M. Louter-Nool

LINPACK routines based on the level 2 BLAS

Department of Numerical Mathematics Report NM-R8811 September

LINPACK routines based on the Level 2 BLAS

Margreet Louter-Nool
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

The Extended or Level 2 BLAS is intended to improve the performance of portable programs on high-performance computers. In this paper we examine where Extended BLAS routines may be inserted in the LINPACK, such that no changes in the parameter list have to be made. We also discuss why, for some algorithms, a simple restructuring in terms of Level 2 BLAS fails. We do not attempt to re-design the algorithms or to change the data structure. We concentrate on the translation of calls to original (Level 1) BLAS into calls to Level 2 BLAS in order to improve readability, modularity and efficiency. This examination results in a still portable subset of the LINPACK with a better performance than the original routines. The measured performances of original and modified LINPACK routines on the CDC CYBER 990, CDC CYBER 205, CRAY X-MP and the NEC SX-2 are compared and analyzed.

1980 Mathematics subject classification: Primary:65V05. Secondary:65FXX
Key Words & Phrases: Matrix-vector operations, BLAS, Extended or Level 2 BLAS, Vector and parallel computers, Efficiency, Portability, Performance Measurements.

Note: This paper is submitted for publication elsewhere

1. Introduction

On vector and parallel computers the number of floating point operations is not a suitable measure to check whether a particular algorithm is faster than another. It is well-known that on such machines optimization on vector-vector level like the original BLAS[14] is not sufficient, and that larger units than the BLAS are required. More precisely, on uniprocessor vector machines the set of matrix-vector operations of the Extended BLAS[5] seems to be well suited. On multiprocessor machines, and machines where the performance is dominated by data traffic, better performance can be achieved by the use of Level 3 BLAS[4]. For those architectures the original matrices are partitioned into blocks and the matrix-matrix operations are performed on those blocks. As a consequence, algorithms must be recasted in terms of matrix-vector or matrix-matrix operations to reduce the number of memory references; such a reduction will increase the efficiency of high-performance computers.

A well-known technique that is used to achieve high rates of execution is to overlap operations if two operations are independent. For the _AXPY operation ($y \leftarrow y + a \, . \, x$) two floating point operations can deliver one result per clock cycle. The same rate can sometimes be obtained for the _DOT operation. Another way to reduce the number of memory references is the technique of loop-unrolling[7, 8], where the output register of one vector instruction is the same as one of the input registers for the next instruction. On a CRAY-1 supervector performance can be achieved without resorting to assembler language when applying this technique.

Van der Vorst[21] discusses the need to adapt and extend general numerical software libraries, so that they perform efficiently on vector and parallel computers as well. We will show that using the Level 2 BLAS may be a major step in this direction. Especially on high-performance machines, the use of existing libraries will be preferable; for small runs or for testing parts of a code, it is desirable to maintain portability to other machines. We have concentrated on the LINPACK as an example of

Report NM-R8811
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

a widely used library. Moreover, the codes are easily available, via the NETLIB facility[8].

Though we realize that often an extensive restructuring of the codes (see, e.g., Dongarra, Kaufman and Hammerling[9], and Dongarra and Sorensen[10]) will be necessary in order to improve the performance, only small modifications are applied here. Our aim is to insert Level 2 BLAS routines in the LINPACK package without changing the parameter list and the data structure and to keep the same round-off pattern. We assume that users of portable software do not like to adapt their programs. Any adaptation of the parameter list will result in changes in the calling program. For that reason, we have excluded many routines from insertion of the Level 2 BLAS rather than to allow additional workspace or data restructuring.

Currently, a new linear algebra package, the LAPACK[1] is being developed based on the LINPACK[3] as well as the EISPACK[12,19] libraries. The use of the BLAS, including Level 2 and 3, will be the basis to achieve efficiency for that package. In order to improve the performance, the codes will be restructured extensively, whereas some algorithms may be deleted, others may be extended. As opposed to our strategy, we expect that the data structure and the calling sequence of the LAPACK library may be fairly deviate from the original LINPACK library.

In Section 2, we discuss the recoding strategy and we also mention which loops cannot be replaced by calls to Level 2 BLAS. Examples of recoding for various types of matrices are given in Section 3. Throughout this paper timings on various machines are given. These timings, of course, depend on the properties of the Level 2 BLAS routines being called, and on the quality of the compiler. In Section 4, the performances of the Level 2 BLAS are considered. We used its model implementation[6], and as far as available, a machine-dependent implementation. In Section 5, execution times of the original LINPACK and the modified codes are compared. Finally, some remarks on the use, the contents and the data structure of the Level 2 BLAS can be found in Section 6.

2. Recoding strategy

The naming convention for the LINPACK routines that we have considered, is as follows. The first character indicates the data type of the matrix :

S	- REAL
C	- COMPLEX

The second and third characters denote the kind of matrix :

GE	- General
GB	- General band
PO	- Positive definite
PP	- Positive definite packed
PB	- Positive definite band
TR	- Triangular
GT	- General tridiagonal
PT	- Positive definite tridiagonal
SI	- Symmetric indefinite
SP	- Symmetric indefinite packed

For each form there are four subroutines :

FA	- Factor
CO	- Estimate condition
SL	- Solve
DI	- Determinant, inverse, inertia

The original LINPACK subroutines are based on the Level 1 BLAS. Most of the loops containing calls to _AXPY or _DOT can be replaced by a single call to a Level 2 BLAS routine. The advantages of the use of the Level 2 BLAS are :

- the modularity and clarity of the LINPACK subroutines are improved.

- operating on the matrix-vector level offers more possibilities for speed improvement.
- LINPACK still remains portable and can be executed on all kinds of serial and vector computers, for which the model implementation or a machine-dependent implementation of the Level 2 BLAS are present.

The subroutines have been rewritten in the following way :

1. FORTRAN66 → FORTRAN77
   Examples :
   . GO TO statements have been replaced by IF THEN ELSE constructions to improve the readability.
   . omission of tests on empty loops.
   . array specifications have been adapted according to FORTRAN77.
2. insertion of Level 1 BLAS routines with increment value greater than 1.
   Examples :
   . to determine the largest off diagonal element in a column, LINPACK uses I_AMAX, and for the largest off-diagonal element in a row a loop is used. Now in both cases I_AMAX is called.
   . to swap two rows a call to _SWAP has been inserted.
3. loops with calls to _AXPY and _DOT have been replaced by a call to a Level 2 BLAS routine, if possible.

Not nearly all LINPACK routines can be restructured in terms of Level 2 BLAS modules for the following reasons :

1. The scaling that is done in _CO routines makes it impossible to substitute Level 2 BLAS routines for sections of the existing code. Nevertheless, the performance can change, since each _CO routine calls its corresponding _FA routine.
2. Not every routine performs a matrix-vector operation. For example, SGBDI computes only the determinant of a factorized matrix.
3. The absence of sufficient workspace prohibits a useful modification for subroutines computing Cholesky, QR or singular value decomposition, updating, downdating and exchanging; those are not considered in this paper. If sufficient workspace is missing in the parameter list, subsequent matrix-vector operations can not pass temporary information. An illustration :
```
      DO 10 J = JL, JU
        T = -SDOT (N-L+1, U(L,L), 1, U(L,J), 1) / U(L,L)
        CALL SAXPY (N-L+1, T, U(L,L), 1, U(L,J), 1)
   10 CONTINUE
```
   Clearly, this loop could be replaced by
```
      T = -1.0E0 / U(L,L)
      CALL SGEMV ('Transpose', N-L+1, JU-JL+1, T, U(L,JL), LDU, U(L,L), 1, 0.0E0, WORK, 1)
      CALL SGER  (N-L+1, JU-JL+1, 1.0E0, U(L,L), 1, WORK, 1, U(L,JL), LDU)
```
   if workspace of length $ju - jl + 1$ were available. Note that in the original code only a scalar value is passed.
4. Operations on complex symmetric matrices have not been included in the Level 2 BLAS. Hence, the CSI_ subroutines could not be restructured. The subroutines for complex Hermitian matrices did not present additional problems, except that we missed a conjugated _SWAP.
5. For the solution of $Ax = b$, by LU-factorization, the solution of $Ly = b$ and then $Ux = y$ is required. Since L is not explicitly stored, the same pivoting and elimination operations are applied to b and to the columns of A in the factorization routine, and this prohibits the use of Level 2 BLAS routines. Of course, the solution of the second equation $Ux = y$ can be translated into calls to BLAS2 modules.

3. Some notes on the recoding

3.1. General matrices

To give an illustration of the recoding we analyze a part of the routine SGEFA. This routine factors a real matrix by Gaussian elimination with partial pivoting. At each step in the original code, the k-th reduced submatrix is modified as follows:

```
      DO 30 J = K+1, N
        T = A(L,J)
        IF (L .EQ. K) GO TO 20
          A(L,J) = A(K,J)
          A(K,J) = T
20      CONTINUE
        CALL SAXPY (N-K, T, A(K+1,K), 1, A(K+1,J), 1)
30    CONTINUE
```

Obviously, the k-th and l-th row starting in element $k+1$ can be interchanged outside the loop. This modification does not only save superfluous tests on l equal k, but also this operation can be vectorized now. The remaining part of the loop is a simple rank-1 update. The original LINPACK code can be replaced by

```
      IF (L .NE. K) THEN
        CALL SSWAP (N-K, A(L,K+1), LDA, A(K,K+1), LDA)
      END IF
      CALL SGER (N-K, N-K, 1.0E0, A(K+1,K), 1, A(K,K+1), LDA, A(K+1,K+1), LDA)
```

We have showed here a straightforward translation of the original code into a Level 2 BLAS code. In the literature, other algorithms are discussed to perform this factorization, including methods that can be carried out by calls to Level 2 BLAS. For instance, in Dongarra and Sorensen[10], three alternative methods are given all updating only the k-th row or column instead of the entire submatrix of order k, as in the LINPACK code. Dongarra and Eisenstat[7, Table III] show how one of these methods, performing only matrix-vector multiplications, produces supervector speed on a CRAY-1 without resorting to assembler language. For this purpose they suggested the technique of loop-unrolling to reduce the number of memory references. This technique is very convenient for vector-register machines, but on a direct-memory access machine, like the CYBER 205, the effect is less obvious.

For SGEFA this has the following consequences. Though both methods have the same number of floating point operations, the number of vector operations is different. The LINPACK algorithm needs k vector operations for $k=1, \cdots, n-1$ per step, whereas the Dongarra-Eisenstat approach requires $n-1$ vector operations per step. It turns out that, on the CYBER 205 with its large startup times the Dongarra-Eisenstat approach doubles the execution time. We observe, that we have found two different algorithms, both performing the same factorization, though with a quite different performance characteristic. It seems that each architecture can claim its best algorithm. The LAPACK project aims to choose the structure that provides the best "average" performance over a range of target machines, we wonder what the "average" will be in this case.

3.2. Band matrices

For band matrix routines the modifications are more complicated. The storage convention of the Level 2 BLAS is similar to that of the LINPACK. Hence matrices are stored in rectangular arrays, such that diagonals of the matrix are stored in rows, and columns are stored in corresponding columns of the array. Consider SGBFA that factors a real band matrix. Again two rows must be interchanged and in the original code the elimination part is carried out by SAXPY-operations:

```
         MM = M
         IF (JU .LT. K+1) GO TO 90
         DO 80 J = K+1, JU
           L = L - 1
           MM = MM - 1
           T = ABD(L,J)
           IF (L .EQ. MM) GO TO 70
             ABD(L,J) = ABD(MM,J)
             ABD(MM,J) = T
   70      CONTINUE
           CALL SAXPY (LM, T, ABD(M+1,K), 1, ABD(MM+1,J), 1)
   80    CONTINUE
   90    CONTINUE
```

The row interchanging can be implemented by means of a SSWAP with increment values LDA-1, i.e. one less than the leading dimension declared in the calling program. The same trick can be applied for the elimination; the reduced submatrix is then stored in a nonrectangular array. The modified code becomes

```
         IF (L .NE. M) THEN
           CALL SSWAP (JU-K, ABD(L-1,K+1), LDA-1, ABD(M-1,K+1), LDA-1)
         END IF
         CALL SGER (LM, JU-K, 1.0E0, ABD(M+1, K), 1,
     +              ABD(M-1,K+1), LDA-1, ABD(M,K+1), LDA-1)
```

In general, the band width and the corresponding vector length will be too small to obtain a high performance.

3.3. Positive definite matrices

The number of floating-point operations for factorizing symmetric positive definite full matrices is about half the corresponding number of operations for nonsymmetric matrices. The triangular factorization can easily be performed by calls to _TRSV, the triangular solver of the Extended BLAS. The basis operation to factorize a general matrix is the rank-1 update _GER, as described in Section 3.1. The performance of this update is more than twice that of the triangular solver (_TRSV) used for the symmetric case. Indeed, our experiments (see Section 5) show that when factorizing symmetric matrices, it does not always pay to exploit the symmetric structure. This is another example of an algorithm that takes advantage of a special structure, although it will not always be faster than one that ignores the special structure. Algorithms designed for structured systems may have to be reworked to be competitive on a vector machine, as is shown by Kaufman[13] and others.

3.4. Symmetric indefinite matrices

The recoding of the factorization of symmetric indefinite matrices (SSIFA, SSPFA, CSIFA, CSPFA, CHEFA and CHPFA) requires some explication. For the real symmetric case, the code used to perform the 2 X 2 pivot block elimination in the original LINPACK looked like

```
For $j = k-2$ to 1 by $-1$
    $mul_k = d_{11}\ a_{jk} + d_{12}\ a_{j,k-1}$
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + mul_k\ a_{ik}$
    End
    $mul_{k-1} = d_{21}\ a_{jk} + d_{22}\ a_{j,k-1}$                (3.4.1)
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + mul_{k-1}\ a_{i,k-1}$
    End
    $a_{jk} = mul_k$
    $a_{j,k-1} = mul_{k-1}$
End
```

where the d_i's can be computed outside the outermost loop. Both i-loops have been implemented using _AXPY's. If we wish to use matrix-vector operations rather than vector-vector operations we might replace (3.4.1) by

```
For $j = k-2$ to 1 by $-1$
    $mul_{jk} = d_{11}\, a_{jk} + d_{12}\, a_{j,k-1}$
End
For $j = k-2$ to 1 by $-1$
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + mul_{jk}\, a_{ik}$
    End
End
For $j = k-2$ to 1 by $-1$
    $mul_{j,k-1} = d_{21}\, a_{jk} + d_{22}\, a_{j,k-1}$                    (3.4.2)
End
For $j = k-2$ to 1 by $-1$
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + mul_{j,k-1}\, a_{i,k-1}$
    End
End
For $j = k-2$ to 1 by $-1$
    $a_{jk}$     $= mul_{jk}$
    $a_{j,k-1} = mul_{j,k-1}$
End
```

Obviously, the matrix updates are no symmetric rank-1 updates and Level 2 BLAS routines are not available. For that purpose we have to split the operations into

```
For $j = k-2$ to 1 by $-1$
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + d_{11}\, a_{jk}\, a_{ik}$
    End
End
For $j = k-2$ to 1 by $-1$
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + d_{12}\, a_{j,k-1}\, a_{ik}$
    End
End
For $j = k-2$ to 1 by $-1$
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + d_{21}\, a_{jk}\, a_{i,k-1}$                    (3.4.3)
    End
End
For $j = k-2$ to 1 by $-1$
    For $i = 1$ to $j$
        $a_{ij} \leftarrow a_{ij} + d_{22}\, a_{j,k-1}\, a_{i,k-1}$
    End
End
For $j = k-2$ to 1 by $-1$
    $a_{jk}$     $\leftarrow d_{11}\, a_{jk} + d_{12}\, a_{j,k-1}$
    $a_{j,k-1} \leftarrow d_{21}\, a_{jk} + d_{22}\, a_{j,k-1}$
End
```

Let $a_{.k}$ and $a_{.k-1}$ denote the k-th and $(k-1)$-st column of A. Then (3.4.3) becomes

$$\begin{aligned} &1: \quad A_k^{(1)} \leftarrow A_k \quad + d_{11}\, a_{.k}\, a_{.k}^T \\ &2: \quad A_k^{(2)} \leftarrow A_k^{(1)} + d_{12}\, a_{.k}\, a_{.k-1}^T \\ &3: \quad A_k^{(3)} \leftarrow A_k^{(2)} + d_{21}\, a_{.k-1}\, a_{.k}^T \\ &4: \quad A_k^{(4)} \leftarrow A_k^{(3)} + d_{22}\, a_{.k-1}\, a_{.k-1}^T \end{aligned} \tag{3.4.4a}$$

and two vector updates

$$\begin{aligned} &1: \quad a_{.k}^{(1)} \quad \leftarrow d_{11}\, a_{.k} + d_{12}\, a_{.k-1} \\ &2: \quad a_{.k-1}^{(1)} \leftarrow d_{21}\, a_{.k} + d_{22}\, a_{.k-1} \end{aligned} \tag{3.4.4b}$$

The first and fourth updates are now symmetric rank-1 updates, and the second and third matrix update can be replaced by a single rank-2 update. However, the original loop takes

$$2\,(\,k\,-\,2\,)\,(\,k\,+\,2\,)$$

operations to compute the submatrix A_k and here we have

$$4\,(\,k\,-\,2\,)\,(\,k\,-\,3/2\,)$$

operations. So, at the cost of about twice as many operations we could translate the j-loop of (3.4.1) into three calls to Level 2 BLAS routines.

Notice that in (3.4.4a) only two vectors $a_{.k}$ and $a_{.k-1}$ are involved for updating. The matrix operation may be carried out by just one single Level 2 BLAS rank-2 update, performing

$$A_k^{(4)} \leftarrow A_k + \alpha\, x\, y^T + \alpha\, y\, x^T \tag{3.4.5a}$$

For that purpose we have to construct the vectors, say

$$\begin{aligned} x &= c_1\, a_{.k} + c_2\, a_{.k-1} \\ y &= c_3\, a_{.k} + c_4\, a_{.k-1} \end{aligned} \tag{3.4.5b}$$

and solve the following equations for the symmetric case

$$\begin{aligned} &2\,\alpha\, c_1\, c_3 = d_{11} \\ &2\,\alpha\, c_2\, c_4 = d_{22} \\ &\alpha\, c_1\, c_4 + \alpha\, c_2\, c_3 = d_{12} = d_{21} \end{aligned} \tag{3.4.6a}$$

and for the complex Hermitian case

$$\begin{aligned} &\alpha\, c_1\, \bar{c}_3 + \bar{\alpha}\, \bar{c}_1\, c_3 = d_{11} \\ &\alpha\, c_2\, \bar{c}_4 + \bar{\alpha}\, \bar{c}_2\, c_4 = d_{22} \\ &\alpha\, c_1\, \bar{c}_4 + \bar{\alpha}\, \bar{c}_2\, c_3 = d_{12}\,, \end{aligned} \tag{3.4.6b}$$

where d_{11} and d_{22} are real numbers and $d_{12} = \bar{d}_{21}$ are complex numbers for complex Hermitian matrices. In the real symmetric case, we can determine real values for the c_i's and α. For real symmetric or complex Hermitian matrices, we can use SSYR2 and SSPR2, or, CHER2 and CHPR2, respectively. The set of Level 2 BLAS, however, does not provide routines for complex symmetric matrices. Consequently, the CSIFA and CSPFA can not be adapted.

Unfortunately, the subroutines to factorize real symmetric and complex Hermitian matrices does not use any workspace. Since we do not wish to change the parameter list, we have to use some space of the matrix A. In (3.4.7), the columns $a_{.k}$ and $a_{.k-1}$ are replaced by the the vectors x and y from (3.4.5b) and after updating of A_k they will contain the values of (3.4.4b); the same information is passed as by the original code. This implies that for instance a call to the modified SSIFA can be followed by a call to the original SSISL, or SSIDI. The algorithm becomes

```
If $k \neq 2$ then
    Compute $\delta_1,\delta_2,\delta_3,\delta_4$, such that $a_{.k}$ and $a_{.k-1}$ will contain $x$ and $y$ of (3.4.5b)
    $a_{.k} \leftarrow \delta_1 a_{.k}$
    $a_{.k} \leftarrow a_{.k} + \delta_2 a_{.k-1}$
    $a_{.k-1} \leftarrow \delta_3 a_{.k-1}$
    $a_{.k-1} \leftarrow a_{.k-1} + \delta_4 a_{.k-1}$
    Compute $\alpha$
    $A_k \leftarrow A_k + \alpha a_{.k} \bar{a}_{.k-1}^T + \bar{\alpha} a_{.k-1} \bar{a}_{.k}^T$          (3.4.7)
    Compute $\gamma_1,\gamma_2,\gamma_3,\gamma_4$, such that $a_{.k}$ and $a_{.k-1}$ become the vector updates (3.4.4b)
    $a_{.k} \leftarrow \gamma_1 a_{.k}$
    $a_{.k} \leftarrow a_{.k} + \gamma_2 a_{.k-1}$
    $a_{.k-1} \leftarrow \gamma_3 a_{.k-1}$
    $a_{.k-1} \leftarrow a_{.k-1} + \gamma_4 a_{.k}$
End if
```

We observe, that we have not changed the strategy, we have only changed the order of computation. Tables 5a-b show that the rank-2 update codes require considerably less execution time. Note that the application of a rank-2 update would be much more profitable, if workspace were on hand.

3.5. Tridiagonal matrices

The general tridiagonal matrices in _GTSL and the symmetric positive definite tridiagonal matrices in _PTSL are not stored according to the convention of general band matrices; their description consists of three or two array names, respectively. So, Level 2 BLAS routines can not be used for such matrices. Moreover, if as usual, the diagonals were stored in rows, the maximum vector length would not exceed the bandwidth of two or three, which would imply scalar speed.

In the last few years, many techniques for bi- and tridiagonal systems have been developed for advanced computer architectures[20, 22]. If diagonals were stored by columns rather than by rows, efficient bi- or tridiagonal solvers could be implemented as special cases for band matrices. A Level 2 BLAS implementation would be of much more interest, if such efficient codes for diagonal systems were included.

4. Performance of the Level 2 BLAS

The performance of both the original and the modified LINPACK is determined mainly by the performance of the Level 1 and Level 2 BLAS, respectively. In this section, we first review the peak performance of the machines considered in this paper. Then we list the speed in Mflops of the Extended BLAS routines. Finally, we show which Level 2 BLAS routines are called by the LINPACK routines.

Table 1 contains the cycle time and peak performance of the various machines.

Machine	cycle time in nanoseconds	peak performance in Mflops
CDC CYBER 990	16	60
CDC CYBER 205	20	200
CRAY X-MP/2 (1 proc.)	8.5	235
NEC SX-2	7	1300

Table 1 Cycle Times and Peak Performances

At this moment, on the CDC CYBER 205 optimized versions of all real Level 2 BLAS routines are available as well as the complex rank 1 and 2 updates. For the other machines, only the model implementation[6], which is written in portable FORTRAN77, is available, although its performance is much below the best possible. Timing results of the model implementation and of the machine-dependent CWI BLAS2, optimized for the CYBER 205, are listed in Tables 2a-b (the symbol - denotes : not yet available). All dense matrices are of order 255 and all band matrices are of order 6500 with a maximum band width of 10, so all matrices contain approximately the same number of elements. For each routine two Mflops values are given, one for INCX = (INCY =) 1 and one for INCX = 1 and INCY > 1, in case of two vector arguments, or INCX > 1 otherwise. These combinations of increment values often occur in the implementation of linear algebra subroutines.

The routines for solving triangular equations _TRSV, _TPSV and _TBSV allow for the matrix to be stored either in the upper or lower triangle, i.e., UPLO = 'Upper', or UPLO = 'Lower'. The parameter TRANS is used to specify whether the operation is performed on the matrix or its transpose. The performances of all possible cases are listed, where the suffices _*i* mean :

_1	UPLO = 'Upper',	TRANS = 'No Transpose'
_2	UPLO = 'Upper',	TRANS = 'Transpose'
_3	UPLO = 'Lower',	TRANS = 'No Transpose'
_4	UPLO = 'Lower',	TRANS = 'Transpose'

Mflops for Real Extended BLAS routines										
	INCX = INCY = 1					INCX = 1, INCY > 1				
	CYBER 990	CYBER 205		CRAY X-MP	NEC SX-2	CYBER 990	CYBER 205		CRAY X-MP	NEC SX-2
	model impl.	model impl.	CWI BLAS2	model impl.	model impl.	model impl.	model impl.	CWI BLAS2	model impl.	model impl.
SGEMV	17	75	89	118	393	4	4	88	116	379
SGBMV	2	3	38	14	16	2	3	33	14	14
SSYMV	9	50	51	82	136	6	7	51	70	132
SSPMV	7	8	49	81	135	7	7	49	75	127
SSBMV	2	2	50	4	7	2	4	42	4	6
STRMV	15	41	55	83	130	3	4	54	54	128
STPMV	4	4	51	84	132	4	4	51	54	124
STBMV	3	4	35	12	13	2	3	32	11	12
STRSV_1	7	8	49	78	114	3	3	47	65	116
STRSV_2	8	31	37	53	91	8	5	37	44	88
STRSV_3	22	35	46	79	117	4	3	45	60	117
STRSV_4	8	11	37	52	89	8	5	36	47	88
STPSV_1	3	4	44	84	116	4	3	43	66	113
STPSV_2	8	5	35	54	91	8	5	34	46	87
STBSV_1	2	2	5	11	11	2	2	5	11	11
STBSV_2	4	3	5	5	11	3	3	5	5	10
SGER	23	77	94	118	410	24	77	94	71	416
SSYR	16	47	60	99	257	4	4	60	63	231
SSPR	4	4	60	99	252	4	4	60	64	217
SSYR2	21	58	73	130	338	7	6	73	88	303
SSPR2	8	6	73	129	333	8	6	73	90	291

TABLE 2a Mflops for Real Extended BLAS routines
$n=6500$, $kl=ku=3 \vee k=9$ for band matrices,
$m=n=255$ for dense matrices.

Mflops for Complex Extended BLAS routines										
	INCX = INCY = 1					INCX = 1, INCY > 1				
	CYBER 990	CYBER 205		CRAY X-MP	NEC SX-2	CYBER 990	CYBER 205		CRAY X-MP	NEC SX-2
	model impl.	model impl.	CWI BLAS2	model impl.	model impl.	model impl.	model impl.	CWI BLAS2	model impl.	model impl.
CGEMV	14	34	-	147	551	9	8	-	131	541
CGBMV	4	4	-	37	27	6	7	-	33	25
CHEMV	8	13	-	110	232	7	12	-	104	228
CHPMV	9	13	-	110	231	7	12	-	105	229
CHBMV	4	10	-	10	11	3	9	-	9	11
CTRMV	10	28	-	128	270	7	8	-	116	266
CTPMV	9	8	-	129	270	7	8	-	113	260
CTRSV_1	7	2	-	127	252	8	8	-	113	255
CTRSV_2	10	8	-	82	166	12	8	-	81	161
CTRSV_3	17	26	-	125	259	8	8	-	114	256
CTRSV_4	7	8	-	76	161	15	8	-	79	160
CTPSV_1	9	8	-	124	254	10	8	-	113	254
CTPSV_2	16	8	-	82	164	12	8	-	78	159
CTBSV_1	2	2	-	33	25	4	6	-	31	25
CTBSV_2	3	6	-	11	18	7	6	-	11	18
CGERC	14	34	124	150	567	17	34	123	149	560
CGERU	17	34	134	150	571	17	35	133	149	564
CHER	13	28	89	129	299	7	8	88	136	298
CHPR	9	8	89	130	305	9	8	89	121	299
CHER2	15	29	99	157	331	10	14	98	146	330
CHPR2	11	14	96	157	336	8	13	95	144	332

TABLE 2b Mflops for Complex Extended BLAS routines
$n=6500$, $kl=ku=3 \vee k=9$ for band matrices,
$m=n=255$ for dense matrices.

Obviously, the CYBER 205 compiler fails to vectorize the routines for packed forms, i.e., the routines with a P as third character. It should be noted that also a nonsequential storage (see the right part of Table 2a) degrades the performance of both CYBERs considerably. The optimized FTN200 version for the CYBER 205 as well as the model implementation on the CRAY and the NEC are less sensitive to nonsequential storage. Because of the specific storage scheme for banded matrices (cf. Section 3.2) the vector-length is at most $kl+ku+1$ for general banded matrices and $k+1$ for other banded matrices, if operations are performed on columns rather than rows. However, operating on diagonals stored in rows of the array, results in vector lengths close to n and consequently for large n high performances can be obtained despite the stride problem. So, both the storage convention and the column-oriented approach of the model implementation cause the low performance for the _GBMV and _TBMV routines. For the CWI BLAS2, which operates on diagonals, much higher Mflops-rates are obtained compared to the other machines. It would have been better if the model implementation had been coded along these lines; its performance for banded matrices is very poor, now.

Note that for the solution of a banded matrix in the _TBSV routines, the recurrence relations prevent vectorization. We remark, however, that for banded systems much higher performances can be achieved than the Mflops-rates listed here. In this context we refer to [22] in which a number of techniques for solving tridiagonal linear systems are discussed. Most techniques are also applicable to more general banded systems.

	SGEMV	SSYMV	SSPMV	STPMV	STRSV1	STRSV2	STRSV3	STRSV4	STPSV1	STPSV2	STBSV1	STBSV2	SGER	SSYR	SSPR	SSYR2	SSPR2
SGEFA	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-
SGESL	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-
SGEDI	+	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-
SGBFA	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-
SGBSL	-	-	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-
SPOFA	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-
SPOSL	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-
SPODI	-	-	-	-	-	-	-	-	-	-	-	-	+	+	-	-	-
SPPFA	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-
SPPSL	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-	-	-
SPPDI	-	-	-	+	-	-	-	-	-	-	-	-	-	-	+	-	-
SPBFA	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-
SPBSL	-	-	-	-	-	-	-	-	-	-	+	+	-	-	-	-	-
SSIFA	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	+	-
SSIDI	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SSPFA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	+
SSPDI	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-
STRSL	-	-	-	-	+	+	+	+	-	-	-	-	-	-	-	-	-
STRDI	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-

TABLE 3 Relation between Real Modified LINPACK routines and Level 2 BLAS

The overview in Table 3 explains which of the specific Level 2 BLAS routines were inserted in the LINPACK subroutines. In practice, the Mflops-rates of Tables 2a-b will not be reached for LINPACK routines: in most algorithms the Level 2 BLAS operations are performed on matrices of order k, where $k=1, \cdots, n$ for dense matrices; for banded matrices k will not exceed the band width.

5. EXPERIMENTS

In this section we compare the performance on vector computers of several original LINPACK routines (i.e., those based on Level 1 BLAS subroutines) with the performance of the modified LINPACK routines (i.e., those based on the Level 2 BLAS subroutines). All new codes were primarily tested on a serial machine, a CDC CYBER 750. As opposed to the high-performance vector computers discussed in this paper, this serial machine can be used interactively, a very pleasant circumstance. Not only correctness tests have been carried out on this machine, but also some timings of the programs have been collected. Since, by replacing calls to the Level 1 BLAS by calls to Level 2 BLAS, the number of floating point operations remains unchanged, we expected only a small effect from this recoding on the CYBER 750. Nevertheless, in some cases we found significant deviations in performance. It turns out that calls to the logical function LSAME in the Level 2 BLAS routines are much more expensive than straightforward comparison of characters.

We expect LINPACK based on Level 2 BLAS to run more efficiently than LINPACK based on Level 1 BLAS because of the following reasons : less subroutine call overhead, less parameter checking within the subroutines and less increment testing in loops. Actually, the use of any BLAS is rather expensive, in particular, if no optimized implementations are available. Central to most algorithms in linear algebra are inner product computations (_DOT operation) and additions of a scalar multiple of one vector to another vector (_AXPY operation). If one should replace the BLAS routines for these operations by single DO-loops, avoiding parameter checks and tests on increment values (always equal to 1 in a LINPACK code), the execution time will decrease extremely. Of course, the compiler must be able to recognize the loops and replace them by fast in line codes, as most compilers do nowadays.

Table 4 shows the various BLAS implementations which we have used on four different vector computers.

	Level 1 BLAS	Level 2 BLAS
CDC CYBER 990	"rolled"	model
CDC CYBER 205	CDC	CWI / model
CRAY X-MP/2	SCILIB	model
NEC SX-2	"rolled"	model

TABLE 4 Survey of available BLAS implementations

We remark that analogous to [2], the "unrolled loops" in the model implementation of the original BLAS were replaced by simple loops in FORTRAN. On the CYBER 205, optimizations of both the Level 1 and Level 2 BLAS are available. It should be mentioned that, as a matter of fact, even two Level 1 BLAS optimizations exist for the CDC CYBER 205: the CDC BLAS, distributed by CDC, and the CWI BLAS[17]. In some cases the CWI BLAS runs significantly faster, e.g., for complex vectors with a non-unit stride[16]. However, for small *n* and for unit stride the CDC BLAS runs faster mainly because of the lack of error testing. If we want to present the actual speedup we have to select the CDC BLAS, because the original LINPACK deals with unit strides. The implementation and optimization of the Level 2 BLAS on the CYBER 205 are described in Lioen, Louter-Nool and Te Riele[15]. The codes are available in the NUMVEC-library[18].

Let us first consider the performances of the real LINPACK routines in Table 5a. Both CYBER compilers do not generate very optimal code for the model implementation of the Level 2 BLAS. When we compare for the CYBERs the performance of the original LINPACK (Columns 1 and 3) with those of the modified LINPACK executed with the model Level 2 BLAS (Columns 2 and 4) we see that the performances do not always improve. Obviously, most _SL routines for solving a system of linear equations become less efficient. Moreover, the modified routines operating on packed arrays are slower; on the CYBER 205 more than a factor 3. By comparing the results of the original LINPACK to those with a coded Level 2 BLAS (cf. Columns 3 and 5), one observes a considerable increase of performance. Only the modified SPBFA has become slower; this is also true for the other machines (cf. Section 3.2).

In Table 5a also the performances of the original and the modified LINPACK are compared on both the CRAY X-MP (Columns 6-7) and the NEC SX-2 (Columns 8-9). At present, the system library of the CRAY in Bracknell provides a coded version of the Level 1 BLAS but not of the Level 2 BLAS. For the real case, the modified LINPACK with the model Level 2 BLAS requires less time than the original LINPACK with a coded Level 1 BLAS, except for the routines for banded matrices. On the NEC, where optimized versions of both Levels of BLAS are lacking the modified LINPACK based on the Level 2 BLAS is the best choice. Note that the performances of the routines for packed (i.e., _PP__ and _SP__) and unpacked (i.e., _PO__ and _SI__) arrays are identical.

Timings of Real LINPACK routines in milliseconds

	CYBER 990		CYBER 205			CRAY X-MP		NEC SX-2	
	original LINPACK	model BLAS2 LINPACK	original LINPACK	model BLAS2 LINPACK	CWI BLAS2 LINPACK	original LINPACK	model BLAS2 LINPACK	original LINPACK	model BLAS2 LINPACK
SGEFA	696.3	380.9	287.5	192.3	150.6	129.5	94.6	91.2	47.1
SGESL_0	9.652	20.3	4.063	11.4	3.386	1.766	1.569	1.533	1.898
SGESL_1	10.1	16.9	4.911	5.971	3.946	2.884	2.477	2.056	2.483
SGEDI	1318.8	774.0	505.7	398.6	322.5	242.2	189.1	164.8	77.4
SGBFA	319.3	252.8	227.6	319.2	219.8	86.8	86.4	137.8	132.4
SGBSL_0	124.8	127.3	87.9	154.0	65.7	25.7	22.4	37.6	31.6
SGBSL_1	137.5	101.4	91.6	124.2	70.6	35.4	40.6	43.7	35.0
SPOFA	514.6	811.9	279.0	258.3	219.7	171.4	142.7	118.9	81.5
SPOSL	9.888	18.4	3.137	9.883	3.078	2.351	1.932	1.754	1.122
SPODI	892.0	519.5	469.2	342.9	260.5	186.0	122.7	162.6	70.8
SPPFA	503.5	731.2	288.5	1055.3	238.4	165.5	140.9	120.8	81.7
SPPSL	9.369	21.7	4.380	29.7	3.354	2.274	1.924	1.792	1.152
SPPDI	901.6	2410.5	450.1	3027.3	301.9	172.7	136.0	165.2	86.2
SPBFA	712.3	778.3	410.3	1143.0	1507.0	169.2	356.9	188.4	360.7
SPBSL	140.5	101.8	94.8	112.2	50.1	35.0	35.0	41.5	22.5
SSIFA	519.6	259.3	287.8	176.4	133.9	108.8	70.0	92.1	32.5
SSIDI	946.7	1131.2	467.7	299.0	291.3	235.7	169.6	214.5	104.6
SSPFA	505.3	1253.0	272.3	1507.3	133.6	105.4	62.8	92.1	33.2
SSPDI	935.4	1441.2	463.3	1502.9	307.5	256.0	159.8	215.8	106.2
STRSL_1	5.339	3.101	2.488	2.084	1.613	0.997	0.843	0.842	0.654
STRSL_2	5.196	6.433	2.493	7.969	1.538	1.061	0.853	0.805	0.560
STRSL_3	5.017	8.885	2.875	6.441	1.964	1.577	1.288	1.121	0.717
STRSL_4	4.257	8.982	2.707	2.294	1.936	1.607	1.261	0.993	0.690
STRDI_1	484.7	241.6	216.7	169.6	128.0	88.2	62.7	87.0	40.9
STRDI_2	476.4	253.8	247.1	171.2	129.6	91.0	62.8	86.3	41.0

TABLE 5a Timings of Real LINPACK routines in milliseconds,
$n=6500$, $kl=ku=3 \vee k=9$ for band matrices,
$m=n=255$ for dense matrices.

The efficiency of the model implementation is highly dependent on the FORTRAN compiler available. Table 5b - containing the results of the complex LINPACK - shows that none of the Level 2 BLAS routines are vectorized well by the CYBER compilers; most operations are performed at scalar speed. Since on the CYBER 205 for the timings of the original LINPACK (Column 3) an optimized Level 1 BLAS was used, the performance decreases using the modified LINPACK executed with the model Level 2 BLAS (Column 4). Nevertheless, when executed with the CWI BLAS2 (Column 5) a considerable speed up is obtained, or can be expected (only COMPLEX rank updates are available, yet). On the CYBER 990, the results of the original and modified complex LINPACK are comparable. Due to the absence here of an optimized Level 1 BLAS also the original LINPACK performs badly. It appears that, on the CYBERs, the use of the model implementation of the complex BLAS and, in many cases, the real BLAS must be dissuaded.

On the CRAY, with a coded Level 1 BLAS, an optimized Level 2 BLAS is needed to improve the performance of the modified LINPACK, and henceforth, this explains the execution time reduction compared to the original LINPACK (cf. Column 6 and 7). From Columns 8 and 9, it follows that, on the NEC, where optimized versions of both Levels of BLAS are lacking the modified LINPACK is the best choice, also for the complex case (except for CPBFA).

Timings of Complex LINPACK routines in milliseconds									
	CYBER 990		CYBER 205			CRAY X-MP		NEC SX-2	
	original LINPACK	model BLAS2 LINPACK	original LINPACK	model BLAS2 LINPACK	CWI BLAS2 LINPACK	original LINPACK	model BLAS2 LINPACK	original LINPACK	model BLAS2 LINPACK
CGEFA	2513.5	2516.4	748.7	1420.4	402.6	336.0	348.1	175.8	122.1
CGESL_0	33.4	32.7	10.5	110.9	-	4.328	4.612	2.612	2.246
CGESL_1	86.7	82.2	14.8	40.0	-	6.879	7.319	4.502	3.997
CGEDI	4901.4	4949.7	1451.7	2867.4	-	645.1	684.1	335.8	227.2
CGBFA	610.1	611.8	387.5	500.5	417.5	106.7	110.3	154.0	138.9
CGBSL_0	294.8	292.9	173.4	326.6	-	37.0	35.2	60.8	49.9
CGBSL_1	400.4	405.8	191.6	152.4	-	46.3	67.8	96.7	80.5
CPOFA	3967.3	4024.7	809.9	3021.5	-	361.7	409.5	296.0	214.2
CPOSL	56.3	60.7	13.2	138.8	-	5.634	5.777	3.619	2.773
CPODI	3090.7	3115.0	1118.1	1873.0	587.7	404.8	400.9	289.2	210.4
CPPFA	3914.5	3919.1	808.2	2856.2	-	363.8	407.4	294.9	216.5
CPPSL	59.7	59.6	13.2	65.4	-	5.604	5.933	3.922	2.820
CPPDI	2998.5	2981.2	1096.0	5582.8	-	400.8	413.1	288.7	211.2
CPBFA	1927.1	1934.0	859.3	1535.5	-	228.6	567.1	436.7	577.7
CPBSL	405.2	399.4	201.6	390.1	-	55.4	67.9	82.1	57.2
CHIFA	1617.2	1624.9	646.8	962.3	333.8	234.0	217.0	189.3	112.2
CHIDI	5580.0	5177.8	1253.5	3531.7	-	548.8	521.2	416.9	254.7
CHPFA	1620.7	1622.6	641.0	2839.7	349.6	233.5	216.3	188.2	108.9
CHPDI	5305.3	5302.7	1241.2	3433.1	-	554.4	525.9	415.2	257.0
CTRSL_1	16.3	16.4	5.909	10.3	-	2.367	2.434	1.427	1.089
CTRSL_2	16.2	16.2	5.978	106.1	-	2.399	2.528	1.393	1.046
CTRSL_3	44.4	44.4	8.119	33.2	-	3.569	4.122	2.326	1.908
CTRSL_4	40.6	45.1	8.082	33.2	-	3.498	4.082	2.302	1.938
CTRDI_1	1561.5	1552.4	549.3	922.7	305.2	197.0	197.4	151.9	106.2
CTRDI_2	1543.7	1536.7	550.7	921.6	301.9	202.1	191.1	149.4	106.1

TABLE 5b Timings of Complex LINPACK routines in milliseconds
$n=6500$, $kl=ku=3 \vee k=9$ for band matrices,
$m=n=255$ for dense matrices.

6. Conclusions and remarks

In this paper we have shown that by replacing in the LINPACK library calls to Level 1 BLAS routines by calls to Level 2 BLAS routines - without changing the algorithm, the data structure and the round-off pattern - a considerable speedup can be obtained, especially when a machine-optimized Level 2 BLAS code is available. Moreover, the use of the Level 2 BLAS enhances the modularity and readability of the programs. The subset, which we have developed, can substitute the corresponding subset of the original LINPACK without the need to adapt any calling program, since the parameter lists have not been changed. Unfortunately, this restriction prohibits the adaptation of a larger set of subroutines, like those for QR, SVD and Cholesky factorization, since all suffer from the lack of sufficient workspace.

Section 4 shows that the model implementation of the Level 2 BLAS achieves a moderate efficiency on vector-processing machines, although we expect that, by means of small modifications in the model code, a considerably higher performance can be obtained. We mentioned the alternative implementation of routines for banded matrices (see Section 4 and [6, Section 3.3]) and the straightforward comparison of characters. In the optimized CYBER 205 implementation[15], the test on zero elements is omitted to achieve better performances for general nonsparse vectors. In [11], techniques to optimize the Level 2 BLAS code on the NEC SX-2 in particular are presented. Many of them will also be suited to a more efficient "model" implementation. As long as specialized implementations are wanting, an efficient portable set of FORTRAN77 is required. We hope that optimized implementations of the original and Extended BLAS will be distributed in a similar way as the source codes of the LINPACK and the model implementations of the BLAS which are easily available via the NETLIB facility[8].

The present LINPACK with Level 1 BLAS routines is not attractive for many vector- and parallel machines. The results of Section 5 illustrate that much better performances can be achieved by means of a library based on the Level 2 BLAS. However, we believe that the production of a new linear algebra library makes more sense than the adaptation of the present library. The substitution of Level 2 BLAS modules by calls to Level 1 BLAS is too limited. New algorithms have been, and will have to be developed minimizing the amount of data transfer rather than the number of floating point operations. Also the data structure becomes more and more important. For instance, the partitioning of matrices into submatrices or blocks can be useful. Obviously, the poor performances of the algorithms for banded matrices are closely related to the data storage. Moreover, this storage does not provide for a fast solution of bi- and tridiagonal systems. On many vector machines, the FORTRAN storage convention of the complex vectors and matrices is rather inconvenient. The COMPLEX data type to maintain a close correspondence between LINPACK routines for REAL and COMPLEX matrices causes many superfluous data transfers, because most operations on complex vectors involve a stride 2 problem. Sometimes, it is better to separate the real and imaginary parts, in other cases operations must be performed controlled by bit vectors. However, in all cases, the performance decreases. Another difficulty arises when compilers don't provide complex arithmetic, like the two CYBER compilers considered in this paper.

References

1. J. Demmel, J.J. Dongarra, J.J. Du Croz, A. Greenbaum, S. Hammerling and D.C. Sorenson (September 1987). Prospectus for the Development of a Linear Algebra Library for High-Performance Computers, *Technical Memorandum 97, Argonne National Laboratory.*
2. J.J. Dongarra (April 1988). Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment, *Technical Memorandum 23, Argonne National Laboratory.*
3. J.J. Dongarra, J.R. Bunch, C.B. Moler and G.W. Stewart (1979). *Linpack User's Guide,* SIAM, Philadelphia, PA.
4. J.J. Dongarra, J.J. Du Croz, I.S. Duff and S. Hammerling (May 1988). A Set of Level 3 Basic Linear Algebra Subprograms, *Technical Memorandum 88 (Revision 1), Argonne National*

Laboratory.

5. J.J. Dongarra, J.J. Du Croz, S. Hammerling and R.J. Hanson (November 1986). An Extended Set of Fortran Basic Linear Algebra Subprograms, *Technical Memorandum 41 (Revision 3), Argonne National Laboratory*.
6. J.J. Dongarra, J.J. Du Croz, S. Hammerling and R.J. Hanson (August 1986). An Extended Set of Fortran Basic Linear Algebra Subprograms: Testing Software and Model Implementation, *Argonne National Laboratory Report*, ANL-TM 81.
7. J.J. Dongarra and S.C. Eisenstat (1984). Squeezing the Most out of an Algorithm in CRAY FORTRAN, *ACM Transactions on Mathematical Software*, 10, 219-230.
8. J.J. Dongarra and E. Grosse (July, 1987). Distribution of Mathematical Software via Electronic Mail, *Comm of the ACM*, 30, 5, 403-407.
9. J.J. Dongarra, Linda Kaufman and Sven Hammarling (1986). Squeezing the Most out of Eigenvalue Solvers on High-Performance Computers, *Linear Algebra and its Applications*, 77, 113-136.
10. J.J. Dongarra and D.C. Sorensen (1986). Linear Algebra on High-Performance Computers, *Parallel Computing 85*, Elsevier Science Publishers B.V., 3-32.
11. R.M. Dubash, J.L. Fredin and O.G. Johnson (1977). *Benchmark of the Extended Basic Linear Algebra Subprograms on the NEC SX-2 Supercomputer*, Lecture Notes in Computer Science, 297, Springer-Verlag, Berlin.
12. B.S. Garbow, J.M. Boyle, J.J. Dongarra and C.B. Moler (1977). *Matrix Eigensystem Routines - EISPACK Guide Extension*, Lecture Notes in Computer Science, 51, Springer-Verlag, Berlin.
13. Linda Kaufman (1984). Banded Eigenvalue Solvers on Vector Machines, *ACM Transactions on Mathematical Software*, 10, 73-86.
14. C.L. Lawson, R.J. Hanson, D.R. Kincaid and F.T. Krogh (1979). Basic Linear Algebra Subprograms for FORTRAN usage, *ACM Transactions on Mathematical Software*, 5, 308-323.
15. W.M. Lioen, M. Louter-Nool and H.J.J. te Riele (1987). Optimization of the real Level 2 BLAS on the CYBER 205, In: *Algorithms and Applications on Vector and Parallel Computers*, H.J.J. te Riele, Th. J. Dekker and H.A. van der Vorst (eds.), North-Holland, Amsterdam-New York-Oxford, 199-212.
16. M. Louter-Nool (1987). Basic linear algebra subprograms (BLAS) on the CDC CYBER 205, *Parallel Computing*, 4, 143-165.
17. M. Louter-Nool (1988). Translation of Algorithm 539: Basic Linear Algebra Subprograms for FORTRAN Usage in FORTRAN 200 for the CDC CYBER 205, to appear in: *ACM Transactions on Mathematical Software*.
18. NUMVEC (1988), *A library of NUMerical software for VECtor and Parallel Processors*,Centre for Mathematics and Computer Science, Amsterdam.
19. B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema and C.B. Moler (1976). *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science, 6, 2nd edition, Springer-Verlag, Berlin.
20. J. Schlichting and H.A. van der Vorst (1987). Solving bidiagonal systems of linear equations on the CDC CYBER 205, NM-R8725, Centre for Mathematics and Computer Science, Amsterdam.
21. H.A. van der Vorst (1988). Vectorial aspects of software libraries, *Supercomputer 23*, V-1, 33-41.
22. H.A. van der Vorst (1987). Large tridiagonal linear systems on vector and parallel computers, *Parallel Computing*, 5, 45-54.