Communications in Mathematics and Applications
Vol. 14, No. 2, pp. 707–719, 2023
ISSN 0975-8607 (online); 0976-5905 (print)
Published by RGN Publications
DOI: 10.26713/cma.v14i2.2129

http://www.rgnpublications.com

Research Article

Divide and Conquer Methods for Solving Linear Systems

Youssouf Mezzar* and Kacem Belghaba

Laboratory of Mathematics and its Applications, University Oran I, Algeria
***Corresponding author:** youcefmezzar@gmail.com

Received: November 27, 2022 **Accepted:** March 24, 2023

Abstract. The Divide and Conquer (D&C) strategy solves a problem by breaking it down into sub-problems, which are themselves smaller cases of the same type of problem. We will see how this technique creates a numerical recursive algorithm to calculate the inverse of square matrices and solve linear systems using Schur's complement, Cholesky, and LU decomposition.

Keywords. Divide and conquer, Recursive algorithm, Schur complement, Matrix decomposition, Matrix inversion

Mathematics Subject Classification (2020). 15A09, 65F05, 65F10, 65Y04

1. Introduction

The direct inverting of matrices is useful for many applications, including solving systems of linear equations. But the effectiveness of the algorithms for this task is related to the size and the algebraic structure of the matrices to be inverted (Reuter and Hill [17]), which prompts us to use approximate iterative solutions instead (Saad [18]).

Our problem is to solve the linear system $Mx = b$. As is well known, if M is a non-singular matrix, the solution is given by $x = M^{-1}b$. This approach suffers from a practical point of view of digital instability (Cormen *et al*. [8]). For this reason, we will try several methods to calculate the inverse of the matrix M, we will compare them in terms of the time spent and calculation precision.

A well-known method is then the LU decomposition (Cholesky decomposition in the case of symmetric positive definite matrices) is numerically stable and has the added benefit of being faster in practice, we will also study is Schur's complement decomposition method, but using a Divide and Conquer (D&C) strategy (Abu-Saman and El-Okur [2], Björck [5], and Cormen *et al*. [8]).

In Divide and Conquer, we solve a problem recursively, applying three steps to each level of recursion:

- *Divide* the problem into two or more sets of sub-problems that represent smaller cases of the same problem.
- *Conquer* the sub-problems by solving them recursively. If the sizes of the sub-problem are small enough, then just solve it directly.
- *Combine* the solutions of sub-problems into solution of original problem.

In linear algebra, several researchers have used a Divide and Conquer strategy. Abu-Samman [1], Abu-Samman and El-Okur [2], Heller [13], and Mahfoudhi [16] were interested in the inversion of matrices using Divide and Conquer algorithms and others like Andersen *et al*. [3], Dongara [11], Georgiev and Waśniewski [12], and Yang *et al*. [21] used the same strategy to decompose matrices using Cholesky and LU decomposition. There are also those who have suggested several algorithms for solving linear systems and eigenvalue problems (Cleary and Dongarra [6], and Climent *et al*. [7]).

Our objective here is to design sequential algorithms based on the Divide and Conquer strategy to invert matrices and to solve linear systems using previous research in the field. We suggest a new Divide and Conquer algorithm for solving linear systems based on Schur's complement decomposition (DC_SchurSol). Then we adopt the forward substitution and back substitution methods (Björck [5]) to get tow Divide and Conquer algorithms (DC_LTriSol, DC_UTriSol) for solving lower and upper triangular linear systems.

This paper is organized as follows. In the first section, we present Divide and Conquer models for inverting matrices using LU, Cholesky, and Schur's complement decomposition In the second section we present some algorithms for solving linear systems by dividing them into small linear sub-systems.

Numerical examples are given to show the efficiency of the algorithms. The algorithms are simulated to work with dense, structural, and sparse matrices. The computational results of (the D&C algorithms) are compared to the MATLAB inverses obtained using built-in functions and the iterative method (CG) (Björck [5], Lyche [15], and Saad [18]) for solving linear systems.

2. Divide and Conquer Methods for Matrix Inversion

We can summarize the steps of Divide and Conquer strategy in the following algorithm:

Algorithm 1: Divide and Conquer algorithm structure (DC)

Step 1: **input** the problem P;
Step 2: **If** P small enough **return** the solution of P;
Step 3: **else** divide P into sub-problems $P_1, P_2, \ldots, P_k$, $k > 1$;
Apply DC to each of these sub-problems;
Step 4: **return** Combine(DC(P_1), DC(P_2),. .,DC(P_k)).

2.1 Schur's Complement Decomposition

Let M be a $n \times n$ block matrix given as:

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \tag{2.1}$$

where A is $p \times p$ matrix and D is $q \times q$ matrix, with $p+q=n$.
We assume that A is invertible, the Schur complement of A in M is $S = D - CA^{-1}B$, the factorization of M is written as follows: (Cottle [9], Sousedík *et al*. [19], [1])

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I & BA^{-1} \\ 0 & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & 0 \\ A^{-1}C & I \end{pmatrix}, \tag{2.2}$$

where I represent the identity matrix.
If S is an invertible matrix then the inverse of the matrix M is:

$$M^{-1} = \begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ -A^{-1}C & I \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & -BA^{-1} \\ 0 & I \end{pmatrix} \tag{2.3}$$

$$= \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}. \tag{2.4}$$

If M is a symmetric positive definite matrix, so that the matrices A and $S = D - CA^{-1}C^T$ are both symmetric and positive definite matrices (Cormen *et al*. [8]). Therefore, the inverses A^{-1} and S^{-1} exist. Then, we have:

$$M = \begin{pmatrix} A & C^T \\ C & D \end{pmatrix} = \begin{pmatrix} I & C^TA^{-1} \\ 0 & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & 0 \\ A^{-1}C & I \end{pmatrix}, \tag{2.5}$$

and

$$M^{-1} = \begin{pmatrix} I & 0 \\ -A^{-1}C & I \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & -C^TA^{-1} \\ 0 & I \end{pmatrix} \tag{2.6}$$

$$= \begin{pmatrix} A^{-1} + A^{-1}C^TS^{-1}CA^{-1} & -A^{-1}C^TS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}. \tag{2.7}$$

So, to invert a matrix M of size n, it suffices only to invert two matrices A and S of size p and q respectively (see Algorithm 2).

2.2 Divide and Conquer Method for Inverting Triangular Matrix

In 1973, Heller used a Divide and Conquer strategy to develop a recursive algorithm for triangular matrix inversion (Heller [13]). The main idea is to split a triangular matrix T and its inverse B (both of size n) into 3 sub-matrices of size $\frac{n}{2}$ as in Mahfoudhi [16].

The procedure is repeated recursively until we reach sub-matrices of size 1. Thus, the inversion of the triangular matrix T of size n consists of inverting two triangular sub-matrices of size $\frac{n}{2}$.

Let T be a lower triangular matrix of size n partitioned as follows:

$$T = \begin{pmatrix} T_1 & 0 \\ T_2 & T_3. \end{pmatrix}. \tag{2.8}$$

By equation (2.4) we obtain (see Datta [10], and Laub [14]):

$$T^{-1} = \begin{pmatrix} T_1^{-1} & 0 \\ -T_3^{-1}T_2T_1^{-1} & T_3^{-1} \end{pmatrix}. \tag{2.9}$$

Note that T_1 and T_3 are also lower triangular matrices, so they can be inverted recursively.

[1] J. Gallier, The Schur complement and symmetric positive semidefinite (and definite) matrices, *preprint* (2019), URL: https://www.cis.upenn.edu/~jean/schur-comp.pdf.

Algorithm 2: D&C to invert matrix using Schur's complement decomposition (DC_SchurInv)

Step 1: input M, with M is $n \times n$ symmetric positive definite matrix;
Step 2: **if** n==1
$V = 1/M$;
else
if $\mathrm{mod}(n,2) == 0$;
$p = q = n/2$;
else
$p = (n-1)/2$;
$q = n - p$;
splitting M as $M = \begin{pmatrix} A_p & C^T \\ C & D_q \end{pmatrix}$;
let V a new $n \times n$ matrix with: $V = M^{-1} = \begin{pmatrix} K_p & R \\ T & U_q \end{pmatrix}$;
compute S where $S = D - CA^{-1}C^T$
compute recursively A^{-1} and S^{-1} as;
$A^{-1} = \mathrm{DC\_SchurInv(A)}$
$S^{-1} = \mathrm{DC\_SchurInv(S)}$ and set $U_p = S^{-1}$
compute:
$Z = A^{-1}C^T S^{-1}$;
$R = -Z$ and set $T = R^T$;
$K_p = A^{-1} + ZCA^{-1}$;
Step 3: reshape the sub-matrices K, R, T and U in V as: $V = \begin{pmatrix} K_p & R \\ T & U_q \end{pmatrix}$.

Algorithm 3: D&C method to invert a lower triangular matrix (DC_LTMInv)

Step 1: input T, with T is $n \times n$ non-singular triangular matrix and B its inverse.
Step 2: **if** $n == 1$;
$B = 1/T$;
else
if $\mathrm{mod}(n,2) == 0$;
$p = q = n/2$;
else
$p = (n-1)/2$;
$q = n - p$;
splitting T as $T = \begin{pmatrix} T_1 & 0 \\ T_2 & T_3 \end{pmatrix}$, where T_1 and T_3 of size p and q respectively;
let B a new matrix with $B = T^{-1} = \begin{pmatrix} B_1 & 0 \\ B_2 & B_3 \end{pmatrix}$, where B_1 and B_3 of size p and q respectively
compute recursively T_1^{-1} and T_3^{-1} as:
$B_1 = \mathrm{DC\_LTMInv}(T_1)$;
$B_3 = \mathrm{DC\_LTMInv}(T_3)$;
compute
$C = -B_3 * T_2$;
$B_2 = C * B_1$;
Step 3: combine the sub-matrices B_1, B_2 and B_3 in B as: $B = \begin{pmatrix} B_1 & 0 \\ B_2 & B_3 \end{pmatrix}$.

For the upper triangular matrix $T = \begin{pmatrix} T_1 & T_2 \\ 0 & T_3 \end{pmatrix}$ we have:

$$T^{-1} = \begin{pmatrix} T_1^{-1} & -T_1^{-1}T_2T_3^{-1} \\ 0 & T_3^{-1} \end{pmatrix}. \tag{2.10}$$

Now the Divide and Conquer algorithm (see Algorithm 3) to invert a lower triangular matrix (DC_LTMInv) can be suggested, which can be easily modified to obtain an algorithm for inverting an upper triangular matrix (DC_UTMInv).

Remark 2.1. We have modified the algorithm above to invert $n \times n$ matrices in which n is not exactly a power of 2.

2.3 Cholesky and LU Decomposition

Using the LU decomposition, we can write the matrix M as a product of two triangular matrices L and U as the form (Banerjee and Roy [4], and Björck [5]):

$$M = LU \tag{2.11}$$

with L is a lower triangular matrix has 1 in its diagonal entries, and U is an upper triangular matrix.

We can use this decomposition to find the inverse of M as: $M^{-1} = U^{-1}L^{-1}$, such that L^{-1} and U^{-1} can be found using Algorithm 3.

In the case where M is a symmetric positive definite matrix, Cholesky decomposition can be used to factorize M as follows: $M = LL^T$, where L is a lower triangular matrix. This can be used to invert the matrix M as: $M^{-1} = L^{-T}L^{-1}$.

In [3], [12], algorithms are proposed for LU and Cholesky decomposition using the Divide and Conquer method. This will be used in the following algorithms for inverting matrices.

Algorithm 4: D&C method to invert a matrix using the LU decomposition (DC_LUInv)

Step 1: input M, with M is an $n \times n$ non-singular matrix;
Step 2: decompose M as $M = LU$ with L and U are lower and upper triangular matrix respectively;
Step 3: compute L^{-1} and U^{-1} as follows:
$L^{-1} =$ DC_LTMInv(L);
$U^{-1} =$ DC_UTMInv(U);
Step 4: finally $M^{-1} = U^{-1}L^{-1}$.

Algorithm 5: D&C method to invert matrix using Cholesky's decomposition (DC_CholInv)

Step 1: input M, with M is an $n \times n$ symmetric positive definite matrix;
Step 2: decompose M as $M = LL^T$ with L is a lower triangular matrix;
Step 3: compute L^{-1} and L^{-T} as follows:
$L^{-1} =$ DC_LTMInv(L);
$L^{-T} = (L^{-1})^T$;
Step 4: finally $M^{-1} = L^{-T}L^{-1}$.

2.4 Numerical Experiment

This section presents our implementation experiments for different versions of the matrix inversion described above (DC_SchurInv, DC_LUInv, DC_CholInv and inv). The experiments use the MATLAB library (The MathWords[2]), remember that "inv" refers to a matrix inversion command in MATLAB.

[2]The MathWorks, *MATLAB User's Guide*, Version 8.1.0.604, (R2013a), URL: https://www.mathworks.com.

Example 2.1. Consider the $n \times n$ matrix given by:

$$m_{1,j} = 1, \quad j = 1,\dots,n;$$
$$m_{i,1} = 1, \quad i = 1,\dots,n;$$
$$m_{i,j} = m_{i-1,j} + m_{i,j-1}, \quad i = 2,\dots,n;\ j = 2,\dots,n.$$

For $n = 8$ and $n = 16$ the matrix M is symmetric positive definite (Abu-Saman and El-Okur [2]), if $n = 8$ we have:

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 6 & 10 & 15 & 21 & 28 & 36 \\ 1 & 4 & 10 & 20 & 35 & 56 & 84 & 120 \\ 1 & 5 & 15 & 35 & 70 & 126 & 210 & 330 \\ 1 & 6 & 21 & 56 & 126 & 252 & 462 & 792 \\ 1 & 7 & 28 & 84 & 210 & 462 & 924 & 1716 \\ 1 & 8 & 36 & 120 & 330 & 792 & 1716 & 3432 \end{pmatrix}. \tag{2.12}$$

Now we invert this matrix using the algorithms above and comparing them in term of the relative residual error computing using the formula: $\max(\|I - AA^{-1}\|, \|I - A^{-1}A\|)/\|A\|$.

Table 1. Computations of the residual relative error for invert the matrix (2.12) shows that the Divide and Conquer algorithms give the inverse accurately and without errors, but using inv gives completely wrong matrices in the case where $n = 16$

Matrix size	$\max(\|I - AA^{-1}\|, \|I - A^{-1}A\|)/\|A\|$			
	DC_SchurInv	DC_LUInv	DC_CholInv	inv(M) by MATLAB
8	0.0000e+000	0.0000e+000	0.0000e+000	97.0529e-012
16	0.0000e+000	0.0000e+000	0.0000e+000	11.9169e+000

Example 2.2. We want to invert the Poisson matrix M of size n given by (Lyche [15]):

$$a_{ii} = 4, \quad i = 1,\dots,n,$$
$$a_{i+1,i} = a_{i,i+1} = -1, \quad i = 1,\dots,n-1,\ i \neq m, 2m, \dots, (m-1)m,$$
$$a_{i+m,i} = a_{i,i+m} = -1, \quad i = 1,\dots,n-m,$$
$$a_{ij} = 0, \quad \text{otherwise},$$

where $m = \sqrt{n}$, for $n = 9$ we have:

$$M = \begin{pmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{pmatrix}. \tag{2.13}$$

M is a symmetric positive definite matrix (Lyche [15]) then we can use the Cholesky and Schur's complement decomposition.

Table 2. Timing of Poisson matrix inversion using Divide and Conquer methods

Matrix size	Time (second)			
	DC_SchurInv	DC_LUInv	DC_CholInv	inv(M) by MATLAB
1600	0.9208	1.0657	0.5875	0.1368
2500	1.2967	2.7775	1.3961	0.3621
3600	2.7470	6.505837	3.5275	0.7616
4900	5.8463	16.2494	7.9505	3.3009
6400	9.8727	32.1943	15.8321	5.5159
8100	18.6535	72.1769	35.3100	9.4356

Table 3. Computations of the residual relative error for different size of Poisson matrix by Divide and Conquer methods

Matrix size	$\max(\|I - AA^{-1}\|, \|I - A^{-1}A\|)/\|A\|$			
	DC_SchurInv	DC_LUInv	DC_CholInv	inv(M) by MATLAB
1600	4.6623e-15	6.1873e-15	5.9117e-15	1.0367e-14
2500	7.5625e-15	1.0766e-14	9.6850e-15	1.6379e-14
3600	1.2089e-14	1.5832e-14	1.4552e-14	2.6336e-14
4900	1.6318e-14	2.3232e-14	1.9394e-14	3.2289e-14

Example 2.3. In Table 4 and Table 5, we will compare the inversion algorithms of random symmetric positive definite matrices of different size, in terms of time and relative residual error.

Table 4. Timing of matrix inversion using Divide and Conquer methods

Matrix size	Time (second)			
	DC_SchurInv	DC_LUInv	DC_CholInv	inv(M) by MATLAB
1600	0.5347	1.7477	0.7307	0.6763
2500	1.5560	6.4414	3.2744	2.6516
3600	3.8330	15.4444	7.7602	6.5551
4900	7.7059	27.9299	13.5771	13.8499
6400	21.0229	67.9501	30.9712	32.5047
8100	107.6112	311.0037	130.0791	62.8132

Table 5. Computations of the residual relative error for different size of dense symmetric positive definite matrix by Divide and Conquer methods

Matrix size	$\max(\|I - AA^{-1}\|, \|I - A^{-1}A\|)/\|A\|$			
	DC_SchurInv	DC_LUInv	DC_CholInv	inv(M) by MATLAB
1600	1.2031e-11	1.1493e-13	8.1339e-14	2.1338e-08
2500	9.2392e-10	7.1712e-12	7.1002e-12	4.0617e-06
3600	1.7450e-12	7.9153e-15	8.7412e-15	2.8820e-08
4900	1.0742e-11	5.7758e-14	3.0743e-14	7.3678e-08

Discussion

The Divide and Conquer algorithms treat all matrices in the same way and are not affected by their algebraic properties. This makes it very effective in dense and poorly preconditioned matrices.

We notice that Shure's algorithm (DC_SchurInv) is faster in the dense matrices despite inverting two matrices at each level of recursion, but it is less accurate than Cholesky algorithm (DC_CholInv).

The problem with Divide and Conquer algorithms is the memory because it has to keep two matrices of size n in all the algorithm steps. This makes it slower in the case of large matrices, as we have noticed that it takes about 40% of the total execution time in the combine.

3. Solving Linear Systems

In this section we would like to solve a linear system:

$$Mx = b \tag{3.1}$$

with M being an $n \times n$ symmetric positive-definite matrix and x, b two vectors in $\mathbb{R}$.

3.1 Divide and Conquer Method Using Schur's Complement Decomposition

Our approach is to divide the linear system (3.1) of size n into two similar linear systems of size p and q with $p+q=n$, using Schur's complement decomposition $M = LDL^T$ as equation (2.5) so $D = L^{-1}ML^{-T}$.

Using our decomposition, we obtain

$$Mx = b \Leftrightarrow ML^{-T}L^T x = b \tag{3.2}$$

$$\Leftrightarrow L^{-1}ML^{-T}L^T x = L^{-1}b \tag{3.3}$$

$$\Leftrightarrow Dy = \widetilde{b} \tag{3.4}$$

We put $\widetilde{b} = L^{-1}b$ and $y = L^T x$, where x is the desired solution vector. Now, we split the matrix D and the vectors y and $\widetilde{b}$ as follows:

$$D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad \widetilde{b} = \begin{pmatrix} \widetilde{b}_1 \\ \widetilde{b}_2 \end{pmatrix}. \tag{3.5}$$

By replacing the previous in (3.4), we obtain the following result:

$$Dy = \widetilde{b} \Leftrightarrow \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \begin{pmatrix} \widetilde{b}_1 \\ \widetilde{b}_2 \end{pmatrix} \tag{3.6}$$

$$\Leftrightarrow \begin{cases} D_1 y_1 = \widetilde{b}_1, \\ D_1 y_1 = \widetilde{b}_2. \end{cases} \tag{3.7}$$

Hence, to solve the linear systems (3.1) of size n, it is sufficient to solve only two linear sub-systems independent of size p and q with $p+q=n$ which can be solved recursively and in parallel.

Algorithm 6: D&C to solve linear systems using Schur's complement (DC_SchurSol)

Step 1: input M, b, with M is $n \times n$ symmetric positive-definite matrix and $b \in \mathbb{R}^n$;
Step 2: **if** $n == 1$;
$x = b/M$;
else
if $\mathrm{mod}(n,2) == 0$;
$p = q = n/2$;
else
$p = (n-1)/2$;
$q = n - p$;
decompose M into $M = LDL^T$ using equation (2.5);
as: $M = \begin{pmatrix} I & L_1 \\ 0 & I \end{pmatrix}\begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}\begin{pmatrix} I & 0 \\ L_1 & I \end{pmatrix}$, where D_1 and D_2 of size p and q respectively
put $\widetilde{b} = L^{-1}b$ and splitting D and $\widetilde{b}$ as:
$D = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}, \widetilde{b} = \begin{pmatrix} \widetilde{b}_1 \\ \widetilde{b}_2 \end{pmatrix}$;
solving in parallel the two sub linear systems $D_1 y_1 = \widetilde{b}_1$ and $D_2 y_2 = \widetilde{b}_2$ recursively as:
$y_1 = \mathrm{DC\_SchurSol}(D_1, \widetilde{b}_1)$;
$y_2 = \mathrm{DC\_SchurSol}(D_2, \widetilde{b}_2)$;
Step 3: combine the solution of two sub linear system as $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$;
Step 4: compute $x = L^{-T}y$.

Remark 3.1. In the algorithm (6), it is easy to find L^{-1} by using equation (2.10). Observe that:

$$L^{-1} = \begin{pmatrix} I & L_1 \\ 0 & I \end{pmatrix}^{-1} = \begin{matrix} I & -L_1 \\ 0 & I \end{matrix}. \tag{3.8}$$

3.2 Divide and Conquer Method for Triangular Linear Systems

We want to solve the following linear system $Tx = b$ with T is an $n \times n$ lower triangular matrix and x, b two vectors of size n.

We split T, x and b as:

$$T = \begin{pmatrix} T_1 & 0 \\ T_2 & T_3 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \tag{3.9}$$

So

$$Tx = b \Leftrightarrow \begin{pmatrix} T_1 & 0 \\ T_2 & T_3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{3.10}$$

$$\Leftrightarrow \begin{cases} T_1 x_1 = b_1, \\ T_2 x_1 + T_3 x_2 = b_2, \end{cases} \tag{3.11}$$

$$\Leftrightarrow \begin{cases} T_1 x_1 = b_1, \\ T_3 x_2 = b_2 - T_2 x_1. \end{cases} \tag{3.12}$$

T_1 and T_3 are also lower triangular matrices, so we can solve the two systems in (3.12) recursively.

Algorithm 7: D&C method to solve lower triangular linear systems (DC_LTriSol)

Step 1: input T, with T is an $n \times n$ lower triangular matrix and $b \in \mathbb{R}^n$;
Step 2: **if** $n == 1$;
$x = b/T$;
else
if $\text{mod}(n,2) == 0$;
$p = q = n/2$;
else
$p = (n-1)/2$;
$q = n - p$;
else decompose T and b as: $T = \begin{pmatrix} T_1 & 0 \\ T_2 & T_3 \end{pmatrix}$, $b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$;
solving two sub linear systems $T_1 x_1 = b_1$ and $T_3 x_2 = b_2 - T_2 x_1$ recursively as:
x_1 =DC_TriSol(T_1, b_1);
$\widetilde{b}_2 = b_2 - T_2 x_1$;
x_2 =DC_TriSol$(T_3, \widetilde{b}_2)$;
Step 3: combine the solution of two sub linear system as $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.

Remark 3.2. The previous algorithm can be modified easily if T is an upper triangular matrix. To find an algorithm for solving an upper triangular systems (DC_UTriSol).

3.3 Divide and Conquer Method to Solve Linear Systems Using LU and Cholesky Decomposition

In Section 2.3, we saw the LU decomposition of a matrix. In this section, we will use it to solve a linear system. Let M be a $n \times n$ non-singular matrix, and x, b two vectors of size n.

The LU decomposition divides the solution of a linear system into two independent steps (Björck [5], and Van Loan [20]):

- The decomposition $M = LU$.
- Solution of the systems $Ly = b$ and $Ux = y$ from the following:

$$Mx = b \Leftrightarrow LUx = b; \tag{3.13}$$

$$\Leftrightarrow Ly = b; \quad \text{with} \quad Ux = y. \tag{3.14}$$

We have the following algorithm:

Algorithm 8: D&C method to solve linear systems using LU decomposition (DC_LUSol)

Step 1: input M, with M is an $n \times n$ non-singular matrix, and b a vector of size n;
Step 2: decompose M as $M = LU$, with L and U are lower and upper triangular matrix respectively;
Step 3: solve the two sub linear systems $Ly = b$ then $Ux = y$ as follows:
y = DC_LTriSol(L,b);
x = DC_UTriSol(U,y).

Remark 3.3. In the case of symmetric positive definite matrices we use Cholesky decomposition $M = LL^T$, and follow the same previous steps with taking $U = L^T$, and we get the algorithm: DC_CholSol.

3.4 Numerical Experiment

We now try to test the efficiency of the Divide and Conquer algorithms in solving the linear system $Mx = b$ with M is an $n \times n$ symmetric positive definite matrix, comparing the iterative methods (CG) (Björck [5], and Lyche [15]).

Example 3.1. Let M be the Poisson matrix defined in the equation (2.13). It is clear that it is sparse, and $b = (1, 1, \ldots, 1)^T$.

Table 6. The timing and the residual relative error of the solution of different size of the Poisson linear system $Mx = b$, by the Divide and Conquer methods

Matrix size	Time (second)			$\|Mx - b\|$		
	DC_SchurSol	DC_CholSol	CG	DC_SchurSol	DC_CholSol	CG
3600	2.6116	0.7999	0.0293	5.8806e-12	5.4534e-12	2.2507e-11
4900	4.7465	1.1429	0.0316	9.0667e-12	8.6216e-12	4.1059e-11
6400	7.9830	1.5576	0.0403	1.4414e-11	1.3024e-11	5.4827e-11
8100	12.9552	2.0846	0.0558	2.0786e-11	1.8538e-11	9.4044e-11
10000	18.9420	3.3982	0.0750	2.9764e-11	2.6081e-11	1.3063e-10

Example 3.2. Now for the randomly selected symmetric positive definite matrices, which are mostly dense.

Table 7. The timing and the residual relative error of the solution of different size of the dense linear system $Mx = b$, by the Divide and Conquer methods

Matrix size	Time (second)			$\|Ax - b\|$		
	DC_SchurSol	DC_CholSol	CG	DC_SchurSol	DC_CholSol	CG
900	0.4572	0.2330	3.4579	2.4306e-09	5.8513e-11	3.4349e-10
1600	1.1124	0.5265	18.8615	4.6683e-07	3.0280e-09	2.1894e-08
2500	2.0733	1.0856	72.2107	1.0346e-05	1.1068e-07	4.2392e-07
3600	4.3217	2.1937	230.3187	9.0728e-06	7.9475e-08	1.1069e-06
4900	8.3134	4.2590	/	4.1107e-05	7.7785e-07	/
6400	16.2687	7.5213	/	2.2387e-07	1.4287e-09	/
8100	34.0075	14.6121	/	2.8928e-07	3.4490e-09	/

Discussion

We have noticed throughout our study that the Divide and Conquer methods are not affected by the algebraic properties of matrices as we said earlier, in contrast to iterative methods whose performance varies from one matrix to another according to its condition number.

In solving linear systems, we notice that the iterative methods are faster in the case of sparse matrices, but very slow in the case of dense matrices, unlike Divide and Conquer methods which seem more stable and give more precise results.We point out that the Schur's algorithm (DC_SchurSol) can be further accelerated and the execution time reduced by about 45% by solving linear sub-systems in parallel.

4. Conclusion

Using Divide and Conquer methods to directly invert matrices and to solve linear systems without decomposition will significantly reduce the approximation error. This technique will also enable calculations of highly sensitive systems. In fact, this technique is really consistent with parallel computing, which makes it much faster.

Acknowledgements

The authors wish to warmly thank the referees for all their useful suggestions.

Competing Interests

The authors declare that they have no competing interests.

Authors' Contributions

All the authors contributed significantly in writing this article. The authors read and approved the final manuscript.

References

[1] A. M. Abu-Saman, Computing matrix inverses by divide-and-conquer method with floating point calculations, *International Journal of Computational and Applied Mathematics* **7**(4) (2012), 479 – 484.

[2] A. M. Abu-Saman and S. S. El-Okur, Divide-and-conquer strategy with Cholesky's factorization for inverting symmetric positive definite matrices, *International Electronic Journal of Pure and Applied Mathematics* **7**(2) (2014), 53 – 62, DOI: 10.12732/iejpam.v7i2.1.

[3] B. S. Andersen, F. Gustavson, A. Karaivanov, M. Marinova, J. Waśniewski and P. Yalamov, LAWRA Linear Algebra with Recursive Algorithms, in: *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia* (PARA 2000), T. Sørevik, F. Manne, A. H. Gebremedhin and R. Moe (editors), *Lecture Notes in Computer Science*, Vol. **1947**, Springer, Berlin — Heidelberg (2001), DOI: 10.1007/3-540-70734-4_7.

[4] S. Banerjee and A. Roy, *Linear Algebra and Matrix Analysis for Statistics*, 1st edition, Chapman and Hall/CRC, New York, 580 pages (2014), DOI: 10.1201/b17040.

[5] Å. Björck, *Numerical Methods in Matrix Computations*, 1st edition, Texts in Applied Mathematics series, Vol. **59**, Springer Cham, xvi + 800 pages (2015), DOI: 10.1007/978-3-319-05089-8.

[6] A. Cleary and J. Dongarra, *Implementation in ScaLAPACK of Divide-and-Conquer Algorithms for Banded and Tridiagonal Linear Systems*, Technical Report UT-CS-97-358, University of Tennessee, Knoxville, TN, (1997), URL: https://library.eecs.utk.edu/files/ut-cs-97-358.pdf.

[7] J.-J. Climent, C. Perea, L. Tortosa and A. Zamora, A BSP recursive divide and conquer algorithm to solve a tridiagonal linear system, *Applied Mathematics and Computation* **159**(2) (2004), 459 – 484, DOI: 10.1016/j.amc.2003.08.130.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 3rd edition, The MIT Press, Cambridge, 1312 pages (2009), URL: https://mitpress.mit.edu/9780262533058/introduction-to-algorithms/.

[9] R. W. Cottle, Manifestations of the Schur complement, *Linear Algebra, and its Applications* **8** (1974), 189 – 211, URL: http://aboutme.samexent.com/classes/fall08/ee8950/cottle.pdf.

[10] B. Datta, *Numerical Methods for Linear Control Systems*, 1st edition, Academic Press, 640 pages (2003), URL: https://www.elsevier.com/books/numerical-methods-for-linear-control-systems/datta/978-0-12-203590-6.

[11] J. Dongarra, V. Eijkhout and P. Luszczek, Recursive approach in sparse matrix LU factorization, *Scientific Programming* **9** (2001), 51 – 60, URL: https://icl.utk.edu/~luszczek/pubs/sciprog.pdf.

[12] K. Georgiev and J. Waśniewski, Recursive version of LU decomposition, in: *Numerical Analysis and Its Applications* (NAA 2000), *Lecture Notes in Computer Science*, L. Vulkov, P. Yalamov and J. Waśniewski (editors), Vol. **1988**, Springer, Berlin — Heidelberg, DOI: 10.1007/3-540-45262-1_38.

[13] D. Heller, A survey of parallel algorithms in numerical linear algebra, *SIAM Review* **20**(4) (1978), 740 – 777, DOI: 10.1137/1020096.

[14] A. J. Laub, *Matrix Analysis for Scientists and Engineers*, SIAM, Philadelphia, xiii + 157 pages (2004), URL: https://my.siam.org/Store/Product/viewproduct/?ProductId=991.

[15] T. Lyche, *Numerical Numerical Linear Algebra and Matrix Factorizations*, 1st edition, Texts in Computational Science and Engineering series, Vol. **22**, Springer, Switzerland, xxiii + 371 pages (2020), DOI: 10.1007/978-3-030-36468-7.

[16] R. Mahfoudhi, *A Fast Triangular Matrix Inversion*, *Proceedings of the World Congress on Engineering* (WCE 2012), Vol. I (2012), July 4-6, 2012, London, UK (2012), URL: https://www.iaeng.org/publication/WCE2012/WCE2012_pp100-102.pdf.

[17] M. G. Reuter and J. C. Hill, An efficient, block-by-block algorithm for inverting a block tridiagonal, nearly block Toeplitz matrix, *Computational Science & Discovery* **5**(1) (2012), 014009, DOI: 10.1088/1749-4699/5/1/014009.

[18] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edition, SIAM, xvii + 520 pages (2003), DOI: 10.1137/1.9780898718003.

[19] B. Sousedík, R. G. Ghanem and E. T. Phipps, Hierarchical Schur complement preconditioner for the stochastic Galerkin finite element methods, *Numerical Linear Algebra with Applications* **21**(1) (2013), 136 – 151, DOI: 10.1002/nla.1869.

[20] C. F. Van Loan, The ubiquitous Kronecker product, *Journal of Computational and Applied Mathematics* **123**(1-2) (2000), 85 – 100, DOI: 10.1016/S0377-0427(00)00393-9.

[21] A. Yang, C. Liu, J. Chang and X. Guo, Research on parallel LU decomposition method and it's application in circle transportation, *Journal of Software* **5**(11) (2010), 1250 – 1255, DOI: 10.4304/jsw.5.11.1250-1255.