A novel approach to level-based preconditioners

Jennifer Scott

STFC Rutherford Appleton Laboratory

Miroslav Tůma

Institute of Computer Science

Academy of Sciences of the Czech Republic

Preconditioning 2011, Bordeaux

Introduction

Consider the large sparse SPD linear system

$$Ax = b$$

A preconditioner should be:

Introduction

Consider the large sparse SPD linear system

$$Ax = b$$

A preconditioner should be:

- cheap to compute

Introduction

Consider the large sparse SPD linear system

$$Ax = b$$

A preconditioner should be:

- cheap to compute
- sparse and fast to apply

Introduction

Consider the large sparse SPD linear system

$$Ax = b$$

A preconditioner should be:

- cheap to compute
- sparse and fast to apply
- provide sufficient approximation of the algebraic problem

Introduction

Consider the large sparse SPD linear system

$$Ax = b$$

A preconditioner should be:

- cheap to compute
- sparse and fast to apply
- provide sufficient approximation of the algebraic problem
- result in rapidly converging preconditioned iterative method.

Introduction

Consider the large sparse SPD linear system

$$Ax = b$$

A preconditioner should be:

- cheap to compute
- sparse and fast to apply
- provide sufficient approximation of the algebraic problem
- result in rapidly converging preconditioned iterative method.

Key target is robustness.

Introduction

Long history of preconditioners based on incomplete factorizations of A:

Introduction

Long history of preconditioners based on incomplete factorizations of A:

- Threshold-based $IC(\tau)$: entries greater than τ dropped.

Introduction

Long history of preconditioners based on incomplete factorizations of A:

- Threshold-based $IC(\tau)$: entries greater than τ dropped.
- Memory-based $IC(m)$: dropping of entries based on memory available.

Introduction

Long history of preconditioners based on incomplete factorizations of A:

- Threshold-based $IC(\tau)$: entries greater than τ dropped.
- Memory-based $IC(m)$: dropping of entries based on memory available.
- Structure-based $IC(\ell)$: potential fill entries allowed only if their level of fill is less than ℓ.

Introduction

Long history of preconditioners based on incomplete factorizations of A:

- Threshold-based $IC(\tau)$: entries greater than τ dropped.
- Memory-based $IC(m)$: dropping of entries based on memory available.
- Structure-based $IC(\ell)$: potential fill entries allowed only if their level of fill is less than ℓ.

Huge range of variants and refinements have been proposed over last 30+ years and used for different classes of problems, with no single approach best overall.

Introduction

Long history of preconditioners based on incomplete factorizations of A:

- Threshold-based $IC(\tau)$: entries greater than τ dropped.
- Memory-based $IC(m)$: dropping of entries based on memory available.
- Structure-based $IC(\ell)$: potential fill entries allowed only if their level of fill is less than ℓ.

Huge range of variants and refinements have been proposed over last 30+ years and used for different classes of problems, with no single approach best overall.

Potential disadvantage of $IC(\tau)$ and $IC(m)$:

structural information may be lost.

Our aim:

Search for more robust incomplete factorization preconditioners

Our aim:

Search for more robust incomplete factorization preconditioners

- Consider the importance of structure of A and its decomposition in incomplete factorization preconditioners.

Our aim:

Search for more robust incomplete factorization preconditioners

- Consider the importance of structure of A and its decomposition in incomplete factorization preconditioners.
- Present the effect separately from the other possible effects.

Our aim:

Search for more robust incomplete factorization preconditioners

- Consider the importance of structure of A and its decomposition in incomplete factorization preconditioners.
- Present the effect separately from the other possible effects.
- Propose a new level-based strategy.

Our aim:

Search for more robust incomplete factorization preconditioners

- Consider the importance of structure of A and its decomposition in incomplete factorization preconditioners.
- Present the effect separately from the other possible effects.
- Propose a new level-based strategy.
- Develop a general approach $IC(\ell, \tau, m)$ that combines level-based approach with memory prediction and dropping small entries.

Our aim:

Search for more robust incomplete factorization preconditioners

- Consider the importance of structure of A and its decomposition in incomplete factorization preconditioners.
- Present the effect separately from the other possible effects.
- Propose a new level-based strategy.
- Develop a general approach $IC(\ell, \tau, m)$ that combines level-based approach with memory prediction and dropping small entries.

Notation: $L = \{l_{ij}\}$ denotes complete factor of A ($A = LL^T$);
$\hat{L} = \{\hat{l}_{ij}\}$ denotes incomplete factor.

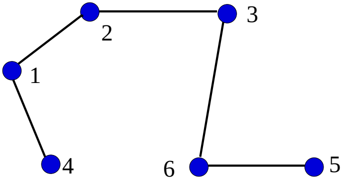
Incomplete factorizations

- Matrix $\rightarrow$ graph

Incomplete factorizations

- Matrix $\rightarrow$ graph

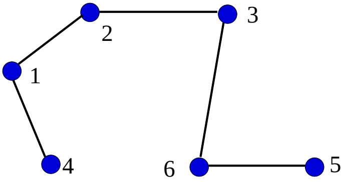
$$A = \begin{bmatrix} 1 & 1 & & 1 & & \\ 1 & 1 & 1 & & & \\ & 1 & 1 & & & 1 \\ 1 & & & 1 & & \\ & & & & 1 & 1 \\ & & 1 & & 1 & 1 \end{bmatrix}$$



Incomplete factorizations

- Matrix $\rightarrow$ graph

$$A = \begin{bmatrix} 1 & 1 & & 1 & & \\ 1 & 1 & 1 & & & \\ & 1 & 1 & & & 1 \\ 1 & & & 1 & & \\ & & & & 1 & 1 \\ & & 1 & & 1 & 1 \end{bmatrix}$$

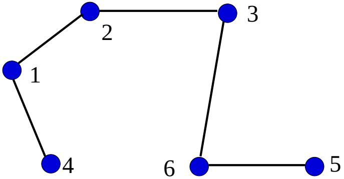


- Fill-path is a path in $\mathcal{G}$ joining nodes i and j via nodes with labels lower than both i and j.

Incomplete factorizations

- Matrix $\rightarrow$ graph

$$A = \begin{bmatrix} 1 & 1 & & 1 & & \\ 1 & 1 & 1 & & & \\ & 1 & 1 & & & 1 \\ 1 & & & 1 & & \\ & & & & 1 & 1 \\ & & 1 & & 1 & 1 \end{bmatrix}$$

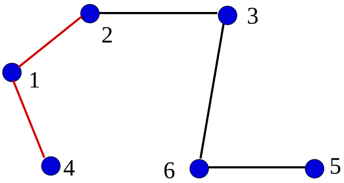


- Fill-path is a path in $\mathcal{G}$ joining nodes i and j via nodes with labels lower than both i and j.
- l_{ij} $(i > j)$ is nonzero if and only if there is a fill path joining i and j in $\mathcal{G}$ (Rose, Tarjan and Leuker, 1978).

Incomplete factorizations

- Matrix $\rightarrow$ graph

$$A = \begin{bmatrix} 1 & 1 & & 1 & & \\ 1 & 1 & 1 & & & \\ & 1 & 1 & & & 1 \\ 1 & & & 1 & & \\ & & & & 1 & 1 \\ & & 1 & & 1 & 1 \end{bmatrix}$$

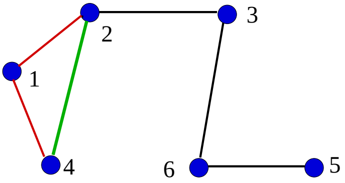


- Fill-path is a path in $\mathcal{G}$ joining nodes i and j via nodes with labels lower than both i and j.
- l_{ij} $(i > j)$ is nonzero if and only if there is a fill path joining i and j in $\mathcal{G}$ (Rose, Tarjan and Leuker, 1978).

Incomplete factorizations

- Matrix $\rightarrow$ graph

$$A = \begin{bmatrix} 1 & 1 & & 1 & & \\ 1 & 1 & 1 & {\color{red}1} & & \\ & 1 & 1 & & & 1 \\ 1 & {\color{red}1} & & 1 & & \\ & & & & 1 & 1 \\ & & 1 & & 1 & 1 \end{bmatrix}$$



- Fill-path is a path in $\mathcal{G}$ joining nodes i and j via nodes with labels lower than both i and j.
- l_{ij} $(i > j)$ is nonzero if and only if there is a fill path joining i and j in $\mathcal{G}$ (Rose, Tarjan and Leuker, 1978).

Level-based incomplete factorizations

- Allow fill-in for fill paths with maximum length ℓ (Watts, 1981).

Level-based incomplete factorizations

- Allow fill-in for fill paths with maximum length ℓ (Watts, 1981).
- In practice: entries of $\hat{L}$ corresponding to nonzero entries of A assigned level 0 and $\hat{l}_{ij}$ allowed in $IC(\ell)$ if $level(i,j) \le \ell$, where

$$level(i,j) = \min_{1 \le l \le \min\{i,j\}} \{level(i,l) + level(l,j) + 1\}$$

(sum rule is one of several definitions).

Problems with $IC(\ell)$

As ℓ increases:

Problems with $IC(\ell)$

As ℓ increases:

- structure can often fill in quickly $\Rightarrow$ expensive to compute and apply

Problems with $IC(\ell)$

As ℓ increases:

- structure can often fill in quickly $\Rightarrow$ expensive to compute and apply
- efficient computation of structure of $IC(\ell)$?

Problems with $IC(\ell)$

As ℓ increases:

- structure can often fill in quickly $\Rightarrow$ expensive to compute and apply
- efficient computation of structure of $IC(\ell)$?

Major break through

Incomplete fill path theorem (Hysom and Pothen, 2002).

Incomplete fill path theorem

level$(i, j) = \ell$ if and only if there exists a shortest fill path of length $\ell + 1$ between i and j in $\mathcal{G}$.

Incomplete fill path theorem

level$(i,j) = \ell$ if and only if there exists a shortest fill path of length $\ell + 1$ between i and j in $\mathcal{G}$.

- Hysom and Pothen use this to develop an algorithm for computing sparsity pattern of single column of $IC(\ell)$.

Incomplete fill path theorem

$level(i, j) = \ell$ if and only if there exists a shortest fill path of length $\ell + 1$ between i and j in $\mathcal{G}$.

- Hysom and Pothen use this to develop an algorithm for computing sparsity pattern of single column of $IC(\ell)$.
- Procedure uses breadth first search that finds shortest path between vertex k and vertices reachable from k via a traversal of at most $\ell + 1$ edges.

Incomplete fill path theorem

level$(i,j) = \ell$ if and only if there exists a shortest fill path of length $\ell + 1$ between i and j in $\mathcal{G}$.

- Hysom and Pothen use this to develop an algorithm for computing sparsity pattern of single column of $IC(\ell)$.
- Procedure uses breadth first search that finds shortest path between vertex k and vertices reachable from k via a traversal of at most $\ell + 1$ edges.
- Nice feature: the structure of each column can be computed independently (and hence in parallel).

Incomplete fill path theorem

level$(i, j) = \ell$ if and only if there exists a shortest fill path of length $\ell + 1$ between i and j in $\mathcal{G}$.

- Hysom and Pothen use this to develop an algorithm for computing sparsity pattern of single column of $IC(\ell)$.
- Procedure uses breadth first search that finds shortest path between vertex k and vertices reachable from k via a traversal of at most $\ell + 1$ edges.
- Nice feature: the structure of each column can be computed independently (and hence in parallel).
- Once sparsity pattern known, separate factorization phase needed to compute entries of $IC(\ell)$.

Preassigning levels

Our aim:
restrict small entries of A to contributing to fewer levels of fill than larger entries.

Preassigning levels

Proposal: given $l > 0$, preassign $level(i, j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

Preassigning levels

Proposal: given $l > 0$, preassign $level(i,j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A (*msmall* and *mbig*).

Preassigning levels

Proposal: given $l > 0$, preassign $level(i,j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A (*msmall* and *mbig*).
- Distribute nonzero entries uniformly by $\log |a_{ij}|$ into

$$[log(mbig) - log(msmall)] + 1$$

  groups, with smallest entries in group with index 1.

Preassigning levels

Proposal: given $l > 0$, preassign $level(i,j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A (*msmall* and *mbig*).
- Distribute nonzero entries uniformly by $\log |a_{ij}|$ into

$$[log(mbig) - log(msmall)] + 1$$

  groups, with smallest entries in group with index 1.
- Sparsify A by dropping any very small entries in group 1.

Preassigning levels

Proposal: given $l > 0$, preassign $level(i,j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A ($msmall$ and $mbig$).
- Distribute nonzero entries uniformly by $\log |a_{ij}|$ into

$$[log(mbig) - log(msmall)] + 1$$

  groups, with smallest entries in group with index 1.
- Sparsify A by dropping any very small entries in group 1.
- Preassign $level(i,j)$ for individual entries according to which group they belong to. Small entries are assigned level $l-1$ and large entries assigned level 0.

Notes on preassigning levels

- Largest entries of A given initial level 0; smaller entries have positive value so contribute to fewer levels of fill.

Notes on preassigning levels

- Largest entries of A given initial level 0; smaller entries have positive value so contribute to fewer levels of fill.
- Large entries may contribute to more than ℓ levels (Strategy II).

Notes on preassigning levels

- Largest entries of A given initial level 0; smaller entries have positive value so contribute to fewer levels of fill.
- Large entries may contribute to more than ℓ levels (Strategy II).
- Straightforward to modify Hysom and Pothen algorithm to accommodate initial levels > 0.

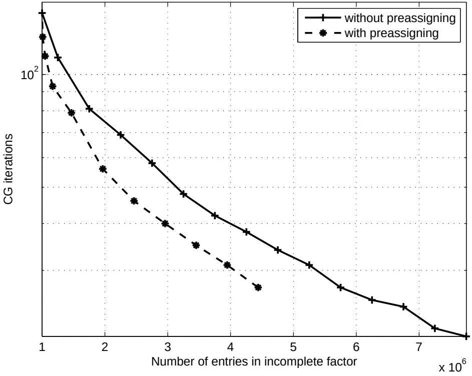
Notes on preassigning levels

- Largest entries of A given initial level 0; smaller entries have positive value so contribute to fewer levels of fill.
- Large entries may contribute to more than ℓ levels (Strategy II).
- Straightforward to modify Hysom and Pothen algorithm to accommodate initial levels > 0.
- Extra cost for symbolic factorization but saves on time to compute and apply $\hat{L}$ if $nz(\hat{L})$ is less than for standard approach.

Effect of preassigning levels

Example:

Kohn-Sham equation (`carsten3`) $n = 250500$.



Test set

- Problems taken from University of Florida Sparse Matrix Collection.
- Selected all SPD matrices of order > 1000.
- CG used with $x_0 = 0$, b computed so that $x = 1$, and stopping criteria

$$\|Ax_k - b\| \leq 10^{-6}\|b\|$$

  with limit of 800 iterations.
- Ran with $\ell = 3$ and removed problems that failed to converge both with and without preassigning initial levels.

Set $\mathcal{T}$ comprises 120 problems.

Performance profile

For solving a given problem, P_i is the most efficient of the preconditioners tested if

$$iter_i \times nz(P_i) \leq \min_{k \neq i}(iter_k \times nz(P_k)).$$

Performance profile

For solving a given problem, P_i is the most efficient of the preconditioners tested if

$$iter_i \times nz(P_i) \le \min_{k \neq i}(iter_k \times nz(P_k)).$$

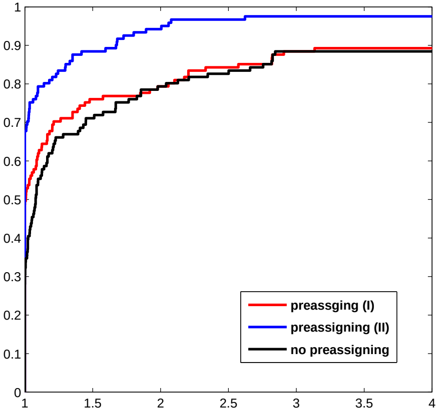
- Suppose P_i run on problem $j \in \mathcal{T}$ has efficiency $e_{ij} \ge 0$.
- Let $\hat{e}_j = \min\{e_{ij}\}$.
- For $\alpha \ge 1$ and each i define

$$k(e_{ij}, \hat{e}_j, \alpha) = \begin{cases} 1 & \text{if } e_{ij} \le \alpha \hat{e}_j \\ 0 & \text{otherwise.} \end{cases}$$

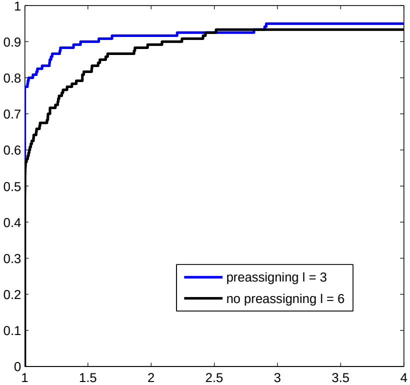
- The performance profile for P_i is then

$$\frac{1}{|\mathcal{T}|} * \sum_{j \in \mathcal{T}} k(e_{ij}, \hat{e}_j, \alpha), \quad \alpha \ge 1.$$

Performance profile $\ell = 3$



Performance profile $\ell = 3$ and 6



Findings so far

- Often advantageous to preassign levels.

Findings so far

- Often advantageous to preassign levels.
- With ℓ fixed, able to solve some problems if levels preassigned that were not solved otherwise.

Findings so far

- Often advantageous to preassign levels.
- With ℓ fixed, able to solve some problems if levels preassigned that were not solved otherwise.
- For other problems, efficiency improved by preassigning levels.

Findings so far

- Often advantageous to preassign levels.
- With ℓ fixed, able to solve some problems if levels preassigned that were not solved otherwise.
- For other problems, efficiency improved by preassigning levels.
- **But** for some problems, better not to preassign levels.
  So far, not able to predict when this is the case.

Findings so far

- Often advantageous to preassign levels.
- With ℓ fixed, able to solve some problems if levels preassigned that were not solved otherwise.
- For other problems, efficiency improved by preassigning levels.
- **But** for some problems, better not to preassign levels.
  So far, not able to predict when this is the case.

Second component of our approach:
keeping structure during factorization phase.

Findings so far

- Often advantageous to preassign levels.
- With ℓ fixed, able to solve some problems if levels preassigned that were not solved otherwise.
- For other problems, efficiency improved by preassigning levels.
- **But** for some problems, better not to preassign levels.
  So far, not able to predict when this is the case.

Second component of our approach:
keeping structure during factorization phase.

Note: in rest of talk, always use preassigning of levels.

Allowing additional memory

- So far memory parameter $m = 1$ ($nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization).

Allowing additional memory

- So far memory parameter $m = 1$ ($nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization).
- What if $m \neq 1$?

Allowing additional memory

- So far memory parameter $m = 1$ ($nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization).
- What if $m \neq 1$?
- $m > 1$ retain the predicted structure but allow some extra entries outside this.

Allowing additional memory

- So far memory parameter $m = 1$ ($nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization).
- What if $m \neq 1$?
- $m > 1$ retain the predicted structure but allow some extra entries outside this.
- Obvious approach: keep the largest extra entries

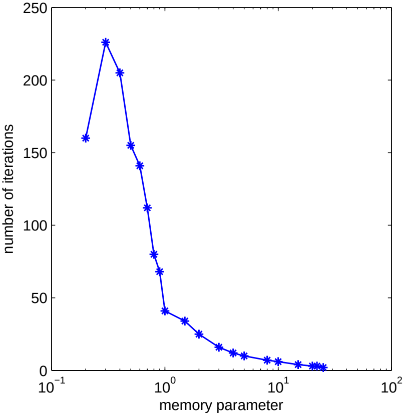
Allowing additional memory

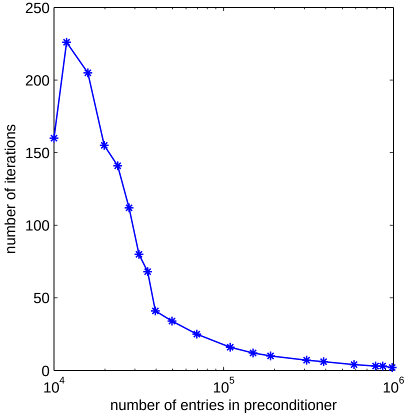
- So far memory parameter $m = 1$ ($nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization).
- What if $m \neq 1$?
- $m > 1$ retain the predicted structure but allow some extra entries outside this.
- Obvious approach: keep the largest extra entries
- How to assign extra space? Two possibilities:
  - uniformly share space between columns
  - non-uniform distribution, using column counts for exact factor L.

Allowing additional memory

- So far memory parameter $m = 1$ ($nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization).
- What if $m \neq 1$?
- $m > 1$ retain the predicted structure but allow some extra entries outside this.
- Obvious approach: keep the largest extra entries
- How to assign extra space? Two possibilities:
  - uniformly share space between columns
  - non-uniform distribution, using column counts for exact factor L.
- $m < 1$ keep only largest entries.

Simple Laplace equation





HB/bcsstk17

For small $\ell > 1$, need $m > 2.7$ to get convergence.

$nz = nz(\hat{L})$ in thousands.

m	$\ell = 0$		$\ell = 1$		$\ell = 2$	
	nz	*iter*	*nz*	*iter*	*nz*	*iter*
2.7	592	†	945	†	1271	11
2.75	603	†	966	23	1292	10
3	657	†	1054	17	1412	8
3.5	767	†	1228	12	1596	1
4	876	†	1402	9	1596	1

Integrate predefined structure with dropping

So far, $nz(\hat{L})$ determined by ℓ and m.

- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner.

Integrate predefined structure with dropping

So far, $nz(\hat{L})$ determined by ℓ and m.

- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner.
- If entries are dropped, we have spare space.

Integrate predefined structure with dropping

So far, $nz(\hat{L})$ determined by ℓ and m.

- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner.
- If entries are dropped, we have spare space.
- Allow entries outside the sparsity structure of $\hat{L}$ to be retained.

Integrate predefined structure with dropping

So far, $nz(\hat{L})$ determined by ℓ and m.

- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner.
- If entries are dropped, we have spare space.
- Allow entries outside the sparsity structure of $\hat{L}$ to be retained.
- If there is insufficient space to accommodate all such extra entries, sort and retain only the largest.

$IC(\ell, \tau, 1)$ versus $IC(\tau)$

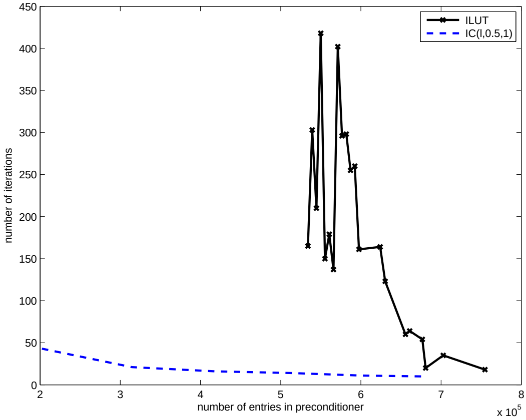
TUBE1, cylindrical shell

$IC(\ell, \tau, 1)$	$\tau = 0.0$		τ =1e-7	
ℓ	*nz*	iter	*nz*	iter
5	2188	283	2105	287
6	2863	223	2711	197
7	3705	159	3431	159
10	7383	230	6346	239
12	10532	158	8527	159
15	13667	83	10404	59

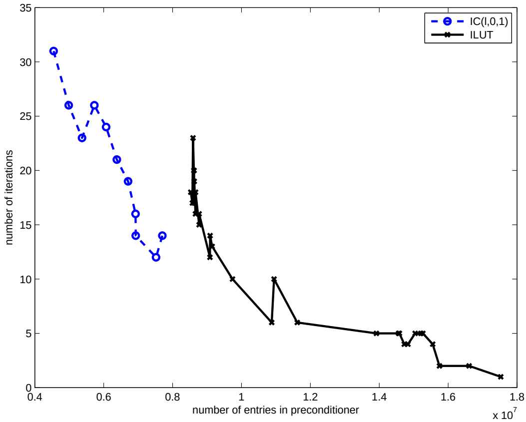
IC(τ)	*nz*	its
5e-5	9649	471
2e-5	9611	87
1e-5	10050	18
5e-6	10741	6
1e-6	12451	2
0	21803	1

- Uni-parameter preconditioner hard to tune.
- Level-based approach converged with much sparser $\hat{L}$.
- Choosing τ is problem and strategy dependent.

Comparison of $ILUT(p, \tau)$ (Saad, 1994) with variable parameters and $IC(\ell, 0.5, 1)$ for `Cylshell/s1rmt3m1`



Comparison of $ILUT(p, \tau)$ with variable parameters and $IC(\ell, 0, 1)$ for `Nasa/nasasrb`



Concluding remarks

- Presented new strategy for setting levels and then exploiting the sparsity structure from symbolic phase during the numerical factorization.

Concluding remarks

- Presented new strategy for setting levels and then exploiting the sparsity structure from symbolic phase during the numerical factorization.
- Numerical results suggest this is a viable approach and is one step in improving incomplete factorization strategies.

Concluding remarks

- Presented new strategy for setting levels and then exploiting the sparsity structure from symbolic phase during the numerical factorization.
- Numerical results suggest this is a viable approach and is one step in improving incomplete factorization strategies.
- More work to be done to refine and extend our approach.

Concluding remarks

- Presented new strategy for setting levels and then exploiting the sparsity structure from symbolic phase during the numerical factorization.
- Numerical results suggest this is a viable approach and is one step in improving incomplete factorization strategies.
- More work to be done to refine and extend our approach.
- Currently developing fast and robust implementation.

Concluding remarks

- Presented new strategy for setting levels and then exploiting the sparsity structure from symbolic phase during the numerical factorization.
- Numerical results suggest this is a viable approach and is one step in improving incomplete factorization strategies.
- More work to be done to refine and extend our approach.
- Currently developing fast and robust implementation.
- Paper to appear in BIT.

Concluding remarks

- Presented new strategy for setting levels and then exploiting the sparsity structure from symbolic phase during the numerical factorization.
- Numerical results suggest this is a viable approach and is one step in improving incomplete factorization strategies.
- More work to be done to refine and extend our approach.
- Currently developing fast and robust implementation.
- Paper to appear in BIT.

Thank you for your attention!