Block Algorithm and Its Implementation for Cholesky Factorization

Jianping Chen, Zhe Jin, Quan Shi, Jianlin Qiu, Weifu Liu
School of Computer Science and Technology
Nantong University
Nantong, P. R. China
e-mail: chen.jp@ntu.edu.cn, jin.z@ntu.edu.cn, shi.q@ntu.edu.cn, qiu.jl@ntu.edu.cn, liu.wf@ntu.edu.cn

***Abstract*—Block algorithm divides a large matrix into small blocks of submatrices to make full use of computer's caches. Blocked smaller submatrices can be directly loaded into the caches to compute. The efficiency of the computation is hence improved. The block algorithm for Cholesky factorization is studied in this paper. Detailed derivation and implementation of the block algorithm are presented. The computations of modules involved in the block algorithm are discussed. A C++ language program is developed. It is tested and compared with the standard (unblocked) version of the Cholesky factorization. The testing results show that the presented block algorithm outperforms the unblocked one in the cases of large matrices, with an increase of execution speed of 20%.**

Keywords-Numerical computation; Cholesky factorization; matrix blocking; cache use.

I. INTRODUCTION

Numerical computations of linear algebra problems including the Cholesky factorization play very important roles in many scientific researches and engineering applications. Today's computers are provided with two or more levels of caches between a high-speed CPU and a relatively low-speed memory. The caches operate at a speed close to that of the CPU. If the data computed currently or used frequently is placed in the caches, the access time of the data can be reduced significantly. The overall efficiency of the computation is hence increased greatly. However, the capacity of the cache memory is much smaller. If the amount of data being computed is very big, such as the case of the Cholesky factorization for a large matrix, it is impossible to load all the data into the cache. The cache can not be well utilized and the computation is less efficient. To solve the problem, the method of matrix blocking can be used [1, 2]. A large matrix is partitioned into blocks of submatrices of small sizes. The computations of these smaller submatrices are more likely to be carried out within caches. Li [3] has discussed the block techniques used in LAPAK (Linear Algebra PACKage, a software library for performing numerical linear algebra computations) and given the testing results on different machines. Andersen et al. [4] presented a recursive block algorithm for Cholesky factorization. It uses the technique of recursive blocking, where a big matrix is blocked recursively (each time split by half) until the final submatrices are small enough. With this recursive blocking method, the blocking process can be made automatically while the block sizes in each level are different. At outer levels, the block sizes are still quite big, and at inner levels, the block sizes can be very small. Ng and Peyton [5] investigated the block algorithm of Cholesky factorization for the sparse matrix, where many of the matrix elements are zeros and special measures are taken to deal with them such as supernodes. The block algorithms mentioned above including the ones in the LAPACK were implemented by using the BLAS (Basic Linear Algebra Subroutine) routines [6], which is an open software library to perform basic linear algebra operations such as vector and matrix multiplications. On the other hand, these algorithms were developed and tested on the machines of more than 10 years ago, at which time the computers had small cache memories of 64K or 128K. Todayôs computers, including microcomputers, have much greater cache memories of more than 1M with very big main memories that reach the order of gigabytes. It is valuable to examine the effects of the block algorithms on these machines. The block algorithm of Cholesky factorization for a general matrix (not the sparse case) is investigated in this paper. The fixed blocking method is used, that is, a big matrix is divided into small blocks in a fixed size linearly part by part. Instead of using the BLAS subroutines, we do the implementation ourselves by C++ programming and discuss the implementation techniques involved in the block algorithm.

The remainder of the paper is organized as follows. Section 2 derives the block algorithm of Cholesky factorization. Section 3 discusses the implementation of the block algorithm and the computations of the modules involved. Presented in Section 4 are testing results and analysis. Finally, conclusions and considerations for future work appear in Section 5.

II. BLOCK ALGORITHM FOR CHOLESKY FACTORIZATION

Cholesky factorization is used to solve linear systems of equations in the case that the coefficient matrix A is symmetric and positive definite:

$$AX = B. \tag{1}$$

A can be factorized into the product of an upper-triangular matrix U with a lower-triangular matrix U^T (U^T is

 ISBN: 978-1-61208-283-7

the transpose of U):

$$A = U^T U \tag{2}$$

with

$$U = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{21} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix}. \tag{3}$$

According to the standard algorithm of the Cholesky factorization [7], the matrix U is computed as follows.

$$u_{ii} = (a_{ii} - \sum_{k=1}^{i-1} u_{ki}^{\ 2})^{1/2} \qquad i=1, 2, \text{é}, n \tag{4}$$

$$u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} u_{ki}u_{kj}) \Big/ u_{ii} \qquad j=i+1, i+2, \text{é}, n. \tag{5}$$

Now, we derive the block algorithm for Cholesky factorization. Since the matrix A is symmetric, only half of the data is needed. Here, the upper triangular part is used. We write (2) in a form of blocked submatrices:

$$\begin{pmatrix} A_{11} & A_{12} \\ & A_{22} \end{pmatrix} = \begin{pmatrix} U_{11}^T & \\ U_{12}^T & U_{22}^T \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix}. \tag{6}$$

Assume that the order of A is $n \times n$. If we set the block size at d, then the order of A_{11} and U_{11} are $d \times d$, A_{12} and U_{12} are $d \times (n\text{-}d)$, and A_{22} and U_{22} are $(n\text{-}d) \times (n\text{-}d)$. Among these blocked submatrices, A_{11}, U_{11}, A_{22} and U_{22} are upper triangular, and A_{12} and U_{12} are rectangular. Computing (6) in the blocked form, we obtain

$$A_{11} = U_{11}^T U_{11} \tag{7}$$

$$A_{12} = U_{11}^T U_{12} \tag{8}$$

$$A_{22} = U_{12}^T U_{12} + U_{22}^T U_{22}. \tag{9}$$

Eq. (7) is the Cholesky factorization of the submatrix A_{11}. It can be computed using the standard Cholesky factorization formulae in (4) and (5). If we properly choose the value of the block size d, i.e., the order of A_{11}, the computation is likely to be carried out within the cache. After U_{11} is solved, U_{12} is computed by (8):

$$U_{12} = (U_{11}^T)^{-1} A_{12}. \tag{10}$$

Let

$$A'_{22} = A_{22} - U_{12}^T U_{12}. \tag{11}$$

Eq. (9) becomes

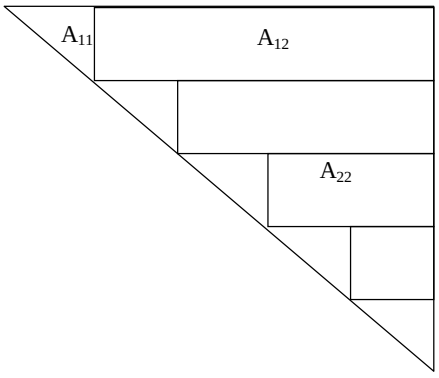

Figure 1. Blocking process of the block algorithm for Cholesky factorization

$$A'_{22} = U_{22}^T U_{22}. \tag{12}$$

Eq. (11) denotes a modification or update of A_{22} through U_{12}. The updated A_{22} is expressed as A'_{22}. Eq. (12) is the Cholesky factorization of A'_{22}. Usually, A'_{22} is still a big matrix. The same blocking process is then applied to A'_{22}. This process is repeated until the order of the final A'_{22} is less than or equal to d. At this time, the Cholesky factorization of the final A'_{22} is computed using the standard formulae of Cholesky factorization in (4) and (5). Fig. 1 illustrates the blocking process.

In summary, the block algorithm for Cholesky factorization can be described as follows.

Step 1: Block matrix A into submatrices A_{11}, A_{12}, A_{22} with a block size of d.

Step 2: Cholesky-factorize A_{11}: $A_{11} = U_{11}^T U_{11}$, yielding U_{11}.

Step 3: Compute the inverse of U_{11}^T, yielding $(U_{11}^T)^{-1}$.

Step 4: Compute U_{12}: $U_{12} = (U_{11}^T)^{-1} A_{12}$.

Step 5: Update A_{22}: $A'_{22} = A_{22} - U_{12}^T U_{12}$.

Step 6: If the order of A'_{22} is less than or equal to d, Cholesky-factorize A'_{22}: $A'_{22} = U_{22}^T U_{22}$, yielding U_{22}. Otherwise, take A'_{22} as A and return to Step 1.

III. IMPLEMENTATION OF BLOCK ALGORITHM

According to the description in above section, the computation of the block algorithm for Cholesky factorization consists of five modules, which include the Cholesky factorization of a small submatrix (A_{11}), the computation of the inverse of a lower-triangular submatrix ($(U_{11}^T)^{-1}$), the multiplication of a lower-triangular submatrix by a rectangular submatrix ($(U_{11}^T)^{-1} A_{12}$), the multiplication of two rectangular submatrices ($U_{12}^T U_{12}$), and the control of the repeated blocking process.

A. Cholesky Factorization of Submatrix

The standard formulae in (4) and (5) are used to compute the Cholesky factorization of the small submatrix A_{11}. Since the matrix A_{11} is upper-triangular, the packed storage method can be applied to save the memory space. If a two dimension array is used to store the matrix, the number of the elements in the *i*th row of the array is *n-i* (*i=0, 1, 2,* é *, n-1*). The outcome matrix (U_{11}) is stored in the same way.

As the elements of the matrix are stored in the packed form, the subscript of each element in the two dimension array differs from its corresponding numbers of the row and the column in the original matrix. The new data structure is shown in Fig. 2.

Since the data structure is changed, the formulae in (4) and (5) need to be modified accordingly. The modified formulae are as follows.

$$u_{i1} = (a_{i1} - \sum_{k=1}^{i-1} u_{k(i-k)}{}^2)^{1/2} \qquad i=1, 2, \text{é}, n \qquad (13)$$

$$u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} u_{k(i-k)} u_{k(j+i-k)}) \Big/ u_{i1} \qquad j=1, 2, \text{é}, n-j \qquad (14)$$

B. Inverse Computation of Lower-Triangular Matrix

It is easy to prove that the inverse of a lower-triangular matrix is still a lower-triangular matrix and the values of its diagonal elements are the reciprocals of the values of the diagonal elements in the original matrix. So, the diagonal elements can be computed separately.

For convenience of the description, we use L to represent a lower-triangular matrix and B to represent its inverse matrix. Assume $L=(l_{ij})_{d \times d}$ and $K=(k_{ij})_{d \times d}$. The diagonal elements of K can be easily computed as

$$k_{ii} = \frac{1}{l_{ii}} \qquad i=1, 2, \text{é}, d. \qquad (15)$$

The other elements can be computed by making use of the characteristics of the triangular matrix instead of using the normal method of Gaussian-Jordan elimination that is less efficient in this case. According to the theorem of linear algebra, the product of the multiplication of a matrix and its inverse matrix is a unit matrix, i.e., a matrix with its elements are zeros except the ones at the diagonal. So, we have

$$\sum_{k=j}^{i} (l_{ik} \times k_{ij}) = 0 \qquad j<i. \qquad (16)$$

Thus, the other elements of K are computed as follows.

$$k_{ij} = -\sum_{k=j}^{i-1} (l_{ik} \times k_{kj}) \quad i=1, 2, \text{é}, d-1; \; j=1, 2, \text{é}, i-1. \quad (17)$$

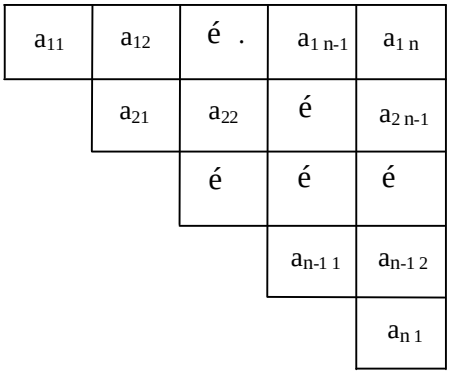

Figure 2. The data structure corresponding to the packed storage

C. Multiplication of Lower-Triangular Matrix by Rectangular Matrix

The order of the lower-triangular matrix ($(U_{11}^T)^{-1}$) is $d \times d$ and the order of the rectangular matrix (A_{12}) is $d \times (n-d)$. The outcome matrix is a also rectangular matrix of $d \times (n-d)$. The data of the rectangular matrix A_{12} comes from the initial input matrix A. The computing result (U_{12}) can be directly put into the output matrix of the block algorithm. For both of the two matrices, we need to determine their starting positions in the arrays.

If it is a normal $n \times n$ two dimension array, we just need to add an offset d to the original starting position of each row. Assuming that the original matrix is R and the matrix actually used is P, the relationship is

$$P[i][j] = R[i][j+d]. \qquad (18)$$

However, the matrix is now stored in the packed form. The offset to be added to the starting position of each row need be modified to *d-i*, where *i* is the number of the row. Therefore, the actual relationship between the two matrices is

$$P[i][j] = R[i][j+d-i]. \qquad (19)$$

The computation method is simply the sequential multiplication of the elements of the corresponding row and column of the two matrices and the summation of the products.

D. Multiplication of Two Rectangular Matrices

The two rectangular matrices multiplied are U_{12}^T and U_{12}, where the order of U_{12}^T is $(n-d) \times d$ and U_{12} is $d \times (n-d)$. If n is very large, $n-d$ is still very large. The two matrices can not be entirely loaded into the cache. Let S represent U_{12}^T and Y represent U_{12}. When doing the multiplication, we need to repeatedly multiply the elements of one row of S with the elements of a column in Y. In C++ language, the elements of an array are stored by rows. The fetch of elements in Y would be across the rows. Each part of data in Y would be

 ISBN: 978-1-61208-283-7

loaded in and out constantly between the cache and memory. This will slow the computation. To avoid the problem, we apply transposing to the matrix Y before computation. Then the computation becomes the multiplication of one row of S with a row in Y successively.

As we know, the matrices S and Y are transposes each other, i.e., $S = Y^T$. After being transposed, the matrix Y becomes exactly the matrix S. Therefore, just one matrix S is needed in the computation. In addition, the result of the multiplication of two mutually transposed matrices is a symmetric matrix. Only half of the elements need to be computed. The operations of multiplication can be reduced by half.

E. Control of Repeated Blocking Process

The function of this part is to control the computing procedure of the block algorithm. It calls each of the modules in the algorithm to perform the corresponding computation. The preliminary work for the computation of each module, such as the matrix transposing and the matrix positioning, is also completed in this part. Recursion is used to implement the repeated blocking processing of the input matrix.

IV. TESTING RESULTS AND ANALYSIS

Based on the discussion of the previous sections, a C++ language program is developed for the block algorithm of Cholesky factorization. The program is run and tested on a microcomputer. Its CPU is Core2 of 2.0GHz with a 2MB L2 cache and the main memory is 2 GB. The compiler used is Microsoft Visual C++ 6.0. Different block sizes and matrix orders are tried to test the execution time of the algorithm. A block size about 50 to 100 turns out to be the best choice for most cases. To make a comparison, the standard algorithm of Cholesky factorization is also programmed and run on the same computer. Table 1 gives the execution times of the block algorithm and the standard algorithm of Cholesky factorization for different matrix orders from 500 to 5000. The data in the fourth row of the table is the speed increase of the block algorithm over the standard algorithm in percentage.

TABLE 1. EXECUTION TIMES OF THE BLOCK ALGORITHM COMPARED THE STANDARD ALGORITHM (IN SECONDS)

Order of matrix	Standard Cholesky	Block Cholesky	Speedup in %
500	0.26	0.35	-34.6
1000	2.39	2.77	-15.9
2000	24.37	24.05	1.3
3000	88.42	82.16	7.1
4000	226.61	194.38	14.2
5000	474.4	382.13	19.4

As the data in the table shows, the block algorithm is superior to the standard algorithm in the case of a large matrix. With the increase of the matrix size, the advantage becomes more and more remarkable. The increase of the execution speed is about 20% when the matrix order reaches 5000.

The arithmetic complexity (total number of float multiplications and divisions) of the direct computation of Cholesky factorization using the equation (4) and (5) is about $n^3/6$ for large values of n where n is the order of the matrix [7]. The block algorithm blocks the matrix for n/d times where d is the block size. At each time, the arithmetic operations contain the direct computation of Cholesky factorization of a matrix ($d \times d$), the inverse computation of a lower-triangular matrix ($d \times d$), the multiplication of a lower-triangular matrix ($d \times d$) with a rectangular matrix ($d \times (n-d)$), and the multiplication of two rectangular matrices (($n-d$)$\times d$ and $d \times (n-d)$), where d is fixed and n is reduced by d each time. It can be counted that the total number of the float multiplications and divisions is at the order of $n^3/6+(n\%d+1)n^2/2$. For a large value of n, the arithmetic complexities of the two algorithms are parallel.

The above testing results and complexity analysis indicate that the block algorithm makes a better use of the cache to improve the computation efficiency for large matrices. For matrices of small sizes, the block algorithm does not perform better than the standard algorithm. This is because the block algorithm is more complicated than the standard one. It has more control operations, which increase the overhead. In the case of a small matrix, the overhead takes a considerable portion of the execution time, making the blocking algorithm less effective.

On the other hand, comparing with the results of the earlier researches [3,4,5] that show that the block algorithms manifest the advantages starting from a matrix order of several hundreds, our results show that the block algorithm manifests the advantage starting from a matrix order of more than a thousand. This implies that with the increase of the cache memories of today's computers, larger matrices can be more easily computed without blocking.

V. CONCLUSION AND FUTURE WORK

The presented block algorithm of Cholesky factorization makes full use of the caches of today's computers to improve the computing efficiency. A considerable increase of execution speed is achieved in the case of large sizes of matrices. It indicates that the matrix blocking is an effective technique for the computation of dense linear algebra. Further work is considered. As mentioned previously, the computations of the modules in the block algorithm contain many loops. The technique of loop unrolling can be used to reduce the execution time spent on the loop control. The method of the recursive blocking will be tried and investigated to make a close comparison of the two blocking techniques. We shall also make more testing to the algorithms on different machines.

 ISBN: 978-1-61208-283-7

ACKNOWLEDGMENT

Financial support from the National Natural Science Foundation (No. 61171132), the Jiangsu Provincial Natural Science Foundation (No. BK2010280) and the Nantong Municipal Application Research Program (No. BK2011026) of P. R. China are acknowledged. The valuable comments and suggestions of the reviewers are also appreciated.

REFERENCES

[1] S. Toledo, ñLocality of reference in LU decomposition with partial pivoting,ò SIAM Journal of Matrix Analysis & Application, vol. 18, April 1997, pp. 1065-1081.

[2] J. Dongarra, Numerical Linear Algebra for High Performance Computers. Philadelphia: SIAM, 1998.

[3] Y. Li, ñBlock algorithms and their effect in LAPACK,ò Journal of Numerical Methods and Computer Applications, vol. 22, March 2001, pp. 172-180.

[4] B. S. Andersen, J. Wasniewski, and F. G. Gustavson, ñA recursive formulation of Cholesky factorization of a matrix in packed storage,ò ACM Trans. Math. Softw., vol. 2 , June 2001, pp. 214-244.

[5] E. G. Ng and B. W. Peyton, ñBlock sparse Cholesky algorithms on advanced uniprocessor computers,ò SIAM Journal on Scientific Computing, vol. 14, May 1993, pp. 1034-1056.

[6] Y. Li and P. Zhu, ñMethods and techniques for speeding up BLAS,ò Journal of Numerical Methods and Computer Applications, vol. 3, Sep. 1998, pp. 227-240.

[7] R. L. Burden and J. D. Faires. Numerical Analysis (Seventh Edition). Beijing: Higher Education Press, 2001.

[8] E. Elmroth, F. Gustavson, and I. Jonsson, ñRecursive blocked algorithms and hybrid data structures,ò SIAM Review, vol. 46, Jan. 2004, pp. 33-45.

[9] X. Wang and B. Feng, ñImproved algorithm of matrix multiplication based on memory,ò Journal of Northwest Normal University, vol. 41, Jan. 2005, pp. 22-24.

[10] S. Xu, Program Collection for Commonly Used Algorithms (in C++ Language). Beijing: Tsinghua University Press, 2009.

[11] K. Ji, J. Chen, Z. Shi, and W. Liu, ñStudy and implementation of block algorithm for matrix triangular factorization,ò Computer Application and Software, vol. 27, Sep. 2010, pp. 72-74.

 ISBN: 978-1-61208-283-7