Fine-grained ILU Factorization

Group J

Gursimran Singh Saluja

Abdullah Mujahid

Basanagouda Somanakatti

Jun. -Prof. Dr. Andreas Vogel

High Performance Computing in the Engineering Sciences

Computational Engineering

Ruhr Universität Bochum

FINE-GRAINED PARALLEL INCOMPLETE LU FACTORIZATION

EDMOND CHOW AND AFTAB PATEL
School of Computational Science and Engineering, College of Computing, Georgia Institute of Technology, Atlanta

Overview

Conventional ILU

- Given sparse matrix: A
- Sparsity pattern S:

$$S(i,j) = 1, \text{for } A(i,j) \neq 0$$
$$S(i,j) = 0, \text{for } A(i,j) = 0$$

- Get $A = LU + r$, where L and U also have the same sparsity pattern as A

Conventional ILU

- Method: Gaussian Elimination
- Inplace algorithm. L has ones on the diagonal (omitted, only entries of U are stored)
- Sequential in nature, parallelization is difficult.
- ILU is mostly used as a *preconditioner*
- **Important property of ILU :**

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

Previous Parallel ILU

Parallel Strategies

- Regular ILU isn't a suitable *Preconditioner* for very large matrices
- Multilevel domain decomposition is preferred
- ILU implemented in each sub domain (smaller matrices)
- Parallelism on a single node (on each sub-domain)
- *Coarse-Grained*

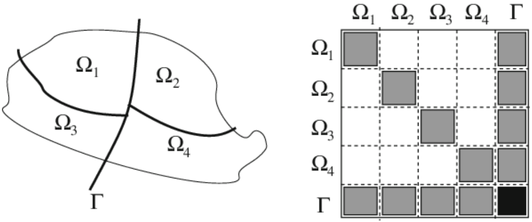


Figure: Domain Decompostion Illustration [3]

New Parallel ILU

Idea

- New *fine-grained* parallel algorithm implements ILU as solving nonlinear equations
- Variables : l_{ij}, u_{ij} which are entries of L and U
- Equations are *constraints* given as follows
- Entry of ILU is exact on the *Sparsity* pattern S

$$(LU)_{ij} = A_{ij} \tag{2}$$

Problem

- Variables:
  - $$l_{ij},\ i > j,\ (i,j) \in S$$
  - $$u_{ij},\ i \leq j,\ (i,j) \in S$$
- Normalization: L has unit diagonal (not computed)
- Number of unknowns: $|S|$
- $$\sum_{k=1}^{min(i,j)} l_{ik}\, u_{ij} = a_{ij},\ (i,j) \in S \tag{3}$$
- Number of constraints: $|S| = m$ (say)
- Problem of solving $|S|$ unknowns with $|S|$ equations
- $|S| > n$, where A is of size $n \times n$

Advantages

- Equations can be solved in parallel with fine-grained parallelism
- Exact solution not necessary for *good ILU preconditioner*
- Good initial guess helps in faster convergence

Solution of constraint equations

- Explicit expression for each unknown in terms of other unknowns
- $$l_{ij} = \frac{1}{u_{ij}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right), \; i > j \tag{4}$$
- $$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \; i \leq j \tag{5}$$
- Form:
$$x^{(p+1)} = G(x^p), \quad p = 0, 1, \ldots$$
- Initial guess: $x^{(0)}$
- Components of $x^{(p+1)}$ can be computed in Parallel.

Ordering

$$g(i,j) = \{(i,j) \longrightarrow k\}$$

Row wise ordering

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ a_{31} & 0 & a_{33} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 3 & 0 & 4 \end{bmatrix}$$

$$g(i,j) = \begin{bmatrix} (1,1) \rightarrow 1 \\ (2,2) \rightarrow 2 \\ (3,1) \rightarrow 3 \\ (3,3) \rightarrow 4 \end{bmatrix}$$

Gussian Elimination ordering

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ a_{31} & 0 & a_{33} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 2 & 0 & 4 \end{bmatrix}$$

$$g(i,j) = \begin{bmatrix} (1,1) \rightarrow 1 \\ (2,2) \rightarrow 3 \\ (3,1) \rightarrow 2 \\ (3,3) \rightarrow 4 \end{bmatrix}$$

Gaussian Elimination ordering :

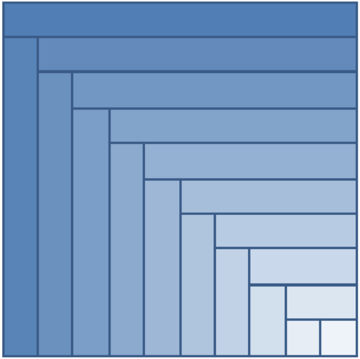


Figure: Ordering illustration

Comparison to exact solution from Gaussian Elimination

Choices

- Different ordering of the unknown variables l_{ij}, u_{ij} can be chosen
- Iterative schemes possible:
  - Asynchronous (parallelizable, Jacobi like)
  - Synchronous (sequential, Gauss-Seidel like)

Case of Exact ILU from GE

$$\xrightarrow[\text{Iteration method : Synchronous}]{\text{Ordering : Gaussian}} \text{Exact ILU in one iteration}$$

Algorithm 1 : Incomplete Factorization

Algorithm 1 Fine-Grained Parallel Incomplete Factorization

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

Algorithm 2: Symmetric Incomplete Factorization

Algorithm 2 Symmetric Fine-Grained Parallel IC

Set unknowns $x_{g(i,j)}$ (l_{ij} and u_{ij}) to initial values
for *sweep* = 1,2, . . . until convergence **do**
    parallel for $(i,j) \in S_U$ **do**
        $s = a_{ij} - \sum_{k=1}^{j-1} x_{g(i,k)} x_{g(k,j)}$
        if $i \neq j$ **then**
            $x_{g(i,j)} = s / x_{g(i,i)}$
        else
            $x_{g(i,i)} = \sqrt{s}$
        end if
    end
end

Convergence Theory

- Nonlinear Equation : $F(x) = x - G(x) = 0$
- Iterative Equation : $x^{(p+1)} = G(x^{(p)})$

Useful Result

Sufficient Condition for **local** linear convergence of fixed point iteration

1. Existence of fixed point (exact ILU guarantees it)
2. G is differentiable around fixed point
3. Spectral radius (maximum eigen value) of $\frac{\partial G}{\partial x}$,

$$\rho(\frac{\partial G(x)}{\partial x}) < 1$$

Convergence Theory : condition 2 check

$$G_{g(i,j)}(x) = \begin{cases} \frac{1}{x_{g(j,j)}} \left(a_{ij} - \sum_{\substack{1 \leq k \leq j-1 \\ (i,k),(k,j) \in S}} x_{g(i,k)} x_{g(k,j)} \right) & \text{if } \ i > j \\ a_{ij} - \sum_{\substack{1 \leq k \leq i-1 \\ (i,k),(k,j) \in S}} x_{g(i,k)} x_{g(k,j)} & \text{if } \ i \leq j \end{cases} \tag{6}$$

Domain of definition of G :

$$D = \{x \in \mathbb{R} \mid x_{g(j,j)} \neq 0, \ 1 \leq j \leq n\}$$

Convergence Theory : condition 2 check

For $i > j$:

$$\frac{\partial G_{g(i,j)}}{\partial u_{kj}} = -\frac{l_{ik}}{u_{jj}},\ k < j,$$

$$\frac{\partial G_{g(i,j)}}{\partial l_{ik}} = -\frac{u_{kj}}{u_{jj}},\ k < j,$$

$$\frac{\partial G_{g(i,j)}}{\partial u_{jj}} = -\frac{1}{u_{jj}^2}(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}),\ k < j,$$

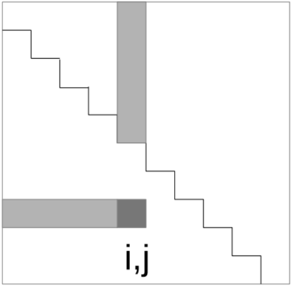
Convergence Theory : condition 2 check

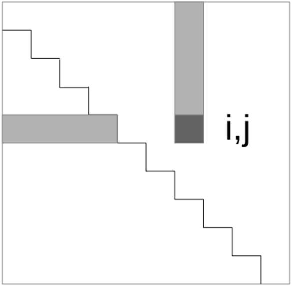
For $i \leq j$:

$$\frac{\partial G_{g(i,j)}}{\partial l_{ik}} = -u_{kj},\ k < i,$$
$$\frac{\partial G_{g(i,j)}}{\partial u_{kj}} = -u_{ik},\ k < i,$$

Convergence Theory : condition 3 check

Variables dependence :





$$\frac{\partial G(x)}{\partial x} = \begin{bmatrix} 0 & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & 0 \end{bmatrix}_{|S|\times|S|}$$

$$\rho\left(\frac{\partial G(x)}{\partial x}\right) = 0$$

Experimental Results

Test platform

- Intel Xeon Phi with 61 cores running at 1.09 GHz
- Each core supporting four way simultaneous multithreading

Aim of tests

1. Convergence of L and U
2. Challenging cases: Nondiagonally dominant
3. Sweeps required for convergence for an effective preconditioner
4. Problem size study
5. Execution times

Experimental Results

Diagnostic tools

- Convergence is determined by taking 1-norm of the residual
- L_1 ***norm :***

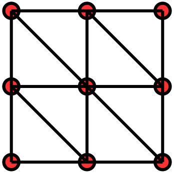
$$\sum_{(i,j)\in S} \left| a_{ij} - \sum_{k=1}^{min(i,j)} l_{ik} u_{kj} \right|$$

- ***sweep*** is the iteration of fixed-point iteration of ILU
- ***solver iteration count*** : number of iterations in the PCG algorithm to get the solution of $Ax = b$.
- Quality of factorization is determined by taking the ***solver iteration count***.

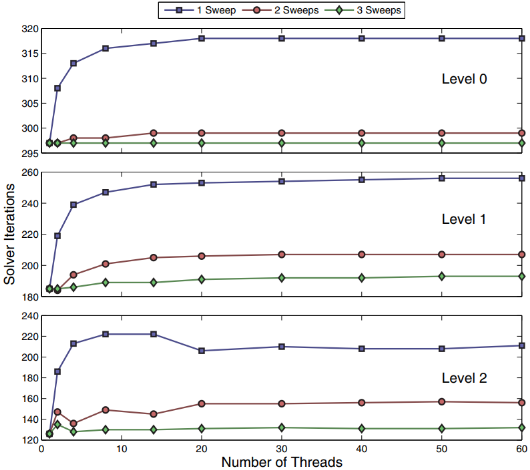
Experiment 1 : Convergence of Algorithm

Symmetric Positive Definite Matrix

- Test Matrix: FEM discretization of Laplacian
- Matrix entries : 203,841 rows and 1,407,811 nonzeros.
- Ordering used : Reverse Cuthill-McKee. (Diagonally dominant)
- Components of *b* are uniformly distributed from [-0.5,0.5]
Sparsity level	0	1	2
Number of non-zeros	805,826	1,008,929	1,402,741

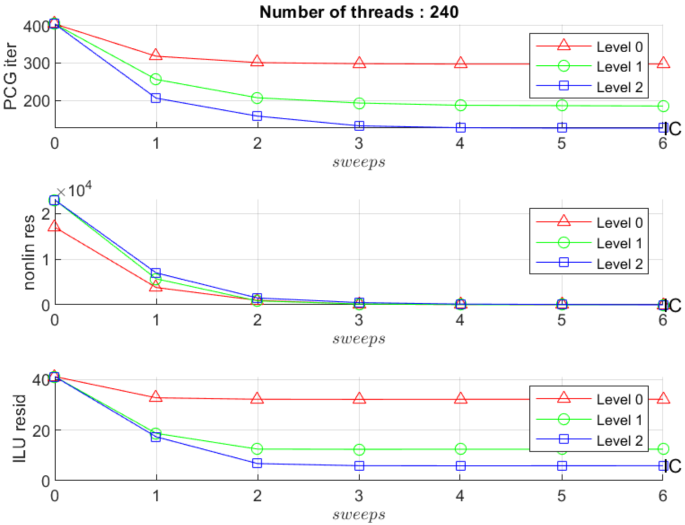


Experiment 1 : Convergence of the algorithm



- Unpreconditioned case : 1223 iterations
- No sweeps : 404 iterations

Experiment 1 : Convergence of the algorithm



Experiment 1 : Convergence of the algorithm

Observations

- Higher level factorizations lead to lower solver iteration counts
level	time for 1 thread	speedup for 60 threads
0	0.189	42.4
1	0.257	44.7
2	0.410	48.8
- ***solver count increase*** till 20 threads, then plateau.
- Algorithm can be highly parallelized
- Good ***preconditioner*** is generated with just a single sweep

Experiment 2 : Nonsymmetric, nondiagonally dominant problems

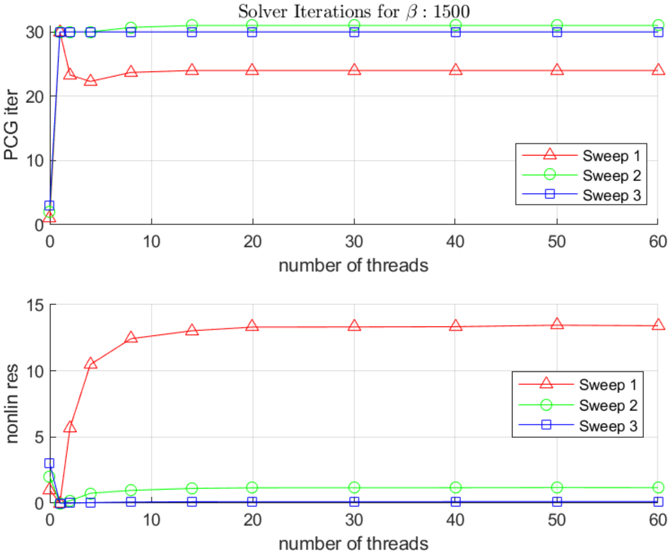
Test Setup

- $2D$ Convection-Diffusion problem :

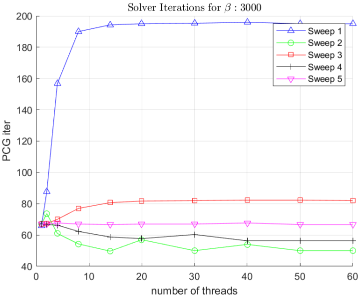
$$-(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + \beta(\frac{\partial e^{xy} u}{\partial x} + \frac{\partial e^{-xy} u}{\partial y}) = g$$

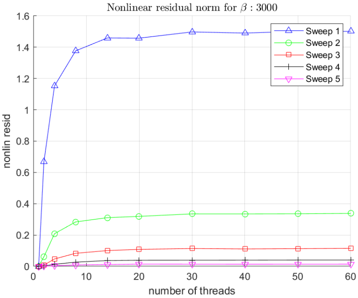
- Central Differencing Scheme on $[0, 1] \times [0, 1]$, Dirichlet Boundary Conditions
- Larger $\beta \to$ nonsymmetric, non diagonally dominant matrices
- Mesh Size: 450×450, Matrix: 202,500 rows and 1,010,700 nonzeros
- Large $\beta \to$ more challenging matrices
- Two tests: $\beta = 1500$ and $\beta = 3000$

Experiment 2 : Nonsymmetric, nondiagonally dominant



Experiment 2 : Nonsymmetric, nondiagonally dominant





Experiment 2 : Nonsymmetric, nondiagonally dominant

Observations

- No preconditioning CG iterations: 1211 and 1301
- Nonlinear residuals:
  - decrease with increasing sweeps
  - larger for higher thread count
- Comparable to that from exact IC
- Few sweeps lead to a good preconditioner

Observations for $\beta = 3000$

- slow convergence, single sweep no longer is good
- effect of number of threads is pronounced
- Algorithm affected by degree of diagonal dominance
- initial guesses are unstable

Experiment 3 : General SPD Problem

Test Setting

- Matrices from University of Florida Sparse Matrix Collection
- level 0 sparsity pattern is used

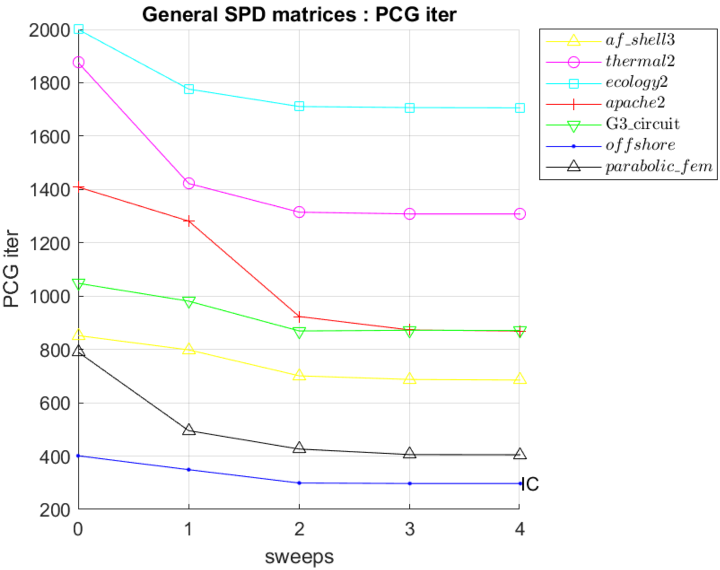
Matrix	No. equations	No. nonzeros
af_shell3	504855	17562051
thermal2	1228045	8580313
ecology2	999999	4995991
apache2	715176	4817870
G3_circuit	1585478	7660826
offshore	259789	4242673
parabolic_fem	525825	3674625

Experiment 3 : General SPD Problem

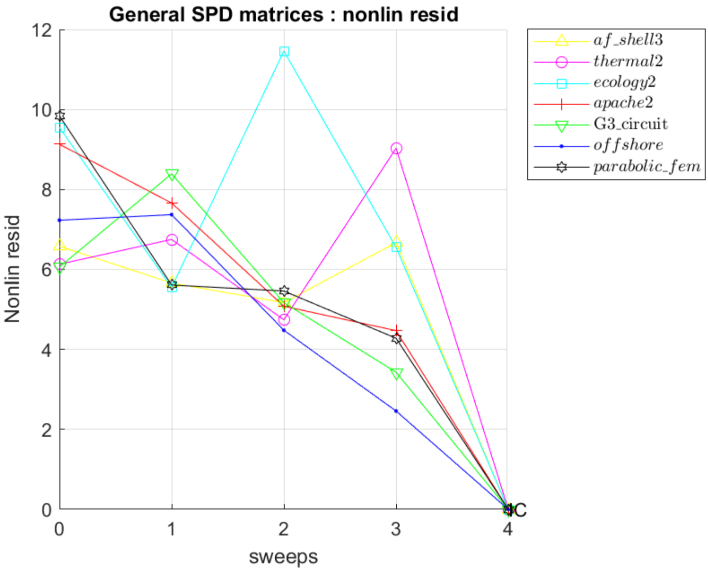
	Sweeps	Nonlin. resid.	PCG iter
af_shell3	0	1.58+05	852
	1	1.66+04	798.3
	2	2.17+03	701
	3	4.67+02	687.3
	IC	0	685
thermal2	0	1.13+05	1876
	1	2.75+04	1422.3
	2	1.74+03	1314.7
	3	8.03+01	1308
	IC	0	1308
ecology2	0	5.55+04	2000+
	1	1.55+04	1776.3
	2	9.46+02	1711
	3	5.55+01	1707
	IC	0	1706
apache2	0	5.13+04	1409
	1	3.66+04	1281.3
	2	1.08+04	923.3
	3	1.47+03	873
	IC	0	869

	Sweeps	Nonlin. resid.	PCG iter
G3_circuit	0	1.06+05	1048
	1	4.39+04	981
	2	2.17+03	869.3
	3	1.43+02	871.7
	IC	0	871
offshore	0	3.23+04	401
	1	4.37+03	349
	2	2.48+02	299
	3	1.46+01	297
	IC	0	297
parabolic_fem	0	5.84+04	790
	1	1.61+04	495.3
	2	2.46+03	426.3
	3	2.28+02	405.7
	IC	0	405

Experiment 3 : General SPD Problem



Experiment 3 : General SPD Problem



Experiment 3 : General SPD Problem

Observations

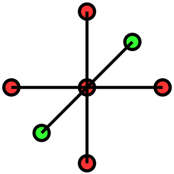
In all cases :

- solver iteration counts corresponding to : 3 sweeps and exact IC are similar
- a good preconditioner achieved without full convergence of ILU

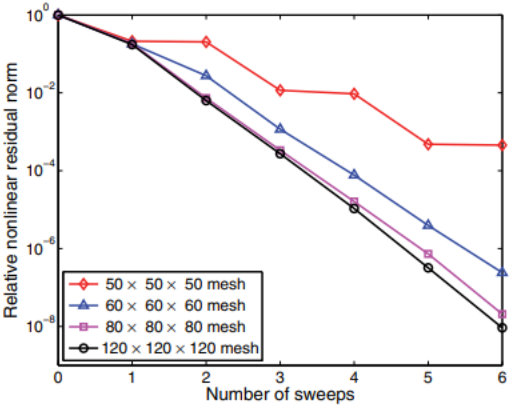
Experiment 4 : Variation of convergence with problem size

Test Setup

- 7-Point Finite Difference 3D Laplacian matrices
- Mesh size :
  - $50 \times 50 \times 50$
  - $60 \times 60 \times 60$
  - $70 \times 70 \times 70$
  - $80 \times 80 \times 80$



Experiment 4 : Convergence with problem size



Experiment 4 : Convergence with problem size

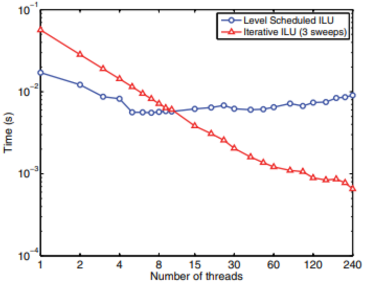
Observations

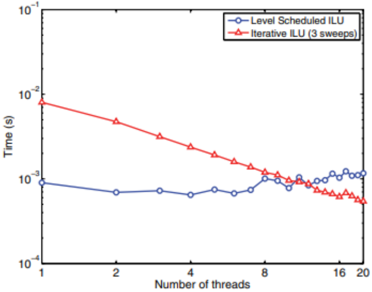
- Convergence is better for larger problem size
- For smaller problems, higher fraction of unknowns updated simultaneously.
- Asynchronous method being closer to Jacobi type fixed point method.
- For large problems little variation with problem size

Experiment 5 : Execution time comparison

Test Setup

- Level scheduled ILU vs Parallel ILU
- Test case: Matrix from 5-point Finite Difference on 100×100 grid
- Non-symmetric version algorithm used, ordering is natural





Experiment 5 : Execution time comparison

Result

- Level scheduled ILU scales well for large parallelism
- New ILU Algorithm is better when parallelism is limited
- For large problems with more parallelism, level scheduling is better because parallel ILU uses more sweeps
- Time for constructing level scheduling is excluded

Approximate Triangular Solves

Parallel Sparse Triangular Solve

- Time for sparse triangular dominates overall solve time
- Methods:
  - Level Schedule Method
  - Inverse as product of sparse triangular factors

Edmond Chow and Aftab Patel
Fine-Grained Parallel Incomplete LU Factorization,
SIAM J. SCI. COMPUT. Vol. 37, No. 2, pp. C169 - C193, 2015.

Rudi Helfenstein and Jonas Koko
Parallel preconditioned conjugate gradient algorithm on GPU,
Journal of Computational and Applied Mathematics Volume 236, Issue 15, September 2012, Pages 3584-3590.

Grasedyck, Lars and Kriemann, Ronald and Le Borne, Sabine
Domain decomposition based $\mathcal{H}$-LU preconditioning,
Volume 112, Numerische Mathematik, May 2009.

Dr. Vasile Gradinaru and Dr. Roger Käppeli
Numerical methods D-PHYS Course at ETH Zürich
Course link