Appendix C

This appendix contains the 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,          TRANSB,          M, N,    ALPHA, A, LDA, B, LDB, BETA, C, LDC)
_HEMM (SIDE, UPLO,          TRANSB,          M, N,    ALPHA, A, LDA, B, LDB, BETA, C, LDC)
_SYRK (      UPLO, TRANSA,                      N, K, ALPHA, A, LDA,          BETA, C, LDC)
_HERK (      UPLO, TRANSA,                      N, K, ALPHA, A, LDA,          BETA, C, LDC)
_TRMM (SIDE, UPLO, TRANSA,          DIAG,    M, N,           A, LDA, B, LDB)
_TRSM (SIDE, UPLO, TRANSA,          DIAG,    M, N,           A, LDA, B, LDB)
```

Appendix B

A model implementation, in Fortran 77, of the Level 3 BLAS routines, together with test programs and timing programs, is available via netlib. To obtain the Level 3 BLAS from netlib send an electronic mail message to
netlib@anl-mcs.arpa or research!netlib. The message should be of the form:

send sblas3 from blas3

which would return the single precision real version of the Level 3 BLAS. A message of the form:

send index from blas3

would return an index of information available. If you do not have access to electronic mail then a copy of the routines can be obtained by sending a request, along with a magnetic tape, to any of the authors.

Note that the model implementation is not claimed to be efficient on any particular types of architecture (the current version uses repeated calls to Level 2 BLAS). Note also that after the Level 3 BLAS have been exposed to public review, their specification is liable to be changed.

```
      SUBROUTINE DPLU(M,N,A,LDA,IPIV,INFO)
*
*     Computes an LU factorization of an m-by-n matrix A, using partial
*     pivoting with row interchanges.
*
      INTEGER           INFO, LDA, M, N
      DOUBLE PRECISION A(LDA,*)
      INTEGER           IPIV(N)
      DOUBLE PRECISION T
      INTEGER           I, IP, J, JP
      INTEGER           IDAMAX
      EXTERNAL          IDAMAX
      EXTERNAL          DGEMV, DSCAL, DSWAP, DTRSV
*
      INFO = 0
      DO 40 J = 1, N
*
*        Apply previous interchanges to jth column.
*
         DO 10 I = 1, J - 1
            IP = IPIV(I)
            T = A(I,J)
            A(I,J) = A(IP,J)
            A(IP,J) = T
   10    CONTINUE
*
*        Compute elements 1:j-1 of j-th column.
*
         CALL DTRSV('Lower','No transpose','Unit',J-1,A(1,1),LDA,A(1,J),
     $              1)
*
*        Update elements j:n of j-th column.
*
         CALL DGEMV('No transpose',M-J+1,J-1,-1.0D0,A(J,1),LDA,A(1,J),1,
     $              1.0D0,A(J,J),1)
*
*        Find pivot and test for singularity.
*
         JP = J - 1 + IDAMAX(M-J+1,A(J,J),1)
         IPIV(J) = JP
         IF (A(JP,J).EQ.0.0D0) GO TO 50
*
*        Apply interchange to columns 1:j.
*
         IF (JP.NE.J) CALL DSWAP(J,A(J,1),LDA,A(JP,1),LDA)
*
*        Compute elements j+1:m of j-th column.
*
         IF (J.LT.M) CALL DSCAL(M-J,1.0D0/A(J,J),A(J+1,J),1)
   40 CONTINUE
      RETURN
*
   50 INFO = J
      RETURN
      END
```

```
      SUBROUTINE DPLUB(M,N,A,LDA,IPIV,INFO,NB)
*
*     Computes an LU factorization of an m-by-n matrix A, using partial
*     pivoting with row interchanges.
*
      INTEGER           INFO, LDA, M, N, NB
      DOUBLE PRECISION A(LDA,*)
      INTEGER           IPIV(N)
      INTEGER           I, IP, J, JB
      EXTERNAL          DGEMM, DPLU, DSWAP, DTRSM
*
      INFO = 0
      DO 40 J = 1, N, NB
         JB = MIN(N-J+1,NB)
*
*        Apply previous interchanges to current block.
*
         DO 10 I = 1, J - 1
            IP = IPIV(I)
            IF (IP.NE.I) CALL DSWAP(JB,A(I,J),LDA,A(IP,J),LDA)
   10    CONTINUE
*
*        Compute superdiagonal block of U.
*
         CALL DTRSM('Left','Lower','No transpose','Unit',J-1,JB,A(1,1),
     $              LDA,A(1,J),LDA)
*
*        Update diagonal and subdiagonal blocks.
*
         CALL DGEMM('No transpose','No transpose',M-J+1,JB,J-1,-1.0D0,
     $              A(J,1),LDA,A(1,J),LDA,1.0D0,A(J,J),LDA)
*
*        Factorize diagonal and subdiagonal blocks and test for
*        singularity.
*
         CALL DPLU(M-J+1,JB,A(J,J),LDA,IPIV(J),INFO)
         DO 20 I = J, J + JB - 1
            IPIV(I) = J - 1 + IPIV(I)
   20    CONTINUE
         IF (INFO.NE.0) GO TO 50
*
*        Apply interchanges to previous blocks.
*
         DO 30 I = J, J + JB - 1
            IP = IPIV(I)
            IF (IP.NE.I) CALL DSWAP(J-1,A(I,1),LDA,A(IP,1),LDA)
   30    CONTINUE
   40 CONTINUE
      RETURN
*
   50 INFO = INFO + J - 1
      RETURN
      END
```

```
      SUBROUTINE DLLT(N,A,LDA,INFO)
*
*     Computes an L*L**T factorization of a symmetric positive-definite
*     matrix A.
*
      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
```

```
      SUBROUTINE DLLTB(N,A,LDA,INFO,NB)
*
*     Computes an L*L**T factorization of a symmetric positive-definite
*     matrix A.
*
      INTEGER           INFO, LDA, N, NB
      DOUBLE PRECISION A(LDA,*)
      INTEGER           J, JB
      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,A(J,J),LDA,A(J+JB,J),LDA)
         END IF
   10 CONTINUE
      RETURN
*
   20 INFO = INFO + J - 1
      RETURN
      END
```

Appendix A

```
      SUBROUTINE DMM(M,N,K,A,LDA,B,LDB,C,LDC)
*
*     This routine performs a matrix multiply of the form C:= A*B.
*     The approach is based on a block scheme which fixes a block of
*     the matrix A. The block size is set by NR and NC.
*
*     The value of NR may be determined, say, by the vector register
*     length, and the value of NC by the size of the cache or local
*     memory.
*
      INTEGER           NR, NC
      PARAMETER         (NR=64,NC=32)
      INTEGER           K, LDA, LDB, LDC, M, N
      DOUBLE PRECISION A(LDA,*), B(LDB,*), C(LDC,*)
      DOUBLE PRECISION BETA
      INTEGER           I, IEND, J, JEND
      EXTERNAL          DGEMM
*
      DO 20 J = 1, N, NC
         IF (J.EQ.1) THEN
            BETA = 0.0D0
         ELSE
            BETA = 1.0D0
         END IF
         JEND = MIN(J+NC-1,N)
         DO 10 I = 1, M, NR
            IEND = MIN(I+NR-1,M)
            CALL DGEMM('No transpose','No transpose',IEND-I+1,K,
     $                 JEND-J+1,1.0D0,A(I,J),LDA,B(J,1),LDB,BETA,C(I,1),
     $                 LDC)
   10    CONTINUE
   20 CONTINUE
*
      RETURN
      END
```

Algebra Subprograms,'' Argonne National Laboratory Report, ANL-MCS-TM-41 (Revision 3), November 1986.

J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, ''An Extended Set of Fortran Basic Linear Algebra Subprograms: Model Implementation and Test Programs,'' Argonne National Laboratory Report, ANL-MCS-TM-81, November, 1986.

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

J. DuCroz, S. Nugent, J. Reid, and D. Taylor, ''Solving Large Full Sets of Linear Equations in a Paged Virtual Store,'' *TOMS* , vol. 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, 1986.

A. George and H. Rashwan, ''Auxiliary Storage Methods for Solving Finite Element Systems,'' *SIAM SISSC*, vol. 6, 1981.

IBM, ''Engineering and Scientific Subroutine Library,'' *IBM*, vol. Program Number: 5668-863.

C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, ''Basic Linear Algebra Subprograms for Fortran Usage,'' *ACM Transactions on Mathematical Software*, vol. 5, 1979.

C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, ''Algorithm 539: Basic Linear Algebra Subprograms for Fortran Usage,'' *ACM Transactions on Mathematical Software*, vol. 5, 1979.

A.C. McKellar and E.G. Coffman Jr., ''Organizing Matrices and Matrix Operations for Paged Memory Systems,'' *CACM*, vol. 12,3, 1969.

R. Schreiber, ''Module Design Specification (Version 1.0),'' *SAXPY Computer Corporation, 255 San Geronimo Way, Sunnyvale, CA 94086.*, 1986.

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.

References

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

C. Bischof and C. Van Loan, ‘‘The WY Representation for Products of Householder Matrices,’’ *SIAM SISSC*, vol. 8, 2, March, 1987.

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

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.

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

A.K. Dave and I.S. Duff, ‘‘Sparse Matrix Calculations on the CRAY-2,’’ AERE Harwell Report CSS 197 (to appear Parallel Computing).

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

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

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

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

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

J.J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson, ‘‘An Extended Set of Fortran Basic Linear

In Appendix A we present a code DPLUB for block *LU*-factorization, and a lower-level routine DPLU, which is called by DPLUB and has been designed to have the same structure. DPLUB calls Level 3 BLAS whereas DPLU calls Level 2 and Level 1 BLAS. Because DPLU needs to operate on a rectangular matrix (so that the factor L is lower trapezoidal rather than triangular), DPLUB has been written with the same specification.

8.4 *QR*-Factorization

We do not discuss this in detail here, but draw attention to the method of (Bischof and Van Loan, 1987), which was designed specifically to exploit efficient matrix-matrix multiplication.

hence:

$$U_{12} = L_{11}^{-1}A_{12}$$

$$L_{22}U_{22} = A_{22} - L_{21}U_{12}$$

$$L_{32} = (A_{32} - L_{31}U_{12})U_{22}^{-1}$$

The computation of one block-column requires the following operations:

1. compute the superdiagonal block of U:

$$U_{12} \leftarrow L_{11}^{-1}A_{12}$$

2. update the diagonal and subdiagonal blocks:

$$\begin{pmatrix} A'_{22} \\ A'_{32} \end{pmatrix} \leftarrow \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} U_{12}$$

3. compute the LU factorization of the diagonal block:

$$A'_{22} \rightarrow L_{22}U_{22}$$

4. compute the subdiagonal block of L:

$$L_{32} \leftarrow A'_{32}U_{22}^{-1}$$

Of course we want to introduce partial pivoting for numerical stability. This means that at step 3 it is not sufficient to consider only the diagonal block A'_{22}: we need to involve the sub-diagonal block A'_{32} as well. Steps 3 and 4 must be combined into the following:

3′. compute the LU factorization with interchanges of the diagonal and subdiagonal blocks:

$$\begin{pmatrix} A'_{22} \\ A'_{32} \end{pmatrix} \rightarrow P \begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22}$$

Here P is a permutation matrix which defines the row interchanges. The row interchanges must also be applied to the other columns of A. In our example code, we apply the interchanges immediately to all preceding block-columns, but delay applying them to subsequent block-columns until they come to be processed. This is similar to the block-column algorithm of (DuCroz *et al.*, 1981): the difference is that their algorithm combined steps 1 and 2, and did not use any BLAS routines.

4. compute the subdiagonal block of L:

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

Operations 1, 3, and 4 can be performed by calls to DSYRK, DGEMM, and DTRSM respectively (in double precision). Operation 2 requires a separate lower-level Cholesky factorization routine (since current standard Fortran forbids recursion).

In Appendix A we give 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. 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

8.3 LU-Factorization

Again, the details of the algorithm may be derived by partitioning the matrices. If we ignore row or column interchanges we have simply:

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{pmatrix} =$$

$$\begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} & L_{11}U_{13} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} & L_{21}U_{13} + L_{22}U_{23} \\ L_{31}U_{11} & L_{31}U_{12} + L_{32}U_{22} & L_{31}U_{13} + L_{32}U_{23} + L_{33}U_{33} \end{pmatrix}$$

This gives

$$A_{12} = L_{11}U_{12}$$

$$A_{22} = L_{21}U_{12} + L_{22}U_{22}$$

$$A_{32} = L_{31}U_{12} + L_{32}U_{22}$$

8.2 Cholesky Factorization

We assume that we are given a positive-definite symmetric matrix A whose lower triangle is stored in the lower triangle of a 2-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, which 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:

$$A_{22}^{'} \rightarrow L_{22}L_{22}^T$$

3. update the subdiagonal block:

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

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 *et al.*, 1986), (IBM, 1986), (Robert and Sguazzero, 1987), (Bischof and Van Loan, 1987), (Bucher and Jordan, 1984), and (Schreiber, 1986), (Calahan, 1986), (Dongarra and Sorensen, 1986).

Here we illustrate how the proposed Level 3 BLAS routines can be used to implement two fundamental algorithms of numerical linear algebra: Cholesky factorization, and LU factorization, together with a simple illustration of use in a high level matrix-multiply routine.

For the factorization routines the strategy in each case is to compute at each stage a block of consecutive columns of the result. The size of the block is a parameter, *nb*, which may be varied to suit the size of the problem and the architecture of the machine. (For transportable software, we need some means of determining *nb* within the routine, rather than requiring it to be passed as a parameter, but we set that issue aside here.)

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 *et al.*, 1984) can easily be extended to algorithms which work by blocks. We have chosen an organization which works by columns rather than rows, and which 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 parameter *nb* 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*×*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.

A similar strategy is used for the matrix multiplication and we present that case first.

8.1 Matrix Multiply

Given matrices A and B we wish to compute the product $C \leftarrow A*B$. The matrices A, B, and C are of dimension $m \times k$, $k \times n$ and $m \times n$ respectively. We can partition the matrices so that

$$\begin{pmatrix} C_1 \\ C_2 \\ . \\ C_p \end{pmatrix} = \begin{pmatrix} & A_{ij} & \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ . \\ B_p \end{pmatrix}.$$

The partitions for the matrices A_{ij}, B_j, and C_i are of dimension $nr \times nc$, $nc \times n$ and $nr \times n$ respectively. The update of one block row of the matrix C follows by computing

$$C_i \leftarrow C_i + A_{ij}B_j.$$

This algorithm is implemented in the subroutine DMM in Appendix A.

likely to be so useful in the type of application we are aiming at. Also such storage is required much less with large memory machines available today and, in addition, we also wish to keep the set of routines as small as possible.

Again in this type of application, we are not aware of a need for simple transposition of complex matrices; this contrasts with the Level 2 BLAS which provide for computing $A^T x$ as a means of achieving $x^T A$, with complex x and A.

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

As with the Level 2 BLAS no check has been included for singularity, or near singularity, in the triangular equation solving routines. 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 conventions, and maintain consistency with, the Level 2 BLAS; however, we have deliberately departed from this 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 update routines, we have consistency between the MM and RK routines. Finally, as mentioned above, we have not allowed transposition in the complex case for the Level 3 BLAS.

Finally we would welcome suggestions for a better way of distinguishing between the two operations performed by SYRK/HERK than our current use of TRANSA.

8. 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), (DuCroz *et al.*, 1981), (Calahan, 1986), and (Dave and Duff, 1986). Both the NAG and the IMSL libraries include such algorithms (F01BTF and F01BXF in NAG; LQEIF and LQEOF 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, 1986)).

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

7. 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 proposal, we have had to make several decisions of this kind, and we are anxious that our reasoning is questioned by as wide a community as possible. Therefore, in this Section, we discuss some of the functionality which we have omitted, and some which we were a little doubtful about including (for example, features which were included without unanimous support from all authors!).

As with the previous BLAS, the three basic matrix-matrix operations were chosen because they occur in a wide range of linear algebra applications. 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 data type, 5 routines performing altogether 48 different operations. The number of routines is much smaller than in the Level 2 BLAS.

The routines that we have proposed are not intended as high level matrix algebra routines, but rather as tools for the construction of such routines. To emphasize the point we have included an example of a high level matrix multiply routine, in Appendix A, based upon the Level 3 BLAS routine DGEMM.

In each case, where appropriate, we include operations involving a matrix and its transpose. 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 allow pre- or post-multiplication of a general matrix by a symmetric or Hermitian matrix, and can devise instances when it is required. However they do not occur frequently, and we would welcome comments on the utility of this operation.

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 proposed 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

For complex matrices

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

e) Solution of triangular systems of equations:

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

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

For real matrices

	TRANSA ='N'	TRANSA ='T' or 'C'
SIDE ='L'	$B \leftarrow A^{-1}B$	$B \leftarrow A^{-T}B$
	A is triangular $m \times m$	A is triangular $m \times m$
SIDE ='R'	$B \leftarrow BA^{-1}$	$B \leftarrow BA^{-T}$
	A is triangular $n \times n$	A is triangular $n \times n$

For complex matrices

	TRANSA ='N'	TRANSA ='C'
SIDE ='L'	$B \leftarrow A^{-1}B$	$B \leftarrow \overline{A}^{-T}B$
	A is triangular $m \times m$	A is triangular $m \times m$
SIDE ='R'	$B \leftarrow BA^{-1}$	$B \leftarrow B\overline{A}^{-T}$
	A is triangular $n \times n$	A is triangular $n \times n$

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

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

Operation (C is always $n \times n$ symmetric or Hermitian):

For real matrices

TRANSA ='N'	TRANSA ='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 complex matrices

TRANSA ='N'	TRANSA ='C'
$C \leftarrow \alpha A\bar{A}^T + \beta C$	$C \leftarrow \alpha \bar{A}^T A + \beta C$
A is $n \times k$	A is $k \times n$

d) Triangular matrix-matrix products:

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

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

For real matrices

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

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

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

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

For real matrices

	TRANSB ='N'	TRANSB ='T' or 'C'
SIDE ='L'	$C \leftarrow \alpha AB + \beta C$ A is symmetric $m \times m$ B is $m \times n$	$C \leftarrow \alpha AB^T + \beta C$ A is symmetric $m \times m$ B is $n \times m$
SIDE ='R'	$C \leftarrow \alpha BA + \beta C$ B is $m \times n$ A is symmetric $n \times n$	$C \leftarrow \alpha B^T A + \beta C$ B is $n \times m$ A is symmetric $n \times n$

For complex matrices

	TRANSB ='N'	TRANSB ='C'
SIDE ='L'	$C \leftarrow \alpha AB + \beta C$ A is Hermitian $m \times m$ B is $m \times n$	$C \leftarrow \alpha A\overline{B}^T + \beta C$ A is Hermitian $m \times m$ B is $n \times m$
SIDE ='R'	$C \leftarrow \alpha BA + \beta C$ B is $m \times n$ A is Hermitian $n \times n$	$C \leftarrow \alpha \overline{B}^T A + \beta C$ B is $n \times m$ A is Hermitian $n \times n$

a) General matrix-matrix products:

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

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

For real matrices

	TRANSA ='N'	TRANSA ='T' or '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 $k \times n$
TRANSB ='T' or 'C'	$C \leftarrow \alpha AB^T + \beta C$ 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 $n \times k$

For complex matrices

	TRANSA ='N'	TRANSA = 'C'
TRANSB ='N'	$C \leftarrow \alpha AB + \beta C$ A is $m \times k$, B is $k \times n$	$C \leftarrow \alpha \overline{A}^T B + \beta C$ A is $k \times m$, B is $k \times n$
TRANSB = 'C'	$C \leftarrow \alpha A\overline{B}^T + \beta C$ A is $m \times k$, B is $n \times k$	$C \leftarrow \alpha \overline{A}^T \overline{B}^T + \beta C$ A is $k \times m$, B is $n \times k$

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, 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 that the first declaration above is replaced by:

```
REAL   ALPHA, BETA
```

For routines whose first letter is Z:

```
COMPLEX*16  ALPHA, BETA 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 that the first declaration above is replaced by:

```
DOUBLE PRECISION   ALPHA, BETA
```

matrix arguments. If M and N > 0, but K = 0, the operation reduces to $C \leftarrow \beta C$ (this applies to the GEMM, SYRK and HERK 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,
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.

5. Storage Conventions

All matrices are stored conventionally in a 2-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 RK 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.

SIDE is used by the routines as follows:

Value	Meaning
'L'	Multiply general matrix by symmetric or triangular matrix on the left.
'R'	Multiply general matrix by symmetric or triangular matrix on the right.

TRANSA, and TRANSB 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 and in the complex case the value 'T' is not allowed.

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 or not the matrix is unit triangular, as follows:

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 and TRANSB have the same values and meanings as TRANS, where TRANSA and TRANSB apply to the matrices A or B, respectively, except that in the complex routines transposition is not allowed.

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.*, 1986a) 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

initial C may be replaced by Z. In the second column, under *real*, the initial S may be replaced by D. See Appendix C 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 which 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	SM
CGE	SGE	*		
CHE	SSY	*	*	
CTR	STR	*		*

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

4. Argument Conventions

We follow a similar convention for the argument lists 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, UPLO and DIAG.

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

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

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

$B \leftarrow BT^{-1}$

$B \leftarrow BT^{-T}$

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 instead of simple transposition and in (b) C is Hermitian and α and β are 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:

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 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
SM	Solve a system of linear equations for a matrix of right-hand sides

The suggested combinations are indicated in Table 3.1 below. In the first column, under *complex*, the

extent for experienced applications programmers.

The details of this proposal 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.

2. Scope of the Level 3 BLAS

The routines proposed 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 updates of a symmetric matrix:

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

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

c) Multiplying a matrix by a triangular matrix:

$$B \leftarrow TB$$

$$B \leftarrow T^T B$$

$$B \leftarrow BT$$

$$B \leftarrow BT^T$$

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.*, 1986a), (Dongarra *et al.*, 1986b). 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 proposed 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 our current proposal provides operations of order $O(n^3)$; hence our use of the term Level 3 BLAS. Section 8 illustrates how some common algorithms can be implemented by calls to the proposed routines. 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 quoted in Section 8); the question of portability has been much less studied but we hope, by proposing a standard set of building blocks, to encourage research into this aspect.

The scope of this proposal is limited. It does not include any routines for matrix factorization; these are covered by LINPACK (Dongarra *et al.*, 1976) and we continue to advocate the use of the LINPACK calling sequences as a standard interface, even if the details of the underlying algorithm are modified to suit particular architectures. Nor does the proposal aim to provide a comprehensive set of routines for elementary matrix algebra. It is intended primarily for software developers and to a lesser

A PROPOSAL FOR A SET OF LEVEL 3 BASIC LINEAR ALGEBRA SUBPROGRAMS

Jack Dongarra[†]

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439-4844

Jeremy Du Croz

Numerical Algorithms Group Ltd.
NAG Central Office, Mayfield House
256 Banbury Road, Oxford OX2 7DE

Iain Duff

Computer Science and Systems Division
Harwell Laboratory
Oxfordshire OX11 ORA

Sven Hammarling

Numerical Algorithms Group Ltd.
NAG Central Office, Mayfield House
256 Banbury Road, Oxford OX2 7DE

Abstract — This paper describes a proposal for 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), (Lawson *et al.*, 1979b), have been very successful and have been used in a wide range of software including LINPACK (Dongarra *et al.*, 1976) 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 Lewis, 1986).

[†] 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 November 16, 1987.

.

Abstract

This paper describes a proposal for 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 PROPOSAL FOR A SET OF LEVEL 3 BASIC LINEAR ALGEBRA SUBPROGRAMS

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

Mathematics and Computer Science Division

Technical Memorandum No. 88

April 1987