A COST-EFFECTIVE ILU PRECONDITIONER FOR WEATHER SIMULATION

GUANGHUI WANG, YANHUA CAO, AND XIANGQIAN GU

Abstract. To date, the most efficient solver used in the weather sciences for the resolution of linear systems in numerical weather prediction is the generalized minimal residual method called GMRES. However, difficulties still appear in matrix resolution when the GMRES iterative method is used without an appropriate preconditioner. For improving the computation speed in numerically solving weather equations, we investigate the use of Incomplete LU (ILU) factorization preconditioner. This is a fast algorithm for solving large scale linear equations. Some strategies for choosing the level of fill and threshold are described in this paper. As an example, we use the GMRES iterative algorithm to solve the finite difference equations of shallow equations and analyze the results that are obtained in preconditioned and un-preconditioned, respectively. It is shown that the computation speed is greatly improved by using ILU factorization preconditioner for the linear system. In addition, this preconditioning algorithm is simple and parallelizable. Therefore, the algorithm is potential in the applications of weather equations.

Key Words. Incomplete LU factorization; preconditioner; shallow water equations; threshold; GMRES iterative algorithm.

1. Introduction

Atmospheric motions are described by coupled nonlinear partial differential equations. The equations not only describe the dynamical processes such as fluid motion, but also the physical processes such as atmospheric radiation, chemistry, etc. (cf.[1]). Because few analytical solution exist, in order for solution to be obtained, the equation are usually discretized and solved numerically. The early primitive equation models, based on explicit time-difference schemes, were computationally very inefficient, requiring a large expenditure of computer time for a given level of accuracy. This resulted from the Courant—Friedrichs—Lewy (CFL) condition for the high
frequency motion, which required the use of time step that were enough small. This results in an algebraic system of equations with a large and sparse matrix.

(1) \[ Ax = b \]

where \( A \in \mathbb{R}^{n \times n} \) is nonsingular, \( x \in \mathbb{R}^n \), and \( b \in \mathbb{R}^n \). The obtained linear system is aimed to be solved with as small computational effort and memory demand as possible. For really large problems \((n > 500000)\), the only way to achieve this is to use an optimal, robust, preconditioned, iterative solution method. Below, the particular meaning of the terminology is explicitly stated (cf. [5]).

(i) Robustness means that the iterative solver converges independently of the parameters of the underlying problem (such as the Poisson number in elasticity problems and the viscosity in fluid dynamics).

(ii) For the iterative method to be optimal, its rate of convergence, i.e. the number of iterations required for the method to converge, must be independent of the size of \( A \).

(iii) Furthermore, in order to handle large scale applications, the iterative solution method should work fast in terms of CPU-time. To achieve this, the iterative solution method must be numerically efficient (few arithmetic operations per unknown)

Generally speaking, there are two approaches to precondition solving system. One popular approach in applications involving PDEs (partial differential equations) is to design specialized algorithms that are optimal (or nearly so) for a narrow class of problems. It is a physics-based preconditioning method (cf. [10], [12], [14]). This application-specific approach can be very successful, but it may require complete knowledge of the problem at hand, including the original (continuous) equations, the domain of integration, the boundary conditions, details of the discretization, and so forth. By making use of available information about the analysis, geometry, and physics of the problem, very effective pre-conditioners can be developed. As pointed out in [13], the method of diffusion synthetic acceleration (DSA) (cf. [2], [11], [17]), which is widely used in the transport community, can be regarded as a physics-based pre-conditioner for the transport equation. Also, multi-grid pre-conditioners are often of this kind (see [10], [12] and [13] for a recent example).

Another preconditioning approach is based directly on linear algebra equations, which is referred as algebraic preconditioning method. It is well known that the rate of convergence of iterative methods for solving (1) is strongly influenced by the spectral properties of matrix \( A \). Preconditioning amounts to transforming the original system into one having the same solution but more favorable spectral properties, such as a clustering of the eigenvalues around 1. A pre-conditioner is a matrix that can be used to accomplish such a transformation. If \( M \) is a nonsingular matrix which approximates \( A^{-1} (M \approx A^{-1}) \), the transformed linear system

(2) \[ MAx = Mb \]
will have the same solution as system (1) but the convergence rate of iterative methods applied to (2) may be much higher. Problem (2) is preconditioned from the left, but right preconditioning is also possible. Preconditioning on the right leads to the transformed linear system

\[ AMy = b \]

In order to construct efficient pre-conditioner \( M \), preconditioning techniques based on Incomplete LU factorization, spare approximate inverses and multi-grid method etc. have been vigorously developed in recent years (cf.[4],[6]-[8],[13],[15],[19],[20]) . Specially, incomplete LU factorization was widely used. The common idea underlying these classes of algorithms are that a spare matrix \( M \approx A^{-1} \) is explicitly computed and used as a preconditioner for Krylov subspace methods (cf.[3],[16]) for the solution of equations (1). The main advantage of these approaches are that the reconditioning operation can easily be implemented in parallel, since it consists of matrix-vector products. Further, the construction and application of pre-conditioners of the approximate inverse type tend to be immune from such numerical difficulties as pivot breakdowns and instability. Approximate inverse techniques rely on the assumption that for a given spare matrix \( A \), it is possible to find a spare matrix \( M \) which is a good approximation, in some sense, of \( A^{-1} \). Since the inverse of a spare matrix is usually dense, this means that for a given irreducible sparsity pattern, it is always possible to assign numerical value to the non-zeros in such a way that all entries of the inverse will be non-zero. Nevertheless, it is often the case that many of the entries in inverse of a spare matrix are small in absolute value, thus making the approximation of \( A^{-1} \) with a spare matrix possible. In this paper, we use the method of [15] to compute a spare matrix \( L \) and \( U \) such that \( (LU)^{-1} \approx A^{-1} \) as the pre-conditioner of the linear system, the sparsity patterns of \( L \) and \( U \) are based on matrix \( A \) and a threshold strategy(cf.[15]). Finally, we apply this preconditioning to shallow water problem, finding that it is very efficient.

The remainder of this paper is organized as follows. In section 2, Incomplete LU factorization methods with threshold and precondition technique are described. Sections 3 is concerned with the application of pre-conditioners to shallow water problem. Numerical results, which include the Krylov iterative number, CPU time and nonzero elements etc., are compared with and without pre-conditioner. A few concluding remarks are given in section 4.

2. ILU Factorization Preconditioners

With the realization that preconditioning is essential for the successful use of iterative methods. Incomplete factorization methods were introduced for the first time by Buleev in the then-Soviet Union in the late 1950s, and independently by Varga (see [18]). Since then, a number of improvements and extensions have been made, including level-of-fills and drop tolerance-based incomplete factorizations, generalizations to block matrices, modified
and stabilized variants, and even efficient parallel implementations([3],[7],[9],[19],[20], etc.).

2.1. Incomplete LU Factorization Methods.

When a sparse matrix is factored by Gaussian elimination, fill-in usually takes place. This means that the triangular factors \( L \) and \( U \) of the coefficient matrix \( A \) are considerably less sparse than \( A \). Even though sparsity-preserving pivoting techniques can be used to reduce fill-in, sparse direct methods are not considered viable for solving very large linear systems such as those arising from the discretization of three-dimensional boundary value problems, due to time and space constraints. However, by discarding part of the fill-in in the course of the factorization process, simple but powerful preconditioners can be obtained in the form \( M = (LU)^{-1} \), where \( L \) and \( U \) are the incomplete (approximate) \( LU \) factors. The precondition matrix is created by the following:

\[
A = LU + R
\]
\[
M = (LU)^{-1}
\]

where, \( L(U) \): lower(upper) triangular matrix with the spare structure by using s threshold strategy. \( R \): error matrix . \( M \): preconditioner.

A general algorithm for building \( ILU \) factorization can be derived by performing \( Gaussian \) elimination and dropping some element in predetermined non-diagonal positions. As a result, the \( ILU \) factorization has fewer non-zero element. This can be done statically by choosing some nonzero pattern in advance. The only restriction on the zero pattern is that it should exclude diagonal element. Therefore, for any zero pattern set \( P \) such that

\[
P \subset \{(i,j)|i \neq j; 1 \leq i,j \leq n}\}
\]

An \( ILU \) factorization can be computed as follows.

**Algorithm 1 General ILU** (cf.[4])

1. For \( i=2,...,n \), Do
2. For \( k=1,...,i-1 \) and if \((i,j) \in P\), Do
3. \( a_{ik} := a_{ik}/a_{kk} \)
4. For \( j=k+1,...,n \) and for \((i,j) \in P\), Do
5. \( a_{i,j} := a_{i,j} - a_{ij} - a_{ik} \times a_{kj} \)
6. End Do
7. End Do
8. End Do

The \( ILU \) factorization technique with no fill-in, denoted by \( ILU(0) \), takes the zero pattern \( P \) to be precisely the zero pattern of \( A \). In the following, we denote by \( b_{i,*} \) the \( i \)th row of a given matrix \( B \) and by \( NZ(B) \) the set of pairs \((i,j)\), \( 1 \leq i,j \leq n \), such that \( b_{i,j} \neq 0 \). Consider now any lower triangular matrix \( L \) with the same structure as the lower part of \( A \) and any matrix \( U \) with the same structure as that of the upper part of \( A \). If the product \( LU \) were performed, it is impossible in general to match \( A \) with this
product for any $L$ and $U$. This is due to the extra diagonals in the product. The entries in these extra diagonal are called \textit{fill-in element}. However, if these fill-in element are ignored, then it is possible to find $L$ and $U$ so that their product is equal to $A$ in the other diagonals. The $ILU(0)$ factorization has just been defined in general terms as any pair of matrices $L$ (unit lower triangular) and $U$ (upper triangular) so that the element of $A - LU$ are zero in the locations of $NZ(A)$.

The accuracy of the $ILU(0)$ incomplete factorization may be insufficient to yield an adequate rate of convergence. More accurate $ILU(\rho)$ factorization are often more efficient as well as more reliable, where parameter $\rho$ ( $\rho$: positive integer) denotes the order of fill-in element. These more accurate factorization differ from $ILU(0)$ by allowing some fill-in. It is a preconditioned technique that the $\rho$-order fill-in element are allowed in the $ILU$ factorization, and $ILU(\rho)$ factorization gives better condition number than the $ILU(0)$ factorization. For all that, there are a number of drawbacks to the $ILU(\rho)$ factorization algorithm. First, the amount of fill-in and computational work for obtaining the $ILU(\rho)$ factorization is not predictable for $\rho > 0$. Second, the cost of updating the level can be high. Most importantly, the level of fill-in for indefinite matrices may not be a good indicator of the size of the elements that are being dropped. Thus, the algorithm may drop large elements and result in an inaccurate incomplete factorization, in that $R = LU - A$ is not small. The techniques that are described in following section will be used to remedy these difficulties by producing incomplete factorization with small error $R$ and controlled fill-in.

2.2. Threshold Strategies and Incomplete \textit{LU} with Threshold.

Incomplete factorization that rely on the level of fill are blind to numerical value because elements that are dropped depend only on the structure of $A$. This can cause some difficulties for realistic problems that arise in many applications. A few alternative methods are available based on dropping elements in the Gaussian elimination process according to their magnitude rather than their locations. with these techniques, the zero pattern $P$ is determined dynamically. A generic $ILU$ algorithm with threshold ($ILUT$) can be derived from the Gaussian elimination process by including a set of rules for dropping small elements. Applying a dropping rule to an element will only mean replacing the element with zero if it satisfies a set of criteria. A dropping rule can be applied to a whole row by applying the same rule to all the elements of the row. In the following algorithm, $w$ is a full-length working row used to accumulate linear combinations of spare rows in the elimination and $w_k$ is the $k$th entry of this row. As usual, $a_{i\ast}$ denotes the $i$th row of $A$.

\textbf{Algorithm 2 ILUT} (cf.[15])

1. For $i=1,...,n$, Do
2. \hspace{1cm} $w := a_{i\ast}$
3. \hspace{1cm} For $k=1,...,i-1$ and when $w_k \neq 0$, Do
4. \hspace{2cm} $w_k := w_k / a_{kk}$
5. \hspace{1cm} Apply a dropping rule to $w_k$
6. If $w_k \neq 0$ then
7. \[ w := w - w_k \cdot u_k \]
8. End If
9. End Do
10. Apply a dropping rule to row $w$
11. \[ l_{i,j} := w_j \text{ for } j = 1, \ldots, i-1 \]
12. \[ u_{i,j} := w_j \text{ for } j = i, \ldots, n \]
13. \[ w := 0 \]
14. End Do

$ILU(0)$ can be viewed as a particular case of the above algorithm. The dropping rule for $ILU(0)$ is to drop elements that are in positions not belonging to the original structure of the matrix. In the factorization $ILUT(\rho, \tau)$, the following rules are used:

(i) In line 5, an element $w_k$ is dropped (i.e., replaced with zero) if it is less than the relative tolerance $\tau_i$ obtained by multiplying $\tau$ by the original norm of the $i$th row (e.g., the 2-norm).

(ii) In line 10, a dropping rule of a different type is applied. First, drop again any element in the row with a magnitude that is below the relative tolerance $\tau_i$. Then keep only the $\rho$ largest elements in the $L$ part of the row and the $\rho$ largest elements in the $U$ part of the row in addition to the diagonal element, which is always kept.

The goal of the second dropping step is to control the number of elements per row. Roughly speaking, $\rho$ can be viewed as a parameter that helps control memory usage, while $\tau$ helps reduce computational cost.

### 2.3. Precondition Technique.

In this section, the precondition technique is described. Overall performance of Krylov methods can be improved when one uses preconditioning. Instead of solving the system $Ax = b$, the system

\[
L^{-1}Au^{-1}(Ux) = L^{-1}b
\]

is solved where $M^{-1} = U^{-1}L^{-1}$ is an approximation to $A^{-1}$ and is easier to compute. Evidently, equation (6) can be translated into

\[
\tilde{A}y = \tilde{b}
\]

where $\tilde{A} = L^{-1}AU^{-1}$, $y = Ux$, $\tilde{b} = L^{-1}b$. Since only matrix-vector products are needed for the GMRES method, it is not necessary to explicitly form $L^{-1}AU^{-1}$. Only two new computational steps "solve $u$ from $\tilde{M}u = v$" are introduced.

To solve equation $Ax = b$, we solve firstly $y$ from equation (7) by the following GMRES($m$) algorithm, and then solve $x$ from equation $Ux = y$.

**Algorithm 3**: GMRES($m$) (restart GMRES iterative method (cf.[16]))

1. Start: Choose $x_0$ and compute $r_0 = \tilde{b} - \tilde{A}x_0$ and $v_1 = r_0/\|r_0\|$.
2. Iterate: for $j = 1, 2, \ldots, m$ do:
   - $h_{i,j} = (\tilde{A}v_j, v_i)$, $i = 1, 2, \ldots, j$,
   - $\hat{v}_{j+1} = \tilde{A}v_j - \sum_{i=1}^{j} h_{i,j}v_i$,
   - $\hat{h}_{j+1} = \|\hat{v}_{j+1}\|_2$. 

3. If \( h_{j+1,j} < \varepsilon \) or \( j = m \), then go to 4, else compute \( v_{j+1} = \tilde{v}_{j+1}/h_{j+1,j} \) and go to 2.

4. Form the approximate solution:
\[
y_m = y_0 + V_m \xi_m \text{ minimizes } \| \beta e_1 - \tilde{H}_m \xi \|, \; \xi \in R^m.
\]

5. Restart:
- Compute \( r_m = \bar{b} - \bar{A}y_m \); if \( \| r_m \|_2 < \varepsilon \) is satisfied then stop
- else compute \( y_0 := y_m, \; v_1 := r_m/\| r_m \| \) and go to 1.

3. Simulation

In this section we will detail how a simple pre-conditioner is incorporated into the iterative solving procedure as well as illustrate the application of ILU preconditioning techniques on the one-dimensional shallow water equations, and in this we will show results from the shallow water model.

3.1. Shallow water model.

For our demonstration of Spare approximate inverse approach we have chosen a simple model problem that solves the one-dimensional shallow-water equations in flux form with no bottom topography (cf., [14]).

We have added additional forcing terms that lead to a closed-form analytical solution. The equations are

\[
\begin{aligned}
\frac{\partial h}{\partial t} + \frac{\partial u h}{\partial x} &= f_h, \\
\frac{\partial u}{\partial t} + \frac{\partial u u h}{\partial x} &= -gh\frac{\partial h}{\partial x} + uf_h + hf_u.
\end{aligned}
\]

where \( h \) is the height of the fluid, \( u \) is the velocity of the fluid, \( g \) is the acceleration due to gravity, and

\[
\begin{aligned}
f_h &= (a_h \omega - a_u h_0 k) \sin (kx + \omega t) + a_h a_u \sin^2 (kx + \omega t), \\
f_u &= (a_h g k - a_u \omega) \sin (kx + \omega t) - 0.5a_h^2 k \sin^2 (kx + \omega t).
\end{aligned}
\]

and equation (8) have the analytic solutions

\[
\begin{aligned}
u &= a_u \cos (kx + \omega t), \\
h &= h_0 - a_u \cos (kx + \omega t).
\end{aligned}
\]

where \( h_0 \) is the average height of the initial wave, \( \omega \) is the frequency of the wave, and \( k \) is the wave-number. Note, the analytical solution are used as initial conditions (set \( t = 0 \)) and periodic boundary conditions are employed in this simple model.

3.2. Fully-implicity difference scheme.

In order to numerically solve differential (8), we use the fully-implicity difference scheme

\[
\begin{aligned}
\frac{h_i^{n+1} - h_i^n}{\Delta t} + \frac{(uh)_{i+1}^{n+1} - (uh)_{i-1}^{n+1}}{2\Delta x} &= f_h^i, \\
\frac{(uh)_{i}^{n+1} - (uh)_{i}^{n}}{\Delta t} + \frac{2\Delta x}{u_{i+1}^{n}(uh)_{i+1}^{n+1} - u_{i-1}^{n}(uh)_{i-1}^{n+1}} &= 0, \\
\frac{\Delta t}{h_i^{n+1} - h_i^{n-1}} + \frac{2\Delta x}{u_{i}^{n}f_h^{n} + h_i^{n}f_u^{n}} &= -gh_i^{n}.
\end{aligned}
\]
We choose the following parameters for our numerical computation. \( g = 1.0ms^{-2}, h_0 = 1.0m, a_h = 0.01m, a_u = 0.01ms^{-1}, k = 2m^{-1}, \) and \( \omega = 0.0s^{-1}. \ x \in [0, 2\pi], t \in [0, 2\pi]. \)

Following [17], we ran the simulations at successively finer resolution while maintaining a fixed Courant number \((\Delta t/\Delta x)\) with each successive \(\Delta t\) and \(\Delta x\) being reduced by a factor of 2. For each simulation, we calculate the truncation error

\[
\text{Error}(N_t, N_x) = \left( \sum_{i=1}^{N_t} \sum_{j=1}^{N_x} (\psi(t_j, x_i) - \psi^*_j)^2 \right)^{1/2}
\]

where \(N_t = T/\Delta t, N_x = L/\Delta x\) with \(T = 2\pi(s)\) and \(L = 2\pi(m)\), and \(\psi(x_i, t_j), \psi^*_j\) are the analytical and numerical solution, respectively, at the point \((x_i, t_j)\).

### 3.3. Simulation results.

In this section, we shall compare the results with and without preconditioner.

We rewrite fully implicit difference equations (10)

\[
\begin{align*}
  h_i^{n+1} + k(P_{i+1}^{n+1} - P_{i-1}^{n+1}) &= h_i^n + \Delta t \frac{P_i^n}{h_i}, \\
  P_i^{n+1} + k(u_i^{n+1}P_{i+1}^{n+1} - u_i^{n-1}P_{i-1}^{n+1}) + gh_i k(h_{i+1}^{n+1} - h_{i-1}^{n+1}) &= d_i^n
\end{align*}
\]

where \(P_i^n = (uh)_i^n, \ d_i^n = P_i^n + \Delta t(u_i^n f_{hi}^n + h_i^n f_{ui}^n), \ k = \Delta t/(2\Delta x).\)

We shall solve \(h_i^{n+1}\) and \(P_i^{n+1}\) from (12) firstly, then solve \(u_i^{n+1}\) from \(P = uh\). Evidently, the coefficient matrix of linear equations (12) is a non-symmetry matrix. Firstly, we write equations (12) into the form of equation (6), and then use Algorithm 2 to get the incomplete LU factorization for some given thresholds \((\tau = 10^{-2}, 10^{-3}, 10^{-4})\), here we keep only the 20 largest elements in the \(L\) part of the row and the 20 largest elements in the \(U\) part of the row in addition to the diagonal element. Finally, we solve \(y\) and \(x\) from equation (7) by using Algorithm 3. Here, iteration matrix is \(L^{-1}AU^{-1}\).

In this paper, we assume that the restart parameter \(m = 10\), convergence parameter in \(GMRES(m), \varepsilon = 10^{-12}\). In order to demonstrate the effectiveness of Incomplete LU factorization method, we have a comparison about average number of Krylov iterations per time-step, total CPU time and the number of nonzero elements of \(L\) and \(U\) with and without preconditioner.

### TABLE. A test run of \(GMRES(10) - ILUT(20, \tau)\) preconditioning

<table>
<thead>
<tr>
<th>Systems</th>
<th>Preconditioner</th>
<th>(\tau = 10^{-2})</th>
<th>(\tau = 10^{-3})</th>
<th>(\tau = 10^{-4})</th>
<th>No preconditioner</th>
</tr>
</thead>
<tbody>
<tr>
<td>100×200</td>
<td>Times [min]</td>
<td>0.547</td>
<td>0.531</td>
<td>0.531</td>
<td>0.688</td>
</tr>
<tr>
<td></td>
<td>Iteration counts</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Nonzero elements</td>
<td>596</td>
<td>664</td>
<td>862</td>
<td></td>
</tr>
<tr>
<td>200×400</td>
<td>Times [min]</td>
<td>2.188</td>
<td>2.094</td>
<td>2.141</td>
<td>2.734</td>
</tr>
<tr>
<td></td>
<td>Iteration counts</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Nonzero elements</td>
<td>1196</td>
<td>1336</td>
<td>1724</td>
<td></td>
</tr>
<tr>
<td>400×800</td>
<td>Times [min]</td>
<td>5.969</td>
<td>5.859</td>
<td>5.797</td>
<td>9.563</td>
</tr>
<tr>
<td></td>
<td>Iteration counts</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Nonzero elements</td>
<td>2396</td>
<td>2693</td>
<td>3508</td>
<td></td>
</tr>
</tbody>
</table>
TABLE show the comparison about total CPU time, iterative number per time-step and non zero elements number after using threshold $\tau$ with the increments of number of halvings of the spatial and the temporal. The iterative number is the average Krylov iterative numbers per time-step. Looking at the results in the $100 \times 200$ system, the case $\tau = 10^{-3}$ (or $10^{-4}$) give the result that reduced the computational time of No preconditioner by 22%. Next, concerning the $200 \times 400$, the $ILU(10^{-3})$ preconditioning method outperforms others. It succeed in reducing the computational effort of no preconditioner by 23%. Furthermore, regarding the $400 \times 800$ system, $LIU(10^{-4})$ preconditioning method was faster than others. It saved 40% of computational time for No preconditioner. In addition, the $ILU(10^{-4})$ preconditioning method in above three systems reduced the iterative number per time-step of no preconditioner by 78%. Therefore, the simulation results have shown that the better results can be obtain with the increasing of temporal-spatial grids.

4. Concluding remarks

Preconditioning based on incomplete $LU$ factorization with threshold can lead to very successful parallel or sequential iterative procedures for solving general spare linear system. In this paper, we examined a class of preconditioning methods for spare linear system arising from finite difference discretization of shallow water problem. The results have shown that the computation speed is greatly improved after we use the preconditioning method and this preconditioning algorithm is simple and parallelizable.

References


State Key Laboratory of Severe Weather (LaSW), Chinese Academy of Meteorological Sciences, Beijing 100081, China
E-mail: wanggh@cams.cma.gov.cn

School of Mathematics and Physics, North China Electric Power University, Beijing 102206, China
E-mail: cao_yanhua@ncepu.edu.cn

State Key Laboratory of Severe Weather (LaSW), Chinese Academy of Meteorological Sciences, Beijing 100081, China
E-mail: guxq@cams.cma.gov.cn