RC24597 (W0807-034) July 7, 2008
Mathematics

IBM Research Report

Clearer, Simpler and More Efficient LAPACK Routines for Symmetric Positive Definite Band Factorization

Fred G. Gustavson
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

Enrique S. Quintana-Ortí, Gregorio Quintana-Ortí, Alfredo Remón
Depto. de Ingeniería y Ciencia de Computadores
Universidad Jaume I
12.071-Castellón
Spain

Jerzy Wasniewski
Department of Informatics and Mathematical Modeling
Technical University of Denmark
DK-2800 Lyngby
Denmark

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Clearer, Simpler and more Efficient LAPACK Routines for Symmetric Positive Definite Band Factorization

Fred G. Gustavson[1], Enrique S. Quintana-Ortí[2], Gregorio Quintana-Ortí[2], Alfredo Remón[2], and Jerzy Waśniewski[3]

[1] IBM T.J. Watson Research Center
Yorktown Heights NY 10598, USA
fg2@us.ibm.com
[2] Depto. de Ingeniería y Ciencia de Computadores
Universidad Jaume I, 12.071–Castellón, Spain
{quintana,gquintan,remon}@icc.uji.es
[3] Department of Informatics and Mathematical Modeling
Technical University of Denmark, DK-2800 Lyngby, Denmark
jw@imm.dtu.dk

Abstract. We describe a minor format change for representing a symmetric band matrix AB using the same array space specified by LAPACK. In LAPACK, band codes operating on the lower part of a symmetric matrix reference matrix element (i, j) as $AB_{1+i-j,j}$. The format change we propose allows LAPACK band codes to reference the (i, j) element as $AB_{i,j}$. Doing this yields lower band codes that use standard matrix terminology so that they become clearer and hence easier to understand. As a second contribution, we simplify the LAPACK Cholesky Band Factorization routine PBTRF by reducing from six to three the number of subroutine calls one needs to invoke during a right-looking block factorization step. Our new routines perform exactly the same number of floating-point arithmetic operations as the current LAPACK routine PBTRF. Almost always they deliver higher performance. The experimental results show that this is especially true on SMP platforms where parallelism is obtained via the use level-3 multi-threaded BLAS. We only consider the lower triangular case of the factorization here; the upper triangular case is currently under investigation.

1 Introduction

LAPACK routine PBTRF [1] uses the Cholesky algorithm to factorize a symmetric positive definite (SPD) band matrix AB of order n, taking into account the band structure of matrix AB to reduce the computational cost of the process by avoiding operations on the zero "off-band" elements. In LAPACK, matrix AB has a half bandwidth of size k_d and it is stored as a $k_d \times n$ rectangle in array `AB` in a standard Fortran data layout (i.e., column major order). In this representation, matrix element $AB_{i,j}$ is physically located at position $(1+i-j, j)$

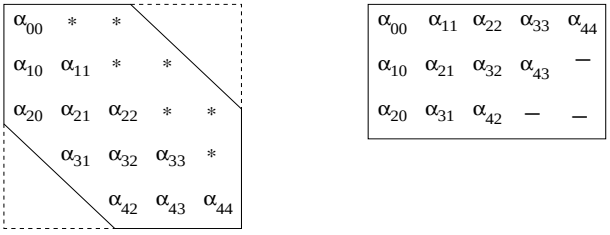

Fig. 1. Symmetric band matrix AB of order $n = 5$ with half bandwidth k_d=2 (left), and packed band storage used in LAPACK as a rectangle `AB` with leading dimension `LDAB` $= k_d+1$ (right). Symbols '$*$' denote strictly upper symmetric entries of the matrix which are not stored.

of array `AB`; Figure 1 illustrates this layout for a special case of AB with $n = 5$, $k_d = 2$, and leading dimension of `AB`, `LDAB`, equal $k_d + 1$.

Now, the lower band of the symmetric matrix can also be viewed as a parallelogram plus a triangle, $P + T$, where P contains the nonzero lower triangular part of submatrix $AB_{1:n,1:k_d+1}$ and T contains the lower triangular part of the full submatrix $AB_{k_d+2:n,k_d+2:n}$; see the shadowed areas in Figure 1 left, where the light color parallelogram corresponds to P and the dark color triangle to T. We claim that this latter representation is more natural and hence code written for it becomes both clearer and more understandable. A standard Fortran layout of $P + T$ can be obtained from `AB` if one changes its leading dimension from `LDAB` to `LDAB`-1. We note that $P + T$ should be isomorphic to `AB` and this is not the case as we have defined it. What is isomorphic to `AB` is an enlarged parallelogram, which we denote as `EP`, obtained from `AB` by making its first row the main diagonal of `EP` via an affine transformation. In what follows we will either refer to $P + T$ or `EP`.

LAPACK blocked routine PBTRF computes a band Cholesky right-looking factorization by processing n_b columns of the matrix per AB iteration. Consider, e.g., the i-th iteration of the routine and, for simplicity, assume $\iota + k_d + n_b \le n$, where $\iota = (i - 1) \cdot n_b$. The partitioning

$$AB \rightarrow \left(\begin{array}{c|c|c} A_{TL} & \star & \\ \hline A_{ML} & A_{MM} & \star \\ \hline & A_{BM} & A_{BR} \end{array}\right) = \left(\begin{array}{c|ccc|c} A_{00} & \star & \star & & \\ \hline A_{10} & A_{11} & \star & \star & \\ A_{20} & A_{21} & A_{22} & \star & \star \\ & A_{31} & A_{32} & A_{33} & \star \\ \hline & & A_{42} & A_{43} & A_{44} \end{array}\right), \tag{1}$$

where $A_{00} \in \mathbb{R}^{\iota\times\iota}$, $A_{11}, A_{33} \in \mathbb{R}^{n_b\times n_b}$, and $A_{22} \in \mathbb{R}^{k\times k}$, $k = k_d - n_b$, then identifies the blocks of the submatrix A_{MM} as those which are updated during that iteration.

In [4] the authors presented two variants of routine PBTRF, named A and B. The purpose of both variants is to concatenate blocks A_{21} and A_{31} in (1) into a

single rectangular block $\bar{A}_{21}$ of size k_d by n_b. The benefit is that the scaling of $\bar{A}_{21}$ can be then performed in a single call to the BLAS TRSM, and the update of A_{22}, A_{32}, and A_{33} can be done by a single call to the BLAS SYRK.

For variant A, they proposed to add extra n_b rows to the (bottom of the) compact storage in `AB` so that the lower band of $P + T$ could become a lower block band and, in doing this, matrix $\bar{A}_{21}$ was contained in the larger lower block band. When $\texttt{LDAB} = k_d+1$, they obtained their variant B. Here they noticed that the strictly lower triangular part of matrix A_{31}, denoted hereafter as $\text{SLT}(A_{31})$, occupies the same locations in the compact storage (i.e., in `AB`) as the lower triangle of elements $(2 : n_b, 2 : n_b)$ of A_{11}, denoted as $\text{LT}(A_{11}(2 : n_b, 2 : n_b))$. Accordingly, they used the auxiliary array `WORK` of the LAPACK routine PBTRF to save submatrix A_{11} and then replace $\text{LT}(A_{11}(2 : n_b, 2 : n_b)) = \text{SLT}(A_{31})$ in `AB` with zeros. Hence, matrix $\bar{A}_{21}$ was now formed inside `AB` instead of the way they formed the matrix $\bar{A}_{21}$ by using extra storage of their previous variant A.

In this contribution we also expose a novel method to accomplish variant A. Like variant B, there is space in the array space `AB` for block $\bar{A}_{21}$ that does not overlap any elements of AB if one chooses $\texttt{LDAB} \geq k_d + n_b$.

With the advent of multi-core platforms with more than four cores, the previous approach based on multi-threaded BLAS becomes suboptimal. The paper [3] discusses a different data layout for `AB` that uses square block (SB) format to represent matrix AB in concert with using BLAS kernels [2]. Their solution introduces dynamic scheduling of the SB operations to honor dependencies and computation on the SB using single-threaded BLAS kernels.

When one changes an algorithm that has been a standard, the migration problem needs to be addressed: there is a lot of existing software that uses the previous algorithm so that backward compatibility is important. This is especially so for a library like LAPACK. This issue was discussed earlier in [2]. We find a similar issue with our variant A. When there is extra unused space in the array holding band matrix AB, specified by $\texttt{LDAB} \geq k_d + n_b$, our variant A can use it. However, using this space will destroy the users' data that resides there. We think this is highly unlikely but possible. So, strictly speaking, in the case of migration one would be prohibited from using our variant A. Because of this, our choice of code for migration must be variant B.

This paper is structured as follows. In Section 2 we discuss how to describe our versions of PBTRF using standard matrix notation. In Section 3 we review variants A and B using standard matrix notation as well as from a different perspective. Section 4 will report some performance results. The paper ends in Section 5, where we give some concluding remarks and discuss future work.

2 New Notation for LAPACK Routine PBTRF

Let matrix A have m rows and n columns. In Fortran, A is stored in a one dimensional array, denoted by `A`, with leading dimension $\texttt{LDA} \geq m$, so that the column elements are stride one apart and row elements are stride `LDA` elements apart. (The word "stride" means the distance between two consecutive matrix elements of A in a given column or row.) Clearly, the array layout `A` is one

dimensional whereas matrix A is two dimensional. The layout allows one to address any element of A in the array A as follows. Assume A is declared as

```
DATA_TYPE A(LDA,N)
```

where N=n and DATA_TYPE refers to any of the standard data types in Fortran; then $A_{i,j}$ resides at location LOC $= (j-1) *$ LDA $+ i$ of A.

Consider now our $n \times n$ symmetric band matrix AB, with bandwidth k_d, stored in a Fortran array declared as

```
DATA_TYPE AB(LDAB,N)
```

This is a one dimensional rectangular layout; see Figure 1. It is assumed that the i-th column of AB has elements $AB_{1:i-1,i}$ and $AB_{i+k_d+1:n,i}$ equal to zero and $AB_{i:i+k_d,i}$ resides in array AB(1:KD+1,I). Note also that LDAB $\geq k_d + 1$ for PBTRF. Hence, array AB contains extra storage of LDAB $- k_d - 1$ elements in each column of P. We define these extra elements of array AB as padding elements of P. For example, when $i = 1$, for matrix AB one does not store element $(1, 2)$ nor elements $(k_d + 2 : N, 1)$ of AB as they lie "outside the band". Also, array AB has padding space in column one for LDAB $- k_d - 1$ additional elements. We will able to use this extra padding space in Section 3.2. Usually LDAB $= k_d + 1$ so there is no padding.

How does one transform array AB representing the rectangle into the EP representation? *One changes* LDAB *to be* LDAB-1 *in the previous array declaration of* AB. This change effects the desired affine transformation mentioned in Section 1. Let us see why this works. In Fortran, LDA is the distance or stride between two consecutive matrix elements in the second dimension of an array A representing a matrix A. However, LDAB is the distance between elements $(1, 1)$ and $(2, 2)$ of array AB representing the band matrix AB. This fact is unknown to Fortran. As a result, one is forced to address matrix element $AB_{i,j}$ as AB(1+i-j,j) in the code. By changing LDAB to be LDAB-1 in the above declaration of AB, one specifies to Fortran that the distance in the second dimension is one less, and hence it now matches the distance between consecutive row elements of AB.

The last paragraph explained how one could "teach" Fortran to view the band rectangle AB as a parallelogram EP, allowing for addressing in LAPACK routine PBTRF using the parallelogram. This means that the $(1 + i - j, j)$ entry for the rectangle (i.e., with the array declared as AB(LDA,N)) becomes the (i, j) entry of the parallelogram EP (i.e., when the declaration of the array is changed to AB(LDA-1,N)).

Here is a simple proof: Using the band rectangle in array AB to represent band matrix AB we have, by definition, that $AB_{i,j}$ resides at location AB(LOC) with LOC = 1+i-j+LDAB*(j-1) = i+(LDAB-1)*(j-1). This means Fortran will now also address $AB_{i,j}$ if array AB is declared in Fortran as AB(LDAB-1,N).

Based on the proof, we are allowed to view the band rectangle as the parallelogram EP. It follows that the code becomes more readable and hence more understandable. Furthermore, the proof shows the execution of old and new code for PBTRF will be identical if we reproduce the original LAPACK code in terms of the new more readable code.

The new code is thus obtained from the original LAPACK routine PBTRF by introducing the following two simple changes:

1. Declare `AB` as `DATA_TYPE AB(LDAB-1,N)`
2. Replace references to `AB(1+i-j,j)` by `AB(i,j)` in PBTRF.

We illustrate this with two code fragments. The factorization of the diagonal block A_{11} from (1) in the original (double-precision real) LAPACK routine DPBTRF is changed as follows:

Original

```
 CALL DPOTF2( UPLO, IB,
$         AB( 1, I ), LDAB-1, II )
```

New

```
 CALL DPOTF2( UPLO, IB,
$         AB( I, I ), LDAB-1, II )
```

(Here, `IB = MIN( NB, N-I+1 )`, where `I` is the iteration counter that starts at 1 and is increased by `NB` at each step.)

Also, the computation of A_{21} in (1) is transformed as

Original

```
 CALL DTRSM( 'Right', 'Lower',
$        'Transpose', 'Non-unit',
$        I2, IB, ONE,
$        AB( 1, I ),    LDAB-1,
$        AB( 1+IB, I ), LDAB-1 )
```

New

```
 CALL DTRSM( 'Right', 'Lower',
$        'Transpose', 'Non-unit',
$        I2, IB, ONE,
$        AB( I, I ),    LDAB-1,
$        AB( I+IB, I ), LDAB-1 )
```

(`I2 = MIN( KD-IB, N-I-IB+1 )` with `KD` the bandwidth of the matrix.)

Note the similarity of the new calls with the analogous factorization of the diagonal block and triangular system solve in the LAPACK routine DPOTRF for the dense Cholesky factorization:

```
 CALL DPOTF2( 'Lower', JB,
$         A( J, J ), LDA, INFO )
 ...
 CALL DTRSM( 'Right', 'Lower',
$        'Transpose', 'Non-unit',
$        N-J-JB+1, JB, ONE,
$        A( J, J ),    LDA,
$        A( J+JB, J ), LDA )
```

3 Simplifying the LAPACK Routine PBTRF

In this Section we reconsider variants A and B proposed in [4]. In our presentation, we will assume that the array which holds matrix AB is declared as `AB(LDAB-1,N)` so that $AB_{i,j}$ and `AB(i,j)` refer to the same element (see previous section). With this consideration, the blocks A_{MM} from (1) which are

updated during iteration $i = \texttt{I}$ lie in the following positions of AB:

rows of AB	columns of AB	I : I + IB − 1	I + IB : I + IB + I2 − 1	I + IB + I2 : I + IB + I2 + I3 − 1
I : I + IB − 1		A_{11}	$\star$	$\star$
I + IB : I + IB + I2 − 1		A_{21}	A_{22}	$\star$
I + IB + I2 : I + IB + I2 + I3 − 1		A_{31}	A_{32}	A_{33}

with $k_d = \texttt{KD}$, $n_b = \texttt{NB}$, IB= min(NB, N-I+1), I2= min(KD-IB, N-I-IB+1), and I3= min(IB, N-I-KD+1). Note that, usually, IB=NB, I3=IB and I2+I3=KD, which occurs when A_{11} lies in P.

3.1 Variant B

The operations that are performed during a factorization and update stage of this variant are:

1. Factorize the diagonal block $A_{11} = L_{11}L_{11}^T$, overwriting the entries of the lower triangle of A_{11} with those of L_{11}.
2. Copy WORK(1:IB,1:IB):= LT(A_{11}) and set LT(A_{31}) := 0.
3. Let $\bar{A}_{21} = \begin{bmatrix} A_{21} \\ A_{31} \end{bmatrix}$. Scale this concatenated matrix by calling BLAS kernel TRSM using the lower triangular factor L_{11} obtained from the factorization of A_{11} stored in WORK: $\bar{A}_{21} := \bar{A}_{21}L_{11}^{-T}$.
4. Let $\bar{A}_{22} = \begin{bmatrix} A_{22} & * \\ A_{32} & A_{33} \end{bmatrix}$. Compute the symmetric rank-IB update of $\bar{A}_{22}$ with a single call to BLAS kernel SYRK: $\bar{A}_{22} := \bar{A}_{22} - \bar{A}_{21}\bar{A}_{21}^T$.
5. Restore SLT(A_{31}) := WORK(2 : IB, 2 : IB).

Assume IB=NB, I3=IB and I2+I3=KD. Then LT(AB(I + 1 : IB, I + 1 : IB)) occupies the same space in the array as SLT(AB(I + KD + 1 : I + KD + IB, I : IB)). Also, A_{11} minus its first column is the first submatrix and A_{31} minus its last column is the second submatrix.

This proof is straight-forward. Array AB is laid out as a set of N vectors each of length KD+1. Thus, $AB_{i+k_d+2,i}$ does not lie in array AB. However, the address of this element in AB is at LOC=(1,I+1) which, due to the declaration of EP, is at LOC=(I+1,I+1). Since the layout of the storage is linear and IB $\leq$ KD, we have our result. Also, as the iteration index I $\rightarrow$ N, A_{31} becomes smaller, and one can appeal to the unused storage in the band rectangle to complete the above proof. Here note that $P + T$ has the same unused storage and is really EP. Clearly, AB and EP are equivalent forms of the Fortran storage layout as they are related via an affine transformation. This proves the validity of the above procedure and hence the correctness of variant B when LDAB=KD+1.

The procedure also holds when LDAB > KD+1. When LDAB $\geq$ KD + NB, SLT(A_{31}) and LT(A_{11}) do not overlap. In case KD + 1 $\leq$ LDAB $<$ KD + NB, SLT(A_{31}) and LT(A_{11}) partially overlap. In both cases, we need to save and restore in WORK the areas of AB that we overwrite with zeros to form SLT(A_{31}). In the overlapping

case we also need to form LT(A_{11}) in `WORK`. This completes our demonstration of our variant B. However, we further discuss the case of `I` near `N`. Here we mean that `I3 < IB` or that we are in the T region of $P+T$. One can see that A_{31} is trapezoidal and that less zeroing is required in Step 3. above because the A_{31} part of $\bar{A}_{21}$ is now smaller (rectangular).

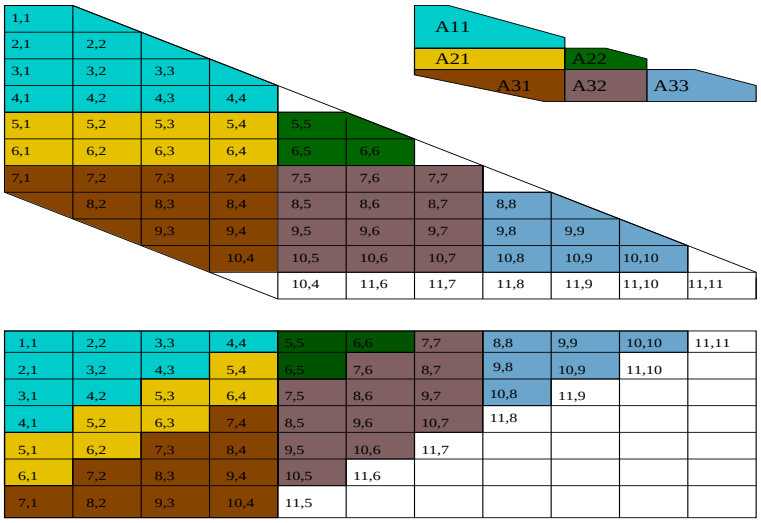

Fig. 2. Symmetric band matrix AB of order n = `N` = 11 with half bandwidth k_d = `KD`=6 (top), and packed band storage as a rectangle `AB` with leading dimension `LDAB`= k_d+1 (bottom). The colors denote the different submatrices $A_{11}, A_{21}, \ldots, A_{33}$ identified during the first iteration of the blocked algorithm with n_b = `NB`=4.

We now illustrate these remarks. Consider the first iteration of the factorization (`I`=1) and the highlighted blocks in Figure 2. Then, `IB`=4, `I2`=2, `I3`=4. Clearly, SLT(A_{31}) = LT(`AB`$(8:10, 1:3)$) overlaps exactly with LT(A_{11}) = LT(`AB`$(2:4, 2:4)$). Now, let `I=5` in the figure. Then, `IB`=4, `I2`=2, `I3`=1, and SLT(A_{31}) is empty as A_{31} = `AB`$(11, 5:8)$. Thus, $\bar{A}_{21}$ = `AB`$(9:11, 5:8)$ is contained entirely in `AB` and $\bar{A}_{22}$ = `AB`$(9:11, 9:11)$. It follows that Steps 2. and 5. become no operations if we use A_{11} instead of `WORK` in Step 3. Finally, let `I=9` in the figure. Then, `IB` = 3, and `I2 = I3` = 0 and only a dense Cholesky factorization of order 3 is performed on `AB(9:11,9:11)`.

The following fragment of LAPACK code corresponds to our variant B:

```
*        Compute the Cholesky factorization of a symmetric band
*        matrix, given the lower triangle of the matrix in band
*        storage. Assume here that LDAB = KD + 1 and KD >= NB.
*        Process the band matrix one diagonal block at a time.
```

```
*
      DO 120 I = 1, N, NB
         IB = MIN( NB, N-I+1 )
*
*        Factorize the diagonal block
*
         CALL DPOTF2( UPLO, IB, AB( I, I ), LDAB-1, II )
         IF( II.NE.0 ) THEN
            INFO = I + II - 1
            GO TO 150
         END IF
         IF( I+IB.LE.N ) THEN
*
*           Update the relevant part of the trailing submatrix.
*           If A11 denotes the diagonal block which has just been
*           factorized, then we need to update the remaining
*           blocks in the diagram:
*                 A11
*                 A21   A22
*                 A31   A32   A33
*           Copy the lower triangle A11 to the work array.
*           Also zero strictly lower part of A31
*
            DO 90 JJ = 1, IB
               DO 80 II = JJ, IB
                  WORK( II, JJ ) = AB( I+II-1, I+JJ-1 )
                  IF( JJ.GT.1 ) AB( I+II-1, I+JJ-1 ) = ZERO
80             CONTINUE
90          CONTINUE
*
*           The numbers of rows and columns in the partitioning
*           are IB, I2, I3 respectively. The blocks A21, A22 and
*           A32 are empty if IB = KD. The lower triangle of A31
*           lies outside the band.
*
            I2 = MIN( KD-IB, N-I-IB+1 )
            I3 = MIN( IB, N-I-KD+1 )
*           Update A21, A31
            CALL DTRSM( 'Right', 'Lower', 'Transpose',
     $                  'Non-unit', I2+I3, IB, ONE, WORK, LDWORK,
     $                  AB( I+IB, I ), LDAB-1 )
*
*           Update A22, A32, A33
*
            CALL DSYRK( 'Lower', 'No Transpose', I2+I3, IB, -ONE,
     $                  AB( I+IB, I ), LDAB-1, ONE,
     $                  AB( I+IB, I+IB ), LDAB-1 )
*
*           Copy the work array back to the lower triangle A11.
*
```

```
                DO 110 JJ = 2, IB
                   DO 100 II = JJ, IB
                      AB( I+II-1, I+JJ-1 ) = WORK( II, JJ )
  100              CONTINUE
  110           CONTINUE
  120        CONTINUE
```

3.2 Variant A

The alternative form of Variant A is to choose LDAB $\geq$ KD + NB. In that case, SLT(A_{31}) never overlaps with LT(A_{11}). Thus, LT(A_{11}) is not destroyed and there is no need for additional work-space. What we are doing here is using the extra padding element described in Section 2 to find room in array EP for the larger block band of Variant A. However, the extra space in array EP is greater than the block band format described in [4] by a factor of two. These brief remarks demonstrate our alternative form of variant A.

4 Experimental Results

All experiments in this section were performed using IEEE double-precision (real) arithmetic and band matrices of order n=5,000, with bandwidth size ranging from $k_d = 1$ to 1,200. Provided $n \gg k_d$, the performance of the routines is only determined by k_d and the block size n_b. In the evaluation, for each bandwidth dimension, k_d, we employed values from 1 to 200 to determine the best block size; only those results corresponding to the optimal value obtained are shown.

Platform	Architecture	#Proc.	Frequency (GHz)	L2 cache (KBytes)	L3 cache (MBytes)	RAM (GBytes)
XEON	Intel Xeon	2	2.4	512	–	1
ITANIUM	Intel Itanium2	4	1.5	256	4	4

Table 1. SMP architectures employed in the evaluation.

Platform	BLAS	Compiler	Optimization Flags	Operating System
XEON	GotoBLAS 1.15 MKL 8.1.1	gcc 3.3.5	-O3	Linux 2.4.27
ITANIUM	GotoBLAS 1.15 MKL 8.0	icc 9.0	-O3	Linux 2.4.21

Table 2. Software employed in the evaluation.

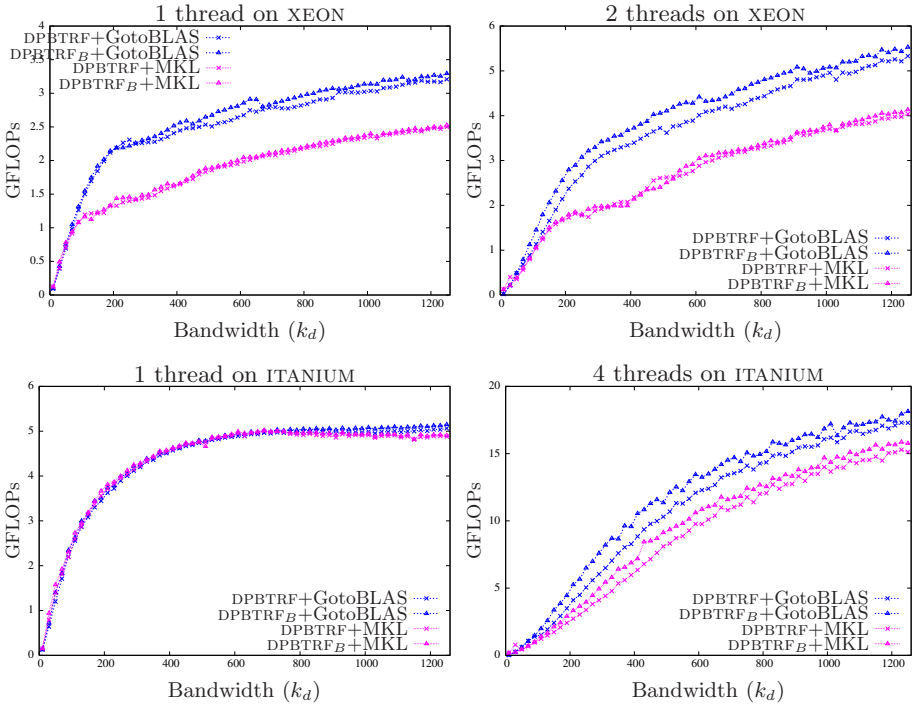

Fig. 3. Performance of the routines for the band Cholesky factorization.

We report the performance of the routines on two different SMP architectures, with 2 and 4 processors; see Table 1. Two threads were employed for the parallel execution on XEON and 4 threads on ITANIUM. As the efficiency of the kernels in BLAS is crucial, for each platform we use the software listed in Table 2.

Figure 3 illustrates the GFLOPs rate (billions of floating-point arithmetic operations per second) of the original code DPBTRF in the lines labeled as 'DPBTRF+GotoBLAS' (linked with GotoBLAS) and 'DPBTRF+MKL' (linked with MKL). We also report in the figure the performance of Variant B of the code in lines 'DPBTRF$_B$+GotoBLAS' and 'DPBTRF$_B$+MKL'. (The performance of Variant A is in practice slightly better than that of Variant B reflecting the cost of the extra copies in Step 2. and 5. of Section 3; see also [4]).

To briefly summarize these results, the gains that Variant B yields by merging the updates of the different parts of the matrix in larger blocks are important only for the parallel (multi-threaded) executions. The reason for this is the *fork-join model* of parallel execution of LAPACK: When a LAPACK routine is executed, all parallelism is extracted from the calls to BLAS. Thus, every time a BLAS kernel is invoked, threads are *spawn* (fork) at the beginning of the execution and *synchronize* (join) at the end. Having multiple calls to BLAS, some of them

for updating small blocks like A_{31} and A_{33} in (1), is therefore harmful for the parallel execution.

5 Conclusions and Future Work

We have shown that a minor change to the storage format representation of symmetric band matrices leads to a much more intuitive description of the program code by allowing it to reference element (i, j) in the band matrix as `AB(i,j)`. This change does not affect the interface of LAPACK routine PBTRF in terms of its function nor of its data layout.

We have also amplified on code modifications which merge several operations that are performed during a block factorization and update stage of PBTRF. These additional modifications provide simpler codes, and also higher performance when the routine is combined with a multi-threaded implementation of BLAS and executed in parallel on an SMP platform.

We have also given proofs of correctness of our clearer and simpler LAPACK Cholesky band codes.

Future work includes addressing the upper symmetric case of the band Cholesky factorization as well as other band factorization routines.

References

1. E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
2. Fred G. Gustavson. High Performance Linear Algebra Algorithms using New Generalized Data Structures for Matrices. *IBM Journal of Research and Development*, 47(1):823–849, January 2003.
3. Gregorio Quintana-Ortí, Enrique S. Quintana-Ortí, Alfredo Remón, and Robert van de Geijn. SuperMatrix for the factorization of band matrices. FLAME Working Note #27 TR-07-51, The University of Texas at Austin, Department of Computer Sciences, September 2007.
4. Alfredo Remon, Enrique Quintana-Orti, and Gregorio Quintana-Orti. Cholesky Factorization of Band Matrices Using Multithreaded BLAS. In *Applied Parallel Computing, State of the Art in Scientific Computing, PARA 2006*, volume LNCS 4699, pages xxx–yyy, Springer-Verlag, Berlin Heidelberg, 2007. Springer.