On Fast Algorithms for Triangular and Dense Matrix Inversion

Ryma Mahfoudhi and Zaher Mahjoub

Abstract We first propose in this paper a recursive algorithm for triangular matrix inversion (TMI) based on the 'Divide and Conquer' (D&C) paradigm. Different versions of an original sequential algorithm are 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 TMI. Afterwards, we generalize our approach for dense matrix inversion (DMI) based on LU factorization (LUF). This latter is used in Mathematical software libraries such as LAPACK xGETRI and MATLAB inv. $A = LU$ being the input dense matrix, xGETRI consists, once the factors L and U are known, in inverting U then solving the triangular matrix system $XL = U^{-1}$ (i.e. $L^TX^T = (U^{-1})^T$, thus $X = A^{-1}$). Two other alternatives may be derived here (L and U being known) : (i) first invert L, then solve the matrix system $UX = L^{-1}$ for X ; (ii) invert both L and U, then compute the product $X = U^{-1}L^{-1}$. Each of these three procedures involves at least one triangular matrix inversion (TMI). Our DMI implementation aims to be used in place of the level 3 BLAS TMI-DMI. Efficient results could be obtained through an experimental study achieved on a set of large sized randomly generated matrices.

Keywords Dense matrix inversion · Divide and conquer · Level 3 BLAS · LU factorization · Recursive algorithm · Triangular matrix inversion

R. Mahfoudhi (✉) · Z. Mahjoub
Faculty of Sciences of Tunis, University of Tunis El Manar,
University Campus - 2092 Manar II, Tunis, Tunisia
e-mail: rimahayet@yahoo.fr

Z. Mahjoub
e-mail: zaher.mahjoub@fst.rnu.tn

1 Introduction

Triangular matrix inversion (TMI) is a basic kernel used in many scientific applications. Given its cubic complexity in terms of the matrix size, say n, several works addressed the design of practical efficient algorithms for solving this problem. Apart the standard TMI algorithm consisting in solving n linear triangular systems of size n, $n-1, \ldots 1$ [1], a recursive algorithm, of same complexity, has been proposed by Heller in 1973 [2–4]. It uses the 'Divide and Conquer' (D&C) paradigm and consists in successive decompositions of the original matrix. Our objective here is two-fold i.e. (i) design an efficient algorithm for TMI that outperforms the BLAS routines and (ii) use our TMI kernel for dense matrix inversion (DMI) through LU factorization, thus deriving an efficient DMI kernel.

The remainder of the paper is organized as follows. In Sect. 2, we present the D&C paradigm. We then detail in Sect. 3 a theoretical study on diverse versions of Heller's TMI algorithm. Section 4 is devoted to the generalization of the former designed algorithm for DMI. An experimental study validating our theoretical contribution is presented in Sect. 5.

2 Divide and Conquer Paradigm

There are many paradigms in algorithm design. Backtracking, Dynamic programming, and the Greedy method to name a few. One compelling type of algorithms is called Divide and Conquer (D&C). Algorithms of this type split the original problem to be solved into (equal sized) sub-problems. Once the sub-solutions are determined, 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 sub-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 **a** (called arity) sub-problems of size n/b. Combining and conquering are assumed to take 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:

$$\begin{aligned} T(n) &= O(1) && \text{if } n = 1 \\ &= aT(n/b) + f(n) && \text{otherwise.} \end{aligned}$$

Let $f(n) = O(n^{\delta})(\delta \geq 0)$, the master theorem for recurrences can in some instances be used to give a tight asymptotic bound for the complexity [1]:

- $a < b^{\delta} \Rightarrow T(n) = O(n^{\delta})$
- $a = b^{\delta} \Rightarrow T(n) = O(n^{\delta} \log_b n)$

- $a > b^{\delta} \Rightarrow T(n) = O(n^{\log_b a})$

3 Recursive TMI Algorithms

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

$$\mathrm{SA(n)} = \mathrm{n}^3/3 + \mathrm{n}^2/2 + \mathrm{n}/6 \tag{1}$$

3.1 Heller's Recursive Algorithm (HRA)

Using the D&C 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 Fig. 1, A being assumed lower triangular). The procedure is recursively repeated until reaching submatrices of size 1. We hence deduce:

$$B_1 = A_1^{-1}, B_3 = A_3^{-1}, B_2 = -B_3 A_2 B_1 \tag{2}$$

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 $B_2 = -B_3 A_2$ and $B_1 = -A_3^{-1} A_2 A_1^{-1}$, let $Q = A_3^{-1} A_2$. From (2), we deduce:

$$A_3 Q = A_2, B_2 A_1 = -Q \tag{3}$$

Hence, instead of two matrix products needed to compute matrix $\mathrm{B_2}$, we have to solve 2 matrix systems of size n/2 i.e. $\mathrm{A_3Q = A_2}$ and $\mathrm{(A_1)^T(B_2)^T = -Q^T}$. We precise that both versions are of $\mathrm{n^3/3 + O(n^2)}$ complexity [3].

Now, for sake of simplicity, we assume that $\mathrm{n = 2^q (q \geq 1)}$. Let RA-k be the Recursive Algorithm designed by recursively applying the decomposition k times i.e. until reaching a threshold size $\mathrm{n/2^k (1 \leq k \leq q)}$. The complexity of RA-k is as

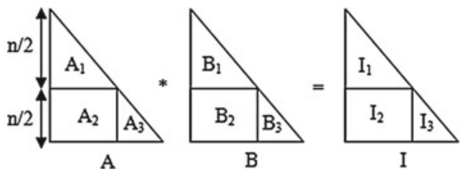


Fig. 1 Matrix decomposition in Heller's algorithm

follows [3]:

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

3.2 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) [5]. Thus, the complexity recurrence formula is:

$$\mathrm{RAMM(n)} = 2\mathrm{RAMM(n/2)} + 2\mathrm{TRMM(n/2)} + \mathrm{O(n^2)}$$

The idea consists in using the fast 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
        $B_1 = RAMM(A_1)$
        $B_3 = RAMM(A_3)$
        $C = TRMM(-B_3,A_2)$
        $B_2 = TRMM(C,B_1)$
    Endif
End
```

- **TRMM algorithm**

To perform the multiplication of a triangular (resp. dense) by a dense (resp. triangular) via block decomposition in halves, we require four recursive calls and two dense matrix-matrix multiplications (MM) Fig. 2.

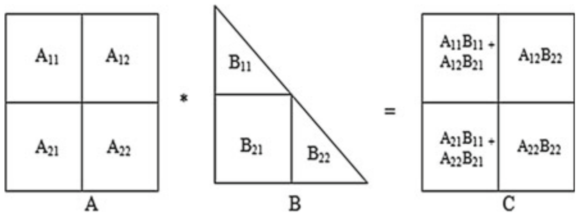


Fig. 2 Matrix decomposition in TRMM algorithm

ALGORITHM 2

TRMM

```
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
      $C_{11} = TRMM(A_{11},B_{11})$
      $C_{12} = TRMM(A_{11},B_{12})$
      $C_{21} = MM(A_{21},B_{11}) + TRMM(A_{22},B_{21})$
      $C_{22} = MM(A_{21},B_{12}) + TRMM(A_{22},B_{22})$
  Endif
End
```

The complexity recurrence formula is thus :

$$\mathrm{TRMM(n) = 4TRMM(n/2) + 2MM(n/2) + O(n^2)}.$$

To optimize this algorithm, we will use a fast algorithm for dense MM i.e. Strassen algorithm.

- **MM algorithm**

In [6, 7], the author reported on the development of an efficient and portable implementation of Strassen MM algorithm. Notice that the optimal number of recursive levels depends on both the matrix size and the target architecture and must be determined experimentally.

3.3 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 (see Sect. 3.1). The algorithm is as follows:

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
      $B_1 = RAMM(A_1)$
      $B_3 = RAMM(A_3)$
      $Q = TSS(A_3, A_2)$
      $B_2 = TSS(A_1^T, -Q^T)$
  Endif
End
```

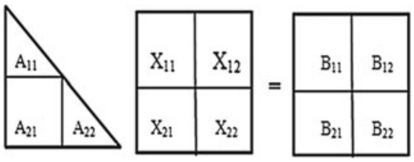
• TSS algorithm

We now discuss the implementation of solvers for triangular systems with matrix right hand side (or equivalently left hand side). This kernel 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 in order to reduce the computations to matrix multiplications [8, 9].

ALGORITHM 4

TSS

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



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

3.4 Algorithms Complexity

As well known, the complexity of the Strassen's Algorithm is $\mathrm{MM(n)} = \mathrm{O(n^{\log_2 7})}$
Besides, the cost RAMM(n) satisfies the following recurrence formula:

$$\mathrm{RAMM(n)} = \mathrm{2RAMM(n/2)} + \mathrm{2TRMM(n/2)} + \mathrm{O(n^2)}.$$

Since

$$\begin{aligned}\mathrm{TRMM(n)} &= \mathrm{4TRMM(n/2)} + \mathrm{2MM(n/2)} + \mathrm{O(n^2)}\\ &= \mathrm{4TRMM(n/2)} + O(n^{log_2 7}) + \mathrm{O(n^2)}\\ &= n^2 + O(n^{log_2 7}) + \mathrm{O(n^2)} = O(n^{log_2 7})\end{aligned}$$

We therefore get :

$$\begin{aligned}\mathrm{RAMM(n)} &= \mathrm{2RAMM(n/2)} + \mathrm{2TRMM(n/2)} + \mathrm{O(n^2)}\\ &= nlog(n) + O(n^{log_2 7}) + \mathrm{O(n^2)} = O(n^{log_2 7})\end{aligned}$$

Following a similar way, we prove that $\mathrm{TRMM(n)} = O(n^{log_2 7})$

4 Dense Matrix Inversion

4.1 LU Factorization

As previously mentioned, three alternative methods may be used to perform a DMI through LU factorization (LUF). The first one requires two triangular matrix inversions (TMI) and one triangular matrix multiplication (TMM) i.e. an upper one by a lower one. The two others both require one triangular matrix inversion (TMI) and a triangular matrix system solving (TSS) with matrix right hand side or equivalently left hand side (Algorithm 4). Our aim is to optimize both LUF, TMI as well as TMM kernels [10].

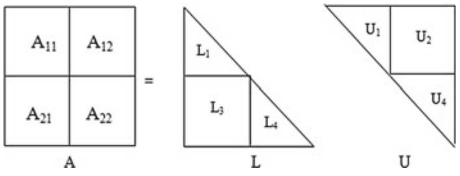
4.2 Recursive LU Factorisation

To reduce the complexity of LU factorization, blocked algorithms were proposed in 1974 [11]. For a given matrix A of size n, the L and U factors verifying A=LU may be computed as follows:

ALGORITHM 5

LUF

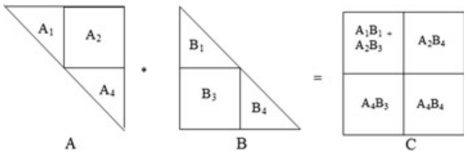
```
Begin
   If (n=1) Then
      L=1;  U=A
   Else /* split matrices into four blocks of sizes n/2
```



```
         $(L_1, [U_1, U_2]) = LUF([A_{11}\ A_{12}])$
         $L_3 = A_{21} U_1^{-1}$
         $H = A_{22} - L_3 U_2$
         $(L_4, U_4) = LUF(H)$
   Endif
End
```

4.3 Triangular Matrix Multiplication (TMM)

Block wise multiplication of an upper triangular matrix by a lower one, can be depicted as follows:



Thus, to compute the dense matrix C = AB of size n, we need:

- Two triangular matrix multiplication (an upper one by a lower one) of size n/2
- Two multiplications of a triangular matrix by a dense one (TRMM) of size n/2.
- Two dense matrix multiplication (MM) of size n/2.

ALGORITHM 6

TMM

```
Begin
   If (n=1) Then
      C = A*B
   Else /* split matrices into four blocks of sizes n/2
      $C_1$ = TMM($A_1$,$B_1$)+MM($A_2$,$B_3$)
      $C_2$ = TRMM($B_4$, $A_2$)
      $C_3$ = TRMM($A_4$,$B_3$)
      $C_4$ = TMM($A_4$,$B_4$)
   Endif
End
```

Clearly, if any matrix-matrix multiplication algorithm with $O(n^{\log_2 7})$ complexity is used, then the algorithms previously presented both have the same $O(n^{\log_2 7})$ complexity instead of $O(n^3)$ for the standard algorithms.

5 Experimental Study

5.1 TMI Algorithm

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 already precised, the optimal number of recursive levels depends on the matrix size and the target architecture and must be determined experimentally). The experiments (as well as the following on DMI) use BLAS library in the last level and were achieved on a 3 GHz, 4Go RAM PC. We used the g++ compiler under Ubuntu 11.01.

We recall that **dtrtri** refers to the BLAS triangular matrix inversion routine with double precision floating points. We named our routines RAMM, RATSS, see fig. 3.

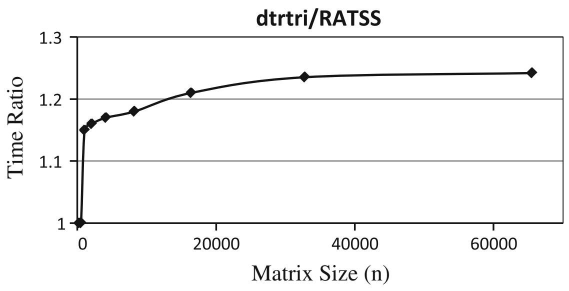


Fig. 3 Time ratio dtrtri/RATSS

Table 1 Timing of triangular matrix inversion (seconds)

Matrix size	dtrtri	RAMM	RATSS	Time ratio dtrtri/RATSS
256	0.01	0.02	0.01	1
512	0.02	0.03	0.02	1
1024	0.23	0.25	0.2	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	102.9	1.18
16384	978.17	981.35	810.68	1.21
32768	7902.14	7927.85	6396.87	1.23
65536	64026.02	64296.97	51548.52	1.24

We notice that for increasing matrix sizes, RATSS becomes even more efficient than **dtrtri** (improvement factor between 15 and 24 %). On the other hand, **dtrtri** is better than RAMM, see table 1.

5.2 *DMI Algorithm*

Table 2 provides a comparison between LU factorization-based algorithms i.e. MILU_1 (one TMI and one triangular matrix system solving), MILU_2 (two TMIs and one TMM), and the BLAS routine where the routine **dgetri** was used in combination with the factorization routine **dgetrf** to obtain the matrix inverse (see Fig. 4).

We remark that the time ratio increases with the matrix size i.e. MILU_1 and MILU_2 become more and more efficient than BLAS (the speed-up i.e. time ratio reaches 4.4 and more).

Table 2 Timing of dense matrix inversion (seconds)

Matrix size	BLAS	MILU_1	MILU_2	Time ratio $\frac{\text{BLAS}}{\text{MILU\_1}}$	Time ratio $\frac{\text{BLAS}}{\text{MILU\_2}}$
256	0.06	0.06	0.06	1.02	1.03
512	0.12	0.07	0.08	1.63	1.56
1024	1.46	0.65	0.73	2.24	1.99
2048	12.00	2.91	3.40	4.12	3.53
4096	96.01	21.97	24.06	4.37	3.99
8192	764.35	174.51	191.09	4.38	4.00
16384	5922.38	1349.06	1473.23	4.39	4.02
32768	50276.71	11400.61	12322.72	4.41	4.08
65536	401295.45	90585.88	97638.80	4.43	4.11

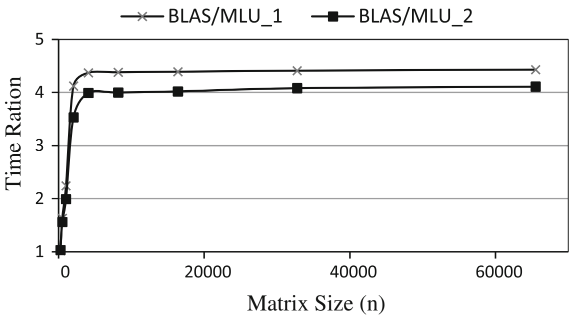


Fig. 4 Time ratio: BLAS/MLU_1 and BLAS/MLU_2

6 Conlusion and Future Work

In this paper we targeted and reached the goal of outperforming the efficiency of the well-known BLAS library for triangular and dense matrix inversion. It has to be noticed that our (recursive) algorithms essentially benefit from both (recursive) Strassen matrix multiplication algorithm, recursive solvers for triangular systems and the use of BLAS routines in the last recursion level. This performance was achieved thanks to (i) efficient reduction to matrix multiplication where we optimized the number of recursive decomposition levels and (ii) reusing numerical computing libraries as much as possible.

These results we obtained lead us to precise some attracting perspectives we intend to study in the future. We may particularly cite the following points.

- Achieve an experimental study on matrix of larger sizes.
- Study the numerical stability of these algorithms
- Generalize our approach to other linear algebra kernels

References

1. Quarteroni A, Sacco R, Saleri F (2007) Méthodes numériques. Algorithmes, analyse et applications, Springer, Milan
2. Heller D (1978) A survey of parallel algorithms in numerical linear algebra. SIAM Rev 20:740–777
3. Nasri W, Mahjoub Z (2002) Design and implementation of a general parallel divide and conquer algorithm for triangular matrix inversion. Int J Parallel Distrib Syst Netw 5(1):35–42
4. Aho AV, Hopcroft JE, Ullman JD (1975) The design and analysis of computer algorithms. Addison-Wesley, Reading
5. Mahfoudhi R (2012) A fast triangular matrix inversion. Lecture notes in engineering and computer science: Proceedings of the world congress on engineering 2012, WCE 2012, London, UK, 4–6 July 2012, pp 100–102
6. Steven H, Elaine M, Jeremy R, Anna T, Thomas T (1996) Implementation of Strassen's algorithm for matrix multiplication. In: Supercomputing '96 proceedings ACM/IEEE conference on supercomputing (CDROM)

7. Strassen V (1969) Gaussian elimination is not optimal. Numer Math 13:354–356
8. Andersen BS, Gustavson F, Karaivanov A, Wasniewski J, Yalamov PY (2000) LAWRA—Linear algebra with recursive algorithms. Lecture notes in computer science, vol 1823/2000, pp 629–632
9. Dumas JG, Pernet C, Roch JL (2006) Adaptive triangular system solving. In: Proceedings of the challenges in symbolic computation software
10. Mahfoudhi R, Mahjoub Z (2012) A fast recursive blocked algorithm for dense matrix inversion. In: Proceedings of the 12th international conference on computational and mathematical methods in science and engineering, cmmse 2012, La Manga, Spain
11. Aho AV, Hopcroft JE, Ullman JD (1974) The design and analysis of computer algorithms. Addison-Wesley, Reading