

THE INCOMPLETE LU PRECONDITIONER USING BOTH CSR AND CSC FORMATS

Khalifa Boumzough^{1*}, Adnane Azzouzi², Abderrahim Boudi¹

¹Faculty of Sciences, Ibn Zohr University, Agadir, Morocco

²Faculty of Sciences Dhar El Mahraz, Sidi Mohamed ben Abdellah University, Fez, Morocco

Abstract. The main idea of this paper is in the determination of the Incomplete LU factors of a given matrix. The idea is based on constructing the Incomplete LU factorization using both storage formats compressed sparse row (CSR) and compressed sparse column (CSC), with the object of minimizing computation time. Numerical examples and comparing standard incomplete LU factorization preconditioner with the Incomplete LU factorization using both storage formats compressed sparse row (CSR) and compressed sparse column (CSC) validate the effectiveness of this preconditioner when they are applied to accelerate Krylov subspace iteration methods such as Generalized minimal residual (GMRES).

Keywords: Preconditioner, Incomplete LU factorization, CSR, CSC, GMRES, Nonsymmetric matrix.

AMS Subject Classification: 65F08, 65F50.

Corresponding author: Khalifa, Boumzough, Ibn Zohr University, Faculty of Sciences Agadir, Mathematics Department, Engineering of Sciences Laboratory, B.P 8106, Agadir 80000, Agadir, Morocco, Tel.: +212611900065, e-mail: khalifa.boumzough@edu.uiz.ac.ma

Received: 30 March 2022; Revised: 12 May 2022; Accepted: 18 July 2022; Published: 5 August

1 Introduction

We consider solution of sparse systems of linear equation of the form:

$$Ax = b, \quad A \in \mathbb{R}^{m \times n}, \quad b \in \mathbb{R}^n \quad \text{and} \quad x \in \mathbb{R}^m \quad (1)$$

where A is a given real matrix, b is a given real right-hand-side vector and x is the unknown vector. A is a sparse matrix that includes many zeros and fewer non zeros compared to $n \times m$. The most large sparse systems are of practical importance in many areas of scientific computing and engineering applications (see for more details (Zhong-Zhi, 2007; Bai et al., 2003; Li et al., 2007; Axelsson, 1996)). The standard methods for solving the linear systems is based on either the direct methods or the iterative methods (Duff et al., 2017; Saad, 2003). The first method gives the exact solution in a finite number of steps using a factorization of the coefficient matrix A to improve the solution. The basic idea of this approach is to consider the most common LU-decomposition for asymmetric systems to factorize the matrix A as $A = LU$ where L is the lower triangular matrix and U is the upper triangular matrix. Then, the system (1) can be solved in two steps, by forwarding substitution in $Ly = b$ and by back substitution in $Ux = y$, which is very efficient for small linear systems. The challenging task is that the direct method can not be applied to large systems due to the fact that the matrix L and U are more full than A , and that lead to high computer storage. Thus another interesting class of methods known as iterative methods has been involved for solving the system (1). This methods are used for a large sparse matrix that contains a lot of zeros.

The methods of Krylov subspace are a field of iterative methods (Bai et al., 2000; Van der Vorst, 2003; Saad, 2003). The idea of extract an approximate solution from a subspace, are convenient and efficient solvers for the system of linear equations (1), one of these iterative methods is GMRES (Saad & Schultz, 1986; Chen et al., 1999). This method is a widely Krylov iterative method for solving nonsymmetric linear systems. The GMRES method consists of minimizing a norm of the residual, it's related to the BCG method and to other variants of the Lanczos method. Among the advantage of the GMRES algorithm on other Krylov methods is their well stability, combined with a nonincreasing residual norm sequence. In order to promote effectiveness, the GMRES algorithm has been combined with a good preconditioner. The preconditioner plays a crucial role in improving the convergence property success of the preconditioned Krylov subspace iterations, whereas the choice of the preconditioner is more important than the choice of the Krylov iterative method. The idea of the preconditioning is only to transform the original linear system into one which, is likely to be easier to solve with an iterative solver. It contributes to the speed of convergence of the method, which is a more crucial part of high-performance computing. In general, there are two well know classes of basic preconditioners are: Jacobi preconditioner and incomplete factorization. The Jacobi preconditioner is based to use the diagonal matrix such as the preconditioner see the refer Saad (2003). The ILU factorization can produce a sparse lower triangular matrix L and a sparse upper triangular matrix U by using Gaussian elimination with certain elimination rules so that the error matrix $E = LU - A$ became small and meets certain constraints such as the existence of zero entries at certain positions (see for examples (Axelsson, 1996; Kraus et al., 2018; Saad, 2003)). The efficiency of the preconditioner depends on how well $M^{-1} = (LU)^{-1}$ approximates A^{-1} . The success of the iterative method is related to the storage method in order to obtain a better solution. There are a lot of methods to store the matrix see (Gao et al., 2021; Bell & Garland, 2009; Li et al., 2014; Duff et al., 2017; Saad, 2003; Mellor-Crummey & Garvin, 2004; Greathouse & Daga, 2014), CSR (compressed sparse row) and CSC (compressed sparse column) are most efficient and most popular. In addition, these two compressed storage schemes can save a great number of memories, however more calculation will be involved. The CSR format is the ease of construction and manipulation, thus it's the most ideal for matrix-by-vector products which is one of the essential operations in the iterative method. The performance of the operations SPMV on CSR can differ significantly on high-performance computers, whereas for some algorithms that proceed by column and rows in this situation, the performance CSR format is not satisfactory. The objective of this paper is to construct an incomplete LU preconditioner (ILU) using the both storage formats. We construct a CSR format from an existing CSC format. We also show that the preconditioner $ILLU$ based on the both storage formats CSR and CSC is the efficient preconditioner in terms of the computation times CPU s.

The remainder of this paper is arranged as follows, Section 2 provides background on various sparse matrix storage formats. In Section 3, we discuss ILU Factorization Preconditioners. Several classes of experimental problems are described in Section 4. Numerical results are given and discussed in Section 5. Finally, some concluding remarks are given in Section 6.

2 Storage scheme

The majority of problems in sciences or engineering lead to sparse linear systems after the discretization of PDEs, which contains a lot of zeros. These zeros give almost no information, but they consumes a lot of memory. Storing the matrix is keeping only the nonzero elements in a vector VA of length nnz and conserve their coordinates in two other vectors JA and IA. There are three ways of compress storage:

- COO IA stores the row indices of length nnz , JA stores the column indices of length nnz .
- CSR Compressed Sparse Row, JA stores the column indices of length nnz , IA stores only

the beginning number of the first element of each row of length $n + 1$.

- CSC Compressed Sparse Column, IA stores the row indices of length nnz , JA stores only the beginning number of the first element of each column of length $n + 1$.

The coordinate formats are the storage scheme used by the Matrix Market database for matrices and systems of linear equations; for details, see the refer Davis & Hu (2011). On the other hand, the Compressed Sparse Row and Column formats are the most general (Duff et al., 2017; Saad, 2003), they keep only the necessary elements, are inefficient, requiring an indirect addressing step for each scalar operation in a matrix-vector product or a preconditioner to solve as shown below.

For example, let A be a square matrix of order 5×5

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 \\ a_{21} & a_{22} & 0 & a_{24} & 0 \\ a_{31} & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & 0 \\ a_{51} & 0 & 0 & a_{54} & a_{55} \end{pmatrix}$$

The arrays VA, JA, and IA are

$$\begin{array}{l} \text{VA} = \overbrace{a_{11} \ a_{12} \ a_{14}}^{\text{row1}} \ \overbrace{a_{21} \ a_{22} \ a_{24}}^{\text{row2}} \ \overbrace{a_{31} \ a_{33} \ a_{34} \ a_{35}}^{\text{row3}} \ \overbrace{a_{43} \ a_{44}}^{\text{row4}} \ \overbrace{a_{51} \ a_{54} \ a_{55}}^{\text{row5}} \\ \text{JA} = \ 1 \ 2 \ 4 \ 1 \ 2 \ 4 \ 1 \ 3 \ 4 \ 5 \ 3 \ 4 \ 1 \ 4 \ 5 \\ \text{IA} = \ 1 \ 4 \ 7 \ 11 \ 13 \ 16 \end{array}$$

We note that the compressed sparse column *CSC* equivalent to represent A^T in format *CSR*.

One of the fundamental primitives used in iterative methods for solving sparse linear systems is sparse Matrix-vector multiplication (*spMV*). We consider the *spMV* operation $y \leftarrow y + Ax$, where A is sparse Matrix and x, y are dense vectors. Algorithmically the *spMV* is as follows (i, j): $a_{i,j} \neq 0$: $y_i = y_i + a_{i,j}.x_j$, Where $a_{i,j}$ denote the elements of A .

The most common data structure used to store a sparse matrix for *spMV* computation is compressed sparse row (CSR), for details one can refer to Mellor-Crummey & Garvin (2004); Greathouse & Daga (2014). One of the advantages of CSR is the ease of construction and manipulation (no preprocessing of the matrix needed), which means that CSR is the most ideal for *spMV*, we now give an algorithmic description of the *spMV* on CSR.

```

1 for  $i = 1$  To  $n$  do
2    $y(i) = 0$ ;
3   for  $j = IA(i)$  To  $IA(i + 1) - 1$  do
4      $jj = JA(j)$ ;
5      $y(i) = y(i) + VA(j) * x(jj)$ ;

```

We observe that *spMV* on CSR creates the products of row vectors and column vectors of VA. The same principle when using the matrix-vector product definition (Goharian et al., 2003). It provides excellent compression of structured and unstructured sparse matrices. The performance of the *spMV* basis of CSR is good on *CPUs*. While some algorithms proceed by columns and rows. In this situation, the CSR format used for the sparse matrix defines the algorithm by impacting the performance. To improve the efficiency, we need to use both storage formats on these algorithms. Among these algorithms, the incomplete *LU* factorization algorithm (without storage of the original matrix) requires one loop to build the elements per row and the second loop to build the elements per column. As a result, we suggested combining

both storage formats (CSR and CSC) by using the advantages of each of them for constructing ILU factorization.

3 ILU Factorization Preconditioners

A general Incomplete LU factorization process computes a sparse lower triangular matrix L and a sparse upper triangular matrix U , such that $E = LU - A$ are small satisfies certain constraints such as containing zero entries in some position Saad (2003). The Incomplete LU factorization technique with no fill-in denoted by $ILU(0)$ take the zero pattern to be precisely the zero pattern of the original matrix A . By definition, the L and U matrices in $ILU(0)$ have the same number of nonzero elements as the original matrix A . The incomplete factorization technique with no fill-in consists of the performing the i, j, k version of Gaussian elimination, and dropping any element in L and U that fills outside the pettern of A . We will denote the simplest choice by $P = \{(i, j)/a_{i,j} = 0\}$

Algorithm 1: $[L, U]=\text{Factorisation-ILU}(A, P)$

```

1 for  $k = 1, \dots, n - 1$  do
2   for  $i = k + 1, \dots, n$  and if  $(i, k) \notin P$  do
3      $a_{i,k} = a_{i,k}/a_{k,k};$ 
4     for  $j = k + 1, \dots, n$  and if  $(i, j) \notin P$  do
5        $a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j};$ 

```

The above algorithm is for the dense storage array. However, most matrices obtained from the real problems are spares, so the above algorithm is inefficient. In this case, we are using the matrix in CSR format as one of the storage methods. Next, we will describe the Incomplete LU factorization applied to a sparse matrix in CSR format.

3.1 The Incomplete LU factorization for spare matrix in CSR format

An incomplete LU factorizations algorithm depend on the representation of the input matrix A by three arrays: VA continues the non-zero elements of length nnz , JA stores column indices, and IA stores only the starting number of the first element in each row. The values of the resulting factors L and U are stored in the array $luval$, except that the entries for those on the main diagonal of the unit lower triangular matrix L are not stored. Thus, it is needed to store the two factors in a single matrix. This matrix is denoted by L/U . Note that since the pattern of L/U is the same as that of the input matrix A , the other integer matrices in the CSR representation for the LU factors are not needed. So, $ja(k)$, which is the column position of element $a(k)$ in the input matrix, is also the column position of element $luval(k)$ in the L/U matrix.

As result, the standard Incomplete LU algorithm with no-fill made no assumption about the sparsity pattern of A . The nonzero pattern of L and U is often taken to be identical to that of the original matrix. This has the advantage that no additional storage space is required for the nonzero structure of the incomplete decomposition. The ILU factorization in format CSR is very complicated, we use a lot of loops for this algorithm, However, the ILU factorization without CSR is simple, we need just three loops. Therefore the ILU without CSR requires less time than ILU in format CSR for the decomposition of the matrices. For that we suggested the new technique for computing ILU, this new technique depends on using two compressed storage schemes CSR and CSC to compute ILU.

3.2 Incomplete LU factorization with CSR and CSC

The preconditioning ILU makes the algorithm more complicated and require more calculation for a single iteration, thus increases the computation cost for one single iteration. Especially if we consider that an algorithm proceeds by row (or column), it is unsuitable to use the CSC format (or CSR) of a matrix, by reason of the CSC slow row slicing operation (consider CSR). However, the algorithm of the ILU factorization proceeds by column (for computation the lower triangular L factor), and by row (for computation the upper triangular U factor), so it is unsuitable to use only one format (CSR or CSC). it turns out that a more efficient implantation of the practical ILU method can result from the appropriate use of both storage formats in various parts of the algorithm. thus it is natural to store L by column and U by row and to have the lower and upper triangular parts of A stored similarly. It mainly depends that on the following steps:

- Step 1** The matrix A is stored in CSR and CSC formats, we define extra help array $Diag[1, \dots, n]$ which points to the diagonal elements of A in CSC format. The array $Point[1, \dots, n]$ is an array of integers, which points to the entries in U of row i .
- Step 2** From an existing CSC format, we will construct a CSR . The factor L is stored by column, we use the coordinates of the matrix A in CSC format for construct the lower triangular part of the matrix, and we assume that the nonzeros in each row are stored in order by column number, then a pointer of row ia_0 contains the number of nonzeros in each row with $next = ia_0(j)$, where j is the index row in CSC format. This pointer array is update after each elimination step by increasing a point in each pointer at the next nonzero in the row $ia_0(j) = next + 1$. Finally, the pointer ia_0 stores the index of the diagonal elements in CSR format. The factor L are stored and calculated in CSR format by column.
- Step 3** For constructing of the Incomplete upper triangular matrix U , we use this $k = ia_0(j) : ia_0(j + 1) - 1$ for to extract the upper part of the matrix A in CSR format by each row, then we can calculated the factor U by the Pseudo-code.

```

for i=ia0(j):ia(j+1)-1
    ii=Point(ja(i))
    if(ii>0)
        a(i)=a(i)-a(next)*a(ii)
    endif
endfor

```

where j is the index row in CSC format. We can use this step for constructing an algorithm that generates the lower triangular matrix L by columns and the upper triangular matrix U by rows. Thus, we developed an algorithm, which combines two formats CSR and CSC

4 Description of experiments

A number of numerical experiments have been performed to assess the stability, accuracy and efficiency of the incomplete LU factorization using the both storages CSR and CSC. This is achieved through comparisons between the numerical behaviour of Incomplete LU factorization with CSR and CSC and standard ILU preconditioners applied to Krylov subspace iteration methods such as GMRES (Saad & Schultz, 1986). The test matrices from the suite Sparse Matrix Collection.

In the experiments, the CSR and CSC formats are used to store all the matrices and all codes are written in Fortran, compiled on a Linux system and run on an Intel Xeon(R) CPU E5420@ 2.50GHZ with 10 GB of main memory. In addition, all initial guesses x_0 for the iterative

solvers are randomly generated with a uniform distribution such that their entries belong to the interval $[-1, 1]$, and the iterations are terminated either when the number of iterations exceeds 7000 or when the current iterate satisfies $\|r_k\| \leq 10^{-7}\|r_0\|$, where $r_k = b - Ax_k$ is the residual at the k th iteration.

4.1 The suite Sparse Matrix Collection

The Suite Sparse Matrix Collection is a large and actively growing set of sparse matrices that arise in many applications in science and engineering with underlying 2D or 3D geometry (as structural engineering, computational fluid dynamics, model reduction, electromagnetic, semiconductor devices, thermodynamics, materials, acoustics, computer graphics/vision, robotics/kinematics, and other discretizations) and those that typically do not have such geometry (optimization, circuit simulation, economic and financial modelling, theoretical and quantum chemistry, chemical process simulation, mathematics and statistics, power networks, and other networks and graphs). For details, one can refer to Davis & Hu (2011).

In Table 1, we list twenty seven sparse matrices as our Suite Sparse Matrix Collection.

Table 1: The suite sparse matrix collection

Name	plot	n	nnz	Application
atmosmodd		1270432	8814880	Computational Fluid Dynamics Problem
cdde5		961	4681	Subsequent Computational Fluid Dynamics Problem
chipcool0		20082	281150	Model Reduction Problem
dc1		116835	766396	Circuit Simulation Problem Sequence
majorbasis		160000	1750416	Optimization Problem
pde900		900	4380	2D/3D Problem
pde2961		2961	14585	2D/3D Problem
sherman4		1104	3786	Computational Fluid Dynamics Problem
steam1		240	2248	Computational Fluid Dynamics Problem
stomach		213360	3021648	2D/3D Problem
t2em		921632	4590832	Electromagnetics Problem
venkat01		62424	1717792	Computational Fluid Dynamics Problem Sequence Problem
wang3		26064	177168	Semiconductor Device Problem Problem
wang4		26068	177196	Semiconductor Device Problem
c-55		32780	403450	Optimization Problem
cage11		39082	559722	Directed Weighted Graph
torso2		115967	1033473	2D/3D Problem
torso3		259156	4429042	2D/3D Problem
trans5		116835	749800	Subsequent Circuit Simulation Problem
atmosmodl		1489752	10319760	Computational Fluid Dynamics Problem
1138_bus		1138	4054	Power Network Problem
atmosmodm		1489752	10319760	Computational Fluid Dynamics Problem
bcstk12		1473	34241	Duplicate Structural Problem
bcstk22		138	696	Structural Problem

bcsstm10		1086	22092	Structural Problem
cage14		1505785	27130349	Directed Weighted Graph
pores_2		1224	9613	Computational Fluid Dynamics Problem

5 Numerical results

We examine the robustness, effectiveness of Incomplete LU using the both storages CSR and CSC preconditioner by numerically comparing it with the standard ILU preconditioner, which is used to accelerate GMRES (Saad & Schultz, 1986). We compare the methods with respect to the preconditioning time (CPU_p), the iteration time (CPU_{it}), the total number of iteration steps (IT) and the total CPU time

Table 2: Notation

Notation	Description
n	Number of rows in the matrix
nnz	The nonzero of the matrix
CPU_p	The CPU time for constructing a preconditioner
IT	The total number iteration
CPU	The total CPU time

The numerical results in this section concern the ILU using the both storages CSR and CSC denoted as ILU_{csc_csr} – and the standard ILU–preconditioned GMRES, denoted ILU_{csc_csr} – GMRES and ILU – GMRES, respectively, for solving systems of linear equations with coefficient matrices from the Suite Sparse Matrix Collection; see Table 1. In Table 3 we list the total CPU time for ILU_{csc_csr} and ILU. It interesting to compare of the efficiency of the Incomplete LU factorization by different techniques of the implementation. The results are listed in the Table 3.

Table 3: The comparison of the efficiency of two Incomplete LU factorization

Matrix no	Matrix name	CPU_p	
		ILU_{csc_csc}	ILU
1	atmosmodd	0.382	0.530
2	cdde5	1.400E-04	2.450E-04
3	chipcool0	1.778E-02	1.785 E-02
4	dc1	8.185E-02	55.988
5	majorbasis	5.843E-02	7.373E-02
6	pde900	1.48E-04	2.17E-04
7	pde2961	3.76E-04	5.329E-04
8	sherman4	1.360E-04	2.480E-04
9	steam1	2.390E-04	3.729E-04
10	stomach	0.125	0.131
11	t2em	0.147	0.241
12	venkat01	0.193	0.173
13	wang3	6.819E-03	8.728E-03
14	wang4	6.451E-03	9.209E-03
15	c-55	4.462E-2	6.598E-2

16	cage11	3.898E-002	4.390E-002
17	torso2	1.986E-002	2.426E-002
18	torso3	0.1197	0.1247
19	trans5	9.200E-02	57.701
20	atmosmodl	0.482	0.582
21	1138_bus	1.769E-04	2.283E-04
22	atmosmodm	0.455	0.648
23	bcstk12	2.399E-03	3.360E-03
24	bcstk22	3.899E-05	1.209E-04
25	bcstm10	2.178E-03	1.993E-02
26	cage14	1.598	1.919
27	pores_2	2.909E-04	3.770E-04

To investigate the performance of the algorithms of the standard ILU and ILU_{csc_csr} , we use twenty seven sparse matrices with different structures, in this study we use the symmetric and non-symmetric matrices. The efficiency is characterized by the CPU time using for constructing the incomplete LU factorization. We see that ILU_{csc_csr} requires less time for matrix decomposition. However, the standard ILU need more time for constructing of L and U factors. We observe that the total computation time for Standard ILU as long as that for the ILU_{csc_csr} for constructing the incomplete LU factorization in the three matrices $dc1$, $t2em$, and $trans5$. For example the total computation time for standard ILU is 51.742 s, on the other hand the CPU time ILU_{csc_csr} is 6.46E – 2 s for the decomposition the matrix $dc1$. This because the standard ILU involves in many more complicated require more calculations. However ILU_{csc_csr} use the same implementation principle of ILU for the dense matrix.

Table 4: IT and CPU for GMRES for matrices in table 1

Matrix no	Matrix name	GMRES		ILU_{csc_csr} – GMRES		ILU – GMRES	
		IT	CPU	IT	CPU	IT	CPU
1	atmosmodd	918	1152.571	125	245.394	125	247.651
2	cdde5	1347	0.540	85	3.772E-02	85	3.265E-02
3	chipcool0	-	-	65	0.497	65	0.506
4	dc1	-	-	551	134.239	551	141.987
5	majorbasis	91	4.476	9	0.268	9	0.277
6	pde900	100	5.111	96	2.845E-02	96	3.568E-02
7	pde2961	187	0.305	48	2.987E-02	48	3.093E-02
8	sherman4	948	2.422	32	9.608E-03	32	9.56E-03
9	steam1	518	0.265	4	5.869E-04	4	5.950E-04
10	stomach	169	11.434	9	0.440	9	0.496
11	t2em	-	-	604	1811.935	604	1990.177
12	venkat01	-	-	18	0.908	18	0.945
13	wang3	5506	361.158	65	0.940	65	0.803
14	wang4	2519	166.707	56	0.673	56	0.667
15	c-55	-	-	463	32.293	463	32.745
16	cage11	17	0.1016	5	8.177E-02	5	8.208E-02
17	torso2	29	0.3031	6	7.328E-02	6	7.553E-02
18	torso3	140	7.7413	23	1.1031	23	1.0712
19	trans5	-	-	159	21.475	159	24.082
20	atmosmodl	938	1107.254	80	128.364	80	148.289

21	1138_bus	461	0.631	142	0.110	142	0.116
22	atmosmodm	149	55.882	29	26.244	29	26.622
23	bcsstk12	1395	7.403	142	0.211	142	0.207
24	bcsstk22	543	2.469E-002	34	2.475E-03	34	2.484E-03
25	bcsstm10	933	2.531	84	7.061E-02	84	7.332E-02
26	cage14	14	3.617	4	1.909	4	1.965
27	pores_2	-	-	40	1.009E-02	40	1.055E-02

In the table 4, we list the total number of iteration steps and the total CPU time for ILU_{csc_csr} – GMRES and ILU-GMRES, for solving the large linear systems using the suite sparse matrix collection. From this table, we can see that the total number of iteration step of the ILU_{csc_csr} and the standard ILU preconditioners are quite comparable, but the execution time of the ILU_{csc_csr} –preconditioned GMRES iterations are much less than those of the ILU–preconditioned GMRES iteration. In general, ILU_{csc_csr} outperforms ILU-GMRES in total computing time. As a result, it turns out that in the sense of execution time a more effective implementation of the practical ILU_{csc_csr} method comes from appropriately using both storage formats in various parts of the algorithm.

We plot the curves of CPU time versus the total number of iterations in Figs.1, 2, and 3 for the

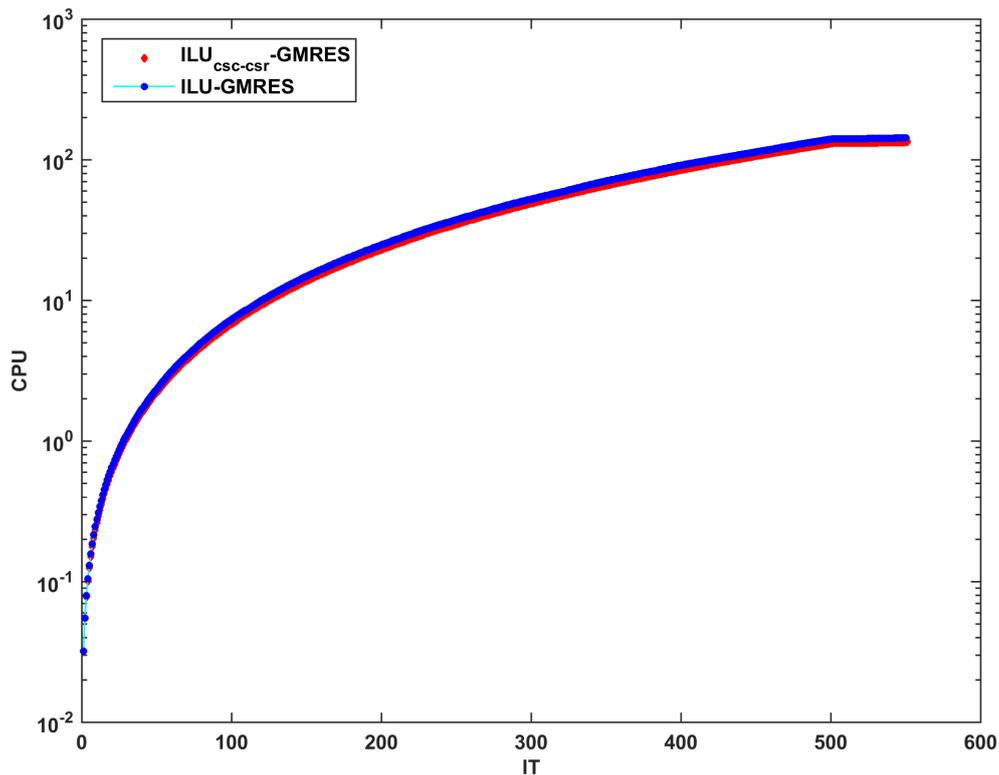


Fig. 1. Curves of the CPU time versus the number of iterations for dc1

matrices dc1, t2em, and trans5, respectively. The red line with diamond markers is the result of preconditioned GMRES with the incomplete LU factorization using both CSR and CSC storage, denoted by ILU_{csc_csr} – GMRES, and the blue line with circle markers is for GMRES preconditioned with the standard incomplete ILU factorization, designated by ILU – GMRES. We see that ILU_{csc_csr} – GMRES took the least time to reach the convergence, although ILU – GMRES required the same number of iterations but took more time to converge at the three matrices.

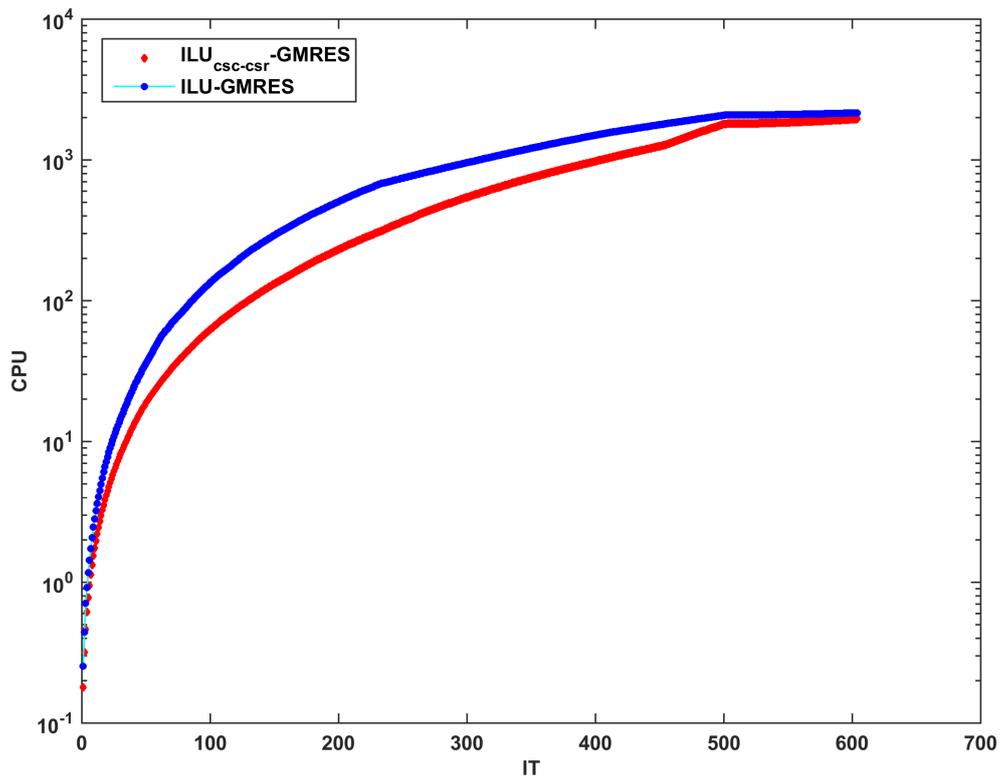


Fig. 2. Curves of the CPU time versus the number of iterations for t2em

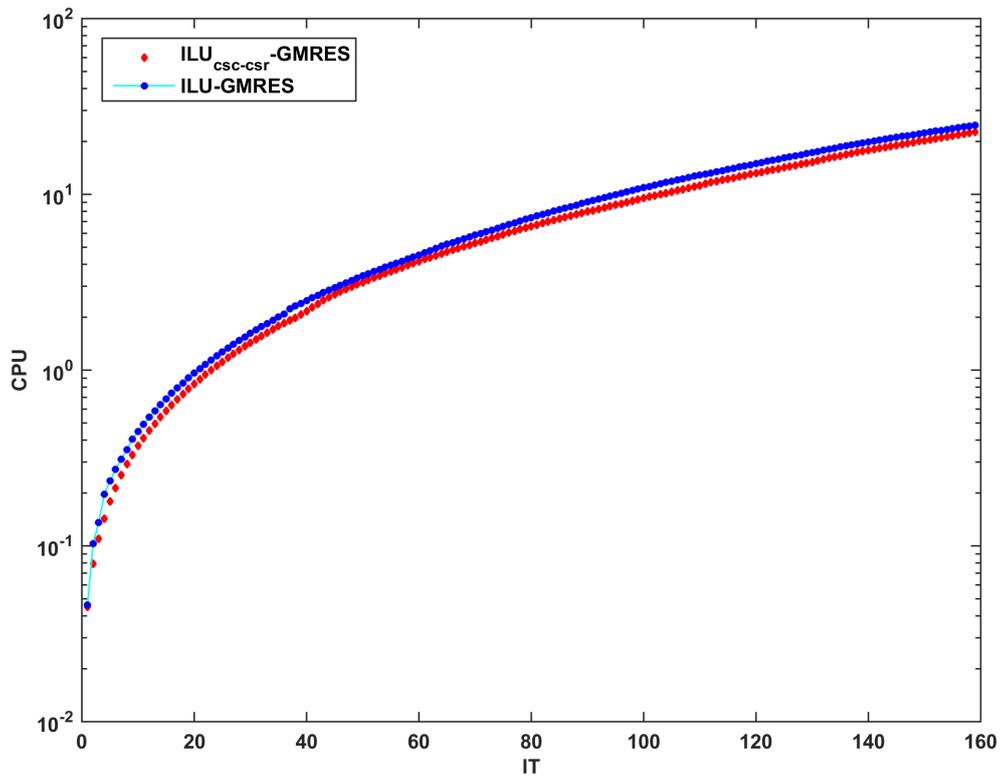


Fig. 3. Curves of the CPU time versus the number of iterations for trans5

6 Conclusion

The purpose of this work is to construct an ILU preconditioner using both CSR and CSC formats. Our numerical experiments have shown that ILU preconditioner using both CSR and CSC formats outperform the standard incomplete triangular factorization preconditioners on aspects of computation time. However, the number of iterations is identical when it is used to accelerate the Krylov subspace iteration methods such as GMRES for several problems. We note that the technique developed in this paper can be very useful for solving ill-conditioned linear systems resulting from the approximation of inverse problems (Mamiyeva, 2021; Ouaiassa et al., 2022; Rasheed et al., 2021; Sultanova, 2021) and presents a means of accelerating solvers proposed in Nachaoui (2004).

Further research, we will interesting to construct an ILU preconditioner using both CSR and CSC formats with some dropping rules, such as having zero entries in some positions or adding entries that are null before doing the incomplete LU factorization.

References

- Axelsson, O. (1996). *Iterative solution methods*. Cambridge University Press.
- Bai, Z.Z. (2000). Sharp error bounds of some Krylov subspace methods for non-Hermitian linear systems. *Applied Mathematics and Computation*, 109(2-3), 273-285.
- Bai, Z.Z., Golub, G.H., & Ng, M.K. (2003). Hermitian and skew-Hermitian splitting methods for non-Hermitian positive definite linear systems. *SIAM Journal on Matrix Analysis and Applications*, 24(3), 603-626.
- Bell, N., Garland, M. (2009). Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proc. Conference High Performance Computing Networking, Storage and Analysis*, SC'09, ACM, New York, NY, 14-19.
- Chen, J.Y., Kincaid, D.R., & Young, D.M. (1999). Generalizations and modifications of the GMRES iterative method. *Numerical Algorithms*, 21(1), 119-146.
- Davis, T.A., Hu, Y. (2011). The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1), 1-25.
- Duff, I.S., Erisman, A.M. & Reid, J.K. (2017). *Direct Methods for Sparse Matrices*. 2nd Edition, Oxford University Press, London.
- Gao, J., Xia, Y., Yin, R., & He, G. (2021). Adaptive diagonal sparse matrix-vector multiplication on GPU. *Journal of Parallel and Distributed Computing*, 157, 287-302.
- Greathouse, J.L., Daga, M. (2014, November). Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 769-780.
- Goharian, N., Jain, A., & Sun, Q. (2003). Comparative analysis of sparse matrix algorithms for information retrieval. *Computer*, 2, 0-4.
- Kraus, J., Lymbery, M. (2018). Incomplete factorization by local exact factorization (ILUE). *Mathematics and Computers in Simulation*, 145, 50-61.
- Li, K., Yang, W., & Li, K. (2014). Performance analysis and optimization for SpMV on GPU using probabilistic modeling. *IEEE Transactions on Parallel and Distributed Systems*, 26(1), 196-205.

- Li, L., Huang, T.Z., & Liu, X.P. (2007). Asymmetric Hermitian and skew-Hermitian splitting methods for positive definite linear systems. *Computers & Mathematics with Applications*, 54(1), 147-159.
- Mamiyeva, T. (2021). On Cauchy and boundary value problems for the third-order discrete-multiplicative derivative equation. *Advanced Mathematical Models & Applications*, 6(2) 174-181.
- Mellor-Crummey, J., & Garvin, J. (2004). Optimizing sparse matrix-vector product computations using unroll and jam. *The International Journal of High Performance Computing Applications*, 18(2), 225-236.
- Nachaoui, A. (2004). Numerical linear algebra for reconstruction inverse problems. *J. Comput. Appl. Math.*, 162(1), 147-164.
- Ouaissa, H., Chakib, A., Nachaoui, A., Nachaoui, M. (2022). On numerical approaches for solving an inverse Cauchy stokes problem. *Appl. Math. Optim.*, 85(1), 1-37.
- Rasheed, S.M., Nachaoui, A., Hama, M.F. & Jabbar, A.K. (2021). Regularized and preconditioned conjugate gradient like-methods methods for polynomial approximation of an inverse Cauchy problem. *Advanced Mathematical Models & Applications*, 6(2), 89-105.
- Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*, 2nd Edition, SIAM.
- Saad, Y., Schultz, M.H. (1986). GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3), 856-869.
- Sultanova, V. (2021). Construction of the adjoint problem to the discrete problems for the second order equation. *Advanced Mathematical Models & Applications*, 6(2), 182-188.
- Van der Vorst, H.A. (2003). *Iterative Krylov Methods for Large Linear Systems* (No. 13). Cambridge University Press.
- Zhong-Zhi, B.A.I. (2007). Splitting iteration methods for non-Hermitian positive definite systems of linear equations. *Hokkaido Mathematical Journal*, 36(4), 801-814.