Evaluating Block Algorithm Variants in LAPACK *

* This work was supported by the National Science Foundation
under Grant No. ASC-8715728.
This paper was submitted to the proceedings of
the Fourth SIAM Conference on Parallel Processing for Scientific
Computing, held in Chicago, Illinois, December 1989.

Edward Anderson and Jack Dongarra
Department of Computer Science
University of Tennessee
107 Ayres Hall
Knoxville, TN 37996

April 23, 1990

Abstract. The LAPACK software project currently under development is intended to provide a portable linear algebra library for high performance computers. LAPACK will make use of the Level 1, 2, and 3 BLAS to carry out basic operations. A principal focus of this project is to implement blocked versions of a number of algorithms to take advantage of the greater parallelism and improved data locality of the Level 3 BLAS. In this paper, we describe our work with variants of some of these algorithms and the performance data we have collected.

LAPACK is planned to be a collection of Fortran 77 subroutines for the analysis and solution of systems of simultaneous linear algebraic equations, linear least-squares problems, and matrix eigenvalue problems [1]. This project will combine the functionality of LINPACK and EISPACK in a single package, incorporate recent algorithmic improvements, and restructure the algorithms to use the Level 2 and 3 BLAS (Basic Linear Algebra Subprograms) for efficiency on today's high-performance computers. We are investigating variant versions of many of the routines in LAPACK. The building blocks of the LAPACK library are the BLAS, a set of standard subroutines for the most common operations in linear algebra [2,3,4]. The original set of BLAS, consisting of vector-vector operations, was used in LINPACK. Recently, specifications have been drawn up for matrix-vector operations (Level 2 BLAS) and matrix-matrix operations (Level 3 BLAS) to meet the demands of multiprocessing, vectorization, and hierarchical memory in today's high-performance computers. In particular, the Level 3 BLAS perform $O(n^3)$ operations on $O(n^2)$ data elements, which helps to improve the ratio of computation to memory references on machines that have a memory hierarchy. This paper describes some of the block factorization routines in LAPACK. The blocked version calls the Level 3 BLAS and, if necessary, an unblocked version of the algorithm to do the processing within a block. The unblocked version calls only Level 1 and 2 BLAS routines and is called directly from the blocked routine if the user has set the blocksize to 1. The LU decomposition is derived by equating the product of a unit lower triangular matrix L and an upper triangular matrix U to the original matrix A. As an illus-

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

In the left-looking algorithm, L_{11}, L_{21}, and L_{31} are already known and we want to solve for the next block column of width NB in L and U. If we equate the second column of the product with the second column of A, we obtain the two matrix equations

$$A_{12} = L_{11} U_{12} \; ; \quad \begin{bmatrix} A_{22} \\ A_{32} \end{bmatrix} = \begin{bmatrix} L_{21} \\ L_{31} \end{bmatrix} U_{12} + \begin{bmatrix} L_{22} \\ L_{32} \end{bmatrix} U_{22}$$

Solving for U_{12} in the first equation requires a solve (with multiple right hand sides) using the lower triangular matrix L_{11}. A matrix-matrix multiply is then used to compute the term involving U_{12} in the second equation and subtract it from the left hand side. An unblocked LU factorization is then applied to the rectangular column of width NB to compute L_{22}, L_{32}, and U_{22}, along with the pivot indices. Block routines have been written for the dense and banded factorizations for solving linear systems, the reductions using orthogonal transformations for eigenvalue computations, and selected other operations. In this section, we provide details on the variants we have implemented for the factorization of dense matrices.

Figure 1: Memory access patterns for variants of LU decomposition

The three block variants we have implemented for the LU factorization of a general matrix are shown in Figure 1. The shaded parts indicate the matrix elements accessed in forming a block row or column, and the darker shading indicates the block row or column being computed. The left-looking variant (described in Section 2) computes a block column at a time using previously computed columns. The right-looking variant (the familiar recursive algorithm) computes a block row and column at each step and uses them to update the trailing submatrix. The Crout variant is a hybrid algorithm in which a block row and column is computed at each step using previously computed rows and previously computed columns. All of the computational work for the LU variants is contained in three routines: the matrix-matrix multiply SGEMM, the triangular solve with multiple right hand sides STRSM, and the unblocked LU factorization for operations within a block column. Table 1 shows the distribution of work among these three routines and the average performance rates on one processor of a Cray 2 for a sample matrix of order 500 using a blocksize of 64. Each variant calls its own unblocked variant, and the row interchanges use about 2% of the total time. The average speed of SGEMM is over 400 megaflops for all three variants, but the average speed of STRSM depends on the size of the triangular matrices. For the left-looking variant, the triangular matrices at each step range in size from NB to $N-NB$, and the average performance is 268 megaflops, while for the right-looking and Crout variants, the triangular matrices are always of order NB and the average speed is only 105 megaflops. Clearly the average performance of the Level 3 BLAS routines in a blocked routine is as important as the percent of Level 3 BLAS work.

Variant	Routine	% operations	% time	avg. megaflops
Left-looking	unblocked LU	10	20	146
(SLUBL)	SGEMM	49	32	438
	STRSM	41	45	268
Right-looking	unblocked LU	10	19	151
(SLUBR)	SGEMM	82	56	414
	STRSM	8	23	105
Crout	unblocked LU	10	16	189
(SLUBC)	SGEMM	82	57	438
	STRSM	8	24	105

Table 1: Breakdown of operations and times for LU variants
for N = 500, NB = 64 (Cray 2-S, 1 processor)

Despite the differences in the performance rates of their components, the block variants of the LU factorization tend to show similar overall performance, with a slight advantage to the right-looking and Crout variants because more of the operations are in SGEMM. Figure 2 shows the performance rates in megaflops of these three variants for different matrix sizes on an 8-processor Cray YMP, along with the performance of the LINPACK routine SGEFA. The optimal blocksize on the Cray computers is 64 for most matrix sizes, but the performance varies less than 10% over a wide range of blocksizes. We have considered three block variants for the Cholesky factorization of a symmetric positive definite matrix. For the purpose of discussion, we consider the factorization $A = LL^T$. In the I-variant, also called the top-looking algorithm, a block row is computed at each step using previously computed rows. The major part of the computation is in updating the current block row using a triangular solve with the leading triangle, which involves the Level 3 BLAS routine STRSM. In the J-variant or left-looking algorithm, a block column is computed at a time using previously computed columns. The major operation is the update of the block column using the matrix-matrix multiply routine SGEMM. In the K-variant or right-looking algorithm, a block column is factored at each step and used to update the trailing submatrix. The update using SSYRK is the dominant operation in this case. Similar performance is observed for these three variants when the three dominant Level 3 BLAS routines are implemented equally well. Figure 2 shows the performance in megaflops vs. N for the three block variants of $A = LL^T$, named SLLTBI, SLLTBJ, SLLTBK, along with the performance of the LINPACK factorization SPOFA, on an 8-processor Cray YMP. As in the case of the LU decomposition, the right-looking variant (SLLTBK) and the variant which calls SGEMM for most of its Level 3 BLAS work (SLLTBJ) are slightly better than the left-looking variant, which does more of its Level 3 BLAS work in STRSM.

Figure 2: Performance of LU and Cholesky variants (Cray YMP, 8 processors)

We have considered two factorizations for symmetric indefinite matrices, the Bunch-Kaufman diagonal pivoting method, which was used in LINPACK, and Aasen's method [5]. The form of the Bunch-Kaufman factorization is

$$PAP^T = LDL^T$$

where P is a permutation matrix, L is unit lower triangular, and D is block diagonal with 1×1 or 2×2 diagonal blocks. A 2×2 pivot block is chosen in order to avoid large entries in the factor L when the diagonal entries are small in magnitude relative to the offdiagonal entries. The factorization from Aasen's method has the form

$$PAP^T = LTL^T$$

where T is tridiagonal. Block versions of each of these factorizations have been developed [6]. The block versions accumulate the elementary transformations and apply them as a rank-k update, in one of the forms

$$A \leftarrow A - XDX^T \text{ (Bunch–Kaufman)}$$
$$A \leftarrow A - XTX^T \text{ (Aasen)}$$

Table 2 compares the performance of the unblocked (Level 2 BLAS) and blocked (Level 3 BLAS) versions of each method on one processor of a Cray 2. We see that while the unblocked form of Aasen's method is better than the unblocked Bunch-Kaufman factorization for large matrices, blocking favors the Bunch-Kaufmann factorization for all matrix sizes.

N	Aasen		Bunch-Kaufman	
	Level 2	Level 3	Level 2	Level 3
100	37	40	44	48
200	79	92	88	109
300	114	147	115	168
400	142	180	132	198
500	165	215	140	238
700	201	257	158	265
1000	236	292	171	314

Table 2: Performance in megaflops of symmetric indefinite factorizations (Cray 2, 1 processor)

The QR decomposition, used in solving linear least squares problems, factors a matrix A as QR, where Q is orthogonal and R is upper triangular. The matrix Q is a product of $n-1$ elementary reflectors (or Householder transformations) $Q = H_1 H_2 \cdots H_{n-1}$, where $H_i = I - \tau_i v_i v_i^T$ and $v_i = [0 \cdots 0\, 1\, x^T]^T$. In the unblocked QR decomposition, a reflector H_i is computed at each step and then applied to the matrix A. A block form of the QR decomposition is obtained by combining several elementary Householder matrices into a block Householder matrix. The product of elementary matrices can be written as

$$\prod_{i=1}^{k} (I - \tau_i v_i v_i^T) = I - VSV^T$$

where $V = [v_1 v_2 \cdots v_k]$ and S is a $k \times k$ upper triangular matrix. Some extra work is required to compute S, so the optimal block size is usually smaller than for LU or Cholesky. In order to obtain a fair comparison of the variants, we always use the operation count for the unblocked algorithm. Table 3 shows the performance in megaflops of four variants of the QR decomposition on one processor of a Cray 2. The two block variants are SQRR, a block right-looking algorithm in which a block Householder matrix is computed and immediately applied to the rest of the matrix as a rank-k update, and SQRL, a block left-looking variant in which the previous updates are first applied to the current block column before the next block Householder matrix is computed. SGEQR2 is the unblocked Level 2 BLAS variant and SQRDC is the Level 1 BLAS variant from LINPACK. We see that the blocked variants only surpass the unblocked variant in performance for matrices of order greater than 200.

QR variant	Matrix size M = N				
	100	200	300	400	500
SQRR (NB = 32)	106	209	269	306	328
SQRL (NB = 48)	102	198	258	293	316
SGEQR2	144	215	242	251	255
SQRDC	24	41	55	66	76

Table 3: Performance in megaflops of QR variants (Cray 2-S, 1 processor)

Block algorithms have also been developed for the Cholesky and LU factorizations of band matrices [7]. The idea is to factor a small diagonal block using an unblocked algorithm, and then to update the matrix within the band using Level 3 BLAS. The use of Level 3 BLAS in the factorization of band matrices can improve the performance of the factorization as long as the bandwidth is not too small. Other block algorithms in LAPACK include the routines for reducing a general rectangular matrix to Hessenberg form, reducing a symmetric matrix to tridiagonal form, generating and multiplying by an orthogonal matrix represented as a product of Householder transformations, and computing the inverse of a square nonsingular matrix. The reductions follow the blocking strategy outlined in [8], and as for the QR decomposition, extra work is required to combine the updates to use Level 3 BLAS. As a result, improvements in performance compared to the unblocked routines are typically observed only for relatively large matrices (order 300 or higher). For each factorization or reduction we have considered, the percentage of BLAS 3 work is the same in the different block variants, but this is not the only consideration. The performance of the Level 3 BLAS routines is dependent on the matrix shapes on which they operate, and in a block algorithm one of the dimensions is always on the order of the blocksize. Small blocksizes will result in performance similar to the unblocked Level 2 BLAS algorithms, for which the choice of variant is highly machine dependent. However, blocking tends to smooth out the differences in the Level 2 BLAS algorithms, so the choice of a block variant on computers such as Crays is not so critical. Although we are considering different block variants now, the public release of LAPACK in 1991 will contain only one variant of each algorithm. For shared memory machines, our current policy is to choose the unblocked variant based on the matrix-vector multiply and the blocked variant for which the dominant Level 3 BLAS operation is the matrix-matrix multiply. For distributed memory machines, we expect the right-looking variants based on the rank-k update to be most useful. Further data is needed before a final decision can be made on which variants will give the best performance over the widest range of high-performance computers.

REFERENCES

C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen, *LAPACK Working Note #5: Provisional Contents*, Argonne National Lab., ANL-88-38, Sept. 1988. C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, *Basic Linear Algebra Subprograms for Fortran Usage*, ACM Trans. Math. Soft., 5 (Sept. 1979), pp. 308-323. J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson, *An Extended Set of Fortran Basic Linear Algebra Subprograms*, ACM Trans. Math. Soft., 14 (Mar. 1988), pp. 1-17. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, to appear in ACM Trans. Math. Soft., Mar. 1990. G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins, Baltimore, 1989. D. Sorensen and C. Van Loan, personal communication. P. Mayes and G. Radicati, *LAPACK Working Note #12: Banded Cholesky Factorization Using Level 3 BLAS*, Argonne National Lab., ANL/MCS-TM-134, Aug. 1989. J. Dongarra, S. Hammarling, and D. Sorensen, *LAPACK Working Note #2: Block Reduction of Matrices to Condensed Forms for Eigenvalue Computations*, Argonne National Lab., ANL/MCS-TM-99, Sept. 1987.

References