International Conference on Control, Engineering & Information Technology (CEIT'13)
Proceedings Engineering & Technology - Vol.2, pp. 136-139, 2013

On Fast Algorithms for Matrix System Solving

Ryma Mahfoudhi[1] and Zaher Mahjoub[2]

Faculty of Sciences of Tunis, University of Tunis El Manar,
University Campus - 2092 Manar II, Tunis, Tunisia
[1]rimahayet@yahoo.fr
[2]zaher.mahjoub@fst.rnu.tn

Abstract— **We design and analyse in this paper two recursive blocked algorithms, based on the 'Divide and Conquer' paradigm, for matrix system solving (MSS) i.e. AX=B where A, X and B are dense square matrices of size n, A and B being known whereas X is unknown. The theoretical analysis leads to algorithms of $O(n^{\log_2 7})$ complexity. An experimental study achieved on both the two algorithms and the level 3 BLAS matrix system solving based on LU factorization kernel permits to evaluate the practical interest of our contribution.**
Keywords— **BLAS, Decomposition, Dense/Triangular matrix system Divide & conquer, LU factorization, Recursive algorithm.**

I. Introduction & Recall

Linear system of equations resolution 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 Gaussian elimination (GE) algorithm, another algorithm, namely LU factorization, (LUF) of same complexity, is frequently used. It consists, in a first phase, in factorizing the input matrix, say A, into a product of a lower (L) triangular matrix and an upper (U) triangular one i.e. A=LU. Afterwards, if Ax=b is the input system, we have to successively solve, in a second phase, the two triangular systems Ly=b and Ux=y. We recall that phase 1 costs $2n^2/3+O(n^2)$ and phase 2 costs $2n^2+O(n)$, thus an overall $2n^2/3+O(n^2)$ complexity [1]. Now, consider the matrix system (MS) : AX=B where A, X and B are three dense square matrices of size n, A and B being known whereas X is unknown. Clearly, a straightforward approach for solving such a matrix system (MS) consists in solving n classical systems of size n. Obviously, this standard algorithm (SA) has a complexity $SA(n)=8n^3/3+ O(n^2)$ since we need only one factorization followed by solving n couples of triangular systems.

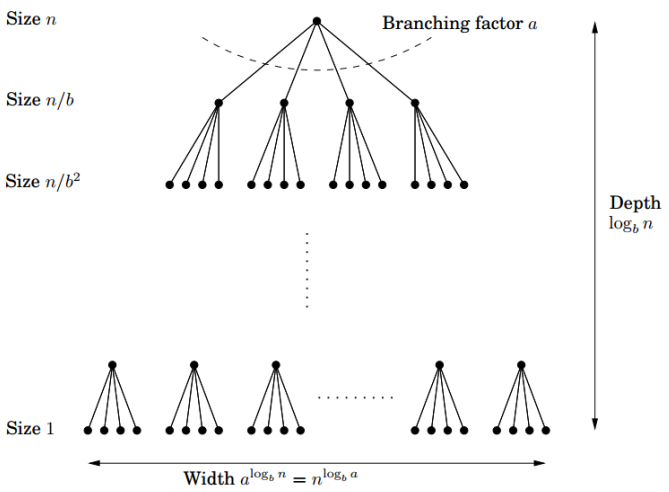
Our objective here is to propose and alternative approach of better complexity for solving AX=B based on the "Divide and conquer" paradigm that outperforms the BLAS routines. It has to be underlined that this study comes within the framework of a generic approach for efficiently solving a generic class of matrix problems already started in previous papers of ours [2,3].

II. Divide And Conquer Paradigm

The 'Divide and Conquer' (D&C) paradigm is widely used to design efficient algorithms for scientific and engineering applications. Algorithms of this type, based on multi-branched recursion, split the original problem into subproblems of (in general) same size [4]. Once the sub-solutions are found, they are combined in order to build 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 subproblem 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 [5].

Consider a D&C algorithm for solving a problem of size **n**. A first decomposition leads to **a** subproblems of same size say **n/b.** Combining and conquering is assumed to take an f(n) time. The base-case corresponds to n=1 (or n_e for an elementary size) and is solved in constant time. Therefore, the time complexity T(n) of such D&C algorithm can be expressed as follows:

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



The master theorem [5] leads, when $f(n)=O(n^{\delta})$ for some constants $a > 0$, $b > 1$, and $\delta \geq 0$ to the following:

$$T(n) = \begin{cases} O\left(n^{\delta}\right) & \text{if } a \prec b^{\delta} \\ O\left(n^{\delta} \log_b n\right) & \text{if } a = b^{\delta} \\ O\left(n^{\log_b a}\right) & \text{if } a \succ b^{\delta} \end{cases}$$

III. Recursive Matrix System Solving Algorithms

A. Recursive Algorithm Using LU Factorization (RLU)

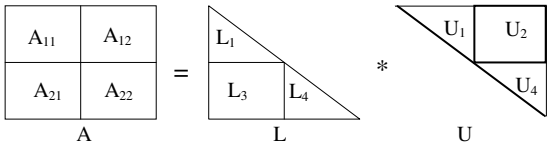
As previously mentioned, solving the MS: AX=B by LU factorization requires one LU factorization (LUF) i.e. A=LU, then solving two triangular matrix systems (TMSS) : LY=C and UX=Y i.e. 2n classical triangular systems of size n. Our aim is to optimize (through the D&C paradigm) both LUF and TMSS kernels in order to obtain a fast algorithm for solving the MS.

1) LU Factorization (LUF)

LU factorization (LUF) is an important numerical algorithm for solving systems of linear equations encountered in Science and Engineering and is characteristic of many dense linear algebra computations. It refers to the factorization of a square matrix into two factors, a lower triangular matrix and an upper one.

To reduce the complexity of LUF, blocked algorithms have been proposed since 1974 [6]. For a given matrix A of size n, the L and U factors verifying A=LU may be computed as follows:

Matrix decomposition:



Formula:

(1) $L_1U_1 = A_{11}$ (3) $L_3U_1 = A_{21}$
(2) $L_1U_2 = A_{12}$ (4) $L_3U_2 + L_4U_4 = A_{22}$

We hence remark that the LUF of matrix A of size n requires :

- One LUF of size n/2 i.e. (1) : $L_1U_1 = A_{11}$ giving L_1 and U_1
- Solving 2 (lower) triangular matrix systems (TMSS) i.e. (2) : $L_1U_2 = A_{12}$ giving U_2 and $(3)^T$: ${U_1}^T{L_3}^T = {A_{21}}^T$ giving L_3
- One matrix multiplication (MM) i.e. L_3U_2
- One LUF of size n/2 i.e. (4) : $L_4U_4 = A_{22} - L_3U_2$ giving L_4 and U_4.

Therefore, the complexity recurrence formula is as follows :
$LUF(n) = 2LUF(n/2) + 2TMSS(n/2) + 1MM(n/2) + O(n^2)$

Remark that $O(n^2)$ is required by matrix addition.

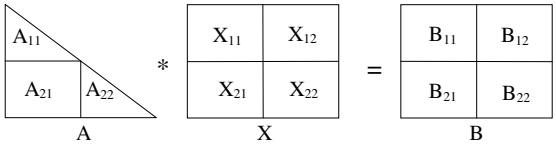
Algorithm

Begin
  If *(n=1)* ***Then*** *L=1;U=A*
  Else */* split matrices into four blocks of sizes n/2*
    $(L_1, [U_1, U_2]) = LUF([A_{11}\ A_{12}])$
    ${}^tU_1\ {}^tL_3 = A_{21}$
    $H = A_{22} - L_3U_2$
    $(L_4, U_4) = LUF(H)$
  Endif
End

2) Triangular Matrix System Solving (TMSS)

We now discuss the design of solvers for triangular matrix system AX=B with matrix right hand side AX=B (or equivalently left hand side XA=B) where A (a triangular matrix) and B (a dense matrix) are known. 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 matrix system with matrix right hand side (AX=B). Our approach is based on a block recursive algorithm in order to reduce the computation to matrix multiplication (MM) [2,3].
To optimize this algorithm, we will use a fast algorithm for dense MM i.e. Strassen algorithm.

Matrix decomposition:



Formula:

(1) $A_{11}X_{11} = B_{11}$ (3) $A_{21}X_{11} + A_{22}X_{21} = B_{21}$
(2) $A_{11}X_{12} = B_{12}$ (4) $A_{21}X_{12} + A_{22}X_{22} = B_{22}$

Hence, solving a TMSS of size n requires 4 TMSS of size n/2 and 2 MM of size n/2. Thus, the resulting complexity recurrence formula is :
$TMSS(n) = 4TMSS(n/2) + 2MM(n/2) + O(n^2)$.

Algorithm

Begin
  If *(n=1)* ***Then***
    $X = B/A$
  Else */* split matrices into four blocks of sizes n/2*
    $X_{11} = TMSS(A_{11}, B_{11})$
    $X_{12} = TMSS(A_{11}, B_{12})$
    $X_{21} = TMSS(A_{22}, B_{21} - MM(A_{21}, X_{11}))$
    $X_{22} = TMSS(A_{22}, B_{22} - MM(A_{21}, X_{12}))$
  Endif
End

3) Complexity evaluation

Since RLU requires one LUF and two TMSSs, we get:
RLU(n)= LUF(n)+2TMSS(n) + $O(n^2)$.
Besides, we have:

TMSS(n)= 4TMSS(n/2) + 2MM(n/2) + $O(n^2)$.

Using a fast Algorithm for dense MM i.e. Strassen algorithm whose complexity is $O(n^{\log_2 7})$ [7], we obtain :
TMSS(n) = 4TMSS(n/2) + $O(n^{\log_2 7})$ = $O(n^{\log_2 7})$

Consequently,
LUF(n) = 2LUF(n/2) + 2TMSS(n/2) + 1MM (n/2) + $O(n^2)$
= 2LUF(n/2) + $O(n^{\log_2 7})$ = $O(n^{\log_2 7})$ =>
Thus RLU(n)= LUF(n)+2 TMSS(n)= $O(n^{\log_2 7})$

B. Recursive Algorithm using Blocked decomposition (RB)

We introduce now another algorithm for solving the MS: AX=B. The main idea consists in decomposing both matrices A, X and B into 4 submatrices of size n/2 as follows.

Matrix decomposition:

A_{11}	A_{12}
A_{21}	A_{22}

*

X_{11}	X_{12}
X_{21}	X_{22}

=

B_{11}	B_{12}
B_{21}	B_{22}

Formula:

(1) $A_{11}X_{11} + A_{12}X_{21} = B_{11} \rightarrow X_{11} = A_{11}^{-1}(B_{11} - A_{12}X_{21})$
(2) $A_{11}X_{12} + A_{12}X_{22} = B_{12} \rightarrow X_{12} = A_{11}^{-1}(B_{12} - A_{12}X_{22})$
(3) $A_{21}X_{11} + A_{22}X_{21} = B_{21}$
$\rightarrow A_{21}A_{11}^{-1}(B_{11} - A_{12}X_{21}) + A_{22}X_{21} = B_{21}$
(4) $A_{21}X_{12} + A_{22}X_{22} = B_{22}$
$\rightarrow A_{21}A_{11}^{-1}(B_{12} - A_{12}X_{21}) + A_{22}X_{22} = B_{22}$

To ensure that the complexity of RB algorithm does not exceed that of the standard algorithm (SA) i.e. $8n^3/3 + O(n^2)$, we must choose the most suitable kernels.

From (3), we deduce:

$(A_{22} - A_{21}A_{11}^{-1}A_{12})X_{21} = B_{21} - A_{21}A_{11}^{-1}B_{11}$

Let $D = A_{21}A_{11}^{-1} \rightarrow A_{21} = DA_{11} \rightarrow {}^tA_{11}\,{}^tD = {}^tA_{21}$
$E = DA_{12}$, $F = DB_{11}$

We get :

(3)→ $(A_{22} - E)X_{21} = B_{21} - F$
(4)→ $(A_{22} - E)X_{22} = B_{22} - DB_{12}$
(1)→ $A_{11}X_{11} = B_{11} - A_{12}X_{21}$
(2)→ $A_{11}X_{12} = B_{12} - A_{12}X_{22}$

Remark that we have to solve 5 dense matrix system i.e. ${}^tA_{11}\,{}^tD = {}^tA_{21}$ and two couples of dense matrix systems i.e. ((3),(4)) and ((1),(2)) where each couple depends on the same matrix. Remark in addition that the second couple depends on the transposed matrix of the first system. Thus we'll use RLU algorithm for each in order to save computations by factorizing each matrix only once.
Hence, to solve the (dense) matrix system AX=B of size n, we need:

- (a) One dense matrix system solving of size n/2 i.e. ${}^tA_{11}\,{}^tD = {}^tA_{21}$ giving D (by RLU algorithm)
- (b) Two matrix multiplications (MM) of size n/2 i.e. $E = DA_{12}$ and $FD = B_{11}$
- (c) One dense matrix system solving of size n/2 i.e. (3) $(A_{22} - E)X_{21} = B_{21} - F$ giving X_{21}(by RLU algorithm)
- (e) Solving 2 triangular matrix systems (TMSS) of size n/2 i.e. (4) $(A_{22} - E)X_{22} = B_{22} - DB_{12}$ giving X_{22}
- (f) Solving 2 triangular matrix systems (TMSS) of size n/2 i.e. (1) $A_{11}X_{11} = B_{11} - A_{12}X_{21}$ giving X_{11}
- (g) Solving 2 triangular matrix systems (TMSS) of size n/2 i.e. (2) $A_{11}X_{12} = B_{12} - A_{12}X_{22}$ giving X_{12}
  Remark that for (f) and (g) we use the LUF already done for solving (a) by RLU algorithm.

So : RB(n) = 2RB(n/2)+ 6TMSS(n/2)+5MM(n/2) + $O(n^2)$
= 2RB(n/2) + $O(n^{\log_2 7})$ = $O(n^{\log_2 7})$

Clearly, if any MM algorithm of $O(n^{\log_2 7})$ 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 corresponding standard algorithms.

IV. EXPERIMENTAL STUDY

This section presents experiments of our implementation of the different versions of dense matrix system solving described above. We have to mention the importance of the determination, for each algorithm used, of the optimal number of recursive levels (nrl) i.e. the one leading to the best execution time. Indeed, the optimal nrl depends on both the matrix size and the target machine architecture and has to be determined experimentally. It is well known that the execution time decreases for increasing nrl until a precise threshold, then increases [8].

Our experiments used BLAS library [9] in the last recursion level and were achieved on two target machines i.e. TM1 (clock 3 GHz, 4Go RAM, 3Mo cache memory) and TM2 (clock 2.5 GHz, 2Go RAM, 3Mo cache memory). We used the g++ compiler under Ubuntu 11.01. All execution times are the means of several runs.

We discuss in this section the variations of the execution time in terms of the matrix size n. For this purpose, n was chosen in the range [512 32768] and the input matrices involving real floating point elements were randomly generated. For sake of simplicity and without loss of generality, we chosed n as a power of 2. WE recall that when this is not the case, there are techniques known in the literature proposing efficient strategies (e.g. padding, dynamic peeling) leading to the power-of-2 case without increasing the complexity order [10].

We named our routines RLU and RB, and the BLAS routine where the routine **dtrsm** was used in combination with the factorization routine **dgetrf** to solve dense systems. We precise that we denote by 'Speed-up alg1/alg2' the ratio execution time of alg1 on execution time of alg2.

TABLE I
EXECUTION TIMES - DENSE MATRIX SYSTEM SOLVING (SECONDS) – TM1

n	BLAS	RLU	RB	Speed-up BLAS/RLU	Speed-up BLAS/RB	Speed-up RLU/RB
512	0.14	0.20	0.14	0.71	1.00	1.43
1024	1.16	1.19	1.01	0.97	1.15	1.18
2048	9.31	9.04	8.03	1.03	1.16	1.13
4096	75.82	71.53	64.80	1.06	1.17	1.10
8192	630.88	595.17	534.65	1.06	1.18	1.11
16384	5406.73	5006.24	4360.27	1.08	1.24	1.15
32768	47633.38	42912.95	37506.60	1.11	1.27	1.14

TABLE II
EXECUTION TIMES – DENSE MATRIX SYSTEM SOLVING (SECONDS) – TM2

n	BLAS	RLU	RB	Speed-up BLAS/RLU	Speed-up BLAS/RB	Speed-up RLU/RB
512	0.16	0.23	0.17	0.69	0.94	1.35
1024	1.45	1.59	1.48	0.91	0.98	1.07
2048	11.86	11.74	10.59	1.01	1.12	1.11
4096	96.03	88.92	84.24	1.08	1.14	1.06
8192	800.12	727.38	683.86	1.10	1.17	1.06
16384	6488.08	5792.93	5406.73	1.12	1.20	1.07
32768	67261.48	59523.43	54684.13	1.13	1.23	1.09

From the above results, we can remark the following:

- With TM1 which has both a higher clock frequency and a RAM capacity than TM2 (i.e. 1.2 times and 2 times more), the execution times are always lower i.e. up to 1.5 times better.
- For both TM1 and TM2 and for any n, RB is always faster than RLU. The corresponding speed-up RLU/RB seems to stabilize for large sizes and reaches better amounts with TM1 than with TM2.
- For TM1 and TM2, both RLU and RB are better than BLAS from n=2048 and on. The corresponding speed-ups increase with n. Indeed, for n=32768, RLU is 11% better than BLAS with TM1 and 13% better with TM2. As to RB and for the same n, it is 27% better than BLAS with TM1 and 23% better with TM2.

We have to add that the recursion is terminated when the size of the size of remaining subproblems to be solved is smaller than the machine block size, which is the only architecture-dependent parameter in our algorithms. We precise that for TM1, the block size is 512 whereas for TM2, it is 256.

V. CONCLUSION AND FUTURE WORK

The two fast recursive algorithms for matrix system solving we designed has been proven enough satisfactory in practice and could outperform some BLAS routines. These performances are tightly related to the target machine and the optimal number of recursion levels. Indeed, this occurs at a threshold reached when the remaining subproblems to be solved are smaller than the memory machine block size. Pursuing recursion until a lower size would in general cause too much overhead and a drop in the overall performance. In this paper we targeted and reached the goal of outperforming the efficiency of the well-known BLAS library for dense matrix system solving. 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, particularly 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.

The 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 large scale matrix systems in order to better evaluate the practical behaviours of our algorithms.
- Study the numerical stability of the designed algorithms since recursive matrix algorithms are known to be (in general) of lower stability than iterative ones [11,12].
- Generalize our approach to other linear algebra kernels such as rectangular matrix system solving.

REFERENCES

[1] P. Lascaux and R. Théodor, "Analyse numérique matricielle appliquée à l'art de l'ingénieur, Tome 1, " *Dunod*, Paris, 2000.

[2] R.Mahfoudhi, and Z. Mahjoub, "A fast recursive blocked algorithm for dense matrix inversion, " *Proceedings of the 12th International Conference on Computational and Mathematical Methods in Science and Engineering*, CMMSE 2012, La Manga, Spain, 2012.

[3] R. Mahfoudhi, "A fast triangular matrix inversion," *Proceedings of the 2012 International Conference on Applied and Engineering Mathematics*, ICAEM 2012, London, 2012.

[4] R. Mahfoudhi, Z. Mahjoub, and W. Nasri, "Une nouvelle méthode de parallélisation optimale pour l'inversion de matrice triangulaire," *Proceedings* Renpar'20 , Saint Malo, France, 2011.

[5] A. Quarteroni, R. Sacco and F. Saleri, "Méthodes numériques. Algorithmes, analyse et applications," *Springer*, Milano, 2007.

[6] A. V. Aho, J. E. Hopcroft, and J.D. Ullman," The design and analysis of computer algorithms," Addison-Wesley, Reading, Mass, 1974.

[7] V. Strassen, "Gaussian elimination is not optimal", Numerische Mathematik, 1969, 13, pp. 354-356.

[8] S. Huss-Lederman, E.M. Jacobson, J.R. Johnson, A. Tsao and T.Turnbull, "Strassen's algorithm for matrix multiplication: Modeling, analysis, and implementation," Technical Report, Center for Computing Sciences, Bowie, Maryland, 1996.

[9] (2013) The BLAS website. [Online]. Available: www.netlib.org/blas/

[10] M. Thottethodi, S. Chatterjee and A. R. Lebeck, "Tuning Strassen's matrix multiplication for memory efficiency," Supercomputing '98 Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, Orlando, Florida, 2012.

[11] J. Demmel, O. Holtz and R. Kleinberg, "Fast linear algebra is stable," Numerische Mathematik., 2007, 108(1), pp. 59-91.

[12] J. Demmel, O. Holtz and R. Kleinberg, "Fast matrix multiplication is stable," Numerische Mathematik., 2007, 106(2), pp. 199-224.