D R A F T
Document for the Basic Linear Algebra Subprograms (BLAS) Standard

Basic Linear Algebra Subprograms Technical (BLAST) Forum

April 25, 1998

Contents

Acknowledgments **vi**

1 Introduction **2**
1.1 Summary . . . 2
1.2 Background, Motivation and History . . . 2
1.3 Organization of the Document . . . 3
1.4 Nomenclature and Conventions Used in this Document . . . 3
1.5 Overall Functionality . . . 3
1.5.1 Justifications for extensions to the BLAS . . . 3
1.5.2 Scalar and Vector Operations . . . 5
1.5.3 Matrix Vector Operations . . . 5
1.5.4 Matrix Operations . . . 7

2 Dense and Banded BLAS **10**
2.1 Overview . . . 10
2.2 Functionality . . . 10
2.2.1 Justifications for extensions to the BLAS . . . 10
2.2.2 Scalar and Vector Operations . . . 11
2.2.3 Matrix Vector Operations . . . 11
2.2.4 Matrix Operations . . . 13
2.2.5 Multiple Instance Cases . . . 14
2.3 Language Independent Specification . . . 17
2.3.1 Reduction Operations . . . 18
2.3.2 Vector Operations . . . 20
2.3.3 Data Movemement with Vectors . . . 22
2.3.4 Matrix-Vector Operations . . . 22
2.3.5 Matrix Operations . . . 25
2.3.6 Matrix-Matrix Operations . . . 28
2.3.7 Data Movement with Matrices . . . 31
2.3.8 Multiple Instances . . . 32
2.4 Language Dependent Spectification . . . 34
2.5 Reference Implementation . . . 34
2.6 Benchmarking, Verification, and Validation . . . 34
2.6.1 Addendum ? . . . 34

3 Sparse BLAS **19**
3.1 Overview . . . 19
3.2 Sparse BLAS Functionality . . . 20
3.2.1 Operations . . . 20
3.2.2 Rationale for Functionality . . . 20
3.2.3 Use on a Distributed Memory Machine . . . 22
3.2.4 Targeted Data Structures . . . 22
3.3 Sparse BLAS Level 2 and Level 3 Toolkit Level Interface . . . 23
3.3.1 Naming Conventions for the Sparse BLAS Toolkit Interface . . . 23
3.3.2 Argument Conventions for Sparse BLAS Toolkit Interface . . . 24
3.4 Storage Conventions for Sparse Matrices . . . 28
3.4.1 Definitions and Notation . . . 28
3.4.2 Descriptor Arguments . . . 30
3.4.3 Point Entry Data Structures . . . 31
3.4.4 Constant Block Entry Data Structures . . . 38
3.4.5 Variable Block Entry Data Structures . . . 40
3.5 Issues for Discussion . . . 43
3.5.1 Language Compatibility . . . 43
3.5.2 Constant Block Entry Details . . . 43
3.5.3 Variable Block Column Data Structure . . . 44
3.5.4 Non-unit Block Diagonal Entries for Triangular Matrices . . . 44
3.5.5 Work Arrays and Dimensions . . . 44
3.5.6 Structurally-symmetric Nonsymmetric Matrices . . . 44
3.5.7 Sparse Matrix Update Kernel . . . 44
3.5.8 General Sparse Permutation Kernels . . . 45
3.5.9 Sparse Matrix-Sparse Matrix Multiplication . . . 45
3.6 Implementation Notes . . . 45
3.6.1 The NIST Sparse BLAS . . . 45
3.6.2 The NIST Fortran Sparse BLAS . . . 46

4 Distributed-memory Dense BLAS Interface **47**
4.1 Introduction to Distributed-memory Dense BLAS . . . 47
4.1.1 Overview and Goals . . . 47
4.1.2 Who Should Use This Standard ? . . . 48
4.1.3 What Platforms Are Targets for Implementation ? . . . 49
4.1.4 What Is Included in the Standard ? . . . 49
4.1.5 What Is Not (Yet ?) Included in the Standard ? . . . 49
4.1.6 Organization of This Chapter . . . 50

5 Extra Precise BLAS (XBLAS) **51**
5.1 Section 1: Introduction . . . 51
5.2 Section 2: Example Applications . . . 53
5.2.1 Example 2.1: Iterative Refinement of Linear Systems . . . 54
5.2.2 Example 2.2: Iterative refinement of other linear algebra problems . . . 55
5.2.3 Example 2.3: Solving Ill-conditioned Triangular Systems . . . 56
5.2.4 Example 2.4: Eigenvalue calculation using Bisection . . . 57
5.2.5 Example 2.5: Orthogonal Eigenvector calculation with Inverse Iteration . . . 58
5.2.6 Example 2.6: Eigenvalue calculation using Lanczos . . . 58

5.2.7 Example 2.7: Using Normal Equations instead of QR for Least Squares . . . 59
5.2.8 Example 2.8: Mixed Real and Complex Arithmetic 59
5.2.9 Example 2.9: Avoiding Pivoting in Gaussian Elimination 60
5.3 Section 3: High-Precision Arithmetic Implementation Technicalities 60
5.4 Section 4: Naming Scheme and Calling Sequence 63
5.5 Section 5: Environmental Enquiries 66
5.6 Section 6: Prioritized List of Extra Precision BLAS 68
5.7 Section 7: Automatic Generation of These Routines 68
5.8 Section 8: Automatic Testing of These Routines 69
5.9 Section 9: Sample Implementations 69
5.10 Section 10: Sample Applications 69
5.11 Section 11: Comparison to a previous proposal 69

A Journal of Development **71**
A.1 LiteBLAS 72
A.2 Interval BLAS 73

B LegacyBLAS **74**
B.1 New routines 74
B.2 C interface to legacy BLAS 74
B.2.1 Naming scheme 74
B.2.2 Indices 74
B.2.3 Character arguments 75
B.2.4 Handling of complex data types 75
B.2.5 Return values of complex functions 76
B.2.6 Array arguments 77
B.2.7 C interface include file 78
B.2.8 Error checking 78
B.2.9 Rules for obtaining the C interface from the Fortran77 79
B.3 cblas.h include file 79
B.4 Using Fortran77 BLAS to support row-major BLAS operations 91
B.4.1 Level 2 BLAS 92
B.4.2 Level 3 BLAS 95
B.4.3 Banded routines 97
B.4.4 Packed routines 99
B.5 Fortran 95 BLAS and an Interface to the Legacy BLAS 100
B.5.1 Introduction 100
B.5.2 Design of the Fortran 95 Interfaces 100
B.5.3 Key Arguments 101
B.5.4 The Fortran 95 Interfaces 101
B.5.5 Error Checking 106
B.5.6 Some Questions 106
B.5.7 A Fortran 95 Thin Interface 107
B.5.8 Appendix: Specific procedures for gemm 108
B.6 gemm – Current Version 111
B.6.1 Description 111
B.6.2 Usage 111
B.6.3 Arguments 111

B.6.4 Examples of usage . . . 113
B.7 gemm – Alternative Version . . . 114
B.7.1 Description . . . 114
B.7.2 Usage . . . 114
B.7.3 Arguments . . . 114
B.7.4 Examples of usage . . . 116

Bibliography **117**

The discussion in each chapter is not limited only to the points listed, any related topics can be included at the discretion of the chapter authors. Final acceptance of each section will be decided at the next meeting using *Robert's Rules*. See BLAST Forum homepage for more detailed rules.

Chapter 2

Dense and Banded BLAS

2.1 Overview

This Chapter provides a basis for the dense and banded BLAS routines. Included are routines that currently exist in the Level 1, 2, and 3 BLAS, as well extensions and new routines. This chapter covers scalars, vectors and dense, banded, and triangular matrices but not sparse data structures.

2.2 Functionality

This section summarizes, in tabular form, the discussions of the Functionality Working Group for dense and banded matrices.

In this version of the document, the functionality is categorized more or less along traditional BLAS lines, but the Forum discussed other schemes and it may be appropriate to categorize the functionality in more than one way in the final document.

If what you need is not listed here, please comment and if possible join the Forum, to let us know what you need.

2.2.1 Justifications for extensions to the BLAS

In the original BLAS, each level was categorised by the type of operation; Level 1 covered scalar and vector operations, Level 2 covered matrix vector operations, while Level 3 covered matrix-matrix operations. The functionality tables in this document are categorised in a similar manner, but with some additional categories to cover operations which it was realized were missing in the original BLAS.

The motivation for the new operations is proven functionality. No function was to be added because it just neatly fit into a framework and no subroutine was to be discarded because it did not fit into a particular framework.

Many of the new functions are auxiliary routines in LAPACK [6]. Only after the LAPACK project was begun was it realized that there were operations like the matrix copy routine, the computation of a norm of a matrix and the generation of Householder transformations that occurred so often that it was wise to make separate subroutines for them.

A second group of these extensions extended the functionality of some of the existing BLAS. For example, the Level 3 BLAS for the rank k update of a symmetric matrix only allows a positive update, which means that it cannot be used for the reduction of a symmetric matrix to tridiagonal form (to facilitate the computation of the eigensystem of a symmetric matrix), or for the decomposition of an indefinite matrix, or for a quasi-Newton update in an optimization routine.

Some of the new BLAS might be considered special cases of the original BLAS. The number of Level 2 and 3 BLAS was purposely kept small, but each had a number of parameters that could extend their functionality. Most of these parameters in practice were either 1.0 or 0.0. Adding these parameters, rather than specifying different subroutines increased the opportunities for the user to make mistakes and increased the amount of overhead of a call to the routine. In this document many of the functions are presented in their general case, but we make recommendations about special cases in the Language Independent Specification chapter.

The extensions considered in the section on multiple instances can be thought of as multiple instances of the Level 1 BLAS. They tend to arise when working with narrow banded and sparse or structured matrices. The canonical example is the decomposition of many narrow banded matrices simultaneously. This type of problem arises when finding several eigenvectors using inverse iteration, when using parallel bisection to determine the eigenvalues of a symmetric matrix, when solving two dimensional separable partial differential equations or preconditioning an iterative method for a nonseparable differential equation. Certainly if one was on a shared memory multi-processor computer using a compiler with compiler directives one could specify that various Level 1 BLAS like IxAMAX, xSWAP and xAXPY be done simultaneously. However, when one is not in this situation and one wishes to use portable code, then the existence of a BLAS to cover this situation is helpful.

Other contenders for multiple instance routines are routines for generating and applying Givens transformation, used for example to reduce a banded matrix to tridiagonal form in order to solve the symmetric eigenvalue problem. Simultaneous Givens transformations are also useful when solving the tridiagonal positive definite symmetric generalized eigenvalue problem and when using the QR algorithm to solve the standard tridiagonal symmetric eigenvalue problem.

For large matrix problems, it might not be feasible or desirable to store the full matrix. In this situation one cannot simply use the matrix-vector multiplication routines in the present BLAS. However, one may be able to form that product with some simultaneous inner product routines because they have some repeated submatrices. This feature is particular true of Toeplitz and Hankel matrices used in signal and image processing.

2.2.2 Scalar and Vector Operations

This sub-section gives the operations corresponding to the traditional Level 1 BLAS operations. Table 2.1 lists the scalar and vector reduction operations, Table 2.2 lists the vector operations, and Table 2.3 lists those vector operations that involve only data movement. The operations are intended to apply to both real and complex arguments. For the sake of compactness the complex operators are usually omitted, so that whenever a transpose operation is given the conjugate transpose should also be assumed for the complex case.

2.2.3 Matrix Vector Operations

This sub-section gives the operations corresponding to the traditional Level 2 BLAS operations. Table 2.4 lists the matrix-vector operations. The operations are intended to apply to both real and complex arguments. The matrix T specifically represents a triangular matrix, either upper or lower, which can be unit or non-unit triangular.

D represents at least a diagonal matrix, and maybe also a 2×2 blocked diagonal or a tridiagonal matrix. Q represents an orthogonal (unitary) matrix.

Inner product	$r \leftarrow x^T y$,
	$r \leftarrow r + x^T y$
	$r \leftarrow \beta r + \alpha x^T y$
Vector norms	$r \leftarrow \lvert\lvert x \rvert\rvert_1$,
	$r \leftarrow \lvert\lvert x \rvert\rvert_\infty$
	$r \leftarrow \lvert\lvert x \rvert\rvert_2$,
Sum	$r \leftarrow \sum_i x_i$
Location of max value	$i = \arg\max_i \lvert x_i \rvert$
Location of min value	$i = \arg\min_i \lvert x_i \rvert$
Min value & location	$i, x_i, ; i = \arg\min_i x_i$
Min absolute value & location	$i, \lvert x_i \rvert, i = \arg\min_i \lvert x_i \rvert$
Max value & location	$i, x_i, ; i = \arg\max_i x_i$
Max absolute value & location	$i, \lvert x_i \rvert, i = \arg\max_i \lvert x_i \rvert$
Sum of squares	$(a, b) \leftarrow \sum x_i^2$,
Generate Givens rotation	$(c, s) \leftarrow \mathrm{rot}(a, b)$
Generate Jacobi rotation	$(c, s) \leftarrow \mathrm{jrot}(a, b)$
Generate Householder transform	$(\alpha, x, \tau) \leftarrow \mathrm{house}(\alpha, x)$,
	$H = I - \alpha u u^T$

Table 2.1: Reduction Operations

Zero	$x \leftarrow 0$
Constant	$x \leftarrow \alpha e$
Conjugate	$x \leftarrow \bar{x}$
Conjugate copy	$y \leftarrow \bar{x}$
Scale	$x \leftarrow \alpha x$,
	$y \leftarrow \alpha x$
Sum	$y \leftarrow x + y$,
	$w \leftarrow x + y$
axpy & axpy like	$y \leftarrow \alpha x + y$,
	$w \leftarrow \alpha x + y$
	$y \leftarrow \alpha x + \beta y$,
	$w \leftarrow \alpha x + \beta y$
Absolute value	$x_i \leftarrow \lvert x_i \rvert$
	$y_i \leftarrow \lvert x_i \rvert$
Elementwise multiply	$x_i \leftarrow x_i y_i$,
	$w_i \leftarrow x_i y_i$
Elementwise divide	$x_i \leftarrow x_i / y_i$,
	$w_i \leftarrow x_i / y_i$
Apply plane rotation	$\begin{pmatrix} x & y \end{pmatrix} \leftarrow \begin{pmatrix} x & y \end{pmatrix} R$

Table 2.2: Vector Operations

For the sake of compactness the complex operators are omitted, so that whenever a transpose operation is given both the conjugate and conjugate transpose should also be assumed for the complex case.

Copy	$y \leftarrow x$
Swap	$y \leftrightarrow x$
Sort into order & return index vector	$(iv, y) \leftarrow \mathrm{sort}(x)$
Sort according to index vector	$y \leftarrow \mathrm{sort}(x, iv)$

Table 2.3: Data Movement with Vectors

In the tables the following notation is used to indicate when an operation applies to general, symmetric, banded and orthogonal matrices:

GE General matrix

SY Symmetric (including Hermitian and complex symmetric)

GB General band

SB Symmetric band (including Hermitian and complex symmetric)

TR Triangular

TB Triangular band

OR Orthogonal (including unitary)

Matrix vector product	$x \leftarrow Ax,\ x \leftarrow A^T x$	GE, SY, GB, SB, TR, TB
	$y \leftarrow Ax,\ y \leftarrow A^T x$	GE, SY, GB, SB
	$y \leftarrow \alpha Ax + \beta y,\ y \leftarrow \alpha A^T x + \beta y$	GE, SY, GB, SB
Triangular solve	$x \leftarrow T^{-1}x,\ x \leftarrow T^{-T}x$	TR, TB
Rank one updates	$A \leftarrow \beta xy^T + \alpha A$	GE
and symmetric ($A = A^T$)	$A \leftarrow \beta xy^T + A$	GE
rank one & two updates	$A \leftarrow \beta xx^T + \alpha A$	SY
	$A \leftarrow \beta xx^T + A$	SY
	$A \leftarrow (\beta x)y^T + y(\beta x)^T + \alpha A$	SY
	$A \leftarrow (\beta x)y^T + y(\beta x)^T + A$	SY
Application of Q from	$x \leftarrow Qx,\ x \leftarrow Q^T x$	OR
QR factorization	$y \leftarrow Q^T x\ y \leftarrow Q^T x$	OR

Table 2.4: Matrix Vector Operations

2.2.4 Matrix Operations

This sub-section gives the operations corresponding to the traditional Level 3 BLAS operations, as well as a number of additional single matrix operations. Table 2.5 lists single matrix operations and matrix operations that involve $O(n^2)$ operations, Table 2.6 lists the $O(n^3)$ matrix-matrix operations and Table 2.7 lists those matrix operations that involve only data movement. The operations are intended to apply to both real and complex arguments. The matrix T specifically represents a triangular matrix, either upper or lower, which can be unit or non-unit triangular.

D represents at least a diagonal matrix, and maybe also a 2×2 blocked diagonal or a tridiagonal matrix. Q represents an orthogonal (unitary) matrix.

For the sake of compactness the complex operators are omitted, so that whenever a transpose operation is given both the conjugate and conjugate transpose should also be assumed for the complex case.

GE General matrix

SY Symmetric (including Hermitian and complex symmetric)

GB General band

SB Symmetric band (including Hermitian and complex symmetric)

TR Triangular

TB Triangular band

OR Orthogonal (including unitary)

Matrix set	α – on diagonal , β – off-diagonal	GE, GB, SY, SB, TR, TB
Matrix norms	$r \leftarrow \|\|A\|\|_1, r \leftarrow \|\|A\|\|_F, r \leftarrow \|\|A\|\|_\infty$	GE, GB, SY, SB, TR, TB
Maximum absolute value	$r \leftarrow \max \|a_{i,j}\|$	GE, GB, SY, SB, TR, TB
Matrix scaling	$A \leftarrow \alpha A$	GE, GB, SY, SB, TR, TB
Diagonal scaling	$A \leftarrow DA,\ A \leftarrow AD,\ A \leftarrow D_1 A D_2$	GE, GB, TR, TB
	$A \leftarrow DAD$	SY, SB
	$A \leftarrow A + BD$	GE, GB
Matrix add	$A \leftarrow A + B,\ C \leftarrow A + B$	GE, GB, SY, SB, TR, TB
Matrix add and scale	$A \leftarrow \alpha A + \beta B,\ C \leftarrow \alpha A + \beta B$	GE, GB, SY, SB, TR, TB
	$A \leftarrow \alpha A + \beta B^T$	GE, GB, SY, SB, TR, TB

Table 2.5: Matrix Operations – $O(n^2)$

2.2.5 Multiple Instance Cases

Most of the BLAS tend to cover operations dealing with dense matrices. The BLAS covered here tend to be most useful when dealing with narrow banded matrices or several narrow-banded matrices or sparse matrices that have particular structures . Most of the operations are generalizations are level-1 BLAS but involve many vectors.

The techniques are indirectly relevant to other denser and random sparse matrix problems. Most ways of solving eigensystem problems(e.g. Lanczos or QR), whether they be dense or sparse, reduce matrices to narrow banded problems and then to find the eigenvalues/eigenvectors of these highly structured matrices. Also some important applications, like the solution of Laplace's equation, or other separable elliptical partial differential equations can be solved using fast direct methods, which reduce a large banded matrix to several smaller narrow banded problems, which can be solved simultaneously.

The new operations fall mainly into 2 categories:

(A) Those involved in solving many (probably banded) linear systems simultaneously or many inner products.

Matrix matrix product	$A \leftarrow AB,\ A \leftarrow A^TB,\ A \leftarrow AB^T,\ A \leftarrow A^TB^T$	GE, SY
	$C \leftarrow \alpha AB + \beta C,\ C \leftarrow \alpha A^TB + \beta C$	GE, SY
	$C \leftarrow \alpha AB^T + \beta C,\ C \leftarrow \alpha A^TB^T + \beta C$	GE, SY
Triangular multiply	$B \leftarrow \alpha TB,\ B \leftarrow \alpha BT$	TR
and solve	$B \leftarrow \alpha T^TB,\ B \leftarrow \alpha BT^T$	TR
	$B \leftarrow \alpha T^{-1}B,\ B \leftarrow \alpha BT^{-1}$	TR
	$B \leftarrow \alpha T^{-T}B,\ B \leftarrow \alpha BT^{-T}$	TR
Symmetric rank k & $k2$	$C \leftarrow \alpha AA^T + \beta C,\ C \leftarrow \alpha A^TA + \beta C$	GE
updates ($C = C^T$)	$C \leftarrow \alpha ADA^T + \beta C,\ C \leftarrow \alpha A^TDA + \beta C$	GE
	$C \leftarrow (\alpha A)B^T + B(\alpha A)^T + \beta C$	GE
	$C \leftarrow (\alpha AD)B^T + B(\alpha AD)^T + \beta C$	GE
Application of Q, where Q	$X \leftarrow QX,\ X \leftarrow XQ,\ X \leftarrow Q^TX,\ X \leftarrow XQ^T$	OR
is a sequence of Householder	$X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ$	OR
transformations	$X \leftarrow QXQ^T,\ X \leftarrow QXQ^T$	OR
	$X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ$	OR ($X = X^T$)
	$X \leftarrow QXQ^T,\ X \leftarrow QXQ^T$	OR ($X = X^T$)
Application of Q, where Q	$X \leftarrow QX,\ X \leftarrow XQ,\ X \leftarrow Q^TX,\ X \leftarrow XQ^T$	OR
is a sequence of plane	$X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ$	OR
rotations	$X \leftarrow QXQ^T,\ X \leftarrow QXQ^T$	OR
	$X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ$	OR ($X = X^T$)
	$X \leftarrow QXQ^T,\ X \leftarrow QXQ^T$	OR ($X = X^T$)

Table 2.6: Matrix Matrix Operations

Matrix copy	$B \leftarrow A$	GE, GB, SY, SB, TR, TB
Matrix transpose	$A \leftarrow A^T,\ B \leftarrow A^T$	GE, GB
Get and put submatrix	$B \leftarrow a_{I,J}$	GE, GB, SY, SB
	$b_{I,J} \leftarrow A,\ b_{I,J} \leftarrow A + b_{I,J}$	GB, SY
Permute Matrix	$A \leftarrow PA,\ A \leftarrow AP$	GE
	$A \leftarrow QAP$	GE, SY ($Q = P^T, P = Q^T$)
Diagonal extract	$d \leftarrow \mathrm{diag}(A)$	GE, GB, SY, SB, TR, TB

Table 2.7: Data Movement with Matrices

Multiple inner products	$r_i \leftarrow x^Ty[i]$
Multiple maximum abs value	$i_j = argmax_i \lvert x_ij \rvert$
Generate muliple Givens rotation	$(c[i], s[i]) \leftarrow \mathrm{rot}(a[i], b[i])$
Multiple AXPYs	$y[i] \leftarrow a_ix + y[i]$
Apply multiple plane rotation	$(\ x[i] \quad y[i]\ ) \leftarrow (\ x[i] \quad y[i]\ )R[i]$

Table 2.8: Multiple Instance

These arise when solving separable elliptical using such techniques as generalized marching [8] or collocation [10] or Galerkin [38]. They also are found when determining eigenvectors of banded systems,once the eigenvalues have been found, using inverse iteration.

If one is on a multiprocessor system, one may wish to give each system to a particular processor and do a coarse grain separation of operations. When one can only use one processor, the BLAS given below might be useful.

(1)Multiple AXPY-Level 1

```
Do i =1,n
   y(iy)=y(iy)+a(ia)*x(ix)
   ix=ix+incx
   iy=iy+incy
   ia=ia+inca
```

Note: The multiple AXPY differs from the traditional AXPY because of index on a.

(2)Multiple Axpy-Level 2

$$X = X + YD$$

where D is a diagonal matrix stored in a vectors

(3) Multiple SWAP

```
do j =1,m
   do i =1,n
      temp =x(ix,jx)
      x(ix,jx)=y(iy,jy)
      y(iy,jy)=temp
      ix=ix+iincx
      iy=iy+iincy
   jx=jx+jincx
   jy=jy+jincy
```

This could be used a general matrix swap.

(4) Multiple ISAMAX,ISAMIN

Given m vectors of length n determine return m indices/values which given the max/min of these vectors

(5) Multiple SDOT given 2m vectors of length n .

do i=1,m
  $a_i = x_i^T y_i$

This operation is like matrix-vector multiplication but not necessarily using contiguous storage

(B) Those operations involving Givens transformations-

These operation are found in the auxiliary routines SLASR, SLARGV, SLARTV in LAPACK and are the bases of the eigenvalue solvers, especially when eigenvectors are required.

(1)Multiple Givens Generation

Generate simultaneously different Givens rotations each of which will annihilate an element in a 2 vector: $(c_i, s_i) \leftarrow \mathrm{rot}(a_i, b_i)$.

Note: This is found in the auxiliary subroutine SLARGV which is used during the reduction of a banded matrix to a tridiagonal matrix and the transformation of a banded symmetric generalized eigenvalue problem($Ax = \lambda Bx$) with positive definite B to a banded standardized eigenvalue problem.

(2)Simultaneous Givens applications

Simultaneous application of Givens transformations to different columns/rows of a matrix as in

Do $i = i_1$ to i_2 by k
  Do $j = 1, k$
    $t = a_{j,i}c_i - s_i a_{j,i+1}$
    $a_{j,i+1} = s_i a_{j,i} + c_i a_{j,i+1}$
    $a_{j,i} = t$

Note: Here k is greater than 1, so that one can interchange the inner and outer loop. This is a generalization of SLARGV ,which is missing the loop on j. It is used while reducing a banded matrix to tridiagonal form and while transforming a banded symmetric generalized eigenvalue problem($Ax = \lambda Bx$) with positive definite B to a bandard standardized eigengenvalue problem.

(3)Sequential Givens transformations

This is similar to (2) except the transformations cannot be done simultaneously.

Do $i = 1$ to m
  Do $j = 1, k$
    $t = a_{j,i}c_i - s_i a_{j,i+1}$
    $a_{j,i+1} = s_i a_{j,i} + c_i a_{j,i+1}$
    $a_{j,i} = t$

One may also run the loop backwards.

In this form we get the portion of DLASR which is called by the QR routine for finding the eigenvectors of a symmetric matrix and by the the singular vector routine to determine the singular vectors.

Another form of (3) involves using the first or last column as in

Do $i = 1$ to $m - 1$
  Do $j = 1, k$
    $t = a_{j,i}c_i - s_i a_{j,m}$
    $a_{j,m} = s_i a_{j,i} + c_i a_{j,m}$
    $a_{j,i} = t$

This type of operation would occur in least squares problems when adding data and in general occurs when updating factorizations in linear/nonlinear programming. See [33]

One may wish to extend these types of transformations to 3 planar transformations which are the core of the unsymmetric dense eigensolvers.

2.3 Language Independent Specification

In this section, the dense banded BLAS are specified using a language-independent notation. The arguments of procedure calls are marked as IN, OUT or INOUT. The meanings of these are:

- the call uses but does not update an argument marked IN,
- the call may update an argument marked OUT,

- the call both uses and updates an argument marked INOUT.

There is one special case – if an argument is a handle, say a pointer, to an object, say a scalar or an array, an this object is updated by the procedure call, then the argument is marked OUT. It is marked this way even though the handle itself is not modified – we use the OUT attribute to denote that what the handle *references* is updated.

The definition of the dense bnaded BLAS tries to avoid, to the largest possible extent, the use of INOUT arguments, because such use is error-prone, especially for scalar arguments.

Unless otherwise specified, an argument of type OUT or type INOUT cannot be aliased with any other argument passed to a dense banded BLAS procedure. An example of argument aliasing in C appears below. If we define a procedure like this,

```
void copyIntBuffer( int *pin, int *pout, int len )
{   int i;
    for( i=0; i<len; ++i ) *pout++ = *pin++;
}
```

then a call to it in the following code fragment has aliased arguments.

```
int a[10];
copyIntBuffer( a, a+3, 7 );
```

Although the C language allows this, such usage of dense banded BLAS procedures is forbidden unless otherwise specified. Note that Fortran prohibits aliasing of arguments.

2.3.1 Reduction Operations

At this stage, a single procedure name is sometimes used to specify more than one function. These functions will be distinguished later in the language dependent specification section. For example, the procedure below called BLAS_DOT(x, y, r) refers to the following functions $r \leftarrow x^T y$ and $r \leftarrow x^H y$.

BLAS_DOT(x, y, r) $r \leftarrow x^T y,\ r \leftarrow x^H y$

IN	x	a vector of length n
IN	y	a vector of length n
OUT	r	a scalar

BLAS_DOTPR(x, y, r) $r \leftarrow r + x^T y,\ r \leftarrow r + x^H y$

IN	x	a vector of length n
IN	y	a vector of length n
INOUT	r	a scalar

BLAS_ADOTPBR(α, x, y, β, r) $r \leftarrow \beta r + \alpha x^T y,\ r \leftarrow \beta r + \alpha x^H y$

IN	α	a scalar
IN	x	a vector of length n
IN	y	a vector of length n
IN	β	a scalar
INOUT	r	a scalar

BLAS_NORM(x, r) $r \leftarrow ||x||_1,\ r \leftarrow ||x||_2,\ r \leftarrow ||x||_\infty$

IN	x	a vector of length n
OUT	r	a scalar

BLAS_SUM(x, r) $r \leftarrow \sum_i x_i$

IN	x	a vector of length n
OUT	r	a scalar

Note: The functions computing the sum of absolute values (legacy BLAS ASUM) and the 'genuine' absolute value sum (LAPACK SUM1) for complex vectors are not included in the functionality tables.

BLAS_AMIN(x, imin) $imin = \arg\min_i |x_i|$

IN	x	a vector of length n
OUT	$imin$	a scalar

BLAS_AMAX(x, imax) $imax = \arg\max_i |x_i|$

IN	x	a vector of length n
OUT	$imax$	a scalar

BLAS_MINLOC(x, imin, x_{imin}) $imin, x_{imin}, ; imin = \arg\min_i x_i$

IN	x	a vector of length n
OUT	$imin$	a scalar
OUT	x_{imin}	a scalar

BLAS_AMINLOC(x, imin, x_{imin}) $imin, |x_{imin}|, imin = \arg\min_i |x_i|$

IN	x	a vector of length n
OUT	$imin$	a scalar
OUT	x_{imin}	a scalar

BLAS_MAXLOC(x, imax, x_{imax}) $imax, x_{imax}, ; imax = \arg\max_i x_i$

IN	x	a vector of length n
OUT	$imax$	a scalar
OUT	x_{imax}	a scalar

BLAS_AMAXLOC(x, imax, x_{imax}) $imax, |x_{imax}|, imax = \arg\max_i |x_i|$

IN	x	a vector of length n
OUT	$imax$	a scalar
OUT	x_{imax}	a scalar

Note: Whether the returned location is an offset or an index in the vector x is not specified by the procedures above. This issue will be resolved later in the language dependent section.
Note: The min and max absolute value and location functions return the absolute value of the min or max absolute value element. This loss of information is considered to be acceptable (these functions could have returned the element itself instead).

BLAS_SSQ(x, a, b) $(a, b) \leftarrow \sum x_i^2$, such that $a \cdot b^2 = \sum x^2$

IN	x	a vector of length n
OUT	a	a scalar
OUT	b	a scalar

BLAS_ROTG(a, b, c, s) $(c, s) \leftarrow \mathrm{rot}(a, b)$, such that ...

IN	a	a scalar
IN	b	a scalar
OUT	c	a scalar
OUT	s	a scalar

BLAS_JROTG(a, b, c, s) $(c, s) \leftarrow \mathrm{jrot}(a, b)$, such that ...

IN	a	a scalar
IN	b	a scalar
OUT	c	a scalar
OUT	s	a scalar

BLAS_HOUSEG(α, x, tau)

INOUT	α	a scalar
INOUT	x	a vector of length $n-1$
OUT	τ	a scalar

This function generates an elementary reflector H of order n, such that

$$H\begin{pmatrix} \alpha \\ x \end{pmatrix} = \begin{pmatrix} \beta \\ 0 \end{pmatrix}, H^{'}H = I.$$

where α and β are scalars, and x is an (n-1)-element vector. H is represented in the form

$$H = I - tau\begin{pmatrix} 1 \\ v \end{pmatrix}\begin{pmatrix} 1 & v^{'} \end{pmatrix}.$$

where tau is a scalar and v is a real (n-1)-element vector.

2.3.2 Vector Operations

BLAS_ZERO(x) $\qquad x \leftarrow 0$

OUT	x	a vector of length n

BLAS_SET(α, x) $\qquad x \leftarrow \alpha e$

IN	α	a scalar
OUT	x	a vector of length n

BLAS_CONJG(x) $\qquad x \leftarrow \bar{x}$

INOUT	x	a vector of length n

Note: The above function BLAS_CONJG is not in the functionality tables, but should probably be added.

BLAS_CONJGCOPY(x, y) $\qquad y \leftarrow \bar{x}$

IN	x	a vector of length n
OUT	y	a vector of length n

BLAS_SCAL(α, x) $\qquad x \leftarrow \alpha x$

IN	α	a scalar
INOUT	x	a vector of length n

BLAS_SCALCOPY(α, x, y) $\qquad y \leftarrow \alpha x$

IN	α	a scalar
IN	x	a vector of length n
OUT	y	a vector of length n

Note: Reverse scaling routines without under/overflow like LAPACK routine _RSCL ?

BLAS_SCALGROUP(α, I, x) $\qquad x_{I_i} \leftarrow a_i x_{I_i}$

IN	α	a scalar
IN	I	a vector of length p
INOUT	x	a vector of length n

Note: Is BLAS_SCALGROUP the same as: scale result of gather operation and then scatter ? Should this operation not be in the sparse chapter ?

BLAS_Y_XPY(x, y) $y \leftarrow x + y$

IN	x	a vector of length n
INOUT	y	a vector of length n

BLAS_W_XPY(x, y, w) $w \leftarrow x + y$

IN	x	a vector of length n
IN	y	a vector of length n
OUT	w	a vector of length n

BLAS_Y_AXPY(α, x, y) $y \leftarrow \alpha x + y$

IN	α	a scalar
IN	x	a vector of length n
INOUT	y	a vector of length n

BLAS_W_AXPY(α, x, y, w) $w \leftarrow \alpha x + y$

IN	α	a scalar
IN	x	a vector of length n
IN	y	a vector of length n
OUT	w	a vector of length n

BLAS_Y_AXPBY(α, x, β, y) $y \leftarrow \alpha x + \beta y$

IN	α	a scalar
IN	x	a vector of length n
IN	β	a scalar
INOUT	y	a vector of length n

BLAS_W_AXPBY(α, x, β, y, w) $w \leftarrow \alpha x + \beta y$

IN	α	a scalar
IN	x	a vector of length n
IN	β	a scalar
IN	y	a vector of length n
OUT	w	a vector of length n

BLAS_ABS(x) $x_i \leftarrow |x_i|$

INOUT	x	a vector of length n

BLAS_ABSCOPY(x, y) $y_i \leftarrow |x_i|$

IN	x	a vector of length n
OUT	y	a vector of length n

BLAS_X_XMULY(x, y) $x_i \leftarrow x_i y_i$

IN	y	a vector of length n
INOUT	x	a vector of length n

BLAS_W_XMULY(x, y) $w_i \leftarrow x_i y_i$

IN	x	a vector of length n
IN	y	a vector of length n
OUT	w	a vector of length n

BLAS_X_XDIVY(x, y) $x_i \leftarrow x_i/y_i$

IN	y	a vector of length n
INOUT	x	a vector of length n

Note: Error code is missing.

BLAS_W_XDIVY(x, y) $w_i \leftarrow x_i/y_i$

IN	x	a vector of length n
IN	y	a vector of length n
OUT	w	a vector of length n

Note: Error code is missing.

BLAS_ROT(c, s, x, y) $\begin{pmatrix} x & y \end{pmatrix} \leftarrow \begin{pmatrix} x & y \end{pmatrix} R$

IN	c	a scalar
IN	s	a scalar
INOUT	x	a vector of length n
INOUT	y	a vector of length n

2.3.3 Data Movemement with Vectors

BLAS_COPY(x, y) $y \leftarrow x$

IN	x	a vector of length n
OUT	y	a vector of length n

BLAS_SWAP(x, y) $y \leftrightarrow x$

INOUT	x	a vector of length n
INOUT	y	a vector of length n

BLAS_SORT(x, iv, y) $(iv, y) \leftarrow \mathrm{sort}(x)$

IN	x	a vector of length n
OUT	iv	a vector of length n
OUT	y	a vector of length n

BLAS_PERM(x, iv, y) $y \leftarrow \mathrm{sort}(x, iv)$

IN	x	a vector of length n
IN	iv	a vector of length n
OUT	y	a vector of length n

Note: The above sorting functions implicitly assume a unique sorting criterium. Do we want more than one sorting routine, or accept on input a user supplied sorting function ?

2.3.4 Matrix-Vector Operations

The storage formats are assumed below to language-dependent. As such, they will be discussed in the language dependent specification section. It follows that packed storage routines will also be discussed in the language dependent specification section. Similarly, it is impossible at this stage to determine the complexity of the following operations.

BLAS_GE_X_AX(trans, A, x) $x \leftarrow Ax$, $x \leftarrow A^T x$ or $x \leftarrow A^H x$

IN	$trans$	the transpose operator
IN	A	a matrix of order n
INOUT	x	a vector of length n

BLAS_SY_X_AX(A, x) $x \leftarrow Ax$, with $A = A^T$ or $A = A^H$

IN	A	a symmetric or Hermitian matrix of order n
INOUT	x	a vector of length n

BLAS_TR_X_AX(trans, A, x) $x \leftarrow Ax$, $x \leftarrow A^T x$ or $x \leftarrow A^H x$

IN	*trans*	the transpose operator
IN	A	a triangular matrix of order n with unit diagonal or not
INOUT	x	a vector of length n

Note: The triangular matrix vector multiply function above is missing in the functionality table.

BLAS_GE_Y_AX(trans, A, x, y) $y \leftarrow Ax$, $y \leftarrow A^T x$ or $y \leftarrow A^H x$

IN	*trans*	the transpose operator
IN	A	a matrix of order m-by-n
IN	x	a vector of length n in the non-transpose case, and m otherwise.
OUT	y	a vector of length m in the non-transpose case, and n otherwise.

BLAS_SY_Y_AX(A, x, y) $y \leftarrow Ax$, with $A = A^T$ or $A = A^H$

IN	A	a symmetric or Hermitian matrix of order n
IN	x	a vector of length n
OUT	y	a vector of length n

BLAS_TR_Y_AX(trans, A, x, y) $y \leftarrow Ax$, $x \leftarrow A^T x$ or $x \leftarrow A^H x$

IN	*trans*	the transpose operator
IN	A	a triangular matrix of order n with unit diagonal or not
IN	x	a vector of length n
OUT	y	a vector of length n

BLAS_GEMV(trans, α, A, x, β, y) $y \leftarrow \alpha Ax + \beta y$, $y \leftarrow \alpha A^T x + \beta y$ or $y \leftarrow \alpha A^H x + \beta y$

IN	*trans*	the transpose operator
IN	α	a scalar
IN	A	a matrix of order m-by-n
IN	x	a vector of length n in the non-transpose case, and m otherwise.
IN	β	a scalar
INOUT	y	a vector of length m in the non-transpose case, and n otherwise.

BLAS_SYMV(α, A, x, β, y) $y \leftarrow \alpha Ax + \beta y$, with $A = A^T$ or $A = A^H$

IN	α	a scalar
IN	A	a symmetric or Hermitian matrix of order n
IN	x	a vector of length n
IN	β	a scalar
INOUT	y	a vector of length n

BLAS_TRMV(trans, α, A, x, β, y) $y \leftarrow \alpha Ax + \beta y$, $y \leftarrow \alpha A^T x + \beta y$ or $y \leftarrow \alpha A^H x + \beta y$

IN	*trans*	the transpose operator
IN	α	a scalar
IN	A	a triangular matrix of order n with unit diagonal or not
IN	x	a vector of length n
IN	β	a scalar
INOUT	y	a vector of length n

BLAS_TRSV(trans, A, x) $x \leftarrow A^{-1}x$, $x \leftarrow A^{-T}x$ or $x \leftarrow A^{-H}x$

IN	*trans*	the transpose operator
IN	A	a triangular matrix of order n with unit diagonal or not
INOUT	x	a vector of length n

BLAS_AGER(β, x, y, α, A) $A \leftarrow \beta xy^T + \alpha A$, $A \leftarrow \beta xy^H + \alpha A$

IN	β	a scalar
IN	x	a vector of length n
IN	y	a vector of length n
IN	α	a scalar
INOUT	A	a matrix of dimension m-by-n

BLAS_GER(β, x, y, A) $A \leftarrow \beta xy^T + A$, $A \leftarrow \beta xy^H + A$

IN	β	a scalar
IN	x	a vector of length n
IN	y	a vector of length n
INOUT	A	a matrix of dimension m-by-n

BLAS_ASYR(β, x, α, A) $A \leftarrow \beta xx^T + \alpha A$ with $A = A^T$, $A \leftarrow \beta xx^H + \alpha A$ with $A = A^H$

IN	β	a scalar
IN	x	a vector of length n
IN	α	a scalar
INOUT	A	a symmetric or Hermitian matrix of order n

BLAS_SYR(β, x, A) $A \leftarrow \beta xx^T + A$ with $A = A^T$, $A \leftarrow \beta xx^H + A$ with $A = A^H$

IN	β	a scalar
IN	x	a vector of length n
INOUT	A	a symmetric or Hermitian matrix of order n

BLAS_ASYR2(β, x, y, α, A) $A \leftarrow \beta xy^T + \beta yx^T + \alpha A$ with $A = A^T$, $A \leftarrow \beta xy^H + y(\beta x)^H + \alpha A$ with $A = A^H$

IN	β	a scalar
IN	x	a vector of length n
IN	y	a vector of length n
IN	α	a scalar
INOUT	A	a symmetric or Hermitian matrix of order n

BLAS_SYR2(β, x, y, A) $A \leftarrow \beta xy^T + \beta yx^T + A$ with $A = A^T$, $A \leftarrow \beta xy^H + y(\beta x)^H + A$ with $A = A^H$

IN	β	a scalar
IN	x	a vector of length n
IN	y	a vector of length n
INOUT	A	a symmetric or Hermitian matrix of order n

Note: The symmetric or Hermitian rank-2 update function above is missing in the functionality table.
Note: The symmetric or Hermitian diagonal rank-1 update ($A \leftarrow xDx^T + A$) is incorrectly defined in the functionality table.

BLAS_ORVQR(trans, Q, x) $x \leftarrow Qx,\ x \leftarrow Q^T x,\ x \leftarrow Q^H x$
BLAS_ORVQL(trans, Q, x)
BLAS_ORVLQ(trans, Q, x)
BLAS_ORVRQ(trans, Q, x)

IN	*trans*	the transpose operator
IN	Q	a unitary matrix of order n
INOUT	x	a vector of lenght n

BLAS_Y_ORVQR(trans, Q, x, y) $y \leftarrow Qx,\ y \leftarrow Q^T x,\ y \leftarrow Q^H x$
BLAS_Y_ORVQL(trans, Q, x, y)
BLAS_Y_ORVLQ(trans, Q, x, y)
BLAS_Y_ORVRQ(trans, Q, x, y)

IN	$trans$	the transpose operator
IN	Q	a unitary matrix of order n
IN	x	a vector of length n
IN	y	a vector of length n

2.3.5 Matrix Operations

BLAS_GESET(α, β, A) $A \leftarrow \alpha I, \beta$ off-diagonal

IN	α	a scalar
IN	β	a scalar
OUT	A	a matrix of dimension m-by-n

BLAS_SYSET(α, β, A) $A \leftarrow \alpha I, \beta$ off-diagonal

IN	α	a scalar
IN	β	a scalar
OUT	A	a symmetric or Hermitian matrix of order n

BLAS_TRSET(α, β, A) $A \leftarrow \alpha I, \beta$ off-diagonal

IN	α	a scalar
IN	β	a scalar
OUT	A	a trapezoidal matrix of dimension m-by-n

BLAS_GENORM(A, r) $r \leftarrow ||A||_1,\ r \leftarrow ||A||_F,\ r \leftarrow ||A||_\infty$

IN	A	a matrix of dimension m-by-n
OUT	r	a scalar

BLAS_SYNORM(A, r) $r \leftarrow ||A||_1,\ r \leftarrow ||A||_F,\ r \leftarrow ||A||_\infty$

IN	A	a matrix of order n
OUT	r	a scalar

BLAS_TRNORM(A, r) $r \leftarrow ||A||_1,\ r \leftarrow ||A||_F,\ r \leftarrow ||A||_\infty$

IN	A	a trapezoidal matrix of dimension m-by-n with unit diagonal or not
OUT	r	a scalar

BLAS_GEAMAX(A, r) $r \leftarrow \max |a_{i,j}|$

IN	A	a matrix of dimension m-by-n
OUT	r	a scalar

BLAS_SYAMAX(A, r) $r \leftarrow \max |a_{i,j}|$

IN	A	a symmetric or Hermitian matrix of order n
OUT	r	a scalar

BLAS_TRAMAX(A, r) $r \leftarrow \max |a_{i,j}|$

IN	A	a trapezoidal matrix of dimension m-by-n with unit diagonal or not
OUT	r	a scalar

BLAS_GESCAL(α, A) $A \leftarrow \alpha A$

IN	α	a scalar
INOUT	A	a matrix of dimension m-by-n

BLAS_SYSCAL(α, A) $A \leftarrow \alpha A$

IN	α	a scalar
INOUT	A	a symmetric or Hermitian matrix of order n

BLAS_TRSCAL(α, A) $A \leftarrow \alpha A$

IN	α	a scalar
INOUT	A	a trapezoidal matrix of dimension m-by-n with unit diagonal or not

Note: The scaling factor α of the three above routines may need to be represented as fraction to avoid under/overflow just like in the LAPACK routine LASCL. Maybe two functions are needed here.

BLAS_GE_LEFT_DIAGSCAL(D, A) $A \leftarrow DA$

IN	D	a diagonal matrix of order m
INOUT	A	a matrix of dimension m-by-n

BLAS_TR_LEFT_DIAGSCAL(D, A) $A \leftarrow DA$

IN	D	a diagonal matrix of order m
INOUT	A	a trapezoidal matrix of dimension m-by-n

BLAS_GE_RIGHT_DIAGSCAL(D, A) $A \leftarrow AD$

IN	D	a diagonal matrix of order n
INOUT	A	a matrix of dimension m-by-n

BLAS_TR_RIGHT_DIAGSCAL(D, A) $A \leftarrow AD$

IN	D	a diagonal matrix of order n
INOUT	A	a trapezoidal matrix of dimension m-by-n

BLAS_GE_DIAGSCAL(D1, D2, A) $A \leftarrow D_1 A D_2$

IN	$D1$	a diagonal matrix of order m
IN	$D2$	a diagonal matrix of order n
INOUT	A	a matrix of dimension m-by-n

BLAS_SY_DIAGSCAL(D, A) $A \leftarrow DAD$

IN	D	a diagonal matrix of order n
INOUT	A	a symmetric or Hermitian matrix of order n

BLAS_TR_DIAGSCAL(D1, D2, A) $A \leftarrow D_1 A D_2$

IN	$D1$	a diagonal matrix of order m
IN	$D2$	a diagonal matrix of order n
INOUT	A	a trapezoidal matrix of dimension m-by-n

BLAS_GE_ADD_RIGHT_DIAGSCAL(B, D, A) $A \leftarrow A + BD$

IN	B	a matrix of dimension m-by-n
IN	D	a diagonal matrix of order n
INOUT	A	a matrix of dimension m-by-n

BLAS_TR_ADD_RIGHT_DIAGSCAL(B, D, A) $A \leftarrow A + BD$

IN	B	a trapezoidal matrix of dimension m-by-n
IN	D	a diagonal matrix of order n
INOUT	A	a trapezoidal matrix of dimension m-by-n

BLAS_GE_A_APB(A, B) $A \leftarrow A + B$

INOUT	A	a matrix of dimension m-by-n
IN	B	a matrix of dimension m-by-n

BLAS_SY_A_APB(A, B) $A \leftarrow A + B$

INOUT	A	a symmetric or Hermitian matrix of order n
IN	B	a symmetric or Hermitian matrix of order n

BLAS_TR_A_APB(A, B) $A \leftarrow A + B$

INOUT	A	a trapezoidal matrix of dimension m-by-n
IN	B	a trapezoidal matrix of dimension m-by-n

BLAS_GE_C_APB(A, B, C) $C \leftarrow A + B$

IN	A	a matrix of dimension m-by-n
IN	B	a matrix of dimension m-by-n
OUT	C	a matrix of dimension m-by-n

BLAS_SY_C_APB(A, B, C) $C \leftarrow A + B$

IN	A	a symmetric or Hermitian matrix of order n
IN	B	a symmetric or Hermitian matrix of order n
OUT	C	a symmetric or Hermitian matrix of order n

BLAS_TR_C_APB(A, B, C) $C \leftarrow A + B$

IN	A	a trapezoidal matrix of dimension m-by-n
IN	B	a trapezoidal matrix of dimension m-by-n
OUT	C	a trapezoidal matrix of dimension m-by-n

BLAS_GE_ACC(trans, α, A, β, B) $A \leftarrow \alpha A + \beta B$, $A \leftarrow \alpha A + \beta B^T$, $A \leftarrow \alpha A + \beta B^H$

IN	$trans$	the transpose operator
IN	α	a scalar
INOUT	A	a matrix of dimension m-by-n
IN	β	a scalar
IN	B	a matrix of dimension m-by-n in the non-transpose case and n-by-n otherwise

BLAS_GE_ADD(α, A, β, B, C) $C \leftarrow \alpha A + \beta B$

IN	α	a scalar
IN	A	a matrix of dimension m-by-n
IN	β	a scalar
IN	B	a matrix of dimension m-by-n
OUT	C	a matrix of dimension m-by-n

BLAS_SY_ADD(α, A, β, B, C) $C \leftarrow \alpha A + \beta B$

IN	α	a scalar
IN	A	a symmetric or Hermitian matrix of order n
IN	β	a scalar
IN	B	a symmetric or Hermitian matrix of order n
OUT	C	a symmetric or Hermitian matrix of order n

BLAS_TR_ADD(α, A, β, B, C) $C \leftarrow \alpha A + \beta B$

IN	α	a scalar
IN	A	a trapezoidal matrix of dimension m-by-n
IN	β	a scalar
IN	B	a trapezoidal matrix of dimension m-by-n
OUT	C	a trapezoidal matrix of dimension m-by-n

2.3.6 Matrix-Matrix Operations

In the following section, $op(X)$ denotes X, or X^T or X^H where X is a matrix.

BLAS_GE_A_AXB(side, transA, transB, A, B) $A \leftarrow op(A)op(B)$ or $A \leftarrow op(B)op(A)$

IN	$side$	the side operator
IN	$transA$	the trans operator specifying $op(A)$
IN	$transB$	the trans operator specifying $op(B)$
INOUT	A	a matrix of dimension m_A-by-n_A
IN	B	a matrix of dimension m_B-by-n_B

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Side = Left
%
%        transB |        N                T/C
%  -------------|------------------------------------
%  transA  N    |   n_A=m_B=n_B        n_A=n_B=m_B
%               |
%         T/C   | m_A=n_A=m_B=n_B    m_A=n_A=m_B=n_B
%  -------------|------------------------------------
%
%  Side = Right
%
%        transB |        N                T/C
%  -------------|------------------------------------
%  transA  N    |   m_A=n_B=m_B        m_A=n_B=m_B
%               |
%         T/C   | m_A=n_A=m_B=n_B    m_A=n_A=m_B=n_B
%  -------------|------------------------------------
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

BLAS_TR_A_AXB(trans, A, B) $A \leftarrow Aop(B)$, $A \leftarrow op(B)A$

IN	$trans$	the trans operator specifying $op(B)$
INOUT	A	a matrix of dimension m_A-by-n_A
IN	B	a triangular matrix of order n_B with unit diagonal or not

When $op(B)$ is applied from the left, $n_B = m_A$, and otherwise $n_A = n_B$.

BLAS_SY_A_AXB(trans, A, B) $A \leftarrow Aop(B)$, $A \leftarrow op(B)A$

IN	$trans$	the trans operator specifying $op(B)$
INOUT	A	a matrix of dimension m_A-by-n_A
IN	B	a symmetric or Hermitian matrix of order n_B

When $op(B)$ is applied from the left, $n_B = m_A$, and otherwise $n_A = n_B$.

BLAS_GEMM(transA, transB, α, A, B, β, C) $C \leftarrow \alpha op(A) op(B) + \beta C$

IN	$transA$	the trans operator specifying $op(A)$
IN	$transB$	the trans operator specifying $op(B)$
IN	α	a scalar
IN	A	a matrix such that op(A) is of dimension m-by-k
IN	B	a matrix such that op(B) is of dimension k-by-n
IN	β	a scalar
INOUT	C	a matrix of dimension m-by-n

BLAS_TRSM(α, A, B) $B \leftarrow \alpha A^{-1} B$, $B \leftarrow \alpha A^{-T} B$, $B \leftarrow \alpha A^{-H} B$, $B \leftarrow \alpha B A^{-1}$, $B \leftarrow \alpha B A^{-1}$, $B \leftarrow \alpha B A^{-H}$

IN	α	a scalar
IN	A	a triangular matrix with unit diagonal or not of order m or n
INOUT	B	a matrix of dimension m-by-n

Note: No check for singularity just like legacy BLAS.

BLAS_SYRK(α, A, β, C) $C \leftarrow \alpha A A^T + \beta C$, $C \leftarrow \alpha A A^H + \beta C$, $C \leftarrow \alpha A^T A + \beta C$, $C \leftarrow \alpha A^H A + \beta C$

IN	α	a scalar
IN	A	a matrix of dimension n-by-k or k-by-n
IN	β	a scalar
INOUT	C	a symmetric or Hermitian matrix of order n

BLAS_DIAG_SYRK(α, A, D, β, C) $C \leftarrow \alpha A D A^T + \beta C$, $C \leftarrow \alpha A D A^H + \beta C$, $C \leftarrow \alpha A^T D A + \beta C$, $C \leftarrow \alpha A^H D A + \beta C$

IN	α	a scalar
IN	A	a matrix of dimension n-by-k or k-by-n
IN	D	a diagonal of order k
IN	β	a scalar
INOUT	C	a symmetric or Hermitian matrix of order n

Note: Is the matrix D diagonal or block-diagonal with blocks of size 1 or 2 ? If D is block-diagonal, maybe we need a special notation for those matrices and check again the previous tables for consistency.

BLAS_SYR2K(α, A, B, β, C) $C \leftarrow (\alpha A) B^T + B (\alpha A)^T + \beta C$, $C \leftarrow (\alpha A) B^H + B (\alpha A)^H + \beta C$, $C \leftarrow (\alpha A)^T B + B^T (\alpha A) + \beta C$, $C \leftarrow (\alpha A)^H B + B^H (\alpha A) + \beta C$

IN	α	a scalar
IN	A	a matrix of dimension n-by-k or k-by-n
IN	B	a matrix of dimension n-by-k or k-by-n
IN	β	a scalar
INOUT	C	a symmetric or Hermitian matrix of order n

BLAS_DIAG_SYR2K(α, A, B, D, β, C) $C \leftarrow (\alpha A D) B^T + B (\alpha A D)^T + \beta C$, $C \leftarrow (\alpha A D) B^H + B (\alpha A D)^H + \beta C$, $C \leftarrow (\alpha A D)^T B + B^T (\alpha A D) + \beta C$, $C \leftarrow (\alpha A D)^H B + B^H (\alpha A D) + \beta C$

IN	α	a scalar
IN	A	a matrix of dimension n-by-k or k-by-n
IN	B	a matrix of dimension n-by-k or k-by-n
IN	D	a diagonal matrix of order k
IN	β	a scalar
INOUT	C	a symmetric or Hermitian matrix of order n

Note: Is the diagonal-symmetric rank-2k update really necessary ?

BLAS_X_ORMQR_X(Q, X) $X \leftarrow QX,\ X \leftarrow XQ,\ X \leftarrow Q^TX,\ X \leftarrow XQ^T,\ X \leftarrow Q^HX,\ X \leftarrow XQ^H$
BLAS_X_ORMQL_X(Q, X)
BLAS_X_ORMLQ_X(Q, X)
BLAS_X_ORMRQ_X(Q, X)

IN	Q	an orthogonal or unitary matrix of order k
INOUT	X	a matrix of dimension m-by-n

BLAS_Y_ORMQR_X(Q, X, Y) $Y \leftarrow QX,\ Y \leftarrow XQ,\ Y \leftarrow Q^TX,\ Y \leftarrow XQ^T,\ Y \leftarrow Q^HX,\ Y \leftarrow XQ^H,$
BLAS_Y_ORMQL_X(Q, X, Y)
BLAS_Y_ORMLQ_X(Q, X, Y)
BLAS_Y_ORMRQ_X(Q, X, Y)

IN	Q	an orthogonal or unitary matrix of order k
IN	X	a matrix of dimension m-by-n
OUT	Y	a matrix of dimension m-by-n

BLAS_GE_OR_SIM_QR(Q, X) $X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ,\ X \leftarrow QXQ^H,\ X \leftarrow Q^HXQ$
BLAS_GE_OR_SIM_QL(Q, X)
BLAS_GE_OR_SIM_LQ(Q, X)
BLAS_GE_OR_SIM_RQ(Q, X)

IN	Q	a orthogonal or unitary matrix of order k
INOUT	X	a matrix of order n

BLAS_SY_OR_SIM_QR(Q, X) $X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ,\ X \leftarrow QXQ^H,\ X \leftarrow Q^HXQ$
BLAS_SY_OR_SIM_QL(Q, X)
BLAS_SY_OR_SIM_LQ(Q, X)
BLAS_SY_OR_SIM_RQ(Q, X)

IN	Q	an orthogonal or unitary matrix of order k
INOUT	X	a symmetric or Hermitian matrix of order n

BLAS_X_ORM_X(Q, X) $X \leftarrow QX,\ X \leftarrow XQ, X \leftarrow Q^TX,\ X \leftarrow XQ^T,\ X \leftarrow Q^HX,\ X \leftarrow XQ^H$

IN	Q	an orthogonal or unitary matrix of order k
INOUT	X	a matrix of dimension m-by-n

BLAS_GE_OR_SIM(Q, X) $X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ,\ X \leftarrow QXQ^H,\ X \leftarrow Q^HXQ$

IN	Q	a orthogonal or unitary matrix of order k
INOUT	X	a matrix of order n

BLAS_SY_OR_SIM(Q, X) $X \leftarrow QXQ^T,\ X \leftarrow Q^TXQ,\ X \leftarrow QXQ^H,\ X \leftarrow Q^HXQ$

IN	Q	a orthogonal or unitary matrix of order k
INOUT	X	a symmetric or Hermitian matrix of order n

BLAS_X_GIVM_X(Q, X) $X \leftarrow QX,\ X \leftarrow XQ,\ X \leftarrow Q^TX,\ X \leftarrow XQ^T,$
$X \leftarrow Q^HX,\ X \leftarrow XQ^H$

IN	Q	an orthogonal or unitary matrix of order k
INOUT	X	a matrix of dimension m-by-n

Note: The representation of the orthogonal/unitary Q matrices is language independent and should be specified here. This also applies to a sequence of Givens rotations.
Note: Procedures explicitly forming such Q matrices are missing.

2.3.7 Data Movement with Matrices

BLAS_GE_MCOPY(A, B) $B \leftarrow A$

IN	A	a real/complex matrix of dimension m-by-n
OUT	B	a real/complex matrix of dimension m-by-n

BLAS_SY_MCOPY(A, B) $B \leftarrow A$

IN	A	a symmetric or Hermitian matrix of order n
OUT	B	a symmetric or Hermitian matrix of order n

BLAS_TR_MCOPY(A, B) $B \leftarrow A$

IN	A	a trapezoidal matrix of dimension m-by-n
OUT	B	a trapezoidal matrix of dimension m-by-n

BLAS_GE_MTRANS(A) $A \leftarrow A^T$, $A \leftarrow A^H$

INOUT	A	a matrix of order n

BLAS_GE_MTRANS_COPY(A, B) $B \leftarrow A^T$, $B \leftarrow A^H$

IN	A	a matrix of dimension n-by-m
OUT	B	a matrix of dimension m-by-n

BLAS_GE_GET(A, I, J, B) $B \leftarrow a_{I,J}$

IN	A	a matrix of dimension m_A-by-n_A
IN	I	a contiguous index interval of length $m \leq m_A$
IN	J	a contiguous index interval of length $n \leq n_A$
OUT	B	a matrix of dimension m-by-n

BLAS_TR_GET(A, I, J, B) $B \leftarrow a_{I,J}$

IN	A	a matrix of dimension m_A-by-n_A
IN	I	a contiguous index interval of length $n \leq m_A$
IN	J	a contiguous index interval of length $n \leq n_A$
OUT	B	a symmetric or Hermitian matrix of order n

BLAS_TR_GET(A, I, J, B) $B \leftarrow a_{I,J}$

IN	A	a matrix of dimension m_A-by-n_A
IN	I	a contiguous index interval of length $m \leq m_A$
IN	J	a contiguous index interval of length $n \leq n_A$
OUT	B	a trapezoidal matrix of dimension m-by-n

BLAS_GE_ROW_OR_COL_PERM(P, A) $A \leftarrow PA$, or $A \leftarrow AP$

IN	P	a permute matrix of order m when aplied from the left, and n otherwise
INOUT	A	a matrix of dimension m-by-n

BLAS_GE_ROW_AND_COL_PERM(Q, P, A) $A \leftarrow QAP$

IN	Q	a permute matrix of order m
IN	P	a permute matrix of order n
INOUT	A	a matrix of dimension m-by-n

BLAS_SY_ROW_AND_COL_PERM(P, A) $A \leftarrow P^TAP$

IN	P	a permute matrix of order n
INOUT	A	a symmetric or Hermitian matrix of order n

Note: The specification of the encoding of the permutation is necessary, but missing.

BLAS_GE_DIAG_EXTRACT(A, d) $d \leftarrow \text{diag}(A)$

IN	A	a matrix of dimension m-by-n
OUT	d	a vector of length $\min(m, n)$

BLAS_SY_DIAG_EXTRACT(A, d) $d \leftarrow \text{diag}(A)$

IN	A	a symmetric or Hermitian matrix of order n
OUT	d	a vector of length n

BLAS_TR_DIAG_EXTRACT(A, d) $d \leftarrow \text{diag}(A)$

IN	A	a trapezoidal matrix of dimension m-by-n
OUT	d	a vector of length $\min(m, n)$

2.3.8 Multiple Instances

Definition: a multi-vector X is a finite collection of vectors of the same length. The order of a multi-vector X of length n is the number of vectors of length n contained in X.

Definition: a multi-scalar X is a finite collection of scalars. The order of a multi-scalar X is the number of scalars contained in X.

Note: The orientation of a multivector might be a source of confusion in the next section, because multi-vectors are likely to be stored in a two-dimensional array. So, one may need to add some arguments to distinguish row- from column multi-vectors.

Note: The code displayed in the text for a multiple AXPY Level 1 is an elementwise multiply + add. Is this really what we want ?

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% General Comments
%
% In the functionality table, sometimes _i is used, sometimes [i].
%
% Some functions in the text are not in the table.
%
% Issues: 1) Multiple inner products: in the table the vector x seems to
%         be reused. In the text, there seems to be as many vectors x than
%         vectors y. Are both operations wanted ? Same question for multiple
%         AXPYs.
%
%         Knowing that there are 3 "simple" dot products, is that ok to provide
%         only one in the multiple instance case ?
%
%         Same question for axpy Level-2 in the text ? What about min/max
%         value and/or location ? The notation in the table for the multiple
%         max absolute value is confusing.
%
%         2) The multiple application of rotations ... In the text, it is
%         said to mimic the LAPACK routine _LASR. The definition given in
%         the table is different than what _LASR does. Indeed _LASR applies
```

```
%         the rotations say to rows 1,2, then 2,3 and so on of the same
%         matrix A. In particular, _LASR takes m-1 or n-1 rotations.
%         The function in the table applies different rotations to different
%         pairs of vectors. In particular, this suggests the same number of
%         rotations than vector pairs.
%         There are also the question of forward/backward and row/column
%         application that are not adressed by the function in the table.
%         Finally, _LASR takes an argument PIVOT. The functionality of this
%         parameter is not reflected in the functionality table.
%
%         3) The text mentions the LAPACK routine _LARTV:
%
%  SLARTV applies a vector of real plane rotations to elements of the
%  real vectors x and y. For i = 1,2,...,n
%
%     ( x(i) ) := (  c(i)  s(i) ) ( x(i) )
%     ( y(i) )    ( -s(i)  c(i) ) ( y(i) )
%
%          This function is not in the table.
%
%          4) The text mentions the LAPACK routine _LARGV:
%
%  SLARGV generates a vector of real plane rotations, determined by
%  elements of the real vectors x and y. For i = 1,2,...,n
%
%     (  c(i)  s(i) ) ( x(i) ) = ( a(i) )
%     ( -s(i)  c(i) ) ( y(i) ) = (   0  )
%
%          On exit, x(i) is overwritten by a(i), y(i) is the sinus of the
%          ith plane rotation, and an output vector get contains the cosines.
%
%          There is a discrepancy between _LARGV and the function proposed in
%          the table.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

BLAS_MDOT(x, y, r) $r_i \leftarrow x[i]^T y[i]$

IN	x	a multi-vector of order m and length n
IN	y	a multi-vector of order m and length n
OUT	r	a multi-scalar of order m

BLAS_MAXPY(d, x, y) $y[i] \leftarrow d_i * x[i] + y[i]$

IN	d	a multi-scalar of order m
IN	x	a multi-vector of order m and length n
INOUT	y	a multi-vector of order m and length n

BLAS_MSWAP(x, y) $y[i] \leftrightarrow x[i]$

INOUT	x	a multi-vector of order m and length n
INOUT	y	a multi-vector of order m and length n

BLAS_MAMAX(x, imax) $imax_i = \arg\max_j |x_{i_j}|$

IN	x	a multi-vector of order m and length n
OUT	$imax$	a integer multi-scalar

BLAS_MAMIN(x, imin) $imin_i = \arg\min_j |x_{i_j}|$

IN	x	a multi-vector of order m and length n
OUT	$imin$	a integer multi-scalar

BLAS_MROTG(a, b, c, s) $(c[i], s[i]) \leftarrow \mathrm{rot}(a[i], b[i])$, such that ...

IN	a	a multi-scalar of order m
IN	b	a multi-scalar of order m
OUT	c	a multi-scalar of order m
OUT	s	a multi-scalar of order m

BLAS_MROT(c, s, x, y) $\begin{pmatrix} x[i] & y[i] \end{pmatrix} \leftarrow \begin{pmatrix} x[i] & y[i] \end{pmatrix} R[i]$

IN	c	a multi-scalar of order m
IN	s	a multi-scalar of order m
INOUT	x	a multi-vector of order m and length n
INOUT	y	a multi-vector of order m and length n

2.4 Language Dependent Specification

```
call BLAS_DGEMM( m, n, k, alpha, A, lda, B, ldb, beta, C, ldc )
\\
INTEGER m, n, k, lda, ldb, ldc
DOUBLE PRECISION alpha, beta, A(lda, *), B(ldb, *), C(ldc, *)
```

2.5 Reference Implementation

2.6 Benchmarking, Verification, and Validation

2.6.1 Addendum ?

```
From: Gary Howell <howell@zach.fit.edu>
```

A Blas 2.5 Operator for Increasing Linear Algebra Kernel Functionality

The Level 2 BLAS GER (rank one update) and GEMV (matix vector multiply) provide a convenient way of to implement many linear algebra algorithms. When a matrix is too large for cache memory, the predominant expense of either GER or GEMV algorithm is not in the $2n^2$ arithmetic operations but in the transport of n^2 floating point numbers from main memory to cache memory and back. One successful way to increase the ratio of computation to memory is to use BLAS-3 operators. Unfortunately, BLAS-3 operators are often inconvenient and/or impossible.

BLAS 2.5 operators interleave rank one updates and matrix vector multiplies so that low rank updates, matrix vector multiplies (or multiplies of matrices by a few vectors) and multiplications of a few vectors by the matrix transpose can be accomplished in one pass of the matrix through cache. Under the assumption that a few columns will simultaneously fit in cache memory, the ratio of arithmetic operations to memory transfers is very often two to four times as high as by using BLAS-2 operators. Speed-ups of two to four over BLAS-2 operators occur. On current architectures, bandwidth is often sufficiently high for that the BLAS 2.5 operator allows near optimal performance.

Introduction

I propose one new operator, GEMVR. If several columns of a matrix will fit in cache memory, then GEMVR performs in one pass through cache,

1. a rank NR1 update;
2. a left multiplication by KR row vectors;
3. a rank NR2 update;
4. a right multiplication by KC column vectors.

If a matrix is too large for several columns to fit in cache, then GEMVR will produce correct answers, but performance will suffer. Currently, the discussion is column oriented.

Some Example Capabilities

The proposed operation is simultaneous computation of AX and A^TY along with a rank k update of A. Implementation should give good cache performance for any large matrix A for the case that X has only a few columns and Y^T only a few rows and k is also small. For more columns and rows and larger k, good performance is already possible by standard level-3 BLAS.

The operation allows efficient implementation of the the following common operations and also many others. IN IMPLEMENTING GEMVR TO ACCOMPLISH THE ADVERTIZED ALGORITHMS I HAVE USED THE IOPT VARIABLE TO ALLOW OPTION SPECIFIC OVERWRITES OF INPUT VARIABLES.

1. Householder similarity transformations. $A \leftarrow Q^TAQ$.
2. Householder transformations of the form $A \leftarrow Q_1^TAQ_2$. IOPT set to allow computation of new Householder transformation during current transformation.
3. Unsymmetric Lanczos methods.A^Tx and Ay are required on each step. Typically A is sparse.
4. Elementary similarity transformations.

 i $A \leftarrow LAL^{-1}$. Useful in reduction to Hessenberg form.

 ii $A \leftarrow R^{-1}LAL^{-1}R)$. Useful in reduction to small-band form.

5. Block Gaussian elimination.
6. Block Householder reduction.
7. Modified Gram-Schidt orthogonalization. For $i = 1 : k, \alpha = w^t * v_i; w = w - alpha * v_i$; end;

In each of cases A)-G), operator GEMVR allows the computation to proceed in fewer cache accesses than would be necessary if DGEMV and DGER (matrix vector multiply and rank-one update respectively) are employed independently.

Pseudo-Code

```
      SUBROUTINE DGEMVR(A,LDA,N,M,iblock,jblock, !pseudo-code
     +          W1,Z1,NR1,
     +          Yin,Yout,KR,
     +          W2,Z2,NR2,
     +          Xin,Xout,KC,
     +          Iopt)

c
c W1 and Z1 form a rank NR1 update
c A(i:i+iblock-1,j:j+jblock-1)
c     <-- A(i:i+iblock-1,j:j+jblock-1) + W1*Z1
c
c KR is the number of columns of Yin, If KR > 0 ,
C Yout <-- A^T * Yin + Yout.
```

```
c
c Iopt
c Specification of integer Iopt allows some options
c in an inner loop.  For example, a mixed Householder
c or an elementary similarity transformation can be
c specified in such a way that the operation can be
c performed in only one access to main memory.
c
c W2 and Z2 form a rank NR2 update
c A(i:i+iblock-1,j:j+jblock-1)
c <-- A(i:i+iblock-1,j:j+jblock-1) + W2*Z2
c
c KC is the number of columns of Xin, If KC > 0 ,
C Xout <-- A * Xin + Xout.
c
c
c


       do j=1,jblock                 ! loop on columns
         do i=1,iblock
           rank NR1 update of column j
         end do
         if (j.eq.1) perform an operation on the 1st column as specified
! by Iopt(1)
         do i =1,jblock
           Yout(j) = Yin(i)*A(i,j) + Yout(j)
             ! Yin has KR columns
         end do
         if (j.eq.1) perform an operation on the 1st column as specified
! by Iopt(2)
         do i=1,jblock
           rank NR2 update of column j
         end do
         if (j.eq.1) perform an operation on the 1st column as specified
! by Iopt(3)
         do i = 1,jblock
           Xout(i) = Xout(i) +  A(i,j)*Xin(j)  ! x and x1 have KC columns
         end do
       end do
```

Calling Arguments for Some Specific Applications of GEMVR

The * items have been implemented in a prototype reference implementation. ** were implemented in previous work.

1* Householder similarity transformations. $A \leftarrow Q^T AQ$

$$A \leftarrow A - w(w^t A) - (Aw)w^t + w(w^t Aw)w^t$$

A call to DGMEVR with KR = KC = 1 and NR1=NR2=0 returns $w^t A$ and Aw. Compute $w^t Aw$ and then call DGEMVR with NR1=3 and KR=KC=NR2=0 . This implementation takes two calls to DGEMVR and thus two accesses of A in main memory.

If a sequence of Householder similarity transformations are needed, they can be chained. The first call to DGEMVR computes $w_1^t A$ and Aw_1 . The next call performs a rank 3 update with NR1=3. As soon as the first column of the rank 3 update is known, then w_2 can be computed in the loop Iopt(2). Taking KR = KC = 1 and NR2 = 0 we can then obtain $w_2^t A$ and Aw_2 in the same call to DGEMVR . The last call to DGEMVR consists only of a rank-3 update. With chaining, only the first and last Householder transformations require two calls to DGEMVR.

2* Householder transformations of the form $A \leftarrow Q_1^T A Q_2$.

$$A <-- A - w_2(w_2^t A) - (Aw_2)w_2^t + w_1(w_1^t A w_2)w_2^t$$

. Implementation is almost the same as A).

3. SPARSE APPLICATIONS. For the sparse case, low rank updates are not usually desirable as they destroy the sparsity. Simultaneous computation of $A * x$ and $y^t * A$ is often desirable. a) Look-ahead Lanczos. Compute $A * x$ and $y^t * A$. Set NR1 and NR2 = 0. Pass in initial values of Yout and Xout as zero. For a single step algorithm, take KR = KC = 1. For a block algorithm, take KR = KC = k. Computaion of Ax and $y^t A$ in one pass through cache is straightforward in compressed row or compressed column storage.

   b) Compute $v = A * x$ and $v^t * A$ in one pass of A through the cache. Useful in steepest descent solution of the normal equations. "Always" converges to solution of $A * x = b$, provided A nonsingular. One pass through cache compuation of $v = AX$ and $v^t A$ is straightforward in compressed column storage, less convenient in compressed row storage.

4** Elementary similarity transformations.

   i)$A \leftarrow LAL^{-1}$. A column operation. NR1 = 0, KR = 1, NR2 = 1, KC = 0.

   ii) $A \leftarrow R^{-1} LAL^{-1} R$. NR1 = 1, KR = 1, NR2 = 1, KC = 1. (NEEDS AN IOPT TO AVOID INTERFERENCE BETWEEN UPDATES TO K+1 COLUMN AND UPDATES).

5. Block Gaussian elimination.

$$A \leftarrow LA$$

   It can be performed with NR1 = 0 = NR2 = Yin, and L stored as A, A input as Xin, updated columns of A as Xout. Of course, the same operation can be performed by triangular solve?

6. Block Householder reduction.

$$A \leftarrow (I - 2 * Q * Q^t)A = A - 2 * Q * (Q^t * A)$$

   Set KR = NR2 = k, NR1 = KC = 0 .

7* Modified Gram-Schmidt.

   Take the matrix A(:,1:jblock) = V and Yin = w.

   Take KR = KC = 1 and NR1 = NR2 = 0 . Take Xin = Yout.

   TAKES AN IOPT TO REDEFINE ROW VECTORS FOR EACH NEW COLUMN. THE FIRST ROW OF Yin IS TAKEN AS THE VECTOR BEING ORTHOGONALIZED. THE SECOND ROW OF Yin IS THE CURRENT COLUMN.

A reference FORTRAN implementation has the * options above and will be expanded to accomplish all of the above. SUGGESTIONS AS TO WHAT ELSE COULD BE ACCOMPLISHED WITH THE SAME BASIC ALGORHITHM ARE IMPORTANT TO ENSURE FULL FUNCTIONALITY.

Verification of Algorithm

The reference implementation gives directions for producing sample matrices in Matlab. It allows the reading of the Matlab generated matrices and returns the transformed matrix in Matlab readable form. It is easy to verify advertized operations against Matlab operations on the same matrix.

Bibliography

[1] *The EM Symmetric Sparse Supernodal Format.* Elegant Mathematics, Inc., Bothell, Washington, 1993.

[2] *The EM Unsymmetric Sparse Supernodal Format.* Elegant Mathematics, Inc., Bothell, Washington, 1993.

[3] M. Aboelaze, N. Chrisochoides, and E. Houstis. The Parallelization of Level 2 and 3 BLAS Operations on Distributed Memory Machines. Technical Report CSD-TR-91-007, Purdue University, West Lafayette, IN, 1991.

[4] R. Agarwal, S. Balle, F. Gustavson, M. Joshi, and P. Palkar. A Three-Dimensional Approach to Parallel Matrix Multiplication. *IBM Journal of Research and Development*, 39(5):575–582, 1995.

[5] R. Agarwal, F. Gustavson, and M. Zubair. A High Performance Matrix Multiplication Algorithm on a Distributed-Memory Parallel Computer, Using Overlapped Communication. *IBM Journal of Research and Development*, 38(6):673–681, 1994.

[6] E. Anderson, Z. Bai, C. H. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. C. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, USA, 2nd edition, 1995. (Also available in Japanese, published by Maruzen, Tokyo, translated by Dr Oguni).

[7] P. Bangalore. The Data-Distribution-Independent Approach to Scalable Parallel Libraries. Master's thesis, Mississippi State University, 1995.

[8] R. E. Bank and D. J. Rose. Marching algorithms for elliptic boundary value problems. the constant coefficient case. *SIAM Journal of Numerical Analysis*, 14:792–829, 1977.

[9] Randolph E. Bank and Craig C. Douglas. Sparse matrix multiplication package (SMMP). *Advances in Computational Mathematics*, 1, 1993.

[10] B. Bialecki and G. Fairweather. Matrix decomposition algorithms in orthogonal spline collocation for separable elliptic boundary value problems. *SIAM J. Sci. Comput.*, 16:330–347, 1995.

[11] J. Bilmes, K. Asanovic, C. Chin, and J. Demmel. Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology. In *Proceedings of the International Conference on Supercomputing*. ACM SIGARC, 1997. Vienna, Austria.

[12] R. Bisseling and J. van der Vorst. Parallel Triangular System Solving on a mesh network of Transputers. *SIAM Journal on Scientific and Statistical Computing*, 12:787–799, 1991.

[13] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide.* SIAM, Philadelphia, PA, 1997.

[14] R. Brent and P. Strazdins. Implementation of BLAS Level 3 and LINPACK Benchmark on the AP1000. *Fujitsu Scientific and Technical Journal*, 5(1):61–70, 1993.

[15] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley. A Proposal for a Set of Parallel Basic Linear Algebra Subprograms. In J. Dongarra, K. Masden, and J. Waśniewski, editors, *Applied Parallel Computing*, pages 107–114. Springer Verlag, 1995. (also LAPACK Working Note No.100).

[16] J. Choi, J. Dongarra, and D. Walker. PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers. *Concurrency: Practice and Experience*, 6(7):543–570, 1994. (also LAPACK Working Note No.57).

[17] J. Choi, J. Dongarra, and D. Walker. PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subroutines. *Concurrency: Practice and Experience*, 8(7):517–535, 1996.

[18] A. Chtchelkanova, J. Gunnels, G. Morrow, J. Overfelt, and R. van de Geijn. Parallel Implementation of BLAS: General Techniques for Level 3 BLAS. Technical Report TR95-49, Department of Computer Sciences, UT-Austin, 1995. Submitted to Concurrency: Practice and Experience.

[19] J. Demmel, J. R. Gilbert, and X. S. Li. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination, February 1997. To appear in SIAM J. Matrix Anal. Appl., special issue on Sparse and Structured Matrix Computations and Their Applications (Also LAPACK Working Note 124).

[20] J. Demmel and X. Li. Faster numerical algorithms via exception handling. *IEEE Trans. Comput.*, 43:983–992, 1994.

[21] D. S. Dodson. Corrigendum: Remark on "Algorithm 539: Basic Linear Algebra Subroutines for FORTRAN usage". *ACM Trans. Math. Software*, 9:140, 1983. (See also [39] and [22]).

[22] D. S. Dodson and R. G. Grimes. Remark on algorithm 539: Basic Linear Algebra Subprograms for Fortran usage. *ACM Trans. Math. Software*, 8:403–404, 1982. (See also [39] and [21]).

[23] D. S. Dodson, R. G. Grimes, and J. G. Lewis. Sparse extensions to the FORTRAN Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 17:253–272, 1991. (Algorithm 692).

[24] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 16:1–28, 1990. (Algorithm 679).

[25] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 14:1–32, 399, 1988. (Algorithm 656).

[26] J. J. Dongarra, S. Hammarling, and S. Ostrouchov. BLAS technical workshop. LAPACK Working Note No.109. Technical Report CS-95-317, Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN 37996-1301, USA, 1995. URL: http://www.netlib.org/lapack/lawns/.

[27] I. S. Duff, A. M. Erisman, and J. K. Reid. Direct methods for sparse matrices. In *Monographs on Numerical Analysis*. Oxford University Press, New York, 1986.

[28] Iain Duff, Michele Marrone, and Giuseppe Radicati. A proposal for user level sparse BLAS. Technical Report TR/PA/92/85, CERFACS, 1992.

[29] A. Elster. Basic Matrix Subprograms for Distributed Memory Systems. In D. Walker and Q. Stout, editors, *Proceedings of the Fifth Distributed Memory Computing Conference*, pages 311–316. IEEE Press, 1990.

[30] R. Falgout, A. Skjellum, S. Smith, and C. Still. The Multicomputer Toolbox Approach to Concurrent BLAS and LACS. In *Proceedings of the Scalable High Performance Computing Conference SHPCC-92.* IEEE Computer Society Press, 1992.

[31] Message Passing Interface Forum. MPI: A Message Passing Interface Standard. *International Journal of Supercomputer Applications and High Performance Computing*, 8(3–4), 1994.

[32] G. Fox, S. Otto, and A. Hey. Matrix Algorithms on a Hypercube I: Matrix Multiplication. *Parallel Computing*, 3:17–31, 1987.

[33] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. Methods for modifying matrix factorizations. *Math. Comp.*, 28:505–535, 1974.

[34] F. G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software*, 4(3):250–269, 1978.

[35] M. Heath and C. Romine. Parallel Solution Triangular Systems on Distributed Memory Multiprocessors. *SIAM Journal on Scientific and Statistical Computing*, 9:558–588, 1988.

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

[37] S. Huss-Lederman, E. Jacobson, A. Tsao, and G. Zhang. Matrix Multiplication on the Intel Touchstone DELTA. *Concurrency: Practice and Experience*, 6(7):571–594, 1994.

[38] L. Kaufman and D. D. Warner. High-order, fast direct methods for separable elliptical equations. *SIAM Journal of Numerical Analysis*, 21:672–695, 1984.

[39] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Software*, 5:308–323, 1979. (Algorithm 539. See also [22] and [21]).

[40] G. Li and T. Coleman. A Parallel Triangular Solver for a Distributed-Memory Multiprocessor. *SIAM Journal on Scientific and Statistical Computing*, 9(3):485–502, 1988.

[41] G. Li and T. Coleman. A New Method for Solving Triangular Systems on Distributed-Memory Message-Passing Multiprocessor. *SIAM Journal on Scientific and Statistical Computing*, 10(2):382–396, 1989.

[42] W. Lichtenstein and S. L. Johnsson. Block-Cyclic Dense Linear Algebra. *SIAM Journal on Scientific and Statistical Computing*, 14(6):1259–1288, 1993.

[43] K. Mathur and S. L. Johnsson. Multiplication of Matrices of Arbitrary Shapes on a Data Parallel Computer. *Parallel Computing*, 20:919–951, 1994.

[44] O. McBryan and E. van de Velde. Matrix and Vector Operations on Hypercube Parallel Processors. *Parallel Computing*, 5:117–126, 1987.

[45] R. Pozo. Template numerical toolkit (tnt), 1995. `http://math.nist.gov/tnt`.

[46] Y. Saad. SPARSKIT: a basic tool kit for sparse matrix computations. Preliminary Version.

[47] P. Strazdins. A High Performance, Portable Distributed BLAS Implementation. In *Proceedings of the Sixth Parallel Computing Workshop*, Fujitsu Parallel Computing Center, 1996.

[48] Thinking Machines Corporation. *CMSSL for Fortran*, 1990.

[49] R. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package.* The MIT Press, Cambridge, Massachusetts, 1997.

[50] R. van de Geijn and J. Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. Technical Report UT CS-95-286, LAPACK Working Note No.96, University of Tennessee, 1995.

[51] T. et al. Yang. Distributed memory parallel sparse lu. Ucsb preprint, University of California, Santa Barbara.