A Fast Triangular Matrix Inversion

R.Mahfoudhi

Abstract— **We propose in this paper a recursive algorithm for triangular matrix inversion (TMI) based on the 'Divide and Conquer' (D&C) paradigm. A series of different versions of an original sequential algorithm are first presented. A theoretical performance study permits to establish an accurate comparison between the designed algorithms. Our implementation is designed to be used in place of dtrtri, the level 3 BLAS Triangular Matrix Inversion. Efficient performance could be obtained for large matrix sizes.**

Index Terms— **Divide and conquer, level 3 BLAS, recursive algorithm, triangular matrix inversion.**

I. Introduction

TRIANGULAR matrix inversion (TMI) is a basic kernel in large and intensive scientific applications. Given its cubic complexity, several works addressed the design of efficient parallel algorithms for solving this problem. Apart the standard TMI algorithm consisting in solving n linear triangular systems of size n, n–1,…,1 [1], a recursive algorithm, of same complexity, has been proposed by Heller in 1973 [2]-[3]. Our objective here is the design of a fast sequential algorithms based on Heller's algorithm.
The remainder of the paper is organized as follows. In section 2, we present the divide and conquer paradigm, then we detail a theoretical study on diverse sequential versions of Heller's algorithm in section 3. Finally, we present in section 4 an experimental study.

II. Divide and Conquer

A. Review Stage

There are many paradigms in algorithm design. Backtracking, dynamic programming, and the greedy method to name a few. One compelling type of algorithm is called Divide and Conquer. Algorithms of this type split the problem into subproblems. After the sub-solutions are found they are combined to form the solution of the original problem. When the subproblems are of the same type as the original problem, the same recursive process can be carried out until the problem size is sufficiently small. This special type of D&C is referred to as D&C recursion. The recursive nature of many D&C algorithms makes it easy to express their time complexity as recurrences. Consider a D&C algorithm working on an input size N. It divides its input into subproblems of size N/b. The combining and conquering takes f (N) time. The base-case corresponds to n = 1 and is solved in constant time. The time complexity of this class of algorithms can be expressed as follows:

$$T(N) = O(1) \quad \text{if} \quad n = 1,$$
$$= aT(n/b) + f(n) \quad \text{otherwise.}$$

The master theorem for recurrences can in some instances be used to give a tight asymptotic bound for the complexity [4]:

If a=b : T(n) = O(nlogn)
If a<b and f(n) >0 : T(n) = O(n)
If a<b and f (n) =0 : T(n) = O(logn)
If a>b : T(n) = O ($n\log_b a$)

III. Sequential Recursive TMI Algorithms

We first recall that the well known standard algorithm (SA) for inverting a triangular matrix (an upper or lower triangular matrix), say A of size n, consists in solving n triangular systems. The complexity of (SA) is as follows [1]:

$$SA(n)=n^3/3+n^2/2+n/6 \qquad (1)$$

A. Heller's Recursive Algorithm (HRA)

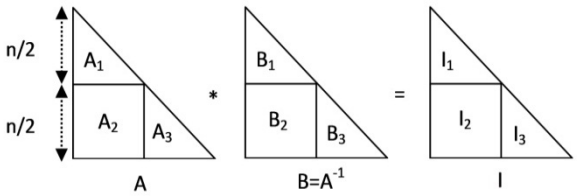


Fig1. Matrix Decomposition in Heller's algorithm

Using the Divide and Conquer paradigm, Heller proposed in 1973 a recursive algorithm [2]-[3] for TMI. The main idea he used consists in decomposing matrix A as well as its inverse B (both of size n) into 3 submatrices of size n/2 (see Figure 1, A being assumed lower triangular). The procedure is recursively repeated until reaching submatrices of size 1. We hence deduce:

$$B_1=A_1^{-1},\ B_3=A_3^{-1},\ B_2=-B_3A_2B_1 \qquad (2)$$

Therefore, inverting matrix A of size n consists in inverting 2 submatrices of size n/2 followed by two matrix products (triangular by dense) of size n/2. In [3] Nasri proposed a slightly modified version of the above algorithm. Indeed, since $B_2=-B_3A_2$ *and* $B_1= -A_3^{-1}A_2A_1^{-1}$, *let*

Manuscript received Mars 23, 2012; revised April 09, 2012. This work is a part of a thesis prepared by the author.

Ryma MAHFOUDHI is with the Computer Sciences Department, University of Tunis El Manar, Faculty of Sciences of Tunis, Campus Universitaire - 2092 Manar II - Tunis, Tunisia , (e-mail: rimahayet@yahoo.fr).

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

$Q = A_3^{-1} A_2$. From (2), we deduce:

$$A_3 Q = A_2 . B_1 A_1 = -Q \qquad (3)$$

Hence, instead of two matrix products needed to compute matrix B2, we have to solve 2 matrix systems of size n/2 i.e. A3Q =A2 and (A1)T(B2)T= –QT. We precise that both versions are of n3/3+O(n2) complexity [3].

Now, for sake of simplicity, we assume that n=2q (q≥1). Let RA-k be the Recursive algorithm designed by recursively applying decomposition until reaching a threshold size n/2k (1≤k≤q. The complexity of RA-k is as follows [3]:

$$\text{RA-k}(n)=n^3/3+n^2/2^{k+1}+n/6 \qquad (4)$$

B. Recursive Algorithm using Matrix Multiplication RAMM

As previously seen, to invert a triangular matrix via block decomposition, one requires two recursive calls and two triangular matrix multiplications (TRMM). The cost is thus RAMM(n) = 2RAMM(n/2) + 2TRMM(n/2). The idea consists in using the Faster Algorithm for TRMM presented below.

ALGORITHM 1

RAMM

Begin

If *(n= 1)* ***then***

$B_1 = 1/A_1$

$B_3 = 1/A_3$

$B_2 = -B_3 * A_2 * B_1$

Else */* splitting matrices into three blocks of sizes n/2*

$B_1 = RAMM(A_1)$

$B_3 = RAMM\ (A_3)$

$C = TRMM(-B_3, A_2)$

$B_2 = TRMM(C, B_1)$

End

- TRMM:

To perform the multiplication of a triangular matrix by a dense matrix via block decomposition in halves, one requires four recursive calls and two dense matrix-matrix multiplications (MM).
The cost is thus TRMM(n) = 4TRMM(n/2)+2MM(n/2).

To optimize this algorithm, we will use a fast Algorithm for dense MM i.e. Strassen algorithm [5]-[6].

- MM:

In [5] the author reported on the development of an efficient and portable implementation of Strassen's MM algorithm. The optimal number of recursive levels depends on the architecture and must be determined experimentally.

ALGORITHM 2

RAMM

Begin
If *(n= 1)* ***then***
$A_{11} * B_{11} = C_{11}$
$A_{11} * B_{12} = C_{12}$
$A_{21} * B_{11} + A_{22} * B_{21} = C_{21}$
$A_{21} * B_{12} + A_{22} * B_{22} = C_{22}$
Else */* splitting matrices into four blocks of sizes n/2*
$C_{11} = TRMM(A_{11}, B_{11}) + MM(A_{12}, B_{21})$
$C_{12} = TRMM(A_{11}, B_{12}) + MM(A_{12}, B_{22})$
$C_{21} = MM(A_{21}, B_{11}) + TRMM(A_{22}, B_{21})$
$C_{22} = MM(A_{21}, B_{12}) + TRMM(A_{22}, B_{22})$
End

C. Recursive Algorithm using Triangular Systems Solving RATSS

In this version, we replace the two matrix products by two triangular systems solving of size n/2. Therefore the algorithm is written as follow:

ALGORITHM 3

RATSS

Begin
If *(n=1)* ***then***
$B_1 = 1/A_1$
$B_3 = 1/A_3$
$Q = A_2/A_3$
$B_2 = -Q/A_1$
Else */* splitting matrices into four blocks of sizes n/2*
$B_1 = RAMM(A_1)$
$B_3 = RAMM(A_3)$
$Q = TSS(A_3, A_2)$
$B_2 = TSS(A_1^T, -Q^T)$
End

- TSS:

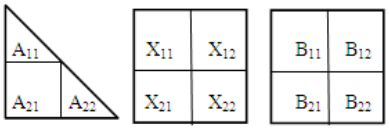
We now discuss the implementation of solvers for triangular systems with matrix right hand side (or equivalently left hand side). This operation is commonly named trsm in the BLAS convention. In the following, we will consider, without loss of generality, the resolution of a lower triangular system with matrix right hand side (AX=B). Our implementation is based on a block recursive algorithm to reduce the computation to matrix multiplications [7]-[8].

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

ALGORITHM 4

TSS

Begin
If *(n=1)* ***then***
$X = B/A$
Else */* splitting matrices into four blocks of sizes n/2*



$X_{11} = TSS(A_{11},B_{11})$
$X_{12} = TSS(A_{11},B_{12})$
$X_{21} = TSS(A_{22}, B_{21}\text{-}MM(A_{21},X_{11}))$
$X_{22} = TSS(A_{22}, B_{22}\text{-}MM(A_{21},X_{12}))$
End

D. Complexity of algorithms

The complexity of the Strassen's Algorithm is $MM(n) = O(n^{\log_2 7})$

The cost function ARMM(n) satisfies the following equation: RAMM(n) = 2RAMM(n/2) + 2TRMM(n/2)

But TRMM(n) = 4TRMM(n/2)+2MM(n/2)

=4TRMM(n/2)+$O(n^{\log_2 7})$= $n^2 + O(n^{\log_2 7})$= $O(n^{\log_2 7})$

So we obtain:

RAMM(n)=2RAMM(n/2)+2TRMM(n/2)

= $n\log(n) + O(n^{\log_2 7})$= $O(n^{\log_2 7})$

By the same method we prove that: TRASS(n)=$O(n^{\log_2 7})$

IV. EXPERIMENT STUDY

This section presents experiments of our implementation of the different versions of triangular matrix inversion described above. We determinate the optimal number of recursive levels for each one (as precised, the optimal number of recursive levels depends on the architecture and must be determined experimentally). The experiments use BLAS library in the last level. We used the g++ compiler under Ubuntu 11.01.

The experiments use BLAS library. We recall that dtrtri refers to the BLAS triangular matrix inversion routine over double precision floating points. We named our routines RAMM, RATSS.

TABLE I
TIMING OF TRIANGULAR MATRIX INVERSION (SECOND)

Matrix Size	dtrtri	RAMM	RATSS	RATSS/dtrtri
256	0.01	0.02	0.01	1
512	0.02	0.03	0.02	1
1024	0.23	0.25	0.20	1.15
2048	2.03	2.08	1.71	1.16
4096	15.54	15.58	13.27	1.17
8192	121.64	127.77	102.90	1.18
16384	978.17	981.35	810.68	1.21

TIMING OF TRIANGULAR MATRIX INVERSION (SECOND)

For increasing matrix dimensions, the RATSS becomes even more efficient (improvement factor between 15% and 21%).

We can notice that dtrtri is quite efficient compare to RAMM.

V. CONCLUSION

We have achieved the goal of outperforming the efficiency of the well known BLAS and LAPACK libraries for triangular matrix inversion. We showed notice that our algorithm benefit from Strassen matrix multiplication algorithm, recursive solvers for triangular systems and the use of Blas routines in the last level. This performance is achieved through efficient reduction to matrix multiplication where we took care of minimizing the ratio and also by reusing the numerical computation as much as possible.

REFERENCES

[1] Quarteroni, A., Sacco, R., Saleri, F. : Méthodes numériques. Algorithmes, analyse et applications, Springer, Milano, 2007.
[2] Heller, D.: A survey of parallel algorithms in numerical linear algebra, SIAM Review 20, pp. 740–777, 1978.
[3] Nasri, W., Mahjoub, Z.: Design and implementation of a general parallel divide and Conquer algorithm for triangular matrix inversion, International Journal of Parallel and Distributed Systems and Networks 5(1), pp. 35–42, 2002.
[4] Time Complexity and the divide and conquer strategy or: how to measure algorithm run-time And: design efficient algorithms, course, Télécom 2A – Algo Complexity, Oct. 2005.
[5] StevenH., Elaine M., Jeremy R., Anna T. and Thomas T., Implementation of Strassen's Algorithm for Matrix Multiplication, IEEE, 1996.
[6] Strassen, V.: Gaussian elimination is not optimal. Numer. Math., 13, 354–356, 1969.
[7] Andersen, B. S., Gustavson, F., Karaivanov, A., Wasniewski, J., and Yalamov, P. Y.: LAWRA - LINEAR ALGEBRA WITH RECURSIVE ALGORITHMS, Lecture Notes in Computer Science, Volume 1823/2000, 629-632, 2000.
[8] Dumas, J.G., Pernet, C. and Roch, J.L.: Adaptive triangular system solving, Challenges in Symbolic Computation Software 2006.

ISBN: 978-988-19251-3-8
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)