Cholesky Factorization of Band Matrices Using Multithreaded BLAS

Alfredo Remón[1], Enrique S. Quintana-Ortí[1], and Gregorio Quintana-Ortí[1]

Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain, {remon,quintana,gquintan}@icc.uji.es

Abstract. In this paper we analyze the efficacy of the LAPACK blocked routine for the Cholesky factorization of symmetric positive definite band matrices on Intel SMP platforms using two multithreaded implementations of BLAS. We also propose strategies that alleviate some of the performance degradation that is observed, and which is basically due to the use of multiple threads when dealing with problems of small scale.

1 Introduction

Exploiting the structure of the coefficient matrix for band linear systems yields huge savings, both in number of computations and storage space. This is recognized in LAPACK [1], which includes a blocked routine for the Cholesky factorization of symmetric positive definite band (SPDB) matrices.

In this paper we perform an experimental evaluation of the LAPACK (double-precision) routine DPBTRF on Intel Xeon^{TM} and Intel Itanium2^{TM} SMP platforms. Underlying this routine, we employ the implementations of BLAS in Goto BLAS and MKL. This evaluation reveals some weaknesses and defines a path to improve the LAPACK routine and the kernels in the BLAS implementations.

The paper is structured as follows. The blocked factorization routine in LAPACK is reviewed in Section 2. Performance results together with some concluding remarks are offered in Section 3.

2 LAPACK blocked routine for band Cholesky factorization

Given a SPDB matrix $A \in \mathbb{R}^{n\times n}$, with bandwidth k_d, the LAPACK routine xPBTRF obtains a decomposition of this matrix into either $A = U^TU$ or $A = LL^T$, where the Cholesky factors $U, L \in \mathbb{R}^{n\times n}$ are, respectively, upper and lower triangular with the same bandwidth as A. We only consider hereafter the latter decomposition, but the elaboration that follows is analogous for the upper triangular case. Also, for brevity, we obviate the description of the packed storage format employed in LAPACK for (symmetric) band matrices.

Consider the partitionings

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \quad \text{and} \quad L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right), \tag{1}$$

where A_{TL}, L_{TL} are both square blocks of the same dimension. (The "$\star$" symbol in A denotes the symmetric quadrant (block) of the matrix and will not be referenced.) From $A = LL^T$, we obtain

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)^T = \left(\begin{array}{c|c} L_{TL}L_{TL}^T & \star \\ \hline L_{BL}L_{TL}^T & L_{BL}L_{BL}^T + L_{BR}L_{BR}^T \end{array}\right),$$

showing that L_{TL} is the Cholesky factor of A_{TL}, $L_{BL} = A_{BL}L_{TL}^{-T}$, and L_{BR} is the Cholesky factor of $A_{BR} - L_{BL}L_{BL}^T$. The LAPACK routine corresponds to what is usually known as a right-looking variant; that is, an algorithm where, at a given iteration, A_{TL} has been completely factorized and overwritten by L_{TL}, A_{BL} has been updated with respect to L_{TL}, and A_{BR} has been overwritten as $A_{BR} := A_{BR} - L_{BL}L_{BL}^T$.

Assume for simplicity that the block size, n_b, is an exact multiple of k_d, and consider now the 5×5 repartitioning

$$\left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c|c|c} A_{00} & \star & \star & & \\ \hline A_{10} & A_{11} & \star & \star & \\ \hline A_{20} & A_{21} & A_{22} & \star & \star \\ \hline & A_{31} & A_{32} & A_{33} & \star \\ \hline & & A_{42} & A_{43} & A_{44} \end{array}\right),$$

where $A_{TL}, A_{00} \in \mathbb{R}^{k\times k}$, $A_{11}, A_{33} \in \mathbb{R}^{n_b\times n_b}$, and $A_{22} \in \mathbb{R}^{l\times l}$, with $l = k_d - n_b$, and an analogous partitioning for L. The computations that are performed in the next iteration of routine xPBTRF are

$$\begin{array}{lll} 1.1)\ A_{11} = L_{11}L_{11}^T, & & \\ 2.1)\ A_{21} := A_{21}L_{11}^{-T}, & 2.2)\ A_{22} := A_{22} - L_{21}L_{21}^T, & \\ 3.1)\ A_{31} := A_{31}L_{11}^{-T}, & 3.2)\ A_{32} := A_{32} - L_{31}L_{21}^T, & 3.3)\ A_{33} := A_{33} - L_{31}L_{31}^T. \end{array} \tag{2}$$

Operation 1.1) is simply obtained as a (dense) Cholesky factorization of A_{11}, while 2.1) and 2.2) are computed, respectively, by invoking the BLAS-3 triangular linear system solver (xTRSM) and the BLAS-3 symmetric rank-k kernel (xSYRK). Given that with these repartitionings, A_{31} and L_{31} are both upper triangular, the implementation of operations 3.1)–3.3) in xPBTRF is as follows. In order to solve the triangular linear system in 3.1), a copy of A_{31} is first obtained in an auxiliary workspace W of dimension $n_b \times n_b$, initialized with zeros in the subdiagonal entries, and then the BLAS-3 solver xTRSM yields $W := WL_{11}^{-T}$. Next, 3.2) is computed as a matrix product using the BLAS-3 kernel xGEMM as $A_{32} := A_{32} - WL_{21}^T$. Finally, the update in 3.3) is obtained using kernel xSYRK as $A_{33} := A_{33} - WW^T$, and the upper triangular part of W is written back to A_{31}. From this particular implementation we observe that:

– Provided $n_b \ll k_d$, a major part of the floating-point arithmetic operations (flops) are performed in 2.2) so that the performance of the LAPACK blocked routine should be similar to that of xSYRK.
– No attempt is made to exploit the upper triangular structure of A_{31}, L_{31} in the computations corresponding to 3.1)–3.3) as there is no appropriate BLAS kernel. Furthermore, the additional space W and the extra copies are only required in order to use BLAS-3 to perform these operations.

3 Experimental Results

All experiments were performed using IEEE double-precision (real) arithmetic and SPDB matrices of order $n = 10000$. In the evaluation of routine DPBTRF, for each bandwidth dimension, we employed values from 1 to 200 to determine the optimal block size, n_b^{opt}, but only those results corresponding to n_b^{opt} are shown.

We report the performance of DPBTRF on two SMP architectures: the first platform, XEON, consists of 2 Intel Xeon processors@2.4 GHz with 512 KB of L2 cache; the second platform, ITANIUM, is composed of 4 Intel Itanium-2 processors@1.5 GHz with 256 KB/4 MB of L2/L3 cache; two threads were used on XEON (2T) and four threads on ITANIUM (4T). The BLAS implementation in MKL 8.0 was used on both platforms; on XEON we also used Goto BLAS 1.00 while on ITANIUM we used Goto BLAS 0.95mt.

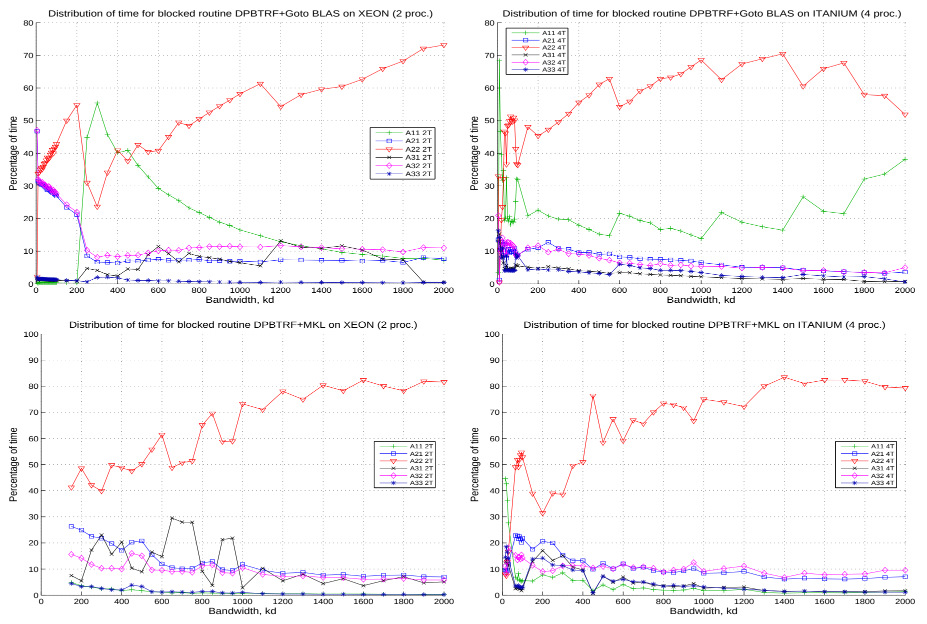

Fig. 1. Distribution of time among the different operations involved in routine DPBTRF on XEON (left) and ITANIUM (right) using multithreaded Goto BLAS (top) and MKL (bottom).

Figure 1 reports the results from a detailed evaluation which shows how the time is distributed among the different operations in DPBTRF: (factorization of) A_{11}, (update of) A_{21}, etc. (see (2)) Two main observations can be drawn from this experiment: The factorization of A_{11} is a major performance bottleneck for the routine in case Goto BLAS is employed. A closer inspection revealed this to be due to the low performance of the multithreaded implementation of DGEMV, invoked from DPOTF2 during the factorization of A_{11}, when the size of this block ($n_b \times n_b$) is small. A similar inspection concluded that MKL employs a single thread in such situation and therefore avoids this bottleneck. A second source of performance degradation is the update of A_{31}, both for MKL and Goto BLAS. Although in theory the time required to update this block should be negligible, the experiments show that this is not the case, specially on XEON.

Figure 2 illustrates the MFLOPs (millions of flops per second) of the original codes DPBTRF in the lines labeled as `Goto` and `MKL`. In order to overcome the first problem, when using the Goto BLAS, we factorize A_{11} using routine DPOTF2 with the code in kernel DGEMV directly inlined. This ensures that a single thread is used during this operation and obtains an important performance enhancement, as shown in Fig. 2 by the lines labeled as `Goto + Inline`.

The second strategy we propose consists in merging the update of A_{31} with that of A_{21} and the update of A_{32} and A_{33} with that of A_{22}. For that purpose, we propose a minor modification of the storage scheme used in LAPACK so that the $k_d \times n$ packed matrix is passed to the routine with n_b additional rows in the bottom part of the matrix, initially set to zeros. This allows to combine the operations of the routine so that only single invocations of DTRSM and DSYRK are required per iteration to operate on the subdiagonal blocks of the band matrix. The improvement that is obtained in the performance is illustrated by the lines labeledd as `Goto + Inline & MS` and `MKL + MS` in Fig. 2.

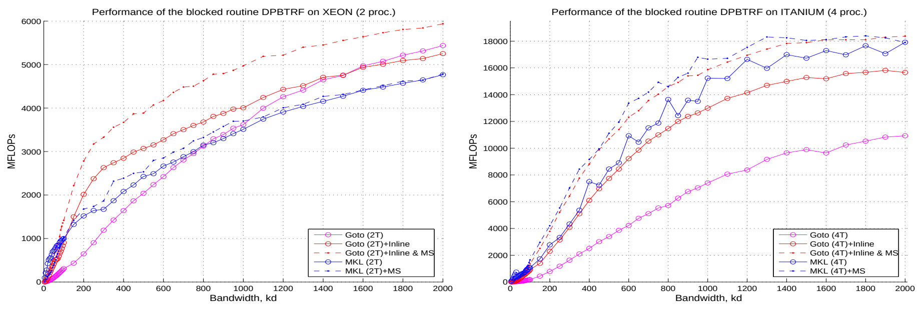

Fig. 2. Performance of routine DPBTRF on XEON (left) and ITANIUM (right) using multithreaded Goto BLAS and MKL.

References

1. E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.