

# A Blas 2.5 Operator for Increasing Linear Algebra Kernel Functionality

Gary Howell

18 April 1998

howell@zach.fit.edu

## Abstract

The Level 2 BLAS GER (rank one update) and GEMV (matrix 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.

# 1 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.

# 2 Some Example Capabilities

The proposed operation is simultaneous computation of  $AX$  and  $A^T Y$  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.  $A \leftarrow Q^T A Q$  .
2. Householder transformations of the form  $A \leftarrow Q_1^T A Q_2$  . IOPT set to allow computation of new Householder transformation during current transformation.
3. Unsymmetric Lanczos methods.  $A^T x$  and  $A y$  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  $i = 1 : k, \alpha = w^t * v_i; w = w - alpha * v_i; \text{end};$

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.

### 3 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

```

## 4 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.  $A \leftarrow Q^T A Q$

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

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  $A \leftarrow Q_1^T A Q_2$  .

$$A \leftarrow A - w_2(w_2^t A) - (Aw_2)w_2^t + w_1(w_1^t Aw_2)w_2^t$$

. 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 computation 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 ALGORITHM ARE IMPORTANT TO ENSURE FULL FUNCTIONALITY.

## 5 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.