

$$C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C$$

with β real.

3. Naming Conventions

The name of a Level 3 BLAS routine follows the conventions of the Level 2 BLAS. The first character in the name denotes the Fortran data type of the matrix, as follows:

S	REAL
D	DOUBLE PRECISION
C	COMPLEX
Z	COMPLEX*16 or DOUBLE COMPLEX (if available)

Characters two and three in the name refer to the kind of matrix involved, as follows:

The details of this paper are concerned specifically with defining a set of subroutines for use in Fortran 77 programs. However, the essential features could be adapted to other programming languages.

In a companion paper (Dongarra *et al.*, 1987) we present a model implementation of the Level 3 BLAS in Fortran 77 (extended to include a COMPLEX*16 data type), and also a set of rigorous test programs.

2. Scope of the Level 3 BLAS

The routines described here have been derived in a fairly obvious manner from some of the Level 2 BLAS, by replacing the vectors x and y with matrices B and C. The advantage in keeping the design of the software as consistent as possible with that of the Level 2 BLAS is that it will be easier for users to remember the calling sequences and parameter conventions.

In real arithmetic the operations proposed for the Level 3 BLAS have the following forms.

a) Matrix-matrix products

$$C \leftarrow \alpha AB + \beta C$$

$$C \leftarrow \alpha A^T B + \beta C$$

$$C \leftarrow \alpha AB^T + \beta C$$

$$C \leftarrow \alpha A^T B^T + \beta C$$

Note that these operations are more accurately described as matrix-matrix multiply-and-add operations; they include rank-k updates of a general matrix.

b) Rank-k and rank-$2k$ updates of a symmetric matrix:

$$C \leftarrow \alpha AA^T + \beta C$$

$$C \leftarrow \alpha A^T A + \beta C$$

$$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C$$

$$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C$$

Lewis, 1985).

An extended set of Fortran BLAS aimed at matrix-vector operations (Level 2 BLAS) were subsequently proposed by Dongarra, Du Croz, Hammarling and Hanson (Dongarra *et al.*, 1988a, 1988b). The Level 2 BLAS were proposed in order to support the development of software that would be both portable and efficient across a wide range of machine architectures, with emphasis on vector-processing machines.

Many of the frequently used algorithms of numerical linear algebra can be coded so that the bulk of the computation is performed by calls to Level 2 BLAS routines; efficiency can then be obtained by utilizing tailored implementations of the Level 2 BLAS routines. On vector-processing machines one of the aims of such implementations is to keep the vector lengths as long as possible, and in most algorithms the results are computed one vector (row or column) at a time. In addition, on vector register machines performance is increased by reusing the results of a vector register, and not storing the vector back into memory.

Unfortunately, this approach to software construction is often not well suited to computers with a hierarchy of memory (such as global memory, cache or local memory, and vector registers) and true parallel-processing computers. (For a description of many advanced-computer architectures see (Dongarra and Duff, 1987).) For those architectures it is often preferable to partition the matrix or matrices into blocks and to perform the computation by matrix-matrix operations on the blocks. By organizing the computation in this fashion we provide for full reuse of data while the block is held in the cache or local memory. This approach avoids excessive movement of data to and from memory and gives a *surface-to-volume* effect for the ratio of operations to data movement. In addition, on architectures that provide for parallel processing, parallelism can be exploited in two ways: (1) operations on distinct blocks may be performed in parallel; and (2) within the operations on each block, scalar or vector operations may be performed in parallel.

The Level 3 BLAS specified here are targeted at the matrix-matrix operations required for these purposes. If the vectors and matrices involved are of order n, then the original BLAS include operations that are of order $O(n)$, the extended or Level 2 BLAS provide operations of order $O(n^2)$, and the routines specified here provide operations of order $O(n^3)$; hence our use of the term Level 3 BLAS. Section 9 illustrates how an algorithm for Cholesky factorization can be implemented by calls to the Level 3 BLAS. Such implementations can, we believe, be portable across a wide variety of vector and parallel computers and also efficient (assuming that efficient implementations of the Level 3 BLAS are available). There is certainly considerable evidence for the efficiency of such algorithms on particular machines (see, for example, the references cited in Section 9); the question of portability has been much less studied but, by proposing a standard set of building blocks, we hope, to encourage research into this aspect.

The scope of the Level 3 BLAS is intentionally limited. No routines are included for matrix factorization, for example; these are currently provided by LINPACK (Dongarra *et al.*, 1979), and will be included in a new linear algebra package currently under development, which will use block algorithms and calls to Level 3 BLAS wherever possible (Demmel *et al.*, 1987). Nor are the Level 3 BLAS intended to be a comprehensive set of routines for elementary matrix algebra. They are intended primarily for software developers and to a lesser extent for experienced applications programmers.

A Set of Level 3

Basic Linear Algebra Subprograms

Jack Dongarra *

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439-4801

Jeremy Du Croz

Numerical Algorithms Group Ltd.
Wilkinson House
Jordan Hill Road
Oxford OX2 8DR

Iain Duff

Computer Science and Systems Division
Harwell Laboratory
Oxfordshire OX11 ORA

Sven Hammarling

Numerical Algorithms Group Ltd.
Wilkinson House
Jordan Hill Road
Oxford OX2 8DR

Abstract — This paper describes a set of Level 3 Basic Linear Algebra Subprograms (Level 3 BLAS). The Level 3 BLAS are targeted at matrix-matrix operations with the aim of providing more efficient, but portable, implementations of algorithms on high-performance computers, especially those with hierarchical memory and parallel processing capability.

1. Introduction

In 1973 Hanson, Krogh, and Lawson wrote an article in the SIGNUM Newsletter (Vol. 8, no. 4, page 16) describing the advantages of adopting a set of basic routines for problems in linear algebra. The original basic linear algebra subprograms, now commonly referred to as the BLAS and fully described in (Lawson *et al.*, 1979a and Lawson *et al.*, 1979b), have been very successful and have been used in a wide range of software including LINPACK (Dongarra *et al.*, 1979) and many of the algorithms published by the ACM Transactions on Mathematical Software. In particular, they are an aid to clarity, portability, modularity and maintenance of software; and they have become a *de facto* standard for the elementary vector operations. An excellent discussion of the *raison d' être* of the BLAS is given in (Dodson and

* Work supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under Contract W-31-109-Eng-38.

Typeset on January 26, 1989.

.

Abstract

This paper describes a set of Level 3 Basic Linear Algebra Subprograms (Level 3 BLAS). The Level 3 BLAS are targeted at matrix-matrix operations with the aim of providing more efficient, but portable, implementations of algorithms on high-performance computers, especially those with hierarchical memory and parallel processing capability.

.

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

A Set of Level 3 Basic Linear Algebra Subprograms

Jack Dongarra, Jeremy Du Croz, Iain Duff and Sven Hammarling

Mathematics and Computer Science Division

Preprint No. 1

August 1988