# Hybrid Fine Grained ILU

Samah Karim
Peter Sentz
Shelby Lockhart

University of Illinois at Urbana-Champaign

# Outline

# Outline

# Iterative Methods

- Direct methods for solving $Ax = b$ can be unnecessarily expensive
- Iterative methods are "infinite" solution techniques that requires computation of $x \to Ax$
- Can sometimes be computationally inefficient if too many iterations are required
- Their convergence can be accelerated by preconditioning

# Preconditioning

$$\boxed{Ax = b}$$

$$\Big\downarrow \text{ Apply preconditioner}$$

$$\boxed{C^{-1}Ax = C^{-1}b}$$

# Preconditioner

A preconditioner is a matrix C such that:

- $\text{cond}(C^{-1}A) < \text{cond}(A)$
- Solution of linear systems $Cz = y$ requires little computational effort
- That is, computation of $y \to C^{-1}y$ is "easy"

# Challenges

- If we choose $C = I$
  - no cost associated with solving linear system $Cz = y$
  - condition number is the same
- If we choose $C = A$
  - resulting $C^{-1}A = I$ perfectly conditioned
  - cost required is equal to that of solving the initial linear system with A itself
- Cost of computing $C$ should be taken into account

# No Perfect Answer!

- Optimal C minimizes total computational effort
- Computational effort depends on:
  - Size of the problem
  - Sparsity pattern of A
  - Spectrum of A
- No general answer about how to choose an optimal C
  - One method is Incomplete Factorization

# Incomplete Factorization

- Given general sparse matrix A, an incomplete LU (ILU) factorization is a pair of matrices:
  - Sparse lower triangular matrix L
  - Sparse upper triangular matrix U

  such that $LU \approx A$
- Take preconditioner $C = LU$. Computing $C^{-1}y = U^{-1}L^{-1}y$ is simple:
  - Solve $Lw = y$ by forward substitution
  - Solve $Uv = w$ by backward substitution

# This paper

- New fine-grained parallel algorithm for ILU factorization
- Computes all nonzero elements in parallel
- Iteratively improves the accuracy of the factorization
- Provides large amounts of parallelism using shared memory

# Outline

1. Iterative Methods
2. Incomplete Factorization
3. New Parallel ILU Algorithm
4. Hybrid Fine Grained ILU
5. Current Project State

# Computing Incomplete Factorization

- Given general sparse matrix A, an incomplete LU (ILU) factorization computes:
  - Sparse lower triangular matrix L
  - Sparse upper triangular matrix U

  such that $LU \approx A$
- Use Gaussian elimination process but only allow nonzeros in specified locations of L and U
- Define sparsity pattern S such that $(i, j) \in S$ if:
  - $l_{ij}$ is permitted to be nonzero when $i \geq j$
  - $u_{ij}$ is permitted to be nonzero when $i \leq j$

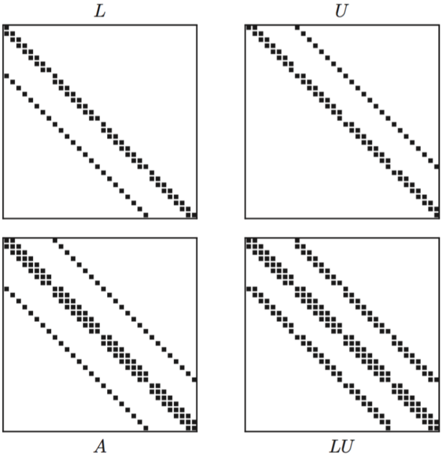
# Incomplete Factorization

**Algorithm 1** Conventional ILU Factorization

**for** $i = 2$ **to** $n$ **do**
    **for** $k = 1$ **to** $i - 1$ **and** $(i, k) \in S$ **do**
        $a_{ij} = a_{ik}/a_{kk}$
        **for** $j = k + 1$ **to** $n$ **and** $(i, j) \in S$ **do**
            $a_{ij} = a_{ij} - a_{ik}a_{kj}$
        **end for**
    **end for**
**end for**

# Example: Zero fill-in ILU (ILU(0))

Take S exactly as the sparsity pattern of A



# Outline

# New Parallel ILU Algorithm

- Parallelism is very fine-grained
- Individual entries of the factorization can be computed in parallel, using an iterative algorithm

# Reformulation of ILU

- New algorithm based on the sometimes overlooked property that:

$$(LU)_{ij} = a_{ij}, \qquad (i,j) \in S \tag{1}$$

  where $(LU)_{ij}$ is the $(i,j)$ entry of the ILU factorization of the matrix with entries $a_{ij}$
- i.e. factorization is exact on the sparsity pattern S

# Reformulation of ILU

- New fine-grained algorithm interprets an ILU factorization as a problem of computing unknowns
  - $l_{ij} \quad , i > j \quad , (i,j) \in S$
  - $u_{ij} \quad , i \leq j \quad , (i,j) \in S$

  using property (1) as constraint.
- $\Rightarrow$ total number of unknowns is $n_S = |S|$, the number of elements in the sparsity pattern $S$

# Reformulation of ILU

- To find these $n_S$ unknowns, rewrite constraints (1) as:

$$\sum_{k=1}^{\min(i,j)} l_{ik} u_{kj} = a_{ij}, \quad (i,j) \in S \tag{2}$$

- Each constraint can be associated with an element in $S \Rightarrow$ there are $n_S$ constraints
- Thus problem of solving for $n_S$ unknowns using $n_S$ nonlinear equations

# Solution of constraint equations

- Write explicit formula for each unknown in terms of the other unknowns:

$$l_{ij} = \frac{1}{u_{jj}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}\right), \quad \textit{if } i > j \tag{3}$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \quad \textit{if } i \leq j \tag{4}$$

- Equations are in the form $x = G(x)$ where $x$ is a vector containing the unknowns $l_{ij}$ and $u_{ij}$

# Solution of constraint equations

- Solve equations via fixed-point iteration:

$$x^{(p+1)} = G(x^{(p)}), \quad p = 0, 1, \ldots \tag{5}$$

  with an initial guess $x^{(0)}$
- Each component of new iterate $x^{(p+1)}$ can be computed in parallel

# Initial Guess

- To begin the fixed point iterations, we need an initial guess for the unknowns
- Simple initial guess:
  - $(L^{(0)})_{ij} = (A)_{ij} \quad , i > j \quad , (i,j) \in S$
  - $(U^{(0)})_{ij} = (A)_{ij} \quad , i \leq j \quad , (i,j) \in S$

# New Parallel ILU

**Algorithm 2** Fine-Grained Parallel Incomplete Factorization

```
Set unknowns l_ij and u_ij to initial values
for sweep = 1, 2, ... until convergence do
    for (i, j) ∈ S, in parallel do
        if i > j then
            l_ij = (a_ij − Σ_{k=1}^{j−1} l_ik u_kj) / u_jj
        else
            u_ij = a_ij − Σ_{k=1}^{i−1} l_ik u_kj
        end if
    end for
end for
```

# New Parallel ILU

- Each fixed-point iteration updating all unknowns is called a sweep
- Algorithm parallelized across elements of S
- Given p compute threads, partition set S into p parts, one for each thread
- Threads run in parallel updating the components of the vector of unknowns x, asynchronously

# Outline

# Distributed Memory Implementation

- To take advantage of available computing resources, algorithm should be adapted to distributed memory systems
- May want to increase problem size, where a matrix may no longer fit in a single node's memory

# Block Jacobi Preconditioner

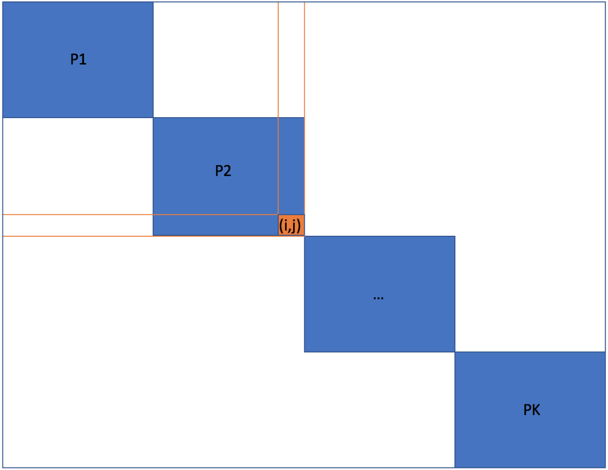
- A simple preconditioner is the diagonal of $A$, i.e. use $D = \operatorname{diag}(a_{11}, a_{22}, \ldots, a_{nn})$
- Can generalize this if we write $A$ in *block* format:

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \cdots & A_{KK} \end{bmatrix}$$

- Use the *block diagonal* of $A$ as a preconditioner:

$$D = \begin{bmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & A_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{KK} \end{bmatrix}$$

# Block Jacobi - Hybrid Implementation



# Hybrid Implementation

- Distribute blocks across processors/nodes
- Compute ILU factorization for each block using OpenMP on each node
- Gather blocks on single processor in order to use preconditioner in iterative method
- Alternatively, could perform iterative method in parallel - no need to gather blocks

# Outline

# Work so far

- Have finished shared memory ILU implementation using OpenMP
- Have finished PCG (an iterative method) code utilizing forward/backward substitution and matrix multiply routines
- Currently finishing Hybrid Block Jacobi-ILU implementation using MPI across nodes

# References

[1] Edmond Chow and Aftab Patel. Fine-grained parallel incomplete lu factorization. *SIAM Journal on Scientific Computing*, 37(2):C169–C193, 2015.

[2] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.