PARALLEL APPROXIMATE LU FACTORIZATIONS FOR SPARSE MATRICES

THOMAS K. HUCKLE *

April 9, 2019

Abstract. In this paper we present iterative methods for computing sparse LU, incomplete LU and modified incomplete LU factorizations for a sparse matrix A. E. Chow showed that the underlying fixed point iteration $x = \Phi(x)$ allows a parallel computation of the factorizations; here, x is a vector of the unknown entries of L and U in some ordering. Furthermore, we introduce and analyze Newton's method replacing the basic fixed point iteration by the problem of finding a zero of $f(x) = \Phi(x) - x$. This leads to fast convergence by additional costs for solving a linear system in each iteration step. Fortunately, the underlying matrix - the Jacobi matrix of derivatives of $f(x)$ - can be shown to be lower triangular with appropriate ordering of the unknowns in the vector x. This allows fast and cheap solution methods.

Key words. preconditioning, incomplete LU factorization, parallel computing

AMS subject classifications. 65F08, 65Y05, 65F50, 65F10

1. Introduction. In many applications a sparse linear system of n equations $Ax = b$ has to be solved on a parallel architecture. Iterative solution methods are efficient in a parallel environment as long as the number of necessary iterations can be limited. Therefore, a preconditioner has to be applied, in the form $P^{-1}Ax = P^{-1}b$, e.g. [8]. Often, the Incomplete LU factorization ILU ([6]) yields a good preconditioner, but—like the Gaussian Elimination algorithm itself— it is inherently sequential. In the classical derivation, the ILU is considered as a restriction of the Gaussian Elimination process to a certain sparsity pattern S delivering an approximate LU factorization $A \approx LU$. Note, that for LU factorizations, usually the diagonal entries of L are chosen as 1. In [1] E. Chow has introduced an iterative approach based on a fixed point equation to derive good approximations for the ILU factors. This approach uses the other property of the ILU, namely $(A = LU)_S$ on pattern S; then A and LU coincide on the chosen sparsity pattern S, and also L and U are restricted to pattern S. This gives rise to a fixed-point iteration in a quadratic function in the unknown entries of L and U. Via this fixed-point iteration, the entries of L and U can be computed iteratively. By allowing asynchronous updates, every update step can be done in parallel. Furthermore, often only a few iterations are necessary to derive good approximations on the exact L and U factors of the ILU. The parallel algorithm is described by Algorithm 1 IILU.

Note that - in view of the sparsity - in all the summations here and in the following, the sum is always taken only over the nonzero entries. Furthermore, in the following we assume that S is the pattern of A, and the triangular parts of S are the pattern of L, and U, resp.

Similarly, for A symmetric positive definite, we can formulate an iterative algorithm for approximating the Incomplete Cholesky Decomposition $A = L \cdot D \cdot L'$ with lower triangular L and diagonal D (Algorithm 2 IICHOLD) or $A = U'U$ with upper triangular U (Algorithm 3 IICHOL).

For many matrices the above method gives meaningful estimates for L and U, resp. L and D, in a few iterations. But if one is interested in exact sparse LU

* Department of Computer Science, Technical University of Munich, Boltzmannstr. 3, 85748 Garching, Munich, Germany (huckle@in.tum.de).

Algorithm 1 Iterative ILU (Chow), IILU

Set unknowns $l_{i,j}$ and $u_{i,j}$ to initial values - Gauss-Seidel factors, e.g.
for sweep = 1,2,... until convergence **do**
  parallel for $(i,j) \in S$ **do**
    if $i > j$ **then**
      $l_{i,j} = \frac{1}{u_{j,j}}(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} \cdot u_{k,j})$
    else
      $u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} \cdot u_{k,j}$
    end if
  end parallel for
end for

Algorithm 2 Iterative ICHOLD, IICHOLD

Set unknowns $l_{i,j}$ of L and d_j of D to initial values - Gauss-Seidel factors, e.g.
for sweep = 1,2,... until convergence **do**
  parallel for $(i,j) \in S, i \geq j$ **do**
    $s = a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} \cdot d_k \cdot l_{k,j}$
    if $i \neq j$ **then**
      $l_{i,j} = s/d(j)$
    else
      $d_i = s$
    end if
  end parallel for
end for

factorizations or for very ill-conditioned A, too many iterates might be necessary to derive reasonable approximations for L and U. This can be especially the case in computing the modified ILU (MILU), see section 3.

In the next section we discuss different one-dimensional orderings of the unknowns in A, L, and U. In the third part we introduce fixed-point iterations for computing the modified ILU. In the fourth section we accelerate all the basic fixed-point iterations by Newton's method for deriving sparse LU, ILU, and MILU factorizations. Numerical examples are included in section 3 and 4.

2. One-dimensional orderings of the entries of a sparse matrix. In the following, different one-dimensional orderings of the nonzero entries of a sparse matrix A, resp. a sparse triangular factor L and U are important. We mainly assume three possibilities:

- Columnwise ordering in A, L, and U:

$$\begin{pmatrix} 1 & k+1 & 2k+1 & \cdots \\ 2 & k+2 & \downarrow & \cdots \\ \downarrow & \downarrow & \downarrow & \cdots \\ k & 2k & \vdots & \cdots \end{pmatrix}, \begin{pmatrix} 1 & & & \\ 2 & k+1 & & \\ \downarrow & \downarrow & 2k & \\ k & 2k-1 & \downarrow & \cdots \end{pmatrix}, \begin{pmatrix} 1 & 2 & 4 & \cdots \\ & 3 & 5 & \cdots \\ & & 6 & \cdots \\ & & & \vdots \end{pmatrix}.$$

Algorithm 3 Iterative ICHOL (Chow), IICHOL

Set unknowns $u_{i,j}$ of U to initial values - Gauss-Seidel factors, e.g.
for sweep = 1,2,... until convergence **do**
  parallel for $(i,j) \in S, i \leq j$ **do**
    $s = a_{i,j} - \sum_{k=1}^{i-1} u_{i,k} \cdot u_{k,j}$
    if $i \neq j$ **then**
      $u_{i,j} = s/u_{i,i}$
    else
      $u_{i,i} = sqrt(s)$
    end if
  end parallel for
end for

- Rowwise ordering in A, L, and U:

$$\begin{pmatrix} 1 & 2 & \rightarrow & k \\ k+1 & k+2 & \rightarrow & 2k \\ 2k+1 & \rightarrow & \rightarrow & \cdots \\ \vdots & \vdots & \vdots & \cdots \end{pmatrix}, \begin{pmatrix} 1 & & & \\ 2 & 3 & & \\ 4 & 5 & 6 & \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}, \begin{pmatrix} 1 & 2 & \rightarrow & k \\ & k+1 & \rightarrow & 2k-1 \\ & & 2k & \cdots \\ & & & \vdots \end{pmatrix}.$$

- Mixed ordering in A:

$$\begin{pmatrix} k & \leftarrow & \leftarrow & 2 & 1 \\ k+1 & 2k & \leftarrow & k+3 & k+2 \\ 2k+1 & 2k+2 & 3k & \leftarrow & 2k+3 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \rightarrow & \rightarrow & \cdot & \leftarrow & \leftarrow \end{pmatrix}.$$

  In the mixed ordering we start with the first row (only entries related to U, no entries to L, because the main diagonal entries of L are all 1) and start numbering from the right. In the second row, the first number is given to the left entry related to L, and the next numbers again related to U, starting from the right. Hence, in every row the first numbers are given to entries on the left in L until the main diagonal, followed by entries in U starting from the right and numbering in the left direction.

In Algorithm 1, for updating an entry $l_{i,j}$ we need the entries $l_{i,k}$ for $k = 1, .., j-1$, and the entries $u_{k,j}$ for $k = 1, ..., j$. By a one-dimensional ordering of the unknowns x in L and U, resp. in A, we want to achieve a triangular dependency of the entries in the derivative, the Jacobi matrix J of the underlying iteration function $\Phi(x)$. Therefore, $l_{i,k}, k = 1, .., j-1$, and $u_{k,j}, k = 1, ..., j$ should have lower indices in the one-dimensional vector than $l_{i,j}$. This is achieved by columnwise or rowwise ordering of the nonzero entries A. In the same way, for updating $u_{i,j}$ we need $l_{i,k}$ and $u_{k,j}$ for $k = 1, .., i-1$ which should have index smaller than $u_{i,j}$.

In general, to derive at a triangular dependency in $\Phi'(x)$, the one-dimensional ordering has to have the following properties:

- For updating $l_{i,j}$, all entries strictly to the left in $L(i,:)$ and above in $U(:,j)$ have to have lower index than $l_{i,j}$.
- For updating $u_{i,j}$, all entries to the left in $L(i,:)$ and strictly above in $U(:,j)$ have to have lower index than $u_{i,j}$.

These two conditions are satisfied by all the above orderings (columnwise, rowwise, and mixed). Therefore, all three orderings lead to an iteration function $\Phi(L, U)$ with derivative Jacobi matrix a strictly lower triangular matrix. See also Fig. 4.1 (a), (b), and (c).

In the symmetric case (Algorithm 3) , we only need unknowns in U (resp. in L). For updating an entry $u_{i,j}$ we need the elements at the i-th row (strictly left of (i, j), and the j-th column (strictly above (i, j)) in U. Hence, the column- and rowwise ordering in U again guarantees a strictly lower triangular form of the derivative of $\Phi(U)$. The same holds true for Algorithm 2 in L and D. Note, that the mixed ordering is not defined for the symmetric case where we consider only the lower or the upper triangular entries.

With the appropriate ordering in strictly lower triangular form, the convergence proofs introduced in [1] can be applied also on the MILU case.

3. Iterative MILU. The modified ILU (MILU) is derived by adding the condition $(A - LU)e = 0$ with $e = (1, 1, ..., 1, 1)^T$. The vector e often represents the ill-conditioned subspace of A. Therefore, forcing the preconditioner to be excact on this subspace can improve the quality of the preconditioner. We can add this additional condition explicitly by replacing the update of the diagonal entries $u_{j,j}$ via the condition

$$L(j, 1 : j)U(1 : j, :)e = A(j, :)e \ .$$

This leads to Algorithm 4 IMILU1.

Algorithm 4 Iterative MILU algorithm 1, IMILU1

```
Set unknowns l_{i,j} and u_{i,j} to initial values
ae = a · e, ue = u · e
for sweep = 1,2,... until convergence do
  parallel for (i, j) ∈ S do
    if i > j then
      l_{i,j} = 1/u_{j,j} (a_{i,j} − Σ_{k=1}^{j−1} l_{i,k} u_{k,j})
    else
      ua = u_{i,j}
      if i < j then
        u_{i,j} = a_{i,j} − Σ_{k=1}^{i−1} l_{i,k} u_{k,j}
      else
        u_{j,j} = ae_j − Σ_{k=j+1}^{n} u_{j,k} − Σ_{k=1}^{j−1} l_{j,k} ue_k
      end if
      ue_i = ue_i − ua + u_{j,j}
    end if
  end parallel for
end for
```

In this algorithm we can save costs by computing $ae := A \cdot e$ in the beginning, and updating $ue := U \cdot e = sum(U)$ for each change in $u_{i,j}$. This allows to apply for the main diagonal entries the update used in Algorithm 4 in the form

$$u_{j,j} = ae_j - \sum_{k=j+1}^{n} u_{j,k} - \sum_{k=1}^{j-1} l_{j,k} ue_k \ .$$

The equation for $u_{j,j}$ is derived by the transformations

$$sum(A(j,:)) = \sum_{k=1}^{n} a_{j,k} = A(j,:) \cdot e =: L(j,:) \cdot ue = L(j,:) \cdot \begin{pmatrix} \sum_{r=1}^{n} u_{1,r} \\ \vdots \\ \sum_{r=n}^{n} u_{n,r} \end{pmatrix} =$$

$$= L(j,1:j-1) \cdot \begin{pmatrix} \sum_{r=1}^{n} u_{1,r} \\ \vdots \\ \sum_{r=j-1}^{n} u_{j-1,r} \end{pmatrix} + \sum_{k=j}^{n} u_{j,k} =$$

$$= L(j,1:j-1) \cdot ue(1:j-1) + \sum_{k=j}^{n} u_{j,k} =$$

$$= \sum_{k=1}^{j-1} l_{j,k} \sum_{r=k}^{n} u_{k,r} + u_{j,j} + \sum_{k=j+1}^{n} u_{j,k} \; .$$

This leads to defining

$$u_{j,j} := \sum_{k=1}^{n} a_{j,k} - \sum_{k=j+1}^{n} u_{j,k} - \sum_{k=1}^{j-1} l_{j,k} \sum_{r=k}^{n} u_{k,r} = ae_j - \Big((\sum_{k=1}^{j} l_{j,k} \sum_{r=k}^{n} u_{k,r}) - u_{j,j} \Big) \; .$$

Here, for k and r the following conditions hold: $k \leq j$ and $k \leq r$; k is contained in the nonzero pattern of $L(j,:)$ and r in the nonzero pattern of $U(k,:)$.

MILU is usually implemented by applying the Gaussian elimination on a restricted pattern like ILU; but instead of deleting every entry outside the allowed pattern, these entries are moved to the related main diagonal position. This leads to the second iterative Algorithm 5 that differs from the first algorithm only for the update of the main diagonal entries $u_{j,j}$.

Algorithm 5 is derived from Algorithm 4 by the following reformulation:

$$(A - LU)e = 0 \rightarrow (diag(A - LU) - (LU)_{\neg S})e = 0$$

in view of $(A - LU)_{S \setminus diag(S)} = 0$. Here, $\neg S$ denotes the indices of $abs(L) \cdot abs(U)$ outside the pattern S. Similarly, $\neg S_j$ denotes the indices in j-th row of $abs(L) \cdot abs(U)$ outside the pattern S, and S_j the indices in the j-th row of pattern S. This leads to

$$a_{j,j} - L(j,:)U(:,j) - (L(j,:)U)_{\neg S}e = 0$$

$$a_{j,j} - u_{j,j} - L(j,1:j-1)U(1:j-1,j) - (L(j,:)U)_{\neg S}e = 0$$

$$u_{j,j} = a_{j,j} - L(j,1:j-1)U(1:j-1,j) - (LU)(j,\neg S_j)e$$

$$u_{j,j} = a_{j,j} - L(j,1:j-1)U(1:j-1,j) - \sum_{j'=1, j' \notin S_j}^{n} (LU)_{j,j'}$$

Algorithm 5 Iterative MILU algorithm 2, IMILU2

```
...
Set unknowns l_{i,j} and u_{i,j} to initial values
for sweep = 1,2,... until convergence do
  parallel for (i,j) ∈ S do
    if i > j then
      l_{i,j} = (1/u_{j,j}) (a_{i,j} − Σ_{k=1}^{j−1} l_{i,k} u_{k,j})
    else
      if i < j then
        u_{i,j} = a_{i,j} − Σ_{k=1}^{i−1} l_{i,k} u_{k,j}
      else
        [s_1, ..., s_k] = indices of pattern of (LU)_{j,:} outside A_{j,:} but including j
        u_{j,j} = a_{j,j} − sum(L(j,:) · U(:,[s_1, ..., s_k])) + u_{j,j}
      end if
    end if
  end parallel for
end for
```

$$u_{j,j} := a_{j,j} - \sum_{j' \in \neg S \cup \{j\}}^{n} (LU)_{j,j'} + u_{j,j} \ .$$

Here, $j' \in [s_1, ..., s_k]$ includes j and additionally the entries $(LU)_{j,j'}$ that are not contained in the pattern S—and that are also subtracted, resp. moved to the diagonal. Note, that the computation of $u_{j,j}$ in Algorithm 5 can be written as

$$u_{j,j} := a_{j,j} + u_{j,j} - sum(l(j,:) \cdot u(:,[s_1, ..., s_k]))$$

where s_j contains the index j and all indices in the j-th row of LU, that are outside the pattern of A.

Applying a columnwise (or rowwise) ordering of the unknown entries $l_{i,j}$ and $u_{i,j}$ results in a strictly lower triangular iteration matrix for the fixed point iteration. This is true for the nondiagonal elements according to Chow [1]; but it also holds for the modified updates of the main diagonal entries for appropriate ordering (see Fig. 4.1 (d) and (e)).

For the incomplete Cholesky factorization we can also formulate an iterative method based on fixed point iteration for approximating U and replacing the condition on the main diagonal entries by $U'Ue = Ae$ (Algorithm 6 IMICHOL1). The equation for determining the diagonal $u_{j,j}$ results in a quadratic equation.

In algorithm IMICHOL1 we again abbreviate $ae = A \cdot e$ and $ue = U \cdot e$. Furthermore, the quadratic equation as update condition for the diagonal entry $ae_j = U(:,j)' \cdot ue$ can be written as

$$ae_j = U(1:j-1,j)' \cdot ue(1:j-1) + u_{j,j} \sum_{k=1}^{n} u_{j,k} + u_{j,j}^2$$

or

$$0 = u_{j,j}^2 + \beta \cdot u_{j,j} + (U(1:j-1,j)' \cdot ue(1:j-1) - ae_j) = u_{j,j}^2 + \beta \cdot u_{j,j} + \alpha \ .$$

This quadratic equation can be solved exactly and the positive solution can be used for the update of $u_{j,j}$. In contrast, it is also possible to use for the linear term $u_{j,j}$

Algorithm 6 Iterative MICHOL algorithm 1, IMICHOL1

Set unknowns $u_{i,j}$ to initial values
$ae = a \cdot e$, $ue = u \cdot e$
for sweep = 1,2,... until convergence **do**
  parallel for $(i,j) \in S, i \leq j$ **do**
    $ua = u_{i,j}$
    if $i \neq j$ **then**
      $u_{i,j} = (a_{i,j} - \sum_{k=1}^{i-1} u_{i,k}u_{k,j})/u_{i,i}$
    else
      $\beta = ue_j - u_{j,j}$
      $\alpha = \sum_{k=1}^{j-1} u_{j,k}ue_k - ae_j$
      $u_{j,j} = -\beta/2 + sqrt(\beta^2/4 - \alpha)$ (or $u_{j,j} = \sqrt{-\alpha - \beta \cdot ua}$)
    end if
    $ue_i = ue_i - ua + u_{i,j}$
  end parallel for
end for

the old value $ua = u_{j,j\,old}$ which leads to the alternative definition

$$u_{j,j} = \sqrt{-\alpha - \beta \cdot u_{j,j\,old}}\,.$$

Numerical examples show that the second option often gives better results and faster convergence. Therefore, the second option was used in the presented numerical examples. Note, that the second option leads to an iteration matrix with nonzero entries on the main diagonal which destroys the convergence proof (that needs strictly lower triangular derivative Φ').

Like in Algorithm 5 we can make use of the equations $(UU')e = Ae$ and $(UU' - A)_S = 0$ which leads to Algorithm 7.

Algorithm 7 Iterative MICHOL algorithm 2, IMICHOL2

Set unknowns $u_{i,j}$ to initial values
$p = spones(-spones(a) + spones(abs(u') \cdot abs(u)) + spones(diag(diag(a))))$
for sweep = 1,2,... until convergence **do**
  parallel for $(i,j) \in S$, $i \leq j$ **do**
    if $i \neq j$ **then**
      $u_{i,j} = (a_{i,j} - \sum_{k=1}^{i-1} u_{k,i}u_{k,j})/u_{i,i}$
    else
      $ua = u_{j,j}$
      $[s_1, ..., s_k] = find(p(:,j))'$
      $s = \sum_{r \in jj} \sum_{k=1}^{j} u_{k,j}u_{k,r}$
      $u_{j,j} = sqrt(a_{j,j} - s + ua^2)$
    end if
  end parallel for
end for

In the update of the diagonal entries the above Algorithm 7 makes use of the equation

$$0 = a_{j,j} - U(:,j)'U(:,j) - (U(:,j)'U)_{\neg S})e =$$

TABLE 3.1
tridiag$(-1,3,-1)$*; IICHOL2 is like IICHOL, only written in L*

n/it	precision	IILU1	IICHOL	IICHOL2	IICHOLD
100	10^{-2}	5	5	5	6
1000	10^{-2}	6	6	6	7
100	10^{-14}	25	25	25	28
1000	10^{-14}	27	26	26	33

TABLE 3.2
tridiag$(-1,2,-1)$*. IICHOL2 is like IICHOL, only written in L*

n/it	precision	IILU	IICHOL	IICHOL2	IICHOLD
100	10^{-2}	52	33	33	115
1000	10^{-2}	119	76	76	1122
100	10^{-14}	113	116	116	*
1000	10^{-14}	1180	1122	1122	*

$$= a_{j,j} - (U(:,j-1)'U(:,j-1) + u_{j,j}^2) - (U(:,j)'U)_{\neg S})e$$

and

$$u_{j,j}^2 := a_{j,j} - (U(:,j)'U)_{[s_1,\ldots,s_k]})e + u_{j,j}^2 \ ,$$

where the s_j comprises the index j and the indices in the j-th row of $(U'U)$ that are not in the pattern of A.

The last variation, Algorithm 8, is based on the representation $A = LDL'$ and condition $Ae = LDL'e$.

Algorithm 8 Iterative MICHOL algorithm D, IMICHOLD

```
Set unknowns l_{i,j} and d_{i,i} to initial values
ae = a · e, le = l · e
for sweep = 1,2,... until convergence do
  parallel for (i,j) ∈ S, i ≥ j do
    if i ≠ j then
      la = l_{i,j}
      l_{i,j} = (a_{i,j} − Σ_{k=1}^{j−1} l_{i,k} · d_k · l_{j,k})/d_j
      le_j = le_j − la + l_{i,j}
    else
      d_j = (ae_j − Σ_{k=1}^{j−1} l_{j,k} · d_k · le_k)/le_j
    end if
  end parallel for
end for
```

In the following we present some iteration count for different iterative LU-factorization algorithms for different matrices and precision: Obviously, the number of necessary iterations is growing heavily for ill-conditioned A (Table 3.2), for the exact LU (Table 3.2), for the modified ILU (Table 3.5), and with the size n (Table 3.2, 3.4, and 3.5).

4. Newton's Method for iterative ILU and MILU. For ill-conditioned A, the iterative ILU and MILU possibly need a lot of iterations until convergence. This is

TABLE 3.3
$tridiag(-1, 3, -1)$

n/it	precision	IMILU1	IMILU2	IMICHOL1	IMICHOL2	IMICHOLD
100	10^{-2}	6	5	6	5	23
1000	10^{-2}	8	6	8	6	30
100	10^{-14}	32	24	39	25	139
1000	10^{-14}	34	27	40	26	150

TABLE 3.4
$tridiag(-1, 2, -1)$

n/it	precision	IMILU1	IMILU2	IMICHOL1	IMICHOL2	IMICHOLD
100	10^{-2}	40	52	*	33	*
1000	10^{-2}	111	119	*	76	*
100	10^{-14}	79	113	*	116	*
1000	10^{-14}	623	1180	*	1122	*

also true in case we want to compute the exact LU-factorization, e.g. for a tridiagonal matrix. In such cases, the Newton method can be applied to allow for faster convergence. As special case, we consider in the following A symmetric positive definite with tridiagonal pattern as illustrative example:

$$A = \begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & \ddots & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & b_{n-1} & a_n \end{pmatrix} = LDL^T =$$

$$= \begin{pmatrix} d_1 & d_1c_2 & & & \\ d_1c_2 & d_2 + d_1c_2^2 & d_2c_3 & & \\ & d_2c_3 & \ddots & \ddots & \\ & & \ddots & \ddots & d_{n-1}c_n \\ & & & d_{n-1}c_n & d_n + d_{n-1}c_n^2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & & & & \\ c_2 & 1 & & & \\ & c_3 & \ddots & & \\ & & \ddots & \ddots & \\ & & & c_n & 1 \end{pmatrix} \begin{pmatrix} d_1 & & & & \\ & d_2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & d_n \end{pmatrix} \begin{pmatrix} 1 & c_2 & & & \\ & 1 & c_3 & & \\ & & \ddots & \ddots & \\ & & & \ddots & c_n \\ & & & & 1 \end{pmatrix}.$$

TABLE 3.5
$tridiag(-1,2,-1) \otimes tridiag(-1,2,-1)$

n/it	precision	IILU	IICHOL1	IMILU3	IMICHOL2
100*100	10^{-2}	2	7	89	66
100*100	10^{-14}	29	28	273	274

Ordering the unknown values columnwise in the vector $x = (d_1, c_2, d_2, c_3, ..., c_{n-1}, d_n)$, we arrive at the function

$$f(d_1, c_2, ..., c_{n-1}, d_n) = \begin{pmatrix} d_1 - a_1 \\ d_1 c_2 - b_1 \\ d_2 + d_1 c_2^2 - a_2 \\ d_2 c_3 - b_2 \\ \vdots \\ d_{n-1} c_n - b_{n-1} \\ d_n + d_{n-1} c_n^2 - a_n \end{pmatrix} = 0 \, .$$

The Jacobi matrix of the vector function $f(x)$ is given by

$$J = \begin{pmatrix} 1 & & & & & & & \\ c_2 & d_1 & & & & & & \\ c_2^2 & 2d_1c_2 & 1 & & & & & \\ & & c_3 & d_2 & & & & \\ & & c_3^2 & 2d_2c_3 & \ddots & & & \\ & & & \ddots & \ddots & 1 & & \\ & & & & \ddots & c_n & d_{n-1} & \\ & & & & & c_n^2 & 2d_{n-1}c_n & 1 \end{pmatrix} .$$

Hence, in every iteration of Newton's method to determine the zero of f, one has to solve a linear system in J via

$$x^{(k+1)} = x^{(k)} - J \backslash f(x^{(k)}) \, ,$$

where J is a lower triangular banded matrix.

Next, let us consider the ILU problems in general for given A with pattern S containing the pattern of A. Lets start with the case of symmetric positive definite $A = LL^T$. Again we have to number the unknowns in L and accordingly the equations $(A = LL^T)_S$ that have to be fulfilled for every entry in pattern S. The typical equation at position (i, j) is given by

$$\sum_{k:((i,k)\in S,(j,k)\in S} l_{i,k} l_{j,k} - a_{i,j} = 0 \, .$$

Ordering the unknowns and the equations columnwise leads to a lower triangular sparse Jacobi matrix J. The ordering is given by $x = (L_S(1:n,1), L_S(2:n,2), ...)$ with $f(x)$ ordered relative to the entries $(A_S(1:n,1), A_S(2:n,2), ...)$; here $L_S(k:n,k)$ and $A_S(k:n,k)$ contain only entries in the pattern S.

The same holds true if we order the unknowns and equations rowwise: $x = (L(1,1), L_S(2,1:2), L_S(3,1:3), ...)$ and f relative to entries $(A(1,1), A_S(2,1:2), A_S(3,1:3), ...)$.

Algorithm 9 Iterative Newton ICHOL algorithm, INICHOL

Initialize L
Find r_j, c_j, $j = 1 : N$, row and column indices to entries in sparsity pattern of L
$p :=$ sparse matrix with entries $[r_j, c_j, j]$ such that $p_{r_j,c_j} = j$
For each j collect the indices that contribute in $l(r_j, k) \cdot l(c_j, k) \longrightarrow [r_j, c_j, k_1, ..., k_{n_j}]$
parallel for Not converged **do**
  parallel for $j = 1 : N$ **do**
    $f(j) = -a(r_j, c_j) + \sum_k l(r_j, k) l(c_j, k)$
  end parallel for
  $J = 0$
  parallel for $j = 1 : N$ **do**
    parallel for $k = k_1, ..., k_{n_j}$ **do**
      $J(j, p(c_j, k) = J(j, p(c_j, k) + l(r_j, k)$
      $J(j, p(r_j, k) = J(j, p(r_j, k) + l(c_j, k)$
    end parallel for
  end parallel for
  $vec(L) = vec(L) - J \backslash f$
end parallel for

For a better understanding of Algorithm 9 (INICHOL) we want to describe the derivatives in the Jacobi matrix J. The function f has j-th component

$$f_j(vec(L)) = -a(r_j, c_j) + \sum_k l(r_j, k) l(c_j, k) \ .$$

Therefore, for the main diagonal entries with $c_j == r_j$ the derivative with respect to $l(r_j, k)$ is given by $2l(r_j, k)$. For the other entries at position (r_j, c_j) the derivative with respect to $l(r_j, k)$ is given by $l(c_j, k)$, and the derivative with respect to $l(c_j, k)$ is given by $l(r_j, k)$.

For nonsymmetric problems we have the unknowns in $(L)_S$ and $(U)_S$ as $x = vec(L, U)$, and the equations from $(A = LU)_S$. At position (i, j) this yields

$$\sum_{k=1,(i,k)\in S,(k,j)\in S}^{n} l_{i,k} u_{k,j} - a_{i,j} = 0 \ .$$

Again by ordering the unknowns in L and U and the equations in A columnwise (resp. rowwise), the Jacobi matrix gets triangular and sparse (see Fig. 4.1).

The derivatives of

$$0 = f_j(vec(L, U)) = -a(r_j, c_j) + \sum_k l(r_j, k) u(k, c_j)$$

in Algorithm 10 (INILU) with respect to $l(r_j, k)$ are given by $u(k, c_j)$, and with respect to $u(k, c_j)$ are given by $l(r_j, k)$.

Now we want to apply Newton's method also to MILU. Therefore, we only have to consider the change in the equation relative to the main diagonal entries. With the condition on the main diagonal entries used in Algorithm 4 ((LUe=Ae) this yields

$$0 = -\sum_{k=1}^{n} a_{j,k} + \sum_{k=j}^{n} u_{j,k} + \sum_{k=1}^{j-1} l_{j,k} \sum_{r=k}^{n} u_{k,r} \ ,$$

Algorithm 10 Iterative Newton ILU algorithm, INILU

Initialize L with diag entries 1, U, and Jacobi matrix $J = 0$
Find r_j, c_j, $j = 1 : N$, row and column indices to entries in sparsity pattern of A
$p :=$ sparse matrix with entries $[r_j, c_j, j]$, hence $p_{r_j,c_j} = j$
For each j collect the indices that contribute in $l(r_j, k) \cdot u(k, c_j) \longrightarrow [r_j, c_j, k_1, ...k_{n_j}]$
parallel for Not converged **do**
  parallel for $j = 1 : N$ **do**
    $f(j) = -a(r_j, c_j) + \sum_k l(r_j, k)u(k, c_j)$
  end parallel for
  parallel for $j = 1 : N$ **do**
    parallel for $k = k_1, ..., k_{n_j}$ **do**
      $J(j, p(k, c_j)) = l(r_j, k)$
      if $k \neq r_j$ **then**
        $J(j, p(r_j, k) = u(k, c_j)$
      end if
    end parallel for
  end parallel for
  $vec(L, U) = vec(L, U) - J \backslash f$
end parallel for

and leads to Algorithm INMILU1 The alternative condition similar to Algorithm 5 (moving deleted entries to the main diagonal) states

$$0 = -a_{j,j} + sum(l(j, :) \cdot u(:, [s_1, ..., s_{m_j}]))$$

resulting in Algorithm INMILU2. Here, the indices $s_1, ..., s_{m_j}$ denote the nonzero entries in the j-th row of $S(LU) \setminus (S(A) \backslash S(E))$.

For INMILU1 the modified condition for diagonal element positions can be written as

$$0 = f_j(vec(L, U)) = -(Ae)_{r_j} + L(r_j, :)Ue =$$

$$= -ae_{r_j} + \sum_k l_{r_j,k} \sum_{r=k}^{n} u_{k,r} \ .$$

Therefore, the derivative with respect to $l(r_j, k)$ is $\sum_{r=k}^{n} u_{k,r}$, and the derivative with respect to $u(k, r)$ is given by $l_{r_j,k}$.

For INMILU2 the modified condition for main diagonal elements is given by

$$0 = f_j(vec(L, U)) = -a_{r_j,r_j} + L(r_j, :) \cdot U(:, [s_1, ..., s_{m_j}]) =$$

$$= -a_{r_j,r_j} + \sum_{k,s} l_{r_j,k} u_{k,s} \ .$$

Therefore, the derivative with respect to $l_{r_j,k}$ is given by $\sum u_{k,s}$ for $r_j > k$, and the derivative with respect to $u_{k,s}$ is $l_{r_j,k}$ (only possible for $p(k, s) > 0$ or there is a nonzero entry at position (k, s)).

For the symmetric positive definite case we get Algorithm 13 INMICHOL1 and Algorithm 14 INMICHOL2. For Algorithm INMICHOL1 the modified equations for

Algorithm 11 Iterative Newton MILU algorithm 1, INMILU1

Initialize L, U, $e = (1, 1, ..., 1)'$, $ae = A \cdot e$
Find r_j, c_j, $j = 1 : N$, row and column indices to entries in sparsity pattern of A
$p :=$ sparse matrix with entries $[r_j, c_j, j]$
For each j collect the indices that contribute in $l(r_j, k) \cdot u(k, c_j) \longrightarrow [r_j, c_j, k_1, ...k_{n_j}]$
parallel for Not converged **do**
  $ue = U \cdot e$, $J = 0$
  parallel for $j = 1 : N$ **do**
    if $r_j == c_j$ **then**
      $f(j) = -ae(r_j) + l(r_j, :) \cdot ue$
    else
      $f(j) = -a(r_j, c_j) + \sum_k l(r_j, k)u(k, c_j)$
    end if
  end parallel for
  parallel for $j = 1 : N$ **do**
    parallel for $k \in [k_1, ..., k_{n_j}]$ **do**
      if $r_j == c_j$ **then**
        if $r_j > k$ **then**
          $J(j, p(r_j, k)) = J(j, p(r_j, k)) + sum(u(k, k : n))$
        end if
        $rr = find(u(k, :))$
        parallel for $s = 1 : length(rr)$ **do**
          $J(j, p(k, rr(s))) = J(j, p(k, rr(s))) + l(r_j, k)$
        end parallel for
      else
        $J(j, p(k, c_j)) = J(j, p(k, c_j)) + l(r_j, k)$
        if $r_j > k$ **then**
          $J(j, p(r_j, k)) = J(j, p(r_j, k)) + u(k, c_j)$
        end if
      end if
    end parallel for
  end parallel for
  $vec(L, U) = vec(L, U) - J \backslash f$
end parallel for

the main diagonal entries are

$$0 = f_j(L) = -ae_j + \sum_{k=1}^{j} l_{r_j,k} \sum_{r=k}^{n} l_{k,r} \ ,$$

and the derivative with respect to $l_{r_j,k}$ is given by $\sum l_{k,r}$, the derivative with respect to $l_{k,r}$ is given by $l_{r_j,k}$.

For Algorithm INMICHOL2 the equation in f related to a main diagonal update is given by

$$L(j, :) \cdot L(:, [s_1, ..., s_{m_j}] - a(j, j) = 0 \ .$$

This leads to the derivative $sum(l([s_1, ..., s_{m_j}], k))$ with respect to $l(r_j, k)$, and $l(r(j), k)$ with respect to all variables $l(k_r, k)$ with $k_r \in [s_1, ..., s_{m_j}]$, where $[s_1, ..., s_{m_j}]$ contains

Algorithm 12 Iterative Newton MILU algorithm 2, INMILU2

Initialize L, U
$pp = spones(abs(l) \cdot abs(u)) - spones(a) + spones(diag(diag(a)))$
Find r_j, c_j, $j = 1 : N$, row and column indices to entries in sparsity pattern of A.
$p :=$ sparse matrix with entries $[r_j, c_j, j]$
For each j collect the indices that contribute in $l(r_j, k) \cdot u(k, c_j) \longrightarrow [r_j, c_j, k_1, ...k_{n_j}]$
parallel for Not converged **do**
  $J = 0$
  parallel for $j = 1 : N$ **do**
    if $r_j = c_j$ **then**
      $find(pp(r_j, :)) =: [s_1, ..., s_{m_j}]$
      $f(j) = -a(r_j, r_j) + sum(l(r_j, :) \cdot u(:, [s_1, ..., s_{m_j}])$
    else
      $f(j) = -a(r_j, c_j) + sum((l(r_j, :) \cdot u(:, c_j)'))$
    end if
  end parallel for
  parallel for $j = 1 : N$ **do**
    parallel for $k \in [k_1, ..., k_{n_j}]$ **do**
      if $c_j == r_j$ **then**
        $find(pp(r_j, :)) =: [s_1, ..., s_{m_j}]$
        if $r_j > k$ **then**
          $J(j, p(r_j, k)) = J(j, p(r_j, k)) + sum(u(k, [s_1, ..., s_{m_j}]))$
        end if
        parallel for $s \in [s_1, ..., s_{m_j}]$ **do**
          if $p(k, s) > 0$ **then**
            $J(j, p(k, s)) = J(j, p(k, s)) + l(r_j, k)$
          end if
        end parallel for
      else
        $J(j, p(k, c_j)) = J(j, p(k, c_j)) + l(r_j, k)$
        if $r_j > k$ **then**
          $J(j, p(r_j, k) = J(j, p(r_j, k) + u(k, c_j)$
        end if
      end if
    end parallel for
  end parallel for
  $vec(L, U) = vec(L, U) - J \backslash f$
end parallel for

all main diagonal entries and additionally all nonzero entries in $abs(L) \cdot abs(L')$ outside S.

For Newtons method it is important that the linear system generated by the Jacobi matrix is easy to solve. Therefore, J should be triangular or at least block triangular with small block size. The following Table 4.1 shows that this is the case for all considered orderings for ILU and ICHOL, and often is the case also for MILU and MICHOL.

Algorithm 13 Iterative Newton MICHOL algorithm 1, INMICHOL1

Initialize L, $e = (1, ..., 1)'$, $ae = A \cdot e$,
Find r_j, c_j, $j = 1 : N$, row and column indices to entries in sparsity pattern of L.
$p =$ sparse matrix with entries $[r_j, c_j, j]$
For each j collect the indices that contribute in $l(r_j, k) \cdot l(c_j, k) \longrightarrow [r_j, c_j, k_1, ..., k_{n_j}]$
parallel for Not converged **do**
  $le = L \cdot e$; $J = 0$
  parallel for $j = 1 : N$ **do**
    if $c_j == r_j$ **then**
      $f(j) = -ae(r_j) + l(r_j, :) \cdot le$
    else
      $f(j) = -a(r_j, c_j) + sum((l(r_j, :). \cdot l(c_j, :)))$
    end if
  end parallel for
  parallel for $j = 1 : N$ **do**
    parallel for $k \in [k_1, ...k_{n_j}]$ **do**
      if $r_j == c_j$ **then**
        $J(j, p(r_j, k)) = J(j, p(r_j, k)) + sum(l(k : n, k))$
        $rr = find(l(:, u))$
        parallel for $r = 1 : length(rr)$ **do**
          $J(j, p(rr(r), k)) = J(j, p(rr(r), k)) + l(r_j, k)$
        end parallel for
      else
        $J(j, p(c_j, k) = J(j, p(c_j, k) + l(r_j, k)$ $J(j, p(r_j, k) = J(j, p(r_j, k) + l(c_j, k)$
      end if
    end parallel for
  end parallel for
  $vec(L) = vec(L) - J \backslash f$
end parallel for

TABLE 4.1
triangular (+) or blocktriangular (with small blocksize) (b) form of the Newton matrix for different orderings

ordering	IICHOL	IILU	IMICHOL1	IMICHOL2	IMILU1	IMILU2
columnwise	+	+	b	+	−	−
rowwise	+	+	−	−	b	+
mixed	/	+	/	/	+	+

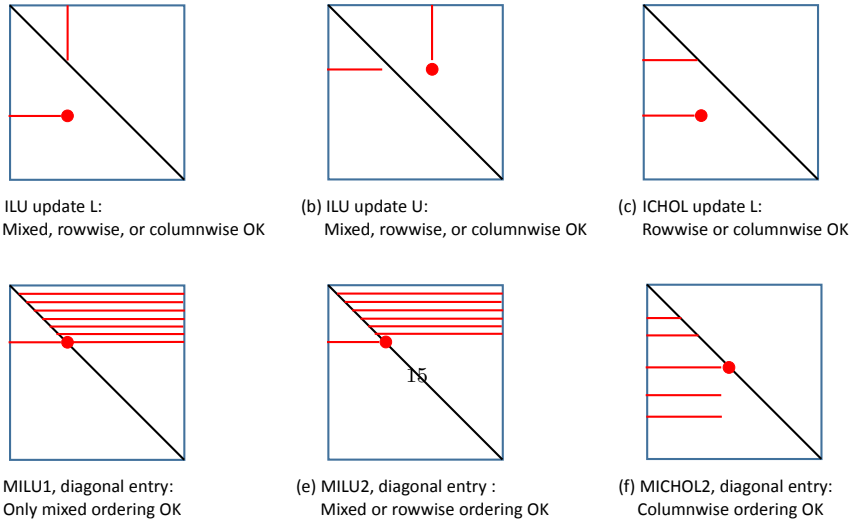

FIG. 4.1. *Ordering in different iterative ILU algorithms that lead to triangular form in J, resp.*

Algorithm 14 Iterative Newton MICHOL algorithm 2, INMICHOL2

Initialize L
Find r_j, c_j, $j = 1 : N$, row and column indices to entries in sparsity pattern of L.
$p :=$ sparse matrix with entries $[r_j, c_j, j]$
$pp = spones(-spones(a) + spones(abs(l) \cdot abs(l')) + spones(diag(diag(a))))$
For each j collect the indices that contribute in $l(r_j, k) \cdot l(c_j, k) \longrightarrow [r_j, c_j, k_1, ..., k_{n_j}]$
parallel for Not converged **do**
  $J = 0$
  parallel for $j = 1 : N$ **do**
    if $c_j == r_j$ **then**
      $find(pp(r(j), :)) =: [s_1, ..., s_{m_j}]$
      $s = sum(l(r(j), :) \cdot l([s_1, ..., s_{m_j}], :)')$
      $f(j) = -a(r(j), r(j)) + s$
    else
      $f(j) = -a(r(j), c(j)) + sum((l(r(j), :). \cdot l(c(j), :)))$
    end if
  end parallel for
  parallel for $j = 1 : N$ **do**
    parallel for $k \in [k_1, ...k_{n_j}]$ **do**
      if $r_j == c_j$ **then**
        $find(pp(r(j), :)) =: [s_1, ..., s_{m_j}]$
        $J(j, p(r(j), k)) = J(j, p(r(j), k)) + sum(l([s_1, ..., s_{m_j}], k))$
        parallel for $s \in [s_1, ..., s_{m_j}]$ **do**
          if $p(s, k) > 0$ **then**
            $J(j, p(s), k)) = J(j, p(s), k)) + l(r(j), k)$
          end if
        end parallel for
      else
        $J(j, p(c(j), k)) = J(j, p(c(j), k)) + l(r(j), k)$
        $J(j, p(r(j), k)) = J(j, p(r(j), k)) + l(c(j), k)$
      end if
    end parallel for
  end parallel for
  $vec(L) = vec(L) - J \backslash f$
end parallel for

TABLE 4.2
tridiag$(-1, 2, -1)$*, iterative Newton method, e.g. INICHOL denoted by ICHOL here*

n/it	precision	ICHOL	ILU	MICHOL1	MICHOL2	MILU1	MILU2
100	10^{-14}	10	10	10	10	10	10
1000	10^{-13}	14	13	14	13	13	14

5. Conclusions. The basic fixed point iteration for computing the ILU or MILU preconditioner can be parallelized. This leads to a preconditioner built on triangular matrices. Therefore, the parallel solution of sparse triangular matrices is the remaining bottleneck.

To achieve faster convergence, Newton's method can be applied also for exact sparse LU, ILU, and MILU. In every Newton step then a triangular system has to

TABLE 4.3
tridiag$(-1, 2, -1) \otimes$ *tridiag*$(-1, 2, -1)$*,iterative Newton method, e.g. INICHOL denoted by ICHOL here*

n/it	precision	ICHOL	ILU	MICHOL1	MICHOL2	MILU1	MILU2
100*100	10^{-13}	5	5	11	11	11	11

solved. Therefore, again efficient parallel triangular solvers are needed.

For the parallel solution of sparse triangular systems there exist direct methods [2, 3, 5, 7, 9]. In [4] we propose also an efficient iterative solver based on recursive preconditioned Jacobi iteration. Hence, combining the parallel ILU with the parallel triangular solvers should lead to an efficient parallel sparse solver. Furthermore, the iterative ILU methods can also be applied on twisted factorizations that are used in eigenvalue computations [10].

The next necessary step is the parallel implementation and testing of the above developed algorithms.

REFERENCES

[1] E. CHOW, A. PATEL, *Fine-grained parallel incomplete LU factorization*, SIAM J. Sci. Comput., 37 (2015), pp. 169–193.

[2] N. J. HIGHAM, *Stability of parallel triangular system solvers*, SIAM Journal on Scientific Computing, 16, (1995).

[3] N. J. HIGHAM, A. POTHEN, *Stability of the partitioned inverse method for parallel solution of sparse triangular systems*, SIAM Journal on Scientific Computing, 15, (1994).

[4] T. HUCKLE, *Accelerated Jacobi Iterations for Bidiagonal and Sparse Triangular Matrices*, preprint 2019.

[5] J. MAYER, *Parallel algorithms for solving linear systems with sparse triangular matrices*, Computing, 86 (2009), pp. 291–312.

[6] J. A. MEIJERINK, H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148162.

[7] M. NAUMOV, *Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU*, Tech. Report NVR-2011-001, NVIDIA, 2011.

[8] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2nd ed., 2003.

[9] J. H. SALTZ, *Aggregation methods for solving sparse triangular systems on multiprocessors*, SIAM J. Sci. Comput., 11 (1990), pp. 123–144.

[10] P. R. WILLEMS, B. LANG, *Twisted factorizations and qd-type transformations for the MR3 algorithm-new representations and analysis*, SIAM J. Matrix Anal. Appl., 33(2):523-553, 2012.