High Performance Computing II
M. Durand and F. El Dabaghi (Editors)
© Elsevier Science Publishers B.V. (North-Holland), 1991

High Performance GEMM-Based Level-3 BLAS: Sample Routines for Double Precision Real Data

Bo Kågström, Per Ling , Charles Van Loan *
Institute of Information Processing
University of Umeå
S-901 87 Umeå, Sweden

Abstract

The Level-3 Basic Linear Algebra Subprograms (BLAS) are designed to perform various matrix multiply and triangular system solving computations. With an optimized Level-3 library it is possible to design portable high performance codes for many matrix computational problems. A drawback of Level-3 BLAS, which consists of nine subprograms, is that the development of optimal code is costly and time consuming because it requires assembly level programming/thinking. It is shown that all one really needs is an optimized `_GEMM`, the BLA subprogram for general matrix multiplication. With suitable partitioning, all the other Level-3 BLA subprograms can be defined in terms of `_GEMM` and a negligible amount of Level-2 computation. Portable high performance algorithms for two Level-3 operations of our GEMM-based library for double precision real data are presented.

1 Introduction

The notion of "level" has assumed great importance in modern matrix computations (see e.g. [10], [15], [6], [7]). The term is used to classify various linear algebra manipulations according to the amount of input data and arithmetic:

Level	Input Data	Arithmetic	Examples
1	linear	linear	dot product, saxpy, vector scale
2	quadratic	quadratic	matrix-vector products, outer products
3	quadratic	cubic	matrix-matrix multiplication

Table 1. Levels of Linear Algebra

Currently, algorithms rich in Level-3 operations are the rage because they imply reduced amounts of memory traffic. For example, an n-by-n matrix multiply $C = AB$ involves $O(n^3)$ arithmetic but $O(n^2)$ data touches (usually). Thus, as n grows the cost of moving

*Department of Computer Science, Cornell University, Ithaca, New York 14853-7501

data (either between processors or within a memory hierarchy) is increasingly dominated by the cost of computation. One mission of the LAPACK project [1], [2] is to rewrite the Linpack and Eispack codes to make them rich in Level-3 BLAS [7]. A fringe benefit of this activity is that it facilitates the tailoring of a particular code to a machine. If a machine is purchased with an optimized Level-3 BLAS library, the LAPACK codes will "automatically" perform well.

The Level-3 BLAS [7] consists of routines for both general and "structured" matrix multiplication. Multiple right hand side triangular system solving is also handled by the package as it is rich in matrix multiplication if properly organized. However, even with just a handful of kernels to optimize, the task is onerous. (One optimized matrix multiply procedure for a well known supercomputer involves about 100,000 lines of assembly code.)

In [14] the concept of GEMM-based Level-3 BLAS was introduced and basic algorithms are described. It was shown that "one can live with" just one highly optimized Level-3 BLA subprogram: `DGEMM`. This subprogram oversees a general matrix multiply of the form $C \leftarrow \alpha AB + \beta C$ (for double precision real data). However, the structured matrix multiplication problems handled by the other Level-3 BLAS can be couched in terms of `DGEMM` with suitable partitionings. Roughly, the idea is to reduce the overall structured multiplication to a set of general multiplications involving "strips". Here, a strip is either a block row or a block column. For performance purposes a strip can be further partitioned into subblocks. At present we are developing a *Fortran 77 implementation* of the GEMM-based Level-3 BLAS [14] that presumably will achieve high performance on a wide range of machines. Hence, our results should be of special interest to manufacturers who do not have the resources to hand-tune each Level-3 BLA subprogram. Notice that the approach of the GEMM-based Level-3 BLAS is also beneficial in designing parallel block algorithms for matrix factorizations [4], [5] [8], [9].

In this paper we will describe algorithms for two Level-3 operations of our GEMM-based library in double precision real arithmetic: `DSYRK` and `DTRSM`. The performance of these routines will be compared with similar Level-3 routines optimized for IBM 3090 VF (J Model) (see [12], [16], [11]). At present, the IBM ESSL library [11] does only provide two of the six level-3 BLAS in double precision (`DGEMM` and `DTRSM`). Within the parallel research group in Umeå a set of high performance Fortran 77 Level-3 BLAS for the IBM 3090 VF has been developed [12], [16]. High performance is achieved by structuring and tuning as to maximize reuse of data in vector registers and in cache memory and to direct the compiler to use compound vector instructions (two instructions executed during one clock cycle). Similar matrix blockings as will be described here are utilized in [12], [16] but no explicit calls are performed to `DGEMM`. Therefore, [16] can be looked at as an *implicit* (in-line coded) GEMM-based Level-3 BLAS library in contrast to the *explicit* GEMM-based approach discussed in this paper.

The rest of this paper is outlined as follows. In section 2 we begin with a discussion of `DGEMM` itself. Sections 3 and 4 deal with GEMM-based versions of `DSYRK` (symmetric rank-k update) and `DTRSM` (triangular system solve with multiple right hand sides), respectively. Experimental performance results are presented in section 5. Finally, in section 6 we summarize and outline some future work.

2 DGEMM - General Matrix Multiply and Add

The key BLA subprogram for us is DGEMM, the general matrix multiplication procedure for double precision real data. A reference to DGEMM [7] has the form

```
call DGEMM(transA, transB, m, n, k, alpha, A, lda, B, ldb, beta, C, ldc)
```

The row dimensions of the arrays A, B, and C are passed through lda, ldb, and ldc. The variables transA and transB are used to communicate transpose information as indicated in Table 2.

Operation	A	B	C	transA	transB
$C \leftarrow \alpha AB + \beta C$	$m \times k$	$k \times n$	$m \times n$	'N'	'T'
$C \leftarrow \alpha A^T B + \beta C$	$k \times n$	$k \times n$	$m \times n$	'T'	'N'
$C \leftarrow \alpha AB^T + \beta C$	$m \times k$	$n \times k$	$m \times n$	'N'	'T'
$C \leftarrow \alpha A^T B^T + \beta C$	$k \times m$	$n \times k$	$m \times n$	'T'	'T'

Table 2. DGEMM functionality

In a full implementation of the Level-3 library there are four versions of the general matrix multiply and add operation: SGEMM for single precision real data, DGEMM for double precision real data, CGEMM for complex single precision data and ZGEMM for double precision complex data. Here, we only consider the double precision real data case. It is possible to invoke DGEMM with proper submatrices of A, B, and C. Consider the update

$$C(m_1 : m_2, n_1 : n_2) \leftarrow C(m_1 : m_2, n_1 : n_2) - A(m_1 : m_2, k_1 : k_2)B(k_1 : k_2, n_1 : n_2)$$

The following code segment performs this calculation:

```
m = m2 - m1 + 1
n = n2 - n1 + 1
k = k2 - k1 + 1
call DGEMM('N','N',m,n,k,-1.,A(m1,k1),lda,B(k1,n1),ldb,1.,C(m1,n1),ldc)
```

With this understanding about DGEMM, we proceed with the description of the two GEMM-based Level-3 BLA subprograms DSYRK and DTRSM.

3 DSYRK - Symmetric Rank-k Update

The BLA subprogram DSYRK [7] is used for the symmetric updates $C \leftarrow \alpha AA^T + \beta C$ and $C \leftarrow \alpha A^T A + \beta C$. A reference to DSYRK has the form

```
call DSYRK(uplo, trans, n, k, alpha, A, lda, beta, C, ldc)
```

where the variable trans specifies which of the two options is in effect:

Operation	A	C	trans
$C \leftarrow \alpha AA^T + \beta C$	$n \times k$	$n \times n$	'N'
$C \leftarrow \alpha A^T A + \beta C$	$k \times n$	$n \times n$	'T'

Table 3. DSYRK functionality ($C = C^T$)

If `uplo = 'L'` then the lower triangular portion of C is updated while `uplo = 'U'` implies that the upper triangular portion of C is updated. With two possibilities for `uplo` and `trans` there are thus four cases to consider.

We detail the case (`uplo`,`trans`) = (`'U'`,`'N'`) first, i.e., the update $C \leftarrow AA^T + C$ with the assumption that we are to reference the upper triangular part of C only. We proceed with $\alpha = \beta = 1$ as we derive the blocking. It is possible to update C by block column or by block row [14]. The main difference is the access pattern of C. Below we present a block row oriented algorithm

$s = 1$; $e = \min(r, n)$
while $s \leq n$
  Compute the upper triangular portion of:$A(s:e,:)A(s:n,:)^T + C(s:e, s:n)$
  $s = e + 1$; $e = \min(s + r - 1, n)$
end

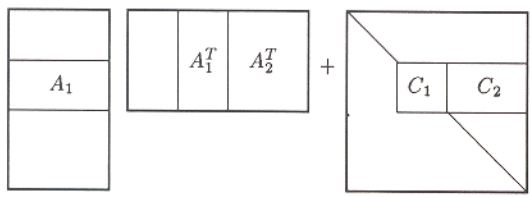
Here, s and e stand for "start" and "end", respectively, and r is the row block size. We break the update of $C(s:e, s:n)$ down into a diagonal block part,

$$C(s:e, s:e) \leftarrow A(s:e,:)A(s:e,:)^T + C(s:e, s:e)$$

and a superdiagonal block part,

$$C(s:e, e+1:n) \leftarrow A(s:e,:)A(e+1:n,:)^T + C(s:e, e+1:n)$$

Schematically, we have



where $A_1 = A(s:e,:)$, $A_2 = A(e+1:n,:)$, $C_1 = C(s:e, s:e)$ and $C_2 = C(s:e, e+1:n)$. The superdiagonal block update $C_2 \leftarrow A_1 A_2^T + C_2$ is handled by a single call to `DGEMM`. The diagonal block update $C_1 \leftarrow A_1 A_1^T + C_1$ is computed in a column-by-column approach by the general gaxpy Level-2 BLA subprogram `DGEMV` [14]:

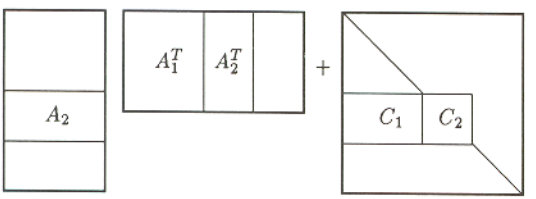
for $j = s:e$
  $C(s:j, j) \leftarrow A(s:j,:)A(j,:)^T + C(s:j, j)$
end

Typically, r is chosen as the vector register length ($r = 256$ on IBM 3090 VF, J Model). However, in many cases $A(s:e,:)$ will be to large to fit in cache memory, and will therefore cause a bad cache utilization. To solve this problem, we introduce an inner level of blocking in the diagonal update. By partitioning the $r \times n$ block rows $A(s:e,:)$ into $r \times c$ subblocks where c (the column block size) is chosen so that a subblock (stored in a temporary array T) requires about 75% of the cache memory. We choose $c = 96$ on the J Model with $256kB$ cache memory. The rest of the cache memory is used for program code and data (vectors to be used in compound instructions). The following algorithm handles the case (`uplo`,`trans`) = (`'U'`, `'N'`):

$s = 1$; $e = \min(r, n)$
while $s \leq n$
  $sc = 1$; $ec = \min(c, k)$
  while $sc \leq k$
    DCOPY: $T \leftarrow A(s : e, sc : ec)$
    DGEMV: $C(s : e, s : e) \leftarrow \alpha TT^T + \beta C(s : e, s : e)$
    $sc = ec + 1$; $ec = \min(sc + c - 1, k)$
  end
  DGEMM: $C(s : e, e + 1 : n) \leftarrow \alpha A(s : e, :)A(e + 1 : n, :)^T + \beta C(s : e, e + 1 : n)$
  $s = e + 1$; $e = \min(s + r - 1, n)$
end

Here, sc and ec stand for "start column" and "end column", respectively. The reason for using T (and not referencing A in DGEMV) is at least twofold. First, we eliminate the problem with (possible) *critical leading dimension* of A. Second, we can work with larger subblocks in cache memory since T allows better *data alignment*. We can only utilize around 50% of the cache memory if $A(s : e, sc : ec)$ is referenced directly. For a more detailed discussion of these issues see [16]. DCOPY copies an $r \times c$ subblock of A to T and after this operation is completed most of T will be in cache memory. Then DGEMV is called $\lceil \frac{k}{c} \rceil$ times. Each call to DGEMV references T, a row of T, and a column of $C(s : e, s : e)$. T resides in cache and is reused $\lceil \frac{k}{c} \rceil - 1$ times. A column of $C(s : e, s : e)$ is stored in a vector register and is replaced in each call (with a copy residing in cache memory). The compound instruction *Multiply and add* reads T directly from cache without using a vector register (see also [16]). Finally, DGEMM is called once per superdiagonal update and references $C(s : e, e + 1 : n)$, $A(s : n, :)$. Here, the reference pattern is controlled by DGEMM.

The cases (uplo, trans) = ('U', 'T'), ('L','N') and ('L','T') are handled similarly. Here is a schematic for the case (uplo, trans) = ('L','N'):



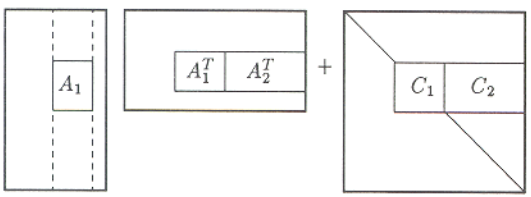
A similar block column oriented algorithm is possible to design.

In the superdiagonal (or subdiagonal) update, the utilization of cache memory and vector registers is controlled by DGEMM. The concept of GEMM-based Level-3 BLAS assumes that a highly optimized DGEMM is available. This implies that DGEMM handles data alignment, critical leading dimension, vector and cache utilization efficiently. However, our experiences are that it can be profitable even to serve an optimized DGEMM with suitable blocks to facilitate data reuse. Therefore, it is also interesting to consider a second level of blocking for these updates.

The following double level block DSYRK algorithm for the case (uplo,trans) = ('U','N') is block column oriented:

$sc = 1$; $ec = \min(c, k)$
while $sc \leq k$
  $s = 1$; $e = \min(r, n)$
  while $s \leq n$
    Compute the upper triangular portion of:
      $A(s : e, sc : ec)A(s : n, sc : ec)^T + C(s : e, s : n)$
    $s = e + 1$; $e = \min(s + r - 1, n)$
  end
  $sc = ec + 1$; $e = \min(sc + c - 1, k)$
end

As before, we break the update of $C(s : e, s : n)$ down into a diagonal block part, and a superdiagonal block part. Schematically, we have



where $A_1 = A(s : e, sc : ec)$, $A_2 = A(e + 1 : n, sc : ec)$, $C_1 = C(s : e, s : e)$ and $C_2 = C(s : e, e + 1 : n)$. The superdiagonal block update $C_2 \leftarrow A_1 A_2^T + C_2$ is handled by a single call to `DGEMM`. The diagonal block update $C_1 \leftarrow A_1 A_1^T + C_1$ is as before computed in a column-by-column approach by `DGEMV`:

for $j = s : e$
  $C(s : j, j) \leftarrow A(s : j, sc : ec)A(j, sc : ec)^T + C(s : j, j)$
end

Similarly, we make use of a temporary array T that stores the current subblock of the (now) current block column of A:

DCOPY: $T \leftarrow A(s : e, sc : ec)$
DGEMV: $C(s : e, s : e) \leftarrow \alpha TT^T + \beta C(s : e, s : e)$
DGEMM: $C(s : e, e + 1 : n) \leftarrow \alpha TA(e + 1 : n, sc : ec)^T + \beta C(s : e, e + 1 : n)$

The reasons for using T (and not referencing A) are the same as mentioned earlier. The main difference between the two algorithms is that `DGEMM` is called within the second loop and references $C(s : e, e + 1 : n)$, $A(e + 1 : n, sc : ec)$ and T. Since T is still residing in cache (due to the cache replacement policy) it can be reused also in `DGEMM`. IBM 3090 VF uses the *LRU* policy.

At the outer level the update is performed as a block-outer-product. A block-inner-product-like update is also possible by exchanging the ordering of the two loops. The main advantage with the block-outer-product update is that the block column of A that we will continue to work with already resides in main memory and will therefore be reused "optimally" at this level of the memory hierarchy.
The three GEMM-based algorithms (block row, block column and double level blocking) for `DSYRK` that follows our Fortran implementations are presented in appendices A.1 –

A.3 of [13]. Given the blocking parameters r and c they compute the updates of Table 3. If `uplo = 'L'` (`uplo = 'U'`) then only the lower (upper) triangular portion of C is referenced. Performance results for these algorithms are presented in section 5.

4 DTRSM - Triangular System Solve

The BLA subprogram `DTRSM` [7] is used for multiple right hand side triangular system solving. A reference to this routine has the form

```
call DTRSM( side, uplo, trans, diag, m, n, alpha, A, lda, C, ldc )
```

The matrix A is triangular and the parameters `side` and `trans` are used to specify its action:

Operation	A	C	`side`	`trans`
$C \leftarrow \alpha X,\ AX = C$	$m \times m$	$m \times n$	`'L'`	`'N'`
$C \leftarrow \alpha X,\ A^T X = C$	$m \times m$	$m \times n$	`'L'`	`'T'`
$C \leftarrow \alpha X,\ XA = C$	$n \times n$	$m \times n$	`'R'`	`'N'`
$C \leftarrow \alpha X,\ XA^T = C$	$n \times n$	$m \times n$	`'R'`	`'T'`

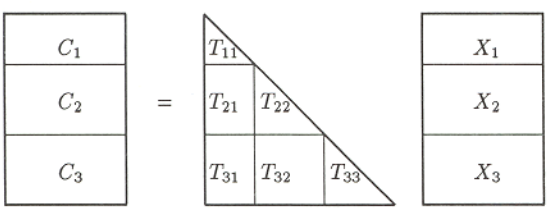
Table 4. `DTRSM` functionality (A triangular)

The matrix A may be lower or upper triangular (`uplo = 'L'` or `'U'`) and may have a unit or nonunit diagonal (`diag = 'U'` or `'N'`). There are four different situations from the standpoint of blocking:

Operation	T	Possibilities
$C \leftarrow X,\ TX = C$	lower triangular	$(T,$ `uplo`$) = (A,$ `'L'`$)$ or $(A^T,$ `'U'`$)$
$C \leftarrow X,\ TX = C$	upper triangular	$(T,$ `uplo`$) = (A,$ `'U'`$)$ or $(A^T,$ `'L'`$)$
$C \leftarrow X,\ XT = C$	lower triangular	$(T,$ `uplo`$) = (A,$ `'L'`$)$ or $(A^T,$ `'U'`$)$
$C \leftarrow X,\ XT = C$	upper triangular	$(T,$ `uplo`$) = (A,$ `'U'`$)$ or $(A^T,$ `'L'`$)$

Table 5. `DTRSM` blocking cases

First, we consider the case $C \leftarrow X,\ TX = C$ where T is lower triangular.



We illustrate by blocking X in 3 block rows:

$$
\begin{aligned}
C_1 &= T_{11}X_1 \\
C_2 &= T_{21}X_1 + T_{22}X_2 \\
C_3 &= T_{31}X_1 + T_{32}X_2 + T_{33}X_3
\end{aligned}
$$

By solving for X in a block forward fashion we can obtain a DGEMM-rich procedure:

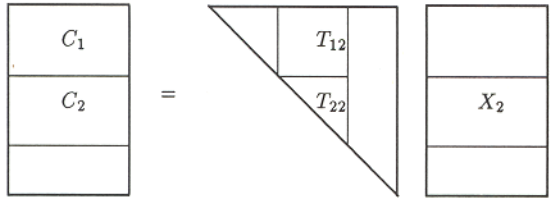
$$X_1 \leftarrow T_{11}^{-1} C_1$$
$$X_2 \leftarrow T_{22}^{-1}(C_2 - T_{21} X_1)$$
$$X_3 \leftarrow T_{33}^{-1}(C_3 - T_{31} X_1 - T_{32} X_2)$$

C_2 and C_3 are updated by calls to DGEMM. The update of C_2 and C_3 with respect to X_1 is performed by one single call.

We could use repeated calls to the Level-2 BLA subprogram DTRSV [6] to find X_1, X_2 and X_3. DTRSV performs matrix-vector products of the form $x \leftarrow T^{-1}x$ where T is a nonsingular triangular matrix described via the variables `uplo`, `diag`, and `trans`. In the absence of an optimized version of DTRSV in [11] we presently use our own DGEMV-based Fortran subroutine DTRSM2. DTRSM2 has the same functionality as DTRSM and is designed to use the vector registers of the IBM 3090 VF efficiently (for details see [16]). By computing X block row-wise from top to bottom we can overwrite C with X:

$e = m - \mathrm{div}(m-1, r) \cdot r$; $s = \max(e - r + 1, 1)$
while $s \leq m$
  DTRSM2: $C(s:e,:) = T(s:e, s:e)^{-1} C(s:e,:)$
  if $e < m$
    DGEMM: $C(e+1:m,:) =$
      $-T(e+1:m, s:e) C(s:e,:) + C(e+1:m,:)$
  end
  $s = e + 1$; $e = s + r - 1$
end

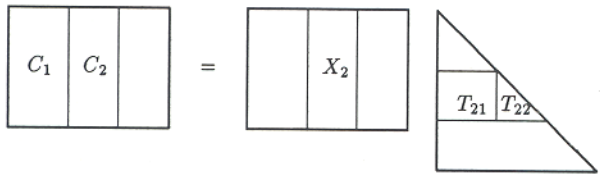
The other three cases listed in Table 4 are similar and require only cursory discussion. For the case $TX = C$ with upper triangular T we have in the i-th step ($i = 2$ and C_1, C_2 are already updated with respect to X_1)



Solve for X_2 : $T_{22} X_2 = C_2$
Update C_1 : $C_1 \leftarrow C_1 - T_{12} X_2$

It follows that if we resolve the block rows of X in reverse order (from bottom to top) we can overwrite C with X.

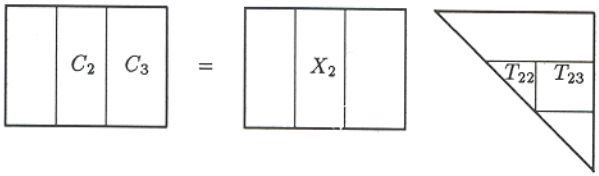
For the case $XT = C$ with T lower triangular we have in the i-th step



Solve for X_2 : $X_2T_{22} = C_2$
Update C_1 : $C_1 \leftarrow C_1 - X_2T_{21}$

It follows that if we resolve the block columns of X from right to left we can overwrite C with X.

Finally, for the case $XT = C$ with T upper triangular we have in the i-th step



Solve for X_2 : $X_2T_{22} = C_2$
Update C_3 : $C_3 \leftarrow C_3 - X_2T_{23}$

It follows that if we resolve the block columns of X from left to right we can overwrite C with X.

Where the possible off-set block is placed is determined by `uplo` and `trans`. If (`uplo`, `trans`) = (`'U'`, `'N'`) or (`'L'`, `'T'`) it sits in the lower rightmost corner, and if (`uplo`, `trans`) = (`'U'`, `'T'`) or (`'L'`, `'N'`) in the upper leftmost corner. This imply that we optimize with respect to daxpy operations (*Multiply and add* instructions). The opposite choices would imply ddot operations (*Multiply and accumulate* instructions).

The GEMM-based `DTRSM` algorithm, that follows our Fortran implementation, is presented in appendix B of [13]. Given the blocking parameter r, it computes the updates of Tables 4 and 5.

5 Performance Results

In this section we present performance results from the Fortran implementations of the GEMM-based `DSYRK` and `DTRSM` algorithms described in sections 3-4. The performance is measured in *Mflops*, million floating point operations per second. The theoretical peak performance of the IBM 3090 VF (J model, one processor) is 138 Mflops (two instructions executed during one clock cycle, 14.5ns). We define the *practical peak performance* of a machine as the performance of the level-3 BLAS `DGEMM` which for the J-model is 108 Mflops i.e. around 80% of the theoretical peak performance. This degradation in performance is mainly due to the characteristics of the vector instructions and the memory hierarchy of the machine [18], [3]. The peak performance of `DGEMV` is around 80 Mflops. All results presented are computed on one processor of a non-dedicated IBM 3090 VF/600J.

The SDCN (Supercomputer Center North) machine is located at Norrdata, Skellefteå in Sweden. Each processor has a $256kB$ cache memory (corresponds to $32k$ double precision data).

Tables 6-8 display some sample performance results for two different leading dimensions, one critical (512) and one non-critical (530) [16] of the DSYRK algorithms: block column, block row and double level blocking. Notice, for given values of n and k, the first line of the two results corresponds to the critical leading dimension (512). The results of the three algorithms are very similar and reach up to 90% of practical peak performance. We conclude, from Tables 6-7, that the DGEMM of ESSL takes care of the necessary optimization and therefore the double level blocking is not necessary on the IBM 3090 VF (J Model). We have also tested the block row and block column algorithms without

n	k	uplo, trans			
		'U','N'	'U','T'	'L','N'	'L','T'
100	2	9.82	8.77	9.55	8.84
100	2	10.38	10.05	10.31	10.05
100	48	49.47	45.62	53.06	46.40
100	48	53.43	52.63	53.24	50.63
100	256	57.43	48.19	57.36	50.59
100	256	58.55	58.02	58.77	57.34
500	2	22.34	20.00	21.47	20.43
500	2	21.65	21.36	21.01	20.97
500	48	85.04	83.29	83.87	82.19
500	48	86.01	88.94	87.36	86.61
500	256	91.53	88.08	92.61	84.87
500	256	93.41	93.18	93.47	93.23

Table 6. Performance of GEMM-based DSYRK - block row algorithm

the inner blocking in the diagonal update. As soon as a block row does not entirely fit into cache memory the performance degrades. For a critical leading dimension and TRANS = 'T' these algorithms show much worse performance (only around 50% of the block row and block column algorithms). Performance results of the GEMM-based DTRSM algorithm is displayed in Table 9. It reaches over 90% of the practical peak performance. For given values of m and n, the first line of the two results corresponds, as before, to the critical leading dimension (512). Similar results for the ESSL routine is shown in Table 10. Notice that our GEMM-based implementation is in many cases faster than the ESSL routine, especially when SIDE = 'L'. In some cases actually more than a factor two.

One way of quantifying the portion of work in DGEMM of these algorithms is the Level-3 fraction [10] (i.e. ratio of DGEMM flops to total flops). For example, the Level-3 fraction of the DSYRK block row algorithm and the DTRSM algorithm is approximately $1 - \frac{1}{N}$ where the matrix dimension is equal to rN [14]. For a fixed matrix dimension this means that a smaller r (≥ 2) gives a larger Level-3 fraction. For a fix r it means that a larger matrix dimension gives a larger Level-3 fraction. It is of course only in the second case that a larger Level-3 fraction normally implies higher performance (see Tables 6-10).

n	k	uplo, trans			
		'U','N'	'U','T'	'L','N'	'L','T'
100	2	9.93	8.96	9.87	8.57
100	2	10.31	9.52	10.27	10.05
100	48	50.60	46.75	49.32	45.45
100	48	51.61	52.85	53.54	52.48
100	256	55.71	51.79	58.59	51.28
100	256	58.44	56.62	58.81	57.73
500	2	21.77	20.01	21.86	20.36
500	2	21.38	21.17	21.64	21.17
500	48	85.88	83.50	85.77	82.87
500	48	86.30	89.31	86.88	88.97
500	256	92.08	87.68	93.04	88.49
500	256	90.92	93.76	93.85	92.62

Table 7. Performance of GEMM-based DSYRK - block column algorithm

n	k	uplo, trans			
		'U','N'	'U','T'	'L','N'	'L','T'
100	2	9.76	8.96	9.82	8.74
100	2	10.44	9.93	10.35	9.94
100	48	50.86	45.79	52.29	46.64
100	48	51.42	51.18	51.99	51.94
100	256	56.27	50.79	58.02	50.30
100	256	58.23	57.97	56.91	57.84
500	2	22.15	20.21	21.67	20.44
500	2	21.57	21.18	21.20	21.40
500	48	86.42	84.91	86.80	85.02
500	48	86.85	86.48	87.42	89.90
500	256	92.83	88.19	92.29	90.45
500	256	93.38	93.51	92.03	93.93

Table 8. Performance of GEMM-based DSYRK - double level blocking

m	n	side, trans			
		'L','N'	'L','T'	'R','N'	'R','T'
100	2	16.88	7.27	2.86	2.61
100	2	19.16	19.40	3.54	3.23
100	48	42.96	46.50	60.28	55.71
100	48	54.58	52.32	59.81	57.57
100	256	61.30	65.19	79.20	70.20
100	256	77.72	77.85	80.79	79.74
500	2	39.42	9.93	7.46	7.58
500	2	39.27	35.27	8.16	7.19
500	48	78.81	69.92	73.60	71.89
500	48	83.60	82.62	78.63	78.83
500	256	91.85	89.77	96.84	90.45
500	256	97.02	96.56	96.29	97.47
2	100	0.80	1.01	3.90	3.02
2	100	1.67	1.86	4.63	4.23
48	100	37.83	46.30	55.03	46.35
48	100	51.57	61.47	55.93	49.94
256	100	73.63	71.19	87.41	84.37
256	100	78.72	81.93	89.86	88.05
2	500	1.04	1.31	6.93	6.69
2	500	1.52	2.63	8.49	8.58
48	500	47.53	54.38	66.56	64.34
48	500	61.34	73.66	72.61	72.10
256	500	73.54	81.79	101.26	90.74
256	500	87.55	85.20	100.31	96.75
500	500	91.47	91.03	99.87	95.63
500	500	96.06	95.28	100.97	94.95

Table 9. Performance results for GEMM-based DTRSM

m	n	side, trans			
		'L','N'	'L','T'	'R','N'	'R','T'
100	2	27.03	21.79	4.94	3.64
100	2	28.41	23.23	5.13	3.77
100	48	43.17	38.31	65.72	62.24
100	48	43.09	42.15	65.44	61.13
100	256	39.14	36.60	81.90	78.76
100	256	45.78	45.87	81.89	78.42
500	2	63.42	48.44	8.97	9.01
500	2	63.19	48.61	9.26	8.81
500	48	78.24	57.55	73.94	76.17
500	48	77.89	75.58	80.96	80.83
500	256	82.01	75.26	98.26	95.16
500	256	83.33	81.53	97.21	96.17
2	100	0.69	0.69	11.59	13.58
2	100	1.16	1.12	13.53	17.27
48	100	22.49	23.40	55.25	49.19
48	100	39.49	39.92	56.42	57.24
256	100	68.36	57.76	90.94	88.26
256	100	71.61	70.61	89.27	88.75
2	500	0.75	0.75	10.41	22.77
2	500	1.20	1.18	22.95	29.61
48	500	23.78	24.12	63.53	59.67
48	500	40.30	40.48	63.39	63.77
256	500	68.95	66.66	103.78	101.38
256	500	73.95	72.33	104.31	104.26
500	500	82.79	80.04	103.70	96.80
500	500	84.52	83.41	104.47	104.25

Table 10. Performance results for ESSL DTRSM

6 Conclusions and Future Work

We have shown that the GEMM-based concept and algorithms are very competitive for implementing a portable high performance Level-3 BLAS library. Here, GEMM-based algorithms for `DSYRK` and `DTRSM` are presented that show performance similar to [11] and [16] (obtained on an IBM 3090 VF, J Model). The portability is managed via at most two architecture dependent parameters (r and c) that determines the partitioning in `DGEMM` and Level-2 operations. On the IBM 3090 r is the vector register length and c is chosen to optimize the cache utilization in successive `DGEMV` operations (see section 3). Notably, the GEMM-based `DTRSM` algorithm for some cases show even better performance than the optimized version in [11] which demonstrates the intricacies in hand-tuning a more complicated Level-3 kernel.

The future work involve developing a complete GEMM-based Level-3 BLAS library for double precision real data that is portable on a wide range of high performance machines. Besides IBM 3090 VF, the architectures we have in mind include the RISC-based architectures ALLIANT FX/2800, IBM RS/6000 and Intel iPSC/860.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, "LAPACK: A Portable Linear Algebra Library for High-Performance Computers", Tech. Report CS-90-105, Univ. of Tennessee, Knoxville, May 1990. (LAPACK Working Note #20).

[2] E. Andersson and J. Dongarra, "Evaluating Block Algorithm Variants in LAPACK", in J. Dongarra, P. Messina, D. Sorensen and R. Voigt (eds), *Parallel Processing for Scientific Computing*, SIAM Publications, 1990, pp 3-8.

[3] E. Cohen, G. King and J. Brady, "Storage Hierarchies", *IBM Systems Journal*, Vol. 28(1) (1988) pp 62-76.

[4] K. Dackland, E. Elmroth, B. Kågström and C. Van Loan, "Parallel Block Matrix Factorizations on the Shared Memory Multiprocessor IBM 3090 VF/600J", Report UMINF-91.07, Inst. of Information Processing, Univ. of Umeå, S-901 87 Umeå, January 1991.

[5] M. Daydé and I. Duff, "Use of Level 3 BLAS in LU Factorization in a Multiprocessing Environment on Three Vector Multiprocessors: the ALLIANT FX/80, the CRAY-2, and the IBM 3090 VF", Technical Report CERFACS, August 1990.

[6] J. Dongarra, J. Du Croz, S. Hammarling and R. Hanson, "An Extended Set of Fortran Basic Linear Algebra Subprograms", *ACM Trans. on Mathematical Software*, Vol. 14 (1988) pp 1-17, 18-32.

[7] J. Dongarra, J. Du Croz, S. Hammarling and I. Duff, "A Set of Level 3 Basic Linear Algebra Subprograms", *ACM Trans. on Mathematical Software*, Vol. 16 (1990) pp 1-17, 18-28.

[8] K. Gallivan, W. Jalby, U. Meier and A. Sameh, "Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design", *Int. J. Supercomputer Applications*, Vol 2 (1988), pp 12-48.

[9] K. Gallivan, R. Plemmons and A. Sameh, "Parallel Algorithms for Dense Linear Algebra Computations", *SIAM Review*, Vol. 32 (1990), pp 54-135.

[10] G. Golub and C. Van Loan, *Matrix Computations*, John Hopkins Press, 2nd edition, 1989.

[11] IBM, *Engineering and Scientific Subroutine Library Guide and Reference*, SC23-0184-3, November 1988.

[12] B. Kågström and P. Ling, "Level 2 and 3 BLAS Routines for IBM 3090 VF: Implementation and Experiences", in J. Dongarra, I. Duff, P. Gaffney and S. McKee(eds), *Vector and Parallel Computing*, Ellis Horwood, 1989, pp 215-228.

[13] B. Kågström, P. Ling and C. Van Loan, "High Performance GEMM-Based Level-3 BLAS: Sample Routines for Double Precision Real Data", Report UMINF-91.09, Inst. of Information Processing, Univ. of Umeå, S-901 87 Umeå, March 1991.

[14] B. Kågström and C. Van Loan, "GEMM-Based Level-3 BLAS", Technical Report, Dept. of Computer Science, Cornell University, December 1989.

[15] C. Lawson, R. Hanson, R. Kincaid and F. Krogh, "Basic Linear Algebra Subprograms for Fortran Usage", *ACM Trans. on Mathematical Software*, Vol. 5 (1979), pp 308-323.

[16] P. Ling, "A Set of High Performance Level-3 BLAS Structured and Tuned for the IBM 3090 VF and Implemented in Fortran 77", Report UMINF-179.90, Inst. of Information Processing, Univ. of Umeå, S-901 87 Umeå, May 1990.

[17] B. Liu and N. Strother, "Programming in VS Fortran on the IBM 3090 for Maximum Vector Performance", *IEEE Computer*, June (1988), pp 65-76.

[18] S. Tucker, "The IBM 3090 System: An Overview", *IBM Systems Journal*, Vol. 25(1), (1986) pp 4-20.