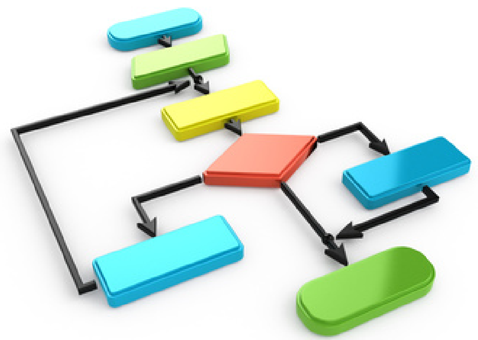
SAND2022-3973C

Sandia National Laboratories

On Iterative Sparse Triangular Solves



Erik G Boman, Sandia National Labs

Copper Mountain Conference, April 2022

U.S. DEPARTMENT OF ENERGY

NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- Sparse triangular solves
- Parallel: Level sets
- Parallel: Iterative methods
- Why would anybody use iterative triangular solves?
- Exact LU vs incomplete LU
- Robustness: Blocking
- Robustness: Scaling
- Ordering, recursion

Sparse triangular solves

- Solve Lx = b, where L is triangular
- Trivial. Just do forward/back substitution!
- Highly sequential. Hard in parallel. Active research topic.
- Important for sparse direct solvers and for ILU preconditioning

Parallel sparse triangular solves

- Well studied problem
  - Anderson & Saad ('89), Saltz ('90), Alvarado & Schreiber ('93), Liu et al. ('16), Chow et al. ('15, '18), …
- Level sets
  - Find level sets in the dependency graph
  - Everything within the same level can be done in parallel
    - Very limited amount of parallelism

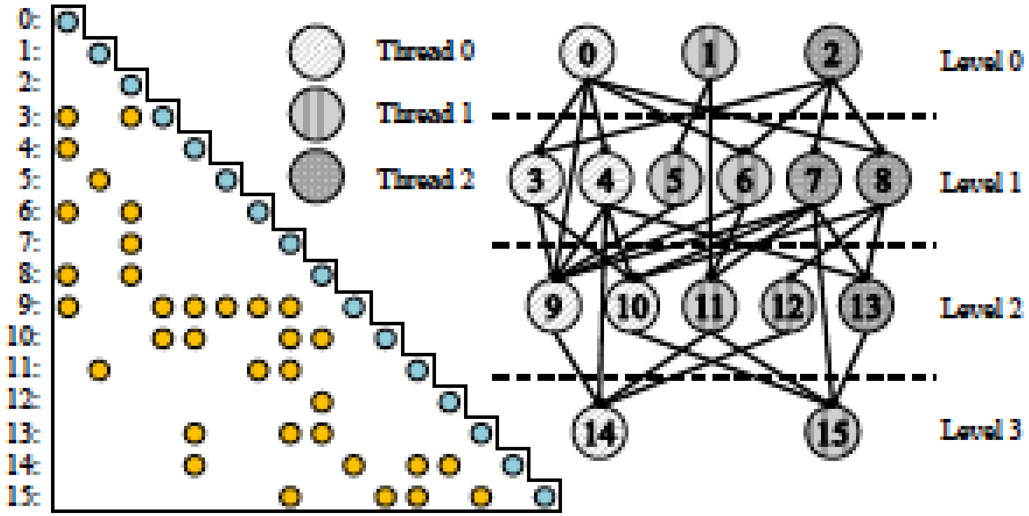


Figure by Wang et al.

Parallel sparse triangular solves

- Iterative methods: (block) Jacobi (Anzt, Chow, et al.)
  - Could use Krylov/GMRES but often not worth it
    - Jacobi good for asynchronous

- Are you crazy? Why compute LU if we must do iterative solves on L and U?
  - Iterative solves on L/U may be faster than on A
  - Iterative solves fast in parallel (e.g., GPU)

- Recent interest motivated by Chow & Patel ('15)
  - Iterative (asynchronous) method for ILU
  - Sparse triangular solves became a bottleneck

Review: Jacobi method

- Let $A = D-L-U$, where D is diagonal, L strictly lower triangular, U strictly upper
- Jacobi: $Dx^{k+1} = (L+U)x^k + b$
- Fixed-point version: $x^{k+1} = T x^k + c$, $T= D^{-1}(L+U)$
- Convergence depends on $\rho(T)$

- Asynchronous Jacobi: Same but overwrite x
  - No "iterations", but asynchronous updates
  - This is actually similar to Gauss-Seidel
  - Convergence depends on $\rho(|T|)$

Polynomial perspective

- Jacobi iteration is really a truncated Neumann series
- Let $A = (I-B)$, then $A^{-1} = I+B+B^2 + \dots$
- k iterations of Jacobi : $x^k = p_k (A) r^0$
  - Where $p_k (A)$ is polynomial degree k

- Question: Can other polynomials do better?
  - Least-squares polynomial (Saad)
  - GMRES polynomial (Loe, Morgan '21)

- In some cases, yes. No clear "winner".

Exact LU vs incomplete LU

- Example: Laplace/Poisson in 2D (n*n)
- Compare Jacobi for Ax=b vs Ly=b, Ux=y
  - Residual reduction: 1e6
- Exact L and U:

n	#it A	#it L	#it U
50	5828	111	111
100	19646	209	208
200	60250	397	396

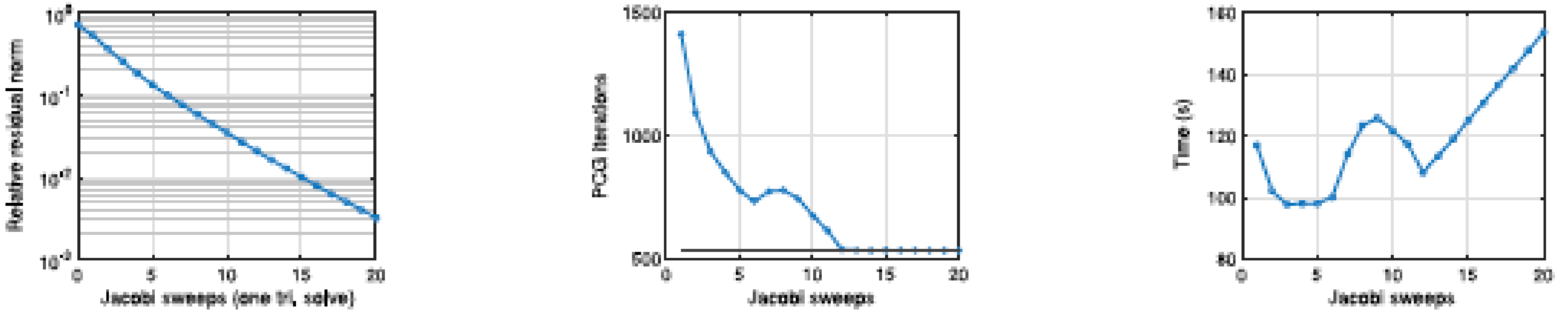
- Incomplete L and U:

n	#it A	#it L	#it U
50	5828	25	25
100	19646	26	26
200	60250	26	26

Note: For preconditioning, we do not need high accuracy. In practice, 5-10 iterations is enough.

Robustness

- Jacobi is not robust, may diverge
  - Convergence for inexact ILU poorly understood
- Several ways to improve robustness:
  - Block Jacobi
  - Scaling
- Difficult case: Highly non-normal matrices



(a) Scalar Jacobi triangular solve convergence.
(b) IC(0)-PCG convergence.
(c) IC(0)-PCG run time.

Example from Chow et al. ('18)

Fig. 2. For Geo_1438, the effects of increasing the number of scalar Jacobi sweeps. Using exact triangular solves requires 533 IC(0)-PCG iterations and 320 s.

Robustness: Blocking

- Block Jacobi
  - Anzt, Chow, Dongarra ('15): "block" asynchronous on GPU
  - Chow, Anzt, Scott, Dongarra ('18):
    - Showed block Jacobi improves robustness on ill-conditioned problems
    - Priority blocking scheme
    - Focus on SPD problems, what about unsymmetric/indefinite?

- Additive Schwarz
  - Anzt, Chow, Szyld, Dongarra ('16): RAS on triangular system
  - Inexact Jacobi on subdomains
  - Got only modest speedup (over Jacobi)
  - Method could be combined with blocking (above)

Robustness: Scaling

- We can scale the matrix A by $A' = D_r A D_c$
- What is a good scaling?
  - Make A better conditioned, "more diag. dom."
  - Chow et al: equilibrate (unit diagonal)
    - works well for SPD problems
  - Thomas et al. ('22):
    - ILU smoothers for multigrid
    - Ruiz scaling (row and columns)
    - Targets the non-symmetric case
    - Tested on real CFD problems
- Scaling is a preprocessing step
  - Can amortize the cost of finding a scaling

Recursive algorithm (synch.)

Let L be split into 2x2 system:

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$$

Now we solve by:

1. Solve L_{11}
2. Update b_2
3. Solve L_{22}

Solve recursively until the blocks are small. Subproblems may use direct or iterative solve.

Unfortunately, need to synchronize after step 1. When does this approach pay off?

1. #iterations for L_{ii} is small
2. L_{21} has a lot of nonzeros

Recursive algorithm (asynch.)

Let L be split into 2x2 system:

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}$$

Now we solve by:

1. Solve L_{11}
2. Update b_2
3. Solve L_{22}

Idea: Solve all steps simultaneously, asynch

Assume: L_{11} and L_{22} have separate resources (cores, GPU).

Who does step 2 (b update)? How often?

If b_2 is updated frequently, the algorithm is Jacobi on L!

Orderings

- Often we can permute A before forming L
    - Permuting a given L is possible but tricky
- Potential ordering methods:
1. RCM (low bandwidth)
2. Graph partitioning
    - L_{22} sparse (not helpful?)
3. Coloring or independent sets
    - L_{11} is diagonal, trivial to invert
    - L_{21} has many nonzeros, good for recursive!

Nested Dissection Ordering (with Separators)

Suppose we reorder A so L looks like:

L =

L11		
	L22	
L31	L32	L33

Then we can solve by:

1. Solve for L_{11} and L_{22} *in parallel*
2. Update $b_3 = b_3 - L_{31}\, y_1 - L_{32}\, y_2$
3. Solve for L_{33}

This algorithm is much more parallel.
Drawback: Finding separators is expensive. Might be useful when solving a sequence of systems.

Conclusions

- Iterative sparse triangular solves make sense
  - Both synchronously and asynch.
- Robustness is a concern
  - Blocking and scaling improve robustness
- Incomplete LU factors typically converge quickly
- Recursive decompositions have potential
  - Natural if solver already uses ND ordering
- Still several issues that need more research

Acknowledgments:

- Thanks to Edmond Chow, Jennifer Loe, Siva Rajamanickam, Daniel Szyld for helpful discussions
- Funded by DOE ASCR.