Science and Engineering Research Council
Rutherford Appleton Laboratory
Chilton DIDCOT Oxon OX11 0QX

RAL-92-087

A Proposal for User Level Sparse BLAS

I Duff M Marrone and G Radicati

December 1992

RAL 92-087

A proposal for user level sparse BLAS *

SPARKER Working Note # 1

Iain S. Duff
Michele Marrone and Giuseppe Radicati †

December 23 1992

Abstract

This paper proposes a set of Level 3 Basic Linear Algebra Subprograms and associated kernels for sparse matrices. We discuss the design, implementation and use of subprograms for the multiplication of a full matrix by a sparse one and for the solution of sparse triangular systems with one or more (full) right-hand sides. We include a routine for checking the input data, generating a new sparse data structure from that input, and scaling a sparse matrix. The new data structure for the transformation can be specified by the user or can be chosen automatically by vendors to be efficient on their machine. We also include routines for permuting the columns of a sparse matrix and one for permuting the rows of a full matrix. A major aim of this exercise is to establish standards to enable efficient, and portable, implementations of iterative algorithms for sparse matrices on high-performance computers. We have designed the routines so that the developer of mathematical software need not be at all concerned with the complexities of the various data structures used for sparse matrices. We have kept the interface and suite of codes as simple as possible while at the same time including sufficient functionality to cover most of the requirements of iterative solvers, and sufficient flexibility to cover most sparse matrix data structures. This document constitutes a proposal for standards in the above areas and the authors solicit comments and suggestions on this draft. Fortran code implementing this proposal is available by anonymous ftp from CERFACS. This proposal is intended to be complementary to the paper "A proposal for a Sparse BLAS Toolkit" by Michael Heroux.

Keywords

sparse matrices, sparse data structures, high-performance computing, programming standards, sparse BLAS, standards for iterative methods

AMS(MOS) subject classifications: 65F10, 65F50.

---

*also appeared as Report TR/PA/92/85, CERFACS, 42 Ave G. Coriolis, 31057 Toulouse Cedex, France

†IBM, Cagliari, Italy

1 Preamble

In developing programs for sparse matrices, the choice of the data structure used to represent the non-zero coefficients of the matrix plays a crucial role. There are several contrasting issues that guide the choice of data structure. Ordinarily the data structure is chosen to limit the number of zeros stored and to avoid unnecessary calculations with zero values during subsequent numerical calculations. Additionally, the data structure must allow the software developer to easily take advantage of any regularity present in the sparsity pattern. Finally the data structure may be chosen so that the software can take advantage of hardware features, such as vector registers or parallel processing capabilities. As a consequence of these somewhat contrasting requirements, a great many different data structures are used for sparse matrices.

Standard computational kernels have been proposed for basic linear algebra operations on full matrices. These Basic Linear Algebra Subprograms include routines for vector operations such as a scalar product (Level 1 BLAS, Lawson, Hanson, Kincaid, and Krogh [17]), for matrix-vector operations such as the product of a vector by a matrix (Level 2 BLAS, Dongarra, Du Croz, Hammarling, and Hanson [8]) and for matrix-matrix operations such as the multiplication of full matrices (Level 3 BLAS, Dongarra, Du Croz, Duff, and Hammarling [9]). All these BLAS are now widely used in the development of software for linear algebra for dense problems. In all these cases, because the matrices involved are stored as full matrices, there are only a very limited number of different forms, each of which has a natural storage scheme: the matrices may be general, symmetric, triangular, or banded. It has been natural to develop different versions of an algorithm for the various forms. In the case of sparse matrices, this is not practical because of the variety of data structures. There are, however, instances when the full BLAS can be used on subproblems within an implementation of a code for sparse matrices (see for example Duff [11] and Amestoy, Daydé, and Duff [2]). Indeed much of the power of frontal and multifrontal techniques for the solution of sparse equations comes from these kernels.

Dodson, Grimes, and Lewis [7] have proposed a standard for extending the Level 1 BLAS to the sparse case. Routines are included for gather and scatter, saxpy and sdot, and application of a Givens rotation. While they are a useful extension, they suffer from the same problem as the Level 1 BLAS in the full case, namely that the data-access requirements are of the same order as the arithmetic and so high efficiency is not obtained on most high-performance computers. More recently there has been some concern over standardizing interfaces to iterative solvers for sparse systems (Ashby and Seager [4]) and it is primarily in this context that our current proposal is oriented. We feel that, particularly with the recent rapid growth in the use of high-performance computers, it is most timely to establish standards to achieve high performance without sacrifice of portability.

Standard computational kernels will facilitate the development of programs that are easy to maintain, easier to port from one machine to another, and yield efficient performance on a wide range of computers. They should hide the programming complexities from most developers, thus separating implementation details connected with performance and data structures from those connected with the algorithm. On the other hand, they provide the flexibility to allow the application programmer to use a data structure meaningful to the particular application.

2 Introduction

We are proposing a standard for a set of subroutines to form the basis for writing mathematical software for implementing iterative methods on sparse matrices. This set includes Level 3 sparse BLAS for sparse-full matrix multiplication and sparse triangular solutions. We also include routines for permuting full and sparse matrices and a routine for data checking, for changing the data format of a sparse matrix, and for performing a diagonal scaling on the input matrix.

We have designed the routines so that the numerical library software developer need not be at all concerned with the complexities of the various data structures used for sparse matrices. We have established a single simple interface that will accommodate most of the data formats in use today and have provided the ability to transform between formats without the need to know their details explicitly. Included in this is the automatic selection and transformation to a data format chosen by vendors to be efficient on their machine. We expect that this transformation will be provided by vendors of high-performance computers.

We are quite intentionally not overly ambitious in this present proposal because we feel it is important to get a standard established in the very near future before different manufacturers develop their own different methods of implementing the functions we seek to standardize.

The primary user community that we are targeting are developers of library software although we recognize that the software could and should be used as building blocks by applications programmers. We expect a sophisticated user community but not necessarily one that is or need be familiar with details of sparse storage schemes. The application programmer would provide the input matrix in a supported format nearest to the natural one for the particular application and would normally request that a transformation be made to the format provided by the vendor for the target machine. We envisage that a major benefit of the provision of this interface will be a much quicker implementation of new algorithmic ideas into complex applications packages.

We have designed the suite of routines to cover the needs of these communities but have tried to keep the number of codes to a minimum to avoid an unnecessary burden on the developer of this suite. We hope that we have met the goals of the earlier standard efforts discussed in Section 1, namely portability, efficiency, maintainability, and flexibility.

In Section 3, we give a general overview of our proposal, with more precise details of the scope in Section 4 and the naming conventions we adopt in Section 5. The storage of sparse matrices is discussed in Section 6 and the use of permutation operations in Section 7. A brief description of argument conventions in Section 8 is followed by our detailed specifications for the Level 3 sparse BLAS in Section 9. Because this report is very much a proposal, Section 10 contains several issues which we feel are still subject to debate and on which we would particularly welcome comment. We have ourselves used these kernels for the implementation of many different iterative solvers and, in the appendix, we include an example showing the use of Level 3 sparse BLAS in the coding of a conjugate gradient algorithm.

A preliminary draft of this proposal was discussed in a workshop at the Copper Mountain

Conference on Iterative methods in April 1992 in conjunction with a paper "A proposal for a sparse BLAS toolkit" by Michael Heroux. The Toolkit paper provides implementation level data dependent routines for Level 2 and Level 3 BLAS and should be viewed as a complementary companion paper to this one. The "User Level Sparse BLAS" routines offer the implementor of an iterative solver a degree of independence from the sparse data structure. They are implemented in terms of the routines in the "A proposal for a sparse BLAS toolkit". We have adopted a consistent standard for naming and interfaces over the two papers.

Although the codes and interfaces in both papers are in Fortran 77, we recognize that Fortran 90 will give a cleaner and more flexible interface, and we will develop a Fortran 90 interface in the near future.

After the Copper Mountain meeting, we distributed a draft proposal to several of our colleagues and have modified the draft to take into account their helpful comments. This version is the first which we have made publicly available.

This paper defines the user interface and a framework to develop sparse kernels. It is not intended as a user guide of existing software. We make no claim that we have implemented every path we describe for every data structure. We hope that the framework we define in this paper is flexible enough to accommodate more data structures and to allow interested users to implement other paths as the need arises. A sample code in Fortran 77 is available through anonymous ftp and shows an implementation of some of the paths. A full specification document for the subroutines can also be obtained using anonymous ftp.

3 General Overview

In many iterative algorithms for solving linear systems and for computing eigenvalues, a very significant fraction of the computing time is spent in multiplying a full vector or vectors by a sparse matrix, or in solving triangular systems of equations with a sparse matrix when implementing the preconditioning. These two kernels are the most obvious candidates for standardization.

The choice of data structure affects the performance of software for sparse matrices and a machine-specific implementation may improve the performance dramatically. In many implementations of iterative algorithms for high-performance computers it is often beneficial to change the data structure provided by the user into a machine-specific one before actually starting the computation. The cost in moving or rearranging the data, and storage, is in many cases more than offset by the gain in performance. Sometimes such rearranging is done explicitly as in the routine sparse_matvec_setup in CMSSL ([5]), and sometimes it is implicit as in the IBM Library ESSL ([15]).

It is proposed to define standard interfaces for the following functions:

- a routine for performing the product of a sparse and a dense matrix,
- a routine for solving a sparse upper or lower triangular system of linear equations

for a matrix of right-hand sides,

- a routine to check the input data, to transform from one sparse format to another, and to scale a sparse matrix,

- a routine to permute the columns of a sparse matrix and a routine to permute the rows of a full matrix.

Note that, although we express the first two functions in Level 3 BLAS terms, the functionality of Level 2 BLAS are available as a trivial subset.

The interface to the routines we describe is designed to accommodate most different data structures. We define a sparse matrix by adding to the usual real and integer arrays (in the following we will use the term real both to cover complex as well as real numbers and to define a floating-point number without prejudice to its precision), a character descriptor array that defines which storage format is used, and a third integer array of length 10.

The data preprocessing routine is essential to this proposal. This routine is designed to be called before the body of the computation. The interface is designed to accept many different data formats and produce many others. In particular it can interrogate the machine it is running on and transform the data into a format that is particularly suited for that machine. Readers interested in writing code for transforming from their own particular structures should consult the sparse toolkit proposal of Heroux [16] which is primarily concerned with implementation issues.

Many algorithms require the permutation of matrices. Additionally, some efficient implementations of sparse matrix-vector products, and of the solution of sparse triangular systems on vector or parallel processors, require the vectors to be reordered. If high efficiency is required, it is necessary to avoid explicit vector permutations in the inner loops and, to enable this, routines have been added to permute sparse matrices and full matrices appropriately. The permutation routines can also be called outside the body of the computation in order to increase efficiency by avoiding permutations within the main loop of the algorithm. This optional facility is discussed more in Section 7.

We illustrate the use of the kernels by a code in the appendix. The algorithm is a version of the preconditioned conjugate gradient algorithm proposed by Henk van der Vorst [6] for efficiency on parallel architectures.

4 Scope of the Level 3 Sparse BLAS

As we said in the introduction, we have fairly modest aims in this present proposal partly because we feel that they cover most requirements of software developers and partly because we are keen that the standard can be accepted quickly and will be embraced by manufacturers. Note that our definition includes operations on vectors as a trivial subset. These may be coded separately at the machine dependent level.

If

- A and H are sparse matrices

- T is a triangular sparse matrix
- B and C are dense matrices
- D is a diagonal matrix
- P, P_L, and P_R are permutation matrices,

then the operations proposed have the following forms:

- Matrix-matrix products
  - $C \leftarrow \alpha P_L A P_R B + \beta C$
  - $C \leftarrow \alpha P_L A^T P_R B + \beta C$
- Solving triangular systems of equations with multiple right-hand sides
  - $C \leftarrow \alpha D P_L T^{-1} P_R B + \beta C$
  - $C \leftarrow \alpha D P_L T^{-T} P_R B + \beta C$
  - $C \leftarrow \alpha P_L T^{-1} P_R D B + \beta C$
  - $C \leftarrow \alpha P_L T^{-T} P_R D B + \beta C$
- Data preprocessing including change of data structure
  - Checks on input data (optional)
  - $(H, P_L, P_R) \leftarrow DA$
- Permuting the columns of a sparse matrix
  - $A \leftarrow AP$
  - $A \leftarrow AP^T$
- Permuting the rows of a dense matrix
  - $C \leftarrow PC$
  - $C \leftarrow P^T C$

5 Naming conventions

The name of a Level 3 sparse BLAS routine follows the conventions of the Level 3 BLAS for dense matrices. The first character in the name denotes the Fortran data type of the matrix as follows:

- S REAL
- D DOUBLE PRECISION

- C COMPLEX
- Z DOUBLE COMPLEX

Characters two and three are 'SP' and denote that the input matrix is SParse.

The fourth and fifth characters denote the operation as follows:

- MM Matrix-matrix product
- SM Solve a system of linear equations for a matrix of right-hand sides
- DP Data preprocessing routine
- RP Permutation on the right (column permutation)
- LP Permutation on the left (row permutation)

In the text of this paper we use precision independent names obtained by dropping the first character. Thus, for example, SPMM covers SSPMM, DSPMM, CSPMM, and ZSPMM.

6 Representation of sparse matrix

A sparse matrix A is represented by five arrays: a character array, a real array, and three integer arrays.

- DESCRA: character*5 array
- A: real array
- IA1 and IA2: integer arrays
- INFO: integer array

DESCRA is a character*5 array of size 10, which defines the format of the sparse matrix. We give some examples of possible formats in the following but stress that this list is neither exhaustive nor supported in any particular implementation.

```
DESCRA(1)    - the storage technique that is used.
               CSC or Compressed Sparse Column
               CSR or Compressed Sparse Row
               COO or Coordinate format
               DIA or Diagonal format
               ELL or Ellpack_Itpack format
               JAD or Jagged_Diagonals
               BDI or Block Diagonal format
               BSC or Block Sparse Column format
               BSR or Block Sparse Row format
```

```
              SKY or Skyline format

DESCRA(2) -   Matrix structure
              G or General
              SU or Symmetric_Upper
              SL or Symmetric_Lower
              HU or Hermitian_Upper
              HL or Hermitian_Lower
              TU or Triangular_Upper
              TL or Triangular_Lower
              SSU or Skew_Symmetric_Upper
              SSL or Skew_Symmetric_Lower
              D or Diagonal

DESCRA(3)   - Diagonal
              U or Unit   (diagonals not stored)
              N or Non_Unit

DESCRA(4:10) - Special information which may be used for a particular
              storage representation
```

Sometimes it is necessary to store supplementary integer information on the sparse data format. One possibility is to use IA1 or IA2. However, this is not acceptable in many cases because it would destroy compatibility with using them for some of the common formats. We therefore include a further integer array INFO that can hold this information, examples being the number of nonzeros when using a coordinate scheme or the block size for block data formats. The INFO array can also be used to convey further information to the preprocessing routine, for example an indication of the likely number of right-hand sides or the number of times the transformed data structure will be used so that the preprocessing routine may choose a better and more efficient data structure. INFO can also be used to indicate that both the matrix and its transpose will be required. The format for the data structure output from the transformation routine is, of course, controlled by the implementor. Although we allow two integer arrays for output, in some implementations only one of these arrays might be used.

There is a potential problem with arrays of character variables when calling Fortran subroutines from C. We feel strongly that the use of a character string is the best way of communicating data on the matrix structure but, since we don't want to pose a difficult problem to the C programmer, we suggest the easier alternative of forbidding the use of the last character so that the maximum string permitted is of length 4.

For example, if we were using the Harwell-Boeing format ([13]), DESCRA(1) would be set to CSC, DESCRA(2) to G, A to the matrix values by columns, IA1 the corresponding row indices, and IA2 pointers to the position of the first entry of each column in the arrays A and IA1. INFO could be used to convey information for the machine-specific transformation if it is required.

In the routine SPDP, the user can request that the format of the sparse matrix is automatically transformed to one that is best on the target machine. This is done by specifying the characters ??? in DESCRA(1) for the output format. In our sample implementation of the codes, we will interpret ??? to mean we leave the input data structure unchanged. We expect the manufacturers to provide appropriate code in their implementations. A nice aspect of this is that the vendor could change the output format to best suit the hardware on which the code is actually being run. In this way, the user is isolated from changes in the hardware. Because of this principal feature of SPDP, auxiliary data might need to be passed to SPDP using the INFO array. For example, the best data structure for SPMM or SPSM may be very dependent on the number of right-hand sides or the number of columns in the full matrix (K, say). The entries of the sparse matrix are used K times, therefore as K increases the cost of indirectly addressing them reduces. When K is very large, a dense vector model of computation works very well [1]. Similarly, it can be helpful to provide SPDP with information on the number of subsequent calls that will be made using the output structure. If the computation is to be repeated a large number of times, it is more worthwhile to spend extra time in the data preprocessing phase in order to obtain a particularly efficient structure for subsequent computation. INFO can also be used to inform the implementor that operations with both the matrix and its transpose will be later required, thus enabling further efficiency in data preprocessing by allowing both structures to be computed at the same time. A vendor chosen code would be output by SPDP into the descriptor associated with the output matrix to describe the data structure used. This code would be used in calls to subsequent subroutines.

It is important to stress that the above formats are provided by way of guidance only. We do not claim that they would all be fully supported and, in addition, users may wish to add their own structures. For example, see results of Erhel ([14]), or the Stripped Jagged Diagonals scheme(Paolini and Radicati [21]) that has proven efficient on an IBM 3090/VF computer. A good example of the influence of data structures on machine performance is given by Agarwal, Gustavson, and Zubair ([1]).

We feel that the five proposed arrays are sufficient to accommodate all of the more widely used sparse matrix storage techniques and we have allowed for future expansion by declaring DESCRA as a character*5 array of length 10 and INFO as an INTEGER array of length 10. Note that we only allow the storage of strings of length up to four in any entry of the character array. Although in much of our software development for this paper, we have primarily considered matrices represented in the CSR format, we have worked on implementations for many other storage techniques.

We have given no details of the storage for each of the structures mentioned above. The reader can find these in the paper by Heroux [16].

7 Permutations

We introduce subroutines to permute the columns of a sparse matrix or the rows of a full matrix explicitly. These subroutines are discussed in Section 9.4. Here we indicate how they may be used when efficient implementation is required. A further example is given in the appendix.

Let us assume that all that is required in the inner loop of the iterative method is:

```
do  i = 1, ...
    y ← Ax                    Multiply by a sparse matrix
    x ← x + ay
end do
```

and that the data conversion routine had converted the sparse matrix A to a matrix H with the same column ordering but a different row ordering. The code would then become:

```
(P_R, H) ← A                  Change data structure and generate permutations

do  i = 1, ...
    y ← Hx                    Multiply by a sparse matrix
    y ← P_R y                 Multiply (left) by a permutation matrix
    x ← x + ay
end do
```

and the additional permutation $P_R y$ (necessary to return y to the original ordering) in each iteration could significantly affect efficiency. By introducing explicit permutation subroutines, we can avoid this as follows:

```
(P_R, H) ← A                  Change data structure
x ← P_R^T x                   Multiply (left) by a permutation
H ← H P_R                     Multiply sparse matrix by permutation
do  i = 1, ...
    y ← Hx                    Multiply by a sparse matrix
    x ← x + ay
end do
x ← P_R x                     Multiply (left) by a permutation to restore
                              to the original ordering
```

8 Argument conventions

We follow a convention for the argument lists similar to that for the Level 3 BLAS for dense matrices with

- Arguments specifying options
- Arguments defining the sizes of the matrices
- Input scalar
- Description of input matrices

- Input scalar (associated with input-output matrix)

- Description of the input-output matrix

We discussed the arguments for the sparse matrices in Section 6. We also have a data format for a diagonal matrix. We hold this matrix in two arguments, a real array D of the values of the entries on the diagonal, and a character*1 argument (UNITD) which indicates whether the diagonal matrix is unit or whether we are doing row or column scaling.

Value	Meaning of UNITD
'U'	Unitary diagonal matrix — ie identity matrix
'L'	Scale on the left (row scaling)
'R'	Scale on the right (column scaling)

TRANS is a character argument specifying an option. It is used by the routines as follows:

Value	Meaning of TRANS
'N'	Operate with or generate the matrix
'T'	Operate with or generate the transpose of the matrix

CHECK is another character argument specifying an option in the data preprocessing routine SPDP. Values for CHECK and their meanings are given in the following table.

Value	Meaning of CHECK
'C'	Perform checks on data and exit
'Y'	Perform checks on data and format transformation
'N'	Do not perform data checks but transform format

In all the calls, if the option parameter is a character other than those in the tables, an immediate error return is made and the data is unchanged. As in the full BLAS, the number of rows and columns in the output array are given by M and N respectively. For SPMM, the number of columns of the input sparse matrix and rows in the input full matrix are given by K. For SPSM, the input sparse matrix is triangular of order M and there are N right-hand sides. 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 performed by SPMM reduces to $C \leftarrow \beta C$

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

A permutation matrix is represented by an integer array. In all cases, if the permutation array is P, entry P(i) is the position of row or column i in the new ordering. If no permutation is needed (that is, the permutation is the identity), the value of the first element of the permutation array can be set to 0.

The description of the input dense matrix consists of a permutation array and the array name B followed by LDB, the leading dimension of the array as declared in the calling (sub)program. The description of the output dense matrix consists of a permutation array and the array name C followed by the leading dimension LDC.

In many instances efficiency is promoted if extra work space is available. We accommodate this in each case by a real array WORK. If the length of array WORK is not present explicitly then it is the maximum of M and N.

In contrast to the full Level 3 BLAS, we provide an error flag, IERROR, in most routines. A positive value of IERROR indicates an error, the actual values are given in specification documents for the subroutines. The data checking option of the data preprocessing routine SPDP does fairly extensive checking but the other routines, since they will normally be called afterwards and in the main loops of the code, do only minimal checking. We feel it is important to introduce this parameter since we have a more complex (and hence more error prone) situation than the full case and there has also been some criticism of the full case for not including such a flag.

9 Specifications of the Level 3 sparse BLAS

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

```
INTEGER       IERROR, LDB, LDC, LWORK, LH1, LH2, LH, M, N, INFO(10),
              INFOH(10)
INTEGER       IA1(*), IA2(*), IT1(*), IT2(*), IH1(*), IH2(*),
              PL(*), PR(*)
CHARACTER*1   CHECK, TRANS, UNITD
CHARACTER*5   DESCRA(10), DESCRT(10), DESCRH(10)
```

For routines whose first letter is an S:

```
REAL          ALPHA, BETA
REAL          A(*), B(LDB,*), C(LDC,*), D(*), H(*), T(*), WORK(*)
```

For routines whose first letter is an D:

```
DOUBLE PRECISION  ALPHA, BETA
DOUBLE PRECISION  A(*), B(LDB,*), C(LDC,*), D(*), H(*), T(*), WORK(*)
```

For routines whose first letter is an C:

```
COMPLEX       ALPHA, BETA
COMPLEX       A(*), B(LDB,*), C(LDC,*), D(*), H(*), T(*), WORK(*)
```

For routines whose first letter is an Z:

```
DOUBLE PRECISION COMPLEX  ALPHA, BETA
DOUBLE PRECISION COMPLEX  A(*), B(LDB,*), C(LDC,*), D(*), H(*), T(*), WORK(*)
```

9.1 Sparse Matrix times Dense matrix

```
_SPMM (TRANS,M,N,K,ALPHA,PL,DESCRA,A,IA1,IA2,INFO,PR,B,LDB,
       BETA,C,LDC,WORK,LWORK,IERROR)
```

Operation (C is always mxn)

TRANS = 'N'	TRANS = 'T'
$C \leftarrow \alpha PL\ A\ PR\ B + \beta C$	$C \leftarrow \alpha PL\ A^T\ PR\ B + \beta C$

When the matrix is complex, the TRANS parameter can be either 'T' for transpose or 'H' for conjugate transpose.

9.2 Solution of triangular systems of equations

```
_SPSM(TRANS,M,N,ALPHA,UNITD,D,PL,DESCRT,T,IT1,IT2,INFO,PR,B,LDB,
      BETA,C,LDC,WORK,LWORK,IERROR)
```

	TRANS = 'N'	TRANS = 'T'
UNITD = 'L'	$C \leftarrow \alpha D\ P_L\ T^{-1}\ P_R\ B + \beta C$	$C \leftarrow \alpha D\ P_L\ T^{-T}\ P_R\ B + \beta C$
UNITD = 'R'	$C \leftarrow \alpha P_L\ T^{-1}\ P_R\ D\ B + \beta C$	$C \leftarrow \alpha P_L\ T^{-T}\ P_R\ D\ B + \beta C$

When the matrix is complex, the TRANS parameter can be either 'T' for transpose or 'H' for conjugate transpose.

9.3 Specification of data preprocessing routine

This is invoked by the following call:

```
_SPDP(CHECK,TRANS,M,N,UNITD,D,DESCRA,A,IA1,IA2,INFO,PL,DESCRH,
      H,IH1,IH2,INFOH,PR,LH,LH1,LH2,WORK,LWORK,IERROR)
```

If CHECK is set to 'Y' or 'C', the above routine will check the input data to see if indices are within range, and, in the case of triangular matrices, to see if they are indeed triangular and have a nonzero diagonal. An error return is also invoked if the data structures requested are not in the implementation. Errors are indicated by a positive value of IERROR. If CHECK is equal to 'C' then the routine immediately returns after checking the input data. If scaling or data transformation is requested, the subroutine then transforms the sparse matrix from the data structure DESCRA, A, IA1, IA2, and INFO to the data structure DESCRH, H, IH1, IH2, and INFOH, optionally (depending on value of UNITD) scaling the matrix by the diagonal matrix D. Any row and column permutations required are provided as output in PL and PR, respectively. Because the storage for the transformed matrix may differ from the original, we input the dimensions of H, IH1, IH2, and WORK in LH, LIH1, LIH2, and LWORK respectively. If these are insufficient for the data format requested, an error return is invoked. The number of locations used (or required in case or error) in H, IH1, IH2, and WORK are returned in LH, LH1, LH2, and LWORK respectively.

TRANS might be used if the user has the matrix A stored but wishes a more efficient way for performing the matrix-matrix multiplication $A^T B$. Then TRANS would be set to 'T' in the SPDP call but be set to 'N' in the call to DPMM. Note that if SPDP is called with TRANS 'N' followed by SPMM with TRANS set to 'T', the same function will be performed but SPMM would possibly be using an inappropriate data structure for multiplication by A^T. It may be that some vendors would wish to discourage this by not supporting calls to SPMM or SPSM with TRANS equal to 'T', returning with an error condition if such a call is attempted.

It is not expected that transformations will be provided between all possible data structures. Note that when SPDP is used to generate a vendor chosen data structure, DESCRH will be set to ??? on input and will be reset to a vendor chosen identifier on output. It is this reset value which should be used in subsequent calls.

9.4 Routines to permute matrices

In order to avoid permutations on each vector algebra operation, we allow permutations on the data structures outside the loop of the iterative algorithm (see Section 7). We believe that this can be accomplished using only a right (column) permutation of a sparse matrix and a left (row) permutation of a dense matrix. Because the permutation of the dense matrix corresponds to a full Level 3 BLAS, we use the appropriate full Level 3 BLAS nomenclature.

We thus have the following calls, SPRP performs the column permutation of a sparse matrix and GELP the row permutation of a dense one.

```
_SPRP( TRANS, M, N, DESCRA, A, IA1, IA2, INFO, P, WORK)
```

and

```
_GELP( TRANS, M, N, P, B, LDB, WORK)
```

Calling SPMM with the two permutation matrices P_L and P_R is the same as calling SPMM with I (identity matrix) and P_R followed by a call to GELP with P_L.

10 Questions for discussion

As we said in the introduction, establishing a standard for Level 3 sparse BLAS involves many compromises. We have tried to be as frugal as practical in introducing new routines and, although the calling sequences are long, we have kept them as short as possible for the functionality and flexibility we think necessary. In this section, we describe some of the compromises explicitly and invite your comments on whether we have sacrificed anything crucial for the sake of simplicity. Also, we stress that the standard is designed to allow future expansion and so the fact that some feature is omitted in the current proposal does not preclude its existence in the future.

- Naming conventions

  Our naming convention follows that of the earlier BLAS. For the matrix-matrix and triangular solve, we have used the characters MM and SM respectively as in the case of full BLAS. Since the routine for creating a new data format, checking the data, and scaling the matrix would normally be called prior to the body of the main code, we have called this a "data preprocessing" routine and have used the characters DP. Our use of the letters SP for sparse, could cause confusion with their use for "*symmetric packed* " in the full BLAS. However, our routines are at Level 3 and symmetric packed is only supported in the full Level 2 BLAS. The only suggestion that we had in our first call for comments was to use the characters 'SS'. We are not so keen on that since the single precision triangular solve routine would become SSSSM.

- Representation of sparse format.

  The format for the sparse matrix is held in the real array A, two integer arrays IA1 and IA2, and a character array DESCRA. We have incorporated UPLO and DIAG, as used in the full BLAS, within the DESCRA array. Because it is sometimes necessary to hold auxiliary information, for example the number of entries for the coordinate scheme or the number of right-hand sides to help in the automatic selection of optimal data formats, we do this by including another integer array INFO. We prefer this to two other solutions of including the integer information in IA1 or IA2 (or in a combined IA1/IA2 array), or providing an auxiliary subroutine PUTCHAR that will store an integer value in DESCRA. Although it would have perhaps been cleaner to have only one integer array instead of three (IA1, IA2, and INFO), we have made a concession to some of the more commonly used formats (CSC, CSR and coordinate scheme (COO) for example) by including the two arrays. We feel it is important and user-friendly to provide this backward compatibility but it is certainly an area that caused much comment (both for and against) in our earlier draft. The format of the output arrays from SPDP (H, IH1, IH2, INFOH) are of course at the discretion of the implementor so it is possible that all integer output is included in only one array

for example. Although we give some examples for possible data formats in Section 6, we must stress that they are not meant to be exhaustive, nor would we expect every implementation to support all those we mention. What is important, however, is that any input data format that you feel important could be included within our framework.

- Transformation between data formats

  In most applications, the ??? output will be used. We feel, however, that it is worth allowing the greater flexibility of permitting the user to transform between two data structures of his or her own choosing, even if such a code is not implemented by the vendors. If people think this overcomplicates things please let us know.

- Use of array of character variables

  There is a problem with the use of arrays of character variables when calling Fortran subroutines from C programs. Since we envisage the use of the kernels from C we need to take some action. We could place the burden on the C programmer or could avoid the use of character arrays altogether. However, we do not like either of these options and have chosen to avoid storage of information in the last character of the character variable. In this case, we only allow a maximum of four characters to be stored in a character*5 variable. We note that problems of this sort will disappear with Fortran 90.

- Finite-element matrices

  We have tried not to prejudice the way the sparse matrix is stored. Indeed we debated about including a routine to assemble a finite-element problem. The comments which we have had so far indicate that it would not be possible to offer full support for finite-element applications without significantly altering the proposal and making it much less suitable for its main purpose. There was some support for a routine for assembling a matrix but that would be at quite a different level from our current set of routines and so we do not propose to include this. We should point out that the Harwell-Boeing format ([13]) does allow for the storage of finite-element matrices and so they can be held in the format proposed in this paper.

- Scaling matrices

  We have not included a diagonal scaling matrix in the SPMM routines but feel the appropriate place for such a scaling is at the preprocessing stage to avoid extra overhead in the inner loops. We have, however, allowed the user to use a diagonal scaling matrix in the case of triangular solves. This could, for example, permit the diagonal of the triangular matrix (or its inverse) to be stored separately for efficiency and flexibility.

- Use of TRANS

  In most cases, SPMM and SPSM will be run with TRANS equal to 'N', since the matrix will have been transposed already, if required, by SPDP. Indeed, some vendors may disable such a call. At the moment, however, we have kept the parameter TRANS because, for example, the user may be developing code on a machine without vendor implementations and may find it easier to avoid explicit transposition in SPDP. Note that in the SPDP routine, the implementor could choose to generate

both structures for the matrix and its transpose. Information on whether both are required is passed in the INFO array. We preferred this to allowing TRANS in SPDP to have the value 'B' (for both) since we feel this is more an implementation issue akin to stipulating the number of subsequent right-hand sides.

- Triangular matrix - matrix multiplication

  There was some interest in a triangular matrix - matrix multiplication routine. We feel that this is more suitably included as a subcase of the SPMM routine and so do not have this as a special case.

- Use of parameter β in SPSM

  Several people suggested dropping the β parameter from SPSM. Although many people at the Copper Mountain meeting commented that we would regret dropping this parameter, we do not have a good example where this is necessary. We have, however, kept it in to await further comments on the grounds that it is easier for us to remove than to add this parameter.

- Omission of TRANSB

  The full Level 3 BLAS allows transposition of the matrix B. However, we do not see the usefulness of this in the present context and do not include this parameter in our Level 3 sparse BLAS.

- Level 2 sparse BLAS

  We have intentionally not provided explicit Level 2 BLAS routines SPMV and SPSV since we feel their functionality is easily incorporated within SPMM and SPSM respectively. We do not believe efficiency need be compromised because special action could be taken by the vendor when the number of columns in `B` is equal to `1`.

- Permutations of matrices

  We have restricted the explicit permutation calls to only two in the belief that it is unnecessary also to include row permutations of sparse matrices or column permutations of full ones. Although this is unæsthetically unsymmetric, we want to keep the demands on implementors to a minimum. Please let us know if this functionality is required. In the permutation routines we allow both operations by a permutation and its transpose although it is trivial to generate one from the other. Again comments on this facility are welcome.

11 Acknowledgments

Many of our colleagues made very helpful comments on an earlier draft of this proposal. We would like to thank all those who participated in the Copper Mountain discussion chaired by Steve Ashby and individuals who have since commented on the draft: Ramesh Agarwal, Richard Brankin, Michel Daydé, Fred Gustavson, Nick Gould, Michael Heroux, Gérard Meurant, Marco Perezzani, Alexander Peters, John Reid, Willi Schönauer, Ray Tuminaro, Carlo Vittoli, and Henk van der Vorst.

12 Appendix — Conjugate Gradient Algorithm

We illustrate the use of the kernels by a code which implements a version of the preconditioned conjugate gradient algorithm proposed by Henk van der Vorst ([6]) for efficiency on parallel architectures. The algorithm can be described in the following way:

$x_{-1} = x_0 =$ initial guess; $r_0 = b - Ax_0$
$p_{-1} = 0$; $\beta_{-1} = 0$; $\alpha_{-1} = 0$
$s = L^{-1} r_0$
$\rho_0 = (s, s)$
for $i = 0, 1, 2, \ldots$
    $w_i = L^{-T} s$;
    $p_i = w_i + \beta_{i-1} p_{i-1}$;
    $q_i = A p_i$;
    $\gamma = (p_i, q_i)$;
    $x_i = x_{i-1} + \alpha_{i-1} p_{i-1}$;
    $\alpha_i = \rho_i / \gamma$;
    $r_{i+1} = r_i - \alpha_i q_i$;
    $s = L^{-1} r_{i+1}$;
    $\rho_{i+1} = (s, s)$;
    if r_{i+1} *small enough* then
        $x_{i+1} = x_i + \alpha_i p_i$;
        quit;
    $\beta_i = \rho_{i+1} / \rho_i$;
end;

The following code assumes that the preconditioner is already available in the form LDL^T. The above algorithm is thus effected by using $LD^{-1/2}$ in place of L. This is easy to do because of our inclusion of a diagonal matrix in the call to SPSM. The main iteration loop (loop 100 on pages 30-31) is very simple: some vector operations, a call to a matrix-vector product (DSPMM), and the preconditioner is implemented with two calls to a solver for a sparse triangular system (DSPSM). The only "extra" requirement is a call to the data preprocessing routine (DSPDP). In this example, we illustrate three different options: using the original data structure, choosing a machine-specific data structure, and combining this with the avoidance of permutations in the inner loops. The data structure used by the kernels depends on the machine so that, depending on which machine the code is run, there may be no data transformation at all, or the data may perhaps be changed to a data structure that requires reordering and permutations. In all cases, the routine generates all the necessary auxiliary information and vectors completely transparently to the user and does not impact the implementation of the conjugate gradient algorithm. The matrix structure input to the main computational kernels is just that which is output from the data preprocessing routine.

```
      SUBROUTINE CG(M,DESCRA,A,IA1,IA2,INFORM,DESCRL,L,IL1,IL2,DESCRU,
     *              U,IU1,IU2,DESCRAN,AN,IAN1,IAN2,LAN,LIAN1,LIAN2,
     *              DESCRLN,LN,ILN1,ILN2,LLN,LILN1,LILN2,DESCRUN,
     *              UN,IUN1,IUN2,LUN,LIUN1,LIUN2,VDIAG,B,X,EPS,ITMAX,
     *              ERR,ITER,IERROR,Q,R,S,W,P,PT1,IAUX,LIAUX,AUX,LAUX)
C
C     Purpose
C     =======
C
C     Driver for routine CGSOLVE ... routine to solve Ax=b by conjugate
C             gradients using algorithm described above.
C
C
C     Parameter
C     =========
C
C     M         - INTEGER
C             On entry M specifies the number of rows of the matrix A.
C             M must be greater than or equal to zero.
C             Unchanged on exit.
C
C     DESCRA    - CHARACTER*5 array of DIMENSION (10)
C             On entry DESCRA defines the format of the sparse matrix.
C             Unchanged on exit.
C
C     A         - DOUBLE PRECISION array of DIMENSION (*)
C             On entry A specifies the values of the input sparse
C             matrix.
C             Unchanged on exit.
C
C     IA1       - INTEGER array of dimension (*)
C             On entry IA1 holds integer information on input sparse
C             matrix.  Actual information will depend on data format used.
C             Unchanged on exit.
C
C     IA2       - INTEGER array of dimension (*)
C             On entry IA2 holds integer information on input sparse
C             matrix.  Actual information will depend on data format used.
C             Unchanged on exit.
C
C     INFORM    - INTEGER array of length 10.
C             On entry can hold auxiliary information on input matrices
C             formats or environment of subsequent calls.
C             Might be changed on exit.
C
C     DESCRL    - CHARACTER*5 array of DIMENSION (10)
C             On entry DESCRL defines the format of the lower factor
```

```
C             of the preconditioner.
C             Unchanged on exit.
C
C     L        - DOUBLE PRECISION array of DIMENSION (*)
C             On entry L specifies the values of the input sparse
C             lower factor.
C             Unchanged on exit.
C
C     IL1      - INTEGER array of dimension (*)
C             On entry IL1 holds integer information on the lower
C             factor for the preconditioning matrix.
C             Unchanged on exit.
C
C     IL2      - INTEGER array of dimension (*)
C             On entry IL2 holds integer information on the lower
C             factor for the preconditioning matrix.
C             Unchanged on exit.
C
C     DESCRU   - CHARACTER*5 array of DIMENSION (10)
C             On entry DESCRU defines the format of the upper
C             factor of the preconditioner.  This is the transpose
C             of the lower triangle L.
C             Unchanged on exit.
C
C     U        - DOUBLE PRECISION array of DIMENSION (*)
C             On entry U specifies the values of the input sparse
C             upper factor.
C             Unchanged on exit.
C
C     IU1      - INTEGER array of dimension (*)
C             On entry IL1 holds integer information on the lower
C             factor for the preconditioning matrix.
C             Unchanged on exit.
C
C     IU2      - INTEGER array of dimension (*)
C             On entry IL1 holds integer information on the lower
C             factor for the preconditioning matrix.
C             Unchanged on exit.
C
C     DESCRAN  - CHARACTER*5 array of DIMENSION (10)
C             On entry DESCRAN defines the format and type of the
C             new data structure for the matrix A
C             If no data transformation is required DESCRAN = DESCRA
C             Unchanged on exit.
C
C     AN       - DOUBLE PRECISION array of DIMENSION (*)
C             It need not be set on entry
C             On exit,
```

```
C            If data transformation is required:
C               AN specifies the values of the new data
C               structure for the sparse matrix.
C            If no data transformation is required
C               Unchanged on exit
C
C     IAN1     - INTEGER array of dimension (*)
C            It need not be set on entry
C            On exit,
C            If data transformation is required:
C               IAN1 specifies the column indices or
C               pointers needed to use the new data
C               structure for the sparse matrix.
C            If no data transformation is required
C               Unchanged on exit
C
C     IAN2     - INTEGER array of dimension (*)
C            It need not be set on entry
C            On exit,
C            If data transformation is required:
C               IAN2 specifies the column indices or
C               pointers needed to use the new data
C               structure for the sparse matrix.
C            If no data transformation is required
C               Unchanged on exit
C
C     LAN      - INTEGER
C            On entry LAN specifies the dimension of AN
C            LAN must be greater than zero.
C            Unchanged on exit
C
C     LIAN1    - INTEGER
C            On entry LIAN1 specifies the dimension of IAN1
C            LIAN1 must be greater than zero.
C            Unchanged on exit
C
C     LIAN2    - INTEGER
C            On entry LIAN1 specifies the dimension of IAN2
C            LIAN2 must be greater than zero.
C            Unchanged on exit
C
C     DESCRLN  - CHARACTER*5 array of DIMENSION (10)
C            On entry DESCRLN defines the format and type of the
C            new data structure for the lower factor of
C            the preconditioner.
C            If no data transformation is required DESCRLN = DESCRL
C            Unchanged on exit.
C
```

```
C     LN       - DOUBLE PRECISION array of DIMENSION (*)
C              It need not be set on entry
C              On exit,
C              If data transformation is required:
C                 LN specifies the values of the new data
C                 structure for the lower factor of
C                 the preconditioner.
C              If no data transformation is required
C                 Unchanged on exit
C
C     ILN1     - INTEGER array of dimension (*)
C              It need not be set on entry
C              On exit,
C              If data transformation is required:
C                 ILN1 specifies the column indices or
C                 pointers needed to use the new data
C                 structure for the lower factor of
C                 the preconditioner.
C              If no data transformation is required
C                 Unchanged on exit
C
C     ILN2     - INTEGER array of dimension (*)
C              It need not be set on entry
C              On exit,
C              If data transformation is required:
C                 ILN2 specifies the column indices or
C                 pointers needed to use the new data
C                 structure for the lower factor of
C                 the preconditioner.
C              If no data transformation is required
C                 Unchanged on exit
C
C     LLN      - INTEGER
C              On entry LLN specifies the dimension of LN
C              LLN must be greater than zero.
C              Unchanged on exit
C
C     LILN1    - INTEGER
C              On entry LILN1 specifies the dimension of ILN1
C              LILN1 must be greater than zero.
C              Unchanged on exit
C
C     LILN2    - INTEGER
C              On entry LILN1 specifies the dimension of ILN2
C              LILN2 must be greater than zero.
C              Unchanged on exit
C
C     DESCRUN  - CHARACTER*5 array of DIMENSION (10)
```

```
C            On entry DESCRUN defines the format and type of the
C            new data structure for the upper factor of
C            the preconditioner.
C            If no data transformation is required DESCRUN = DESCRU
C            Unchanged on exit.
C
C     UN      - DOUBLE PRECISION array of DIMENSION (*)
C            It need not be set on entry
C            On exit,
C            If data transformation is required:
C               UN specifies the values of the new data
C               structure for the lower factor of
C               the preconditioner.
C            If no data transformation is required
C               Unchanged on exit
C
C     IUN1    - INTEGER array of dimension (*)
C            It need not be set on entry
C            On exit,
C            If data transformation is required:
C               IUN1 specifies the column indices or
C               pointers needed to use the new data
C               structure for the upper factor of
C               the preconditioner.
C            If no data transformation is required
C               Unchanged on exit
C
C     IUN2    - INTEGER array of dimension (*)
C            It need not be set on entry
C            On exit,
C            If data transformation is required:
C               IUN2 specifies the column indices or
C               pointers needed to use the new data
C               structure for the upper factor of
C               the preconditioner.
C            If no data transformation is required
C               Unchanged on exit
C
C     LUN     - INTEGER
C            On entry LUN specifies the dimension of UN
C            LUN must be greater than zero.
C            Unchanged on exit
C
C     LIUN1   - INTEGER
C            On entry LIUN1 specifies the dimension of IUN1
C            LIUN1 must be greater than zero.
C            Unchanged on exit
C
```

```
C     LIUN2    - INTEGER
C              On entry LIUN1 specifies the dimension of IUN2
C              LIUN2 must be greater than zero.
C              Unchanged on exit
C
C     VDIAG    - DOUBLE PRECISION array of dimension (M)
C              On entry VDIAG contains the values of the diagonal
C              of the preconditioner.
C              Unchanged on exit.
C
C     B        - DOUBLE PRECISION array of dimension (M)
C              On entry B contains the right-hand side of matrix problem
C              Unchanged on exit.
C
C     X        - DOUBLE PRECISION array of dimension (M)
C              On entry X contains the initial guess of the solution,
C              if no guess is available, it should be set to the zero vector.
C              On exit X contains the solution of linear system.
C
C     EPS      - DOUBLE PRECISION
C              On entry EPS contains the tolerance required to
C              stop the iterative methods using a chosen stopping
C              criteria.
C              Unchanged on exit.
C
C     ITMAX    - INTEGER
C              On entry ITMAX contains the maximum number of iterations
C              allowed.
C              Unchanged on exit.
C
C     ERR      - DOUBLE PRECISION
C              On exit ERR contains the norm of the residual
C
C     ITER     - INTEGER
C              On exit ITER contains the number of iterations performed when
C              iteration process stops.
C
C     IERROR   - INTEGER
C              On exit IERROR contains the values of error flag as follows:
C              IERROR = 0   no error
C              IERROR = -2  the method failed to converge in ITMAX iterations
C              IERROR = -4  error on sigma in CGSOLVE
C              IERROR = 2   error on dimension of vector IAUX
C              IERROR = 4   error on dimension of vector AUX
C              IERROR = 8   this data conversion not implemented
C              IERROR = 16  error on dimensions of vectors AN, IAN1, IAN2
C              IERROR = 64  LWORK  <=  0
C
```

```
C     Q R S W P PT1 - DOUBLE PRECISION arrays of DIMENSION (M)
C             Work areas used in the Conjugate gradient methods
C
C     IAUX     - INTEGER array of DIMENSION(LIAUX)
C             Work area
C
C     LIAUX    - INTEGER
C             On entry LIAUX specifies the dimension of IAUX
C             LIAUX must be greater than zero.
C             The value required depends on the options invoked but it
C             will not exceed 9M+2.
C             Unchanged on exit.
C
C     AUX      - DOUBLE PRECISION array of DIMENSION (LAUX)
C             Work area
C
C     LAUX     - DOUBLE PRECISION
C             On entry LAUX specifies the dimension of AUX
C             LAUX must be greater than zero.
C             The value of LAUX depends on the data transformation
C             being performed.
C             Unchanged on exit.
C
C
C
C     .. Scalar Arguments ..
      DOUBLE PRECISION EPS, ERR
      INTEGER       IERROR,  ITER, ITMAX, LAUX, LIAUX, M
C     .. Array Arguments ..
C
      DOUBLE PRECISION A(*), AN(*), AUX(LAUX), B(*), L(*), LN(*),
     *                 S(*), Q(*), R(*), U(*), UN(*), VDIAG(*), W(*),
     *                 X(*),P(*),PT1(*)
      INTEGER       IA1(*), IA2(*), INFORM(*), IAN1(*), IAN2(*),
     *              IL1(*), IL2(*), IAUX(LIAUX),
     *              ILN1(*), ILN2(*), IU1(*), IU2(*), IUN1(*), IUN2(*)

      CHARACTER     DESCRA(10)*5, DESCRAN(10)*5, DESCRL(10)*5,
     *              DESCRLN(10)*5, DESCRU(10)*5, DESCRUN(10)*5
C     .. Local Scalars ..
      INTEGER       I, IP1, IP2, IP3, IP3N, IP4, IP5,
     *              LAN, LDB, LIAN1,
     *              LIAN2, LILN1, LILN2, LIUN1, LIUN2, LLN, LUN,
     *              NOPERM
      DOUBLE PRECISION D
      CHARACTER     TRANSA, TRANSL, TRANSP, TRANSU, CHECK, UNITD
C     .. Parameters ..
      PARAMETER     (NOPERM=0)
```

```
C     .. External Subroutines ..
      EXTERNAL      CGSOLVE, DGELP, DSPDP
C     .. Executable Statements ..
      UNITD='U'
      IF (DESCRA(1).EQ.DESCRAN(1)) THEN
C
C        No data conversion (or permutation) is required for matrices A L and U
C
         CALL CGSOLVE(M,DESCRA,A,IA1,IA2,INFORM,NOPERM,NOPERM,DESCRL,
     *                L,IL1,IL2,NOPERM,NOPERM,DESCRU,U,IU1,IU2,
     *                NOPERM,NOPERM,VDIAG,B,X,EPS,ITMAX,ERR,ITER,
     *                IERROR,Q,R,S,W,P,PT1,AUX,LAUX)
         IF (IERROR .NE. 0) RETURN
      ELSE IF (DESCRA(1).NE.DESCRAN(1)) THEN
C
C          Data conversion is required for matrices A L and U
C
         IF (DESCRAN(1).NE.'???') THEN
C
C           Set pointers to integer work area IAUX for permutation vectors
C
            IP1 = 1
            IP2 = IP1 + M
            IP3 = IP2 + M
            IP4 = IP3 + M
            IP5 = IP4 + M
            IP6 = IP5 + M
            IP7 = IP6 + M
C
C           Check on IAUX
C
            IF (IP7+1.GT.LIAUX) THEN
C
C              Error on the dimension of vector IAUX
C
               IERROR=2
               RETURN
            END IF
            TRANSA = 'N'
            TRANSL = 'N'
            TRANSU = 'N'
            CHECK = 'N'
            CALL DSPDP(CHECK,TRANSA,M,M,UNITD,D,DESCRA,A,IA1,IA2,
     *                 INFORM,IAUX(IP1),DESCRAN,AN,IAN1,IAN2,INFORM,
     *                 IAUX(IP2),LAN,LIAN1,LIAN2,AUX,LAUX,IERROR)
            IF (IERROR .NE. 0) RETURN
            CALL DSPDP(CHECK,TRANSL,M,M,UNITD,D,DESCRL,L,IL1,IL2,
     *                 INFORM,IAUX(IP3),DESCRLN,LN,ILN1,ILN2,INFORM,
```

```
     *                IAUX(IP4),LLN,LILN1,LILN2,AUX,LAUX,IERROR)
            IF (IERROR .NE. 0) RETURN
            CALL DSPDP(CHECK,TRANSU,M,M,UNITD,D,DESCRU,U,IU1,IU2,
     *                INFORM,IAUX(IP5),DESCRUN,UN,IUN1,IUN2,INFORM,
     *                IAUX(IP6),LUN,LIUN1,LIUN2,AUX,LAUX,IERROR)
            IF (IERROR .NE. 0) RETURN
            CALL CGSOLVE(M,DESCRAN,AN,IAN1,IAN2,INFORM,IAUX(IP2),
     *                  IAUX(IP1),DESCRLN,LN,ILN1,ILN2,IAUX(IP4),
     *                  IAUX(IP3),DESCRUN,UN,IUN1,IUN2,IAUX(IP6),
     *                  IAUX(IP5),VDIAG,B,X,EPS,ITMAX,ERR,ITER,
     *                  IERROR,Q,R,S,W,P,PT1,AUX,LAUX)
            IF (IERROR .NE. 0) RETURN
          ELSE IF (DESCRAN(1).EQ.'???') THEN
C
C           Format at discretion of implementor
C           Set pointers to integer work area IAUX for permutation vectors
C
            IP1 = 1
            IP2 = IP1 + M
            IP3 = IP2 + M
            IP4 = IP3 + M
            IP5 = IP4 + M
            IP6 = IP5 + M
            IP3N = IP6 + M
            IP4N = IP3N + M
            IP5N = IP4N + M
            IP6N = IP5N + M
            IP7 = IP6N + M
C
C           Check on IAUX
C
            IF (IP7+1.GT.LIAUX) THEN
C
C              Error on the dimension of vector IAUX
C
               IERROR=2
               RETURN
            END IF
            TRANSA = 'N'
            TRANSL = 'N'
            TRANSU = 'N'
            CHECK = 'N'
            CALL DSPDP(CHECK,TRANSA,M,M,UNITD,D,DESCRA,A,IA1,IA2,
     *                INFORM,IAUX(IP1),DESCRAN,AN,IAN1,IAN2,INFORM,
     *                IAUX(IP2),LAN,LIAN1,LIAN2,AUX,LAUX,IERROR)
            IF (IERROR .NE. 0) RETURN
            CALL DSPDP(CHECK,TRANSL,M,M,UNITD,D,DESCRL,L,IL1,IL2,
     *                INFORM,IAUX(IP3),DESCRLN,LN,ILN1,ILN2,INFORM,
```

```
     *                IAUX(IP4),LLN,LILN1,LILN2,AUX,LAUX,IERROR)
            IF (IERROR .NE. 0) RETURN
            CALL DSPDP(CHECK,TRANSU,M,M,UNITD,D,DESCRU,U,IU1,IU2,
     *                INFORM,IAUX(IP5),DESCRUN,UN,IUN1,IUN2,INFORM,
     *                IAUX(IP6),LUN,LIUN1,LIUN2,AUX,LAUX,IERROR)
            IF (IERROR .NE. 0) RETURN
C
C           Introduce explicit permutations of input vectors in order
C           to avoid additional permutation in the inner loop
C           of iterative method
C
            TRANSP = 'N'
            LDB = M
            CALL DGELP(TRANSP,M,1,IAUX(IP1),B,LDB,AUX)
            CALL DGELP(TRANSP,M,1,IAUX(IP1),X,LDB,AUX)
            CALL DGELP(TRANSP,M,1,IAUX(IP1),VDIAG,LDB,AUX)
C
C           Define the permutation matrices for subroutine CGSOLVE
C
            DO 20 I = 1, M
               IAUX(IP3N+I-1) = IAUX(IP4+IAUX(IP1+I-1)-1)
               IAUX(IP4N+I-1) = IAUX(IP2+IAUX(IP3+I-1)-1)
               IAUX(IP5N+I-1) = IAUX(IP6+IAUX(IP1+I-1)-1)
               IAUX(IP6N+I-1) = IAUX(IP2+IAUX(IP5+I-1)-1)
   20       CONTINUE

            CALL CGSOLVE(M,DESCRAN,AN,IAN1,IAN2,INFORM,NOPERM,NOPERM,
     *                   DESCRLN,LN,ILN1,ILN2,IAUX(IP3N),IAUX(IP4N),
     *                   DESCRUN,UN,IUN1,IUN2,IAUX(IP5N),IAUX(IP6N),
     *                   VDIAG,B,X,EPS,ITMAX,ERR,ITER,IERROR,Q,R,S,W,
     *                   P,PT1,AUX,LAUX)
            IF (IERROR .NE. 0) RETURN
            CALL DGELP(TRANSP,M,1,IAUX(IP2),X,LDB,AUX)
            CALL DGELP(TRANSP,M,1,IAUX(IP2),B,LDB,AUX)
         END IF
       END IF
       RETURN
       END

C ---------------------------------------------------------------------

       SUBROUTINE CGSOLVE(M,DESCRA,A,IA1,IA2,INFORM,P1,P2,
     *                    DESCRL,L,IL1,IL2,P3,P4,
     *                    DESCRU,U,IU1,IU2,P5,P6,VDIAG,B,X,EPS,ITMAX,ERR,
     *                    ITER,IERROR,Q,R,S,W,P,PT1,WORK,LWORK)
C
C      Purpose
C      =======
```

```
C
C     CGSOLVE  performs the preconditioned conjugate gradient method
C              Van der Vorst's version
C              Note: The preconditioner is given in the form
C                                 -1  T
C                       K = L D   L
C
C     Parameters
C     ==========
C
C     M,DESCRA,A,IA1,IA2,INFORM,DESCRL,L,IL1,IL2,DESCRU,U,IU1,IU2
C     As defined in Subroutine CG
C
C     P1        - INTEGER array of dimension(M)
C              On entry P1 specifies the values of a permutation matrix
C              Unchanged on exit.
C
C     P2        - INTEGER array of dimension(M)
C              On entry P2 specifies the values of a permutation matrix
C              Unchanged on exit.
C
C
C     P3        - INTEGER array of dimension(M)
C              On entry P3 specifies the values of a permutation matrix
C              Unchanged on exit.
C
C     P4        - INTEGER array of dimension(M)
C              On entry P4 specifies the values of a permutation matrix
C              Unchanged on exit.
C
C     P5        - INTEGER array of dimension(M)
C              On entry P5 specifies the values of a permutation matrix
C              Unchanged on exit.
C
C     P6        - INTEGER array of dimension(M)
C              On entry P6 specifies the values of a permutation matrix
C              Unchanged on exit.
C
C     VDIAG     - DOUBLE PRECISION array of dimension (M)
C              On entry VDIAG contains the values of the diagonal
C               of the preconditioner.
C              On exit VDIAG contains the square root of initial values.
C
C     B,X,EPS,ITMAX,ERR,ITER  ... as in subroutine CG.
C
C     IERROR    - INTEGER
C              On exit IER contains the values of error flag as shown in
C              subroutine CG
```

```
C
C     Q R S W P PT1 - DOUBLE PRECISION arrays of DIMENSION (M)
C             Work areas used in the Conjugate gradient methods
C
C     WORK     - DOUBLE PRECISION array of DIMENSION(*)
C             Work area
C
C     LWORK    - INTEGER
C             On entry LWORK specifies the dimension of WORK
C             LWORK must be greater than zero.
C             Unchanged on exit
C
      DOUBLE PRECISION  EPS, ERR
      INTEGER           IERROR, ITER, ITMAX, M
C     .. Array Arguments ..
      DOUBLE PRECISION  B(*), S(*), Q(*), R(*), A(*), VDIAG(*),
     *                  L(*), U(*), W(*), X(*),WORK(*), PT1(*),P(*)
      INTEGER           IA1(*),IA2(*),IU1(*),IU2(*),IL1(*),IL2(*),
     *                  INFORM(*),P1(*),P2(*),P3(*),P4(*),P5(*),P6(*)
      CHARACTER*5       DESCRA(10),DESCRL(10),DESCRU(10)

C     .. Local Scalars ..

      DOUBLE PRECISION  ALFA, BETA, RHO, RH1, EPSTOL, SIGMA, ONE
      DOUBLE PRECISION  ZERO
      INTEGER           I, K, IREST, IT, LDB, LDC
      CHARACTER         TRANSA,DIAGL,DIAGU
C     .. External Subroutines ..
      EXTERNAL DSPMM, DSPSM
C     .. External Functions ..
      DOUBLE PRECISION  DDOT
      EXTERNAL          DDOT
C     .. Intrinsic Functions ..
      INTRINSIC         DABS, DSQRT
C     .. Data statements ..
      DATA              EPSTOL/1.D-35/
C     .. Executable Statements ..

C     STEP 0: INITIALIZATION
C
C     Sparse BLAS parameters set
C

      K      =  1
      TRANSA = 'N'
      DIAGL  = 'R'
      DIAGU  = 'L'
      ONE    = 1.D0
```

```
      ZERO   = 0.D0
      LDB    = M
      LDC    = M
C
C     Method parameter
C
      ITER  = 1
      IREST = 0
      BETA  = 0.D0
      ALFA  = 0.D0
      RHO   = 0.D0
      DO 10 I = 1, M
         PT1(I) = 0.D0
         VDIAG(I) = DSQRT(VDIAG(I))
 10   CONTINUE

C
C     X = initial guess
C     R = A*X
C     R = B - R
C
      CALL DSPMM(TRANSA,M,M,K,ONE,P1,DESCRA,A,IA1,IA2,INFORM,P2,
     *           X,LDB,ZERO,R,LDC,WORK,LWORK,IERROR)
      IF (IERROR .NE. 0) RETURN
      DO 20 I = 1, M
         R(I) = B(I) - R(I)
   20 CONTINUE
C
C     S   = (L-1)*R
C     RHO = (S,S)
C

      CALL DSPSM(TRANSA,M,K,ONE,DIAGL,VDIAG,P3,DESCRL,L,IL1,IL2,
     *           INFORM,P4,R,LDB,ZERO,S,LDB,WORK,LWORK,IERROR)
      IF (IERROR .NE. 0) RETURN
      RHO = DDOT(M,S,1,S,1)
C
C
C     Conjugate gradient iteration
C
      DO 100 IT = ITER, ITMAX - 1
C
C        W = (L-1T)*S
C        P = W + BETA*PT1
C        Q = A*P
C        SIGMA = (P,Q)
C
         CALL DSPSM(TRANSA,M,K,ONE,DIAGU,VDIAG,P5,DESCRU,U,IU1,IU2,
```

```
     *              INFORM,P6,S,M,ZERO,W,M,WORK,LWORK,IERROR)
           IF (IERROR .NE. 0) RETURN
           DO 30 I = 1, M
              P(I) = W(I) + BETA*PT1(I)
 30        CONTINUE
           CALL DSPMM(TRANSA,M,M,K,ONE,P1,DESCRA,A,IA1,IA2,INFORM,P2,
     *              P,LDB,ZERO,Q,LDC,WORK,LWORK,IERROR)
           IF (IERROR .NE. 0) RETURN
           SIGMA = DDOT(M,P,1,Q,1)
           IF (DABS(SIGMA).LT.EPSTOL) THEN
C             ERROR ON SIGMA
              IERROR=-4
              RETURN
           END IF
C
C          X    = X + ALFA*PT1
C          ALFA = RHO/SIGMA
C          R    = R - ALFA*Q
C          S    = (L-1)*R
C          RHO = (S,S)
C
C
           DO 40 I = 1, M
              X(I) = X(I) + ALFA*PT1(I)
  40       CONTINUE
           ALFA = RHO/SIGMA
           DO 50 I = 1, M
              R(I) = R(I) - ALFA*Q(I)
  50       CONTINUE
           CALL DSPSM(TRANSA,M,K,ONE,DIAGL,VDIAG,P3,DESCRL,L,IL1,IL2,
     *              INFORM,P4,R,LDB,ZERO,S,LDB,WORK,LWORK,IERROR)
           IF (IERROR .NE. 0) RETURN
           RH1 = DDOT(M,S,1,S,1)
           ERR = DDOT(M,R,1,R,1)
C
C          Verify if method has converged
C          This is a very crude test and might be supplemented or replaced
C          in a library code.
C
           IF (ERR.LE.EPS) THEN
              DO 60 I = 1, M
                 X(I) = X(I) + ALFA*P(I)
  60          CONTINUE
              RETURN
           ELSE
C
C             Set parameters for next iteration
C
```

```
            ITER = ITER + 1
            BETA = RH1/RHO
            RHO  = RH1
            DO 70 I = 1, M
               PT1(I) = P(I)
 70         CONTINUE
         END IF
 100 CONTINUE
C
C     Method failed to converge in ITMAX iterations
C
      IERROR = -2
      RETURN
C
      END
```

References

[1] Agarwal, R. C., Gustavson, F. G., and Zubair, M. A high performance algorithm using pre-processing for the sparse matrix-vector multiplication. *In Proceedings of Supercomputing '92*, Minneapolis, MN. Nov 16-20, 1992.

[2] Amestoy, P. R., Daydé, M., and Duff, I. S. Use of Level 3 BLAS in the solution of full and sparse linear equations. *In High Performance Computing. Edited by J-L. Delhaye and E. Gelenbe.* North-Holland, 19–31, 1989.

[3] Anderson, E. C. and Saad, Y. *Solving sparse triangular systems on parallel computers.* Technical Report 794, University of Illinois, CSRD, Urbana, IL, 1988.

[4] Ashby, S. F. and Seager, M. K. A proposed standard for iterative solvers. Technical report 102860, LLNL, Livermore, CA, 1990.

[5] CMSSL for CM Fortran. Version 3.0. Thinking Machines Corporation, October 1992.

[6] Demmel, J. W., Heath, M. T., and Van der Vorst, H. A. *Parallel Numerical Linear Algebra* Acta Numerica 1993, Cambridge University Press, 1993.

[7] Dodson, D. S., Grimes, R. G., and Lewis, J. G. Sparse extensions to the Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 17:253–263, 1991.

[8] Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 16:1–17, 1990.

[9] Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. J. An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 14:1-17, 1988.

[10] Duff, I. S. A survey of sparse matrix research. *Proceedings of the IEEE.*, 65:500-535, 1977.

[11] Duff, I. S. Full matrix techniques in sparse Gaussian elimination. *In Numerical Analysis Proceedings, Dundee 1981. Lecture Notes in Mathematics 912. Edited by G.A. Watson*, Springer-Verlag, 71–84, 1981.

[12] Duff, I. S., Erisman A. M., and Reid, J. K.. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[13] Duff, I. S., Grimes, R. G., and Lewis, J. G. Sparse matrix test problems. *ACM Trans. Math. Softw.*, 15:1–14, 1989.

[14] Erhel, J. Sparse matrix multiplication on vector computers. *Int J High Speed Comput*, 2:101–116, 1990.

[15] IBM Engineering and Scientific Subroutine Library. Guide and Reference. Release 4. IBM Corporation, 1990.

[16] Heroux, M. A proposal for a sparse BLAS toolkit. Technical Report TR/PA/92/90, CERFACS, Toulouse, France, 1992.

[17] Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.*, 5:308–323, 1979.

[18] Oppe, T. C., Joubert, W., and Kincaid, D. R. *NSPCG User's guide. A Package for Solving Large Linear Systems by Various Iterative Methods.* Technical Report, The University of Texas at Austin, 1988.

[19] Oppe, T. C. and Kincaid, D. R. The performance of ITPACK on vector computers for solving large sparse linear systems arising in sample oil reservoir simulation problems. *Communications in Applied Numerical Methods*, 2:1–7, 1986.

[20] Osterby O. and Zlatev, Z. *Direct methods for sparse matrices.* Springer Verlag, New York, 1983.

[21] Paolini, G. V. and Radicati di Brozolo, G. Data structures to vectorize CG algorithms for general sparsity patterns. *BIT*, 29:703–718, 1989.

[22] Saad, Y. Krylov subspace methods on supercomputers. *SIAM J. Scient. Stat. Comput.*, 10:1200–1232, 1989.

[23] Young, D. M., Oppe, T. C., Kincaid, D. R., and Hayes, L. J. *On the use of vector computers for solving large sparse linear systems.* Technical Report CNA-199, Center for Numerical Analysis, University of Texas at Austin, Austin, Texas, 1985.

[24] Zlatev, Z., Schaumburg, K., and Wasniewski, J. A testing scheme for subroutines solving large linear problems. *Computers and Chemistry*, 5:91–100, 1981.