Cache Efficient and Parallel Householder Bidiagonalization[*]

Gary W. Howell
Hewlett Packard Corporation
gary.howell@hp.com
Charles T. Fulton
Sumit Malhotra
Jim Parker
Florida Institute of Technology
fulton@zach.fit.edu

1 Abstract

Two new BLAS operators, GEMVT and GEMVER, are discussed. Their application to a substantial re-organization of the standard Golub-Kahan Householder bidiagonalization algorithm is sketched. These new operators accomplish two matrix-vector multiplications in one transfer of the matrix from main memory to the L_2 cache.

2 Introduction

Current computer architectures store data in a hierarchy of distances from the computational registers. Data in a small number of registers can be used for computations in the current clock cycle. Data in several levels of cache memory is

[*]This research partially supported by National Science Foundation Grant EIA-0103642 to Florida Institute of Technology.

available in at most a few clock cycles. Accessing data in main memory requires several dozen clock cycles and is constrained by bus bandwidth. Matrices larger than a few hundred square are typically too large to fit in cache memory and must be stored in main storage (RAM). In this paper, we discuss the use of BLAS 2.5 operators to halve data transfer for operations which require two coupled matrix-vector multiplications. In computers with cache architectures, halving data transfer substantially reduces computational time.

The two BLAS 2.5 routines of interest in this paper are _GEMVER, which performs the operations:

$$\hat{A} \leftarrow A + u_1 v_1^T + u_2 v_2^T$$
$$x \leftarrow \beta \hat{A}^T y + z$$
$$w \leftarrow \alpha \hat{A} x$$

and _GEMVT, which performs the operations:

$$x \leftarrow \beta A^T y + z$$
$$w \leftarrow \alpha A x$$

Specifications for these two routines are part of the new BLAS standard [3].

3 Blocked GEMVT Operator

We found it efficient to implement _GEMVT on top of the BLAS-2 _GEMV operator as follows. Partition a general $m \times n$ rectangular matrix A as

$$\left[\begin{array}{ccc} A_1 & A_2 & \ldots A_j \end{array} \right] \quad (1)$$

Then _GEMVT is performed as follows:

_GEMVT

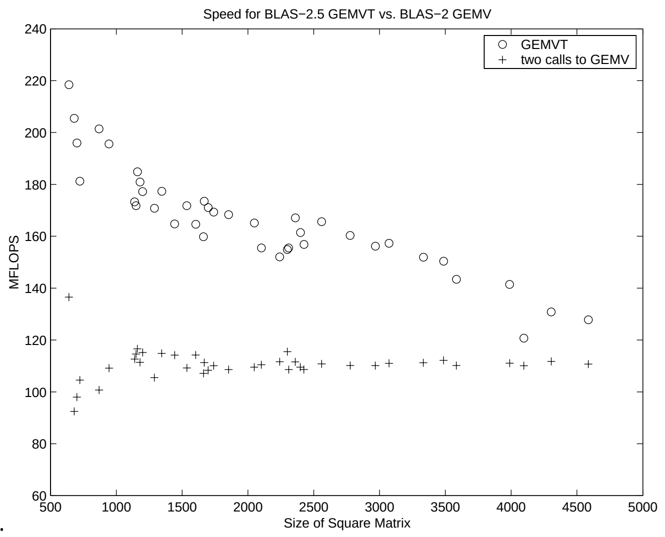
```
For $i = 1, j$
        $x_i^T \leftarrow \beta\, y^T A_i$                Call to _GEMV
        $w \leftarrow w + \alpha A_i x_i$                 Another call to GEMV
End for
```

where k is the number of columns in each block A_i and , for simplicity, we assume $k * j = n$. This version of _GEMVT loops only once through the columns of A, with each successive block of columns of A read into cache and used in two calls to _GEMV.

For runs on a 866 Mhz Intel Pentium III Processor the block size k in 1 is determined dynamically inside subroutine GEMVT by a step function that selects k from the input parameter m for the number of rows of the input matrix.The 866 Mhz Intel Pentium III Processor has 256 KB of L_2Cache. From experiments with square matrices from size 100 to 5120, run times were minimized when the step function for the block size k as a function of m was chosen according to the following Table.

Table 1. Dynamic Block Size Selection in GEMVT for the Pentium III

m = Rows Input Matrix	Block for GEMVT	Max Dble-prec Reals/Block	Min KBytes /Block	PerCent (L_2) Cache Used/Block
$1 < m \leq 640$	24	15360	122.880	48.0
$640 < m \leq 1152$	16	18432	147.456	57.6
$1152 < m \leq 1664$	12	19968	159.744	62.4
$1664 < m \leq 2304$	8	18432	147.456	57.6
$2304 < m \leq 5120$	4	20480	163.840	64.0



..

Figure 1. *Comparison of dynamically blocked DGEMVT with two DGEMV calls These runs were on a 866 MHz Pentium III with 256 KB of L2 cache.*

The plot above gives the megaflop rates for the above-described implementation of GEMVT with the block size k automatically linked to the number of rows m of the A-matrix. The _DGEMV calls made use of the tuned c ATLAS BLAS library [11]. The plot compares these megaflop rates with those from performing the same operation with two calls to ATLAS _GEMV.

4 Parallel GEMVT Codes

We have also made two implementations of parallel _GEMVT. These are described in detail in Sumit Malhotra's thesis [8].

- PBLAS implementation: This consists of a driver which makes two successive calls to the PBLAS routine PDGEMV [4].

- MPI implementation: This a column cyclic implementation in which the serial version of _GEMVT is utilized on each column block.

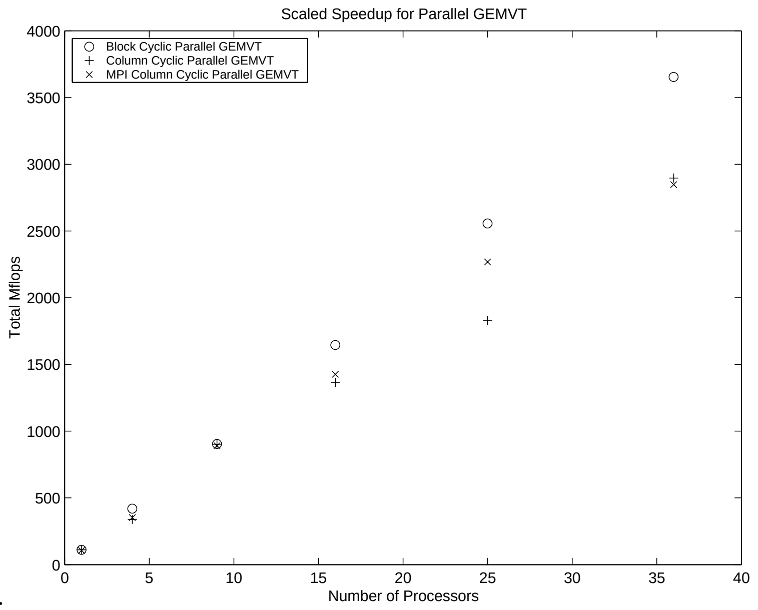


Figure 2. *Parallel GEMVT codes. These runs were on a 866 MHz Pentium III with 256 KB of L2 cache.*

Cleve Moler [9] suggested measurement of parallel efficiency of an algorithm by looking at scaled speedup as the total number of computations and data is increased proportionally to the number of processors. The above PBLAS block cyclic plot for was for square matrices having 7200 by 7200 submatrices on each node, i.e., 51.84 million matrix elements per CPU. The column cyclic algorithms (PBLAS and MPI) also have the same number of elements per CPU, i.e., each CPU has m rows and $51.84e6/m$ total columns. For the PBLAS block cyclic implementation, the megaflop rates were not sensitive to block sizes, for square block sizes > 16. For the PBLAS block cyclic case, the process grid was square, and the blocks were taken 16×16. For the PBLAS column cyclic case and for the MPI column cyclic case the process grid was $1 \times \#CPUs$. For the PBLAS column cyclic case, the cylic blocks had 100 columns. The MPI code had one column block per processor. We plotted the total megaflop rates for 1, 4, 9, 16, 25, 36 processors on the Florida Tech Bluemarlin Beowulf [2]. One conclusion is that the block cyclic data distribution gave near perfect scaled speedups, as indicated by the nearly linear plot. The MPI and PBLAS column cylic algorithms had very similar timings, with good but not perfect speedups.

5 Algorithm BLAS-2

Recall that Householder bidiagonalization can be accomplished in a straightforward BLAS-2 level algorithm by alternating matrix vector multiplies with rank one updates. This corresponds to the sequence of operations for Householder bidiagonalization originally introduced by Golub and Kahan in 1965. Such an algorithm is simple to implement but has excessive data transfer.

The basic step in the reduction of a real matrix to upper bidiagonal form is to choose Householder transformation matrices Q and P

$$Q = I - \tau_q uu^T, \; P = I - \tau_p vv^T$$

so that the matrix

$$B = Q^T AP$$

has zeros in the first column after the diagonal element and in the first row after the first superdiagonal element. After $(i-1)$ steps the leading $(i-1)$ by $(i-1)$ matrix B_{i-1} of A is bidiagonal. Thus we have a natural (but cache-inefficient) BLAS-2 algorithm for bidiagonalization

```
    **Algorithm BLAS-2**
For $i = 1 : n-1$,
    1. Construct $u_i$ of length $m-i+1$ to eliminate elements $A$(i+1:m,i);
    2. $y_i^T \leftarrow u_i^T$ $A$(i:m,i+1:n);                       /* Call _GEMV
    3. $A$(i:m,i+1:n) $\leftarrow$ $A$(i:m,i:n) $- u_i y_i^T$;        /* Call _GER
    If $(i < n-1)$
        4. Construct $v_i$ of length $n-i$ to eliminate elements $A$(i,i+2:n);
        5. $w_i \leftarrow$ $A$(i+1:m,i+1:n) $v_i$;                   /* Call _GEMV
        6. $A$(i+1:m,i+1:n) $\leftarrow$ $A$(i+1:m,i+1:n) $- w_i v_i^T$;   /* Call _GER
    Endif
End for;
If (m > n),
    Choose $u_n$ of length $m-n+1$ to eliminate $A$(n+1:m,n);
End if;
```

When A is too large to fit in cache memory, the BLAS-2 algorithm entails excessive transfer of data between main memory (RAM) and cache. For each _GEMV, A must be read from main memory to cache. For each _GER A must be read from main memory and then written back from cache to main memory.

6 A _GEMVER (BLAS 2.5) Algorithm

Here we describe a new algorithm which combines the two _GEMVs and two _GERs in the above algorithm to be accomplished by one call to the new BLAS routine _GEMVER. The _GEMVER routine is more cache efficient as it involves transfer of data from RAM to cache and back (one read and one write), as opposed to four reads and two writes for the BLAS 2 algorithm of the last section. This type of cache efficiency is similar to that obtained by Stanley [10]. The order of operations is changed. Instead of alternating calls to _GEMV followed by a call to _GER, we rearrange steps of Algorithm BLAS-2, so that the (conceptually simplified) algorithm is

```
    **Algorithm BLAS 2.5–first attempt**
For $i = 1 : n-1$,
    1. Construct $u_i$ to eliminate $A$(i+1:m,j);
    4. Construct $v_i$ to eliminate $A$(i,i+2:m);
    2. $y \leftarrow u_i^T A$;                    /* Call_GEMV
    5. $w \leftarrow A v_i$ ;                     /* Call _GEMV
    3. $A \leftarrow A - u y^T$;                  /* Call _GER
    6. $A \leftarrow A - w v^T$;                  /* Call _GER
End for;
```

So far we have two _GEMV calls followed by two _GER calls. But _GEMVER is a rank two update followed by two _GEMV calls. Therefore, rearrange the algorithm as follows:

```
    **Algorithm BLAS 2.5–second attempt**
Construct $u_1$ to eliminate $A$(2:m,1);
Construct $v_1$ to eliminate $A$(1,3:m);
$y_1 \leftarrow u_1^T A$; $w_1 \leftarrow A v_1$;
For $i = 2 : n-1$,
    1. Construct $u_i$ to eliminate $A$(i+1:m,j);
    2. Construct $v_i$ to eliminate $A$(i,i+2:m);
    3. Call _GEMVER to perform
            (I) $A \leftarrow A - u_{i-1} y_{i-1}^T - w_{i-1} v_{i-1}$ ;
            (II)$y_i^T \leftarrow u_i^T A$ ; $w_i \leftarrow A v_i$;
    4. Modify $v_i$ and $w_i$ appropriately;
End for;
$A \leftarrow A - u_{n-1} y_{n-1}^T w_{n-1} v_{n-1}$ ;
```

The rearrangement of the algorithm introduces some addtional complications. One main problem is that v_i is not known before the call to _GEMVER. Instead, we use a pre-Householder vector $\tilde{v_i}$. We take $\tilde{v_i}$ to be ith row of A (the row to be eliminated by the Householder vector v_i). Letting e_1 be

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

the Householder and pre-Householder vector are related by

$$v = \frac{\tilde{v_i} + s e_1}{k_r}$$

where $s = \pm\|\tilde{v}_i\|_2$ and $k_r = |s|(|s| + |\tilde{v}_i(1)|)$. Similarly $\tilde{w_i} = A\tilde{v_i}$ is related to $w_i = Av_i$ by

$$w_i = Av_i = \frac{A\tilde{v_i} + sAe_1}{K_r} = \frac{\tilde{w_i} + sAe_1}{K_r}$$

Hence the necessary modification of w can be done with access to only a single column of A.

Further details can be found in Howell, et al [6]. In that paper, we also discuss a mixed BLAS 2.5 - BLAS 3 algorithm. It defers the rank one updates so they can be performed as a matrix-matrix (BLAS 3) multiplication. This follows the main approach to blocked algorithms of Dongarra, Hammarling and Sorensen [5]. Paired matrix vector and transposed matrix vector multiplications are performed by the _BLAS 2.5 operator _GEMVT which does step II of _GEMVER in the above Algorithm. As with the above algorithm, the technique of using a pre-Householder vector and correcting it afterwards is required. The timing speed-ups in our blocked version of *_GEMVT* given in Section 3 result in significant speed-up of the serial version of our new bidiagonalization code. The new bidiagonalization code using a blocked version of the above algorithm has been made into a new version of subroutine _GEBRD (from the LAPACK library [1]) and is reported on in [6].

Bibliography

[1] E. Andersen, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchev, D. Sorensen, *LAPACK User's Guide*, 3rd. Ed. 1999 SIAM, Philadelphia.

[2] The web site is http://olin.fit.edu/beowulf.

[3] BLAS technical forum, *www.netlib.org/utk/papers/blast-forum.html*, 1999.

[4] 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*, Computer Science Dept. Tech. Report CS-95-292, University of Tennessee, Knoxville, TN, May 1995. (Also LAPACK Working Note #100).

[5] J. Dongarra, S. Hammmarling, and D. Sorensen, Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math.* 27:215–227, 1989.

[6] G. Howell, C. Fulton, J. Demmel, S. Hammarling, and K. Marmol, Cache Efficient Bidiagonalization Using BLAS 2.5 Operators, submitted to ACM Trans. on Math. Software (available from http://my.fit.edu/∼ cfulton/bidiag.html)

[7] G. Golub and W. Kahan, *Calculating the Singular Values and Psuedo-Inverse of a Matrix*, *SIAM J. Num. Anal.* 2:205–24, 1965.

[8] S. Malhotra,*Parallelization of BLAS 2.5 Operator GEMVT*, M.S. Thesis 2003, Computer Science, Florida Institute of Technology, Melbourne, Florida.

[9] C. Moler, *Matrix Computation on Distributed Memory Multiprocessors, (Ed. M. Heath)*, SIAM, Philadelphia (1986), 181-195.

[10] K.S. Stanley,*Execution Time of Symmetric Eigensolvers*, Ph.D. dissertation, 1997, CS, University of California, Berkeley.

[11] C. Whaley and J. Dongarra,*Automatically Tuned Linear Algebra Software Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999, available on CD-ROM from SIAM.