ASP-DAC 2023

Parallel Incomplete LU Factorization Based Iterative Solver for Fixed-Structure Linear Equations in Circuit Simulation

Lingjie Li, Zhiqiang Liu, Kan Liu, Shan Shen, Wenjian Yu
Dept. Computer Science & Tech., BNRist,
Tsinghua University, Beijing, China
Jan. 18, 2023

Speaker's bio

- **Lingjie Li** is a Ph.D candidate from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His supervisor is Prof. Wenjian Yu.
- His research interests include
  - High-performance numerical algorithms
  - Circuit simulation
  - Tensor computation
  - Data compression

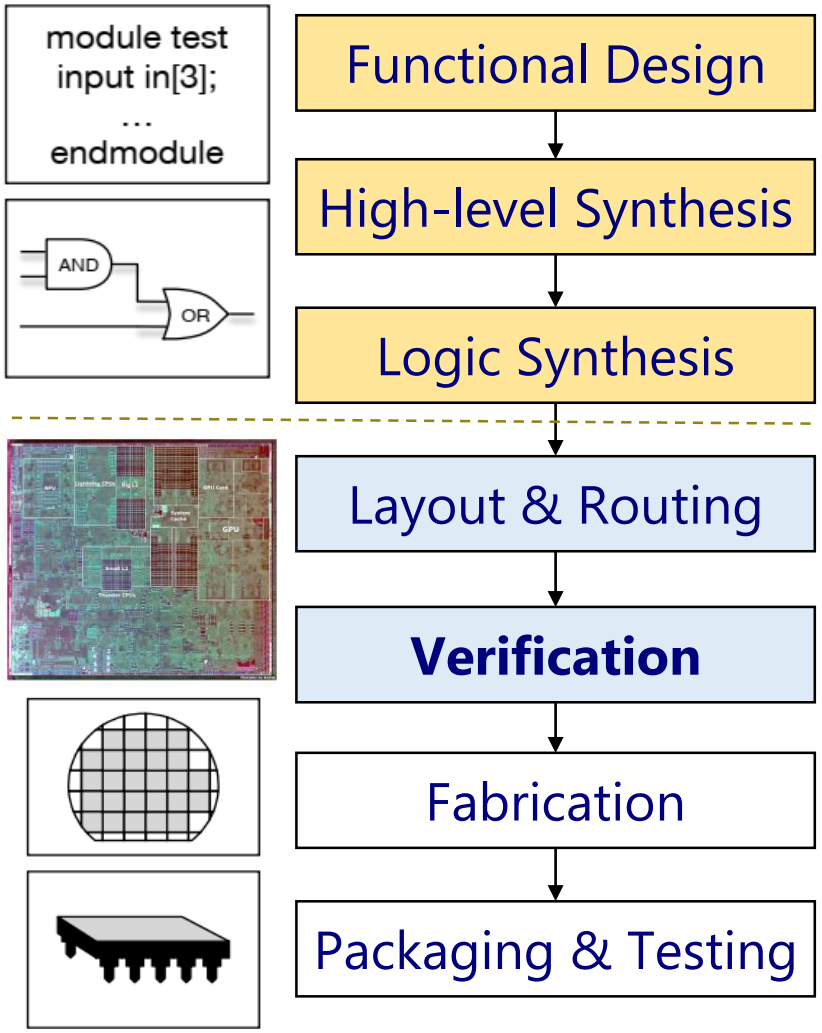
Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

Background



Parasitic Extraction

Circuit Simulation

…

SPICE

Netlist

Circuit Analysis

DAEs

FDM

Nonlinear Equations

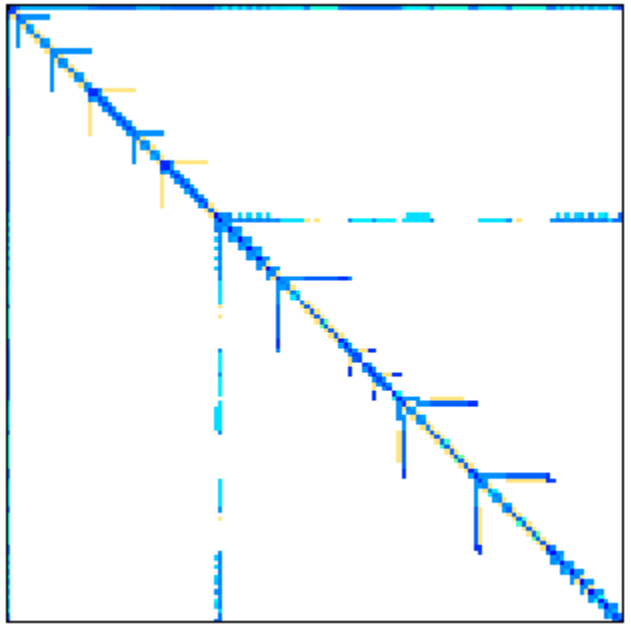
Newton's Method

Linear Equations

Background

- Core problem of circuit simulation
  - Solving the systems of linear equation generated from circuits
- Features of the systems
  - Large
    - ~100K nodes, even more
  - Very sparse, nonzero diagonal
  - Fixed-structure
    - Linear circuit: values don't change
    - Nonlinear circuit: same pattern, changed values

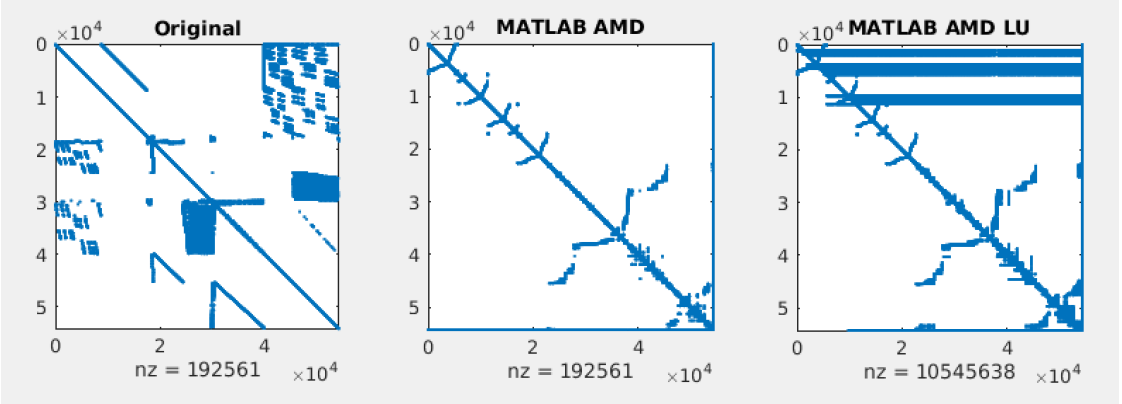
$$\boldsymbol{Ax = b}$$



Background

- Direct solver
  - LU factorization
  - Good for linear circuit
    - Only need to factorize once
  - Potentially huge number of fill-ins
    - Even after reorder

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$



Background

- Iterative solver
  - Preconditioned GMRES

$$\boldsymbol{Ax} = \boldsymbol{b} \Longleftrightarrow \boldsymbol{P}^{-1}\boldsymbol{Ax} = \boldsymbol{P}^{-1}\boldsymbol{b}$$

  - Preconditioner
    - $\mathrm{diag}(\boldsymbol{A})$
    - **Incomplete LU factorization (ILU)**
    - graph spectral sparsification
    - …

Background

- ILU(k) factorization
  - Level of fill-ins

fill-in; lower part nonzero; upper part nonzero

$$\text{level}(a_{ij}) = \begin{cases} \max\limits_{k \mid a_{ik}, a_{kj} \in S} (\text{level}(a_{ik}) + \text{level}(a_{kj}) + 1), & a_{ij} \notin S \\ 0. & a_{ij} \in S \end{cases} \quad (1)$$

	1	2	3	4	5	6	7	8	9	10
1	x		x	x					x	
2	x	x	1	x					1	x
3	x	x	x	x					1	
4	x	x	1	x					1	
5					x	x		x		
6			x	1	x	x		1	2	
7							x			x
8					x	1	x	x	4	1
9		x	2	1		x		2	x	1
10		x	2	x			x	1	2	x

  - Ignore fill-ins whose level > k
  - ILU(0): ignore all fill-ins

Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

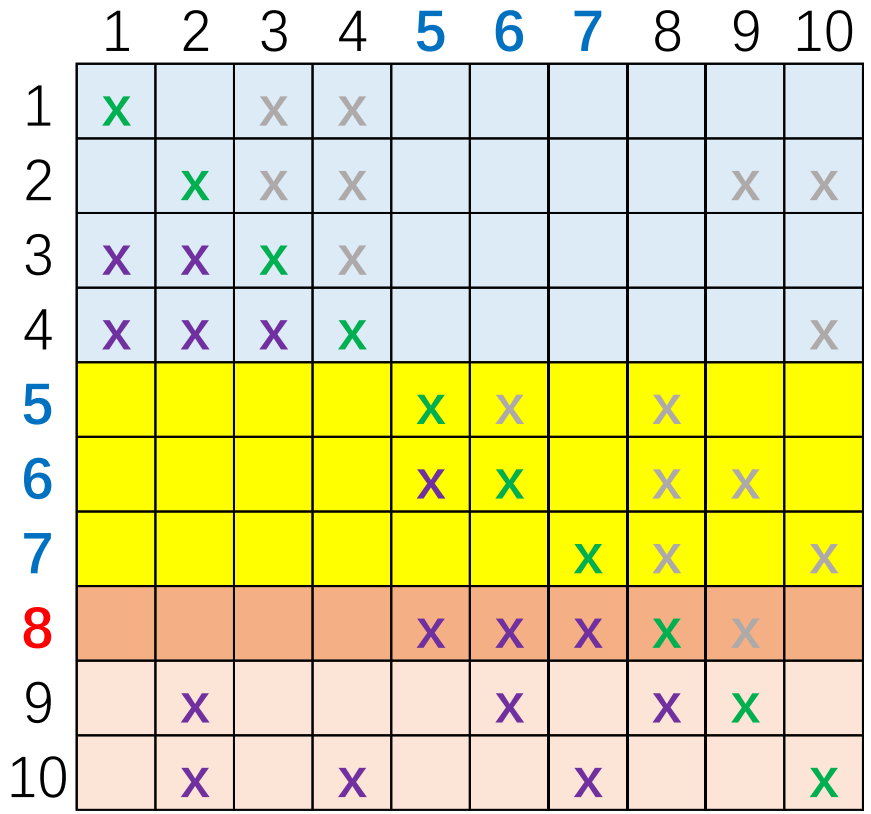
Related work

- Up-looking algorithm for ILU(0)

Algorithm 1 Conventional ILU(0) factorization

Input: $A \in \mathbb{R}^{N \times N}$.

```
1: for i = 2, ..., N do                          ▷ up-looking(i)
2:     for k = 1, ..., i − 1 do                  ▷ row-update(i, k)
3:         if a_ik ∈ S then
4:             a_ik /= a_kk.
5:             for j = k + 1, ..., N do
6:                 if a_ij ∈ S and a_kj ∈ S then
7:                     a_ij −= a_ik a_kj.
```

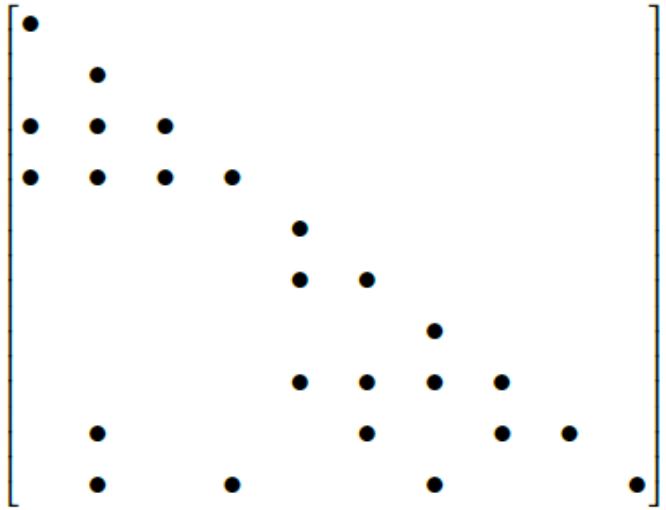


- ILU(k)? Add a symbolic factorization

Related work

- Row-level parallelism and data dependencies

	1	2	3	4	5	6	7	8	9	10
1	x		x	x						
2		x	x	x					x	x
3	x	x	x	x						
4	x	x	x	x						x
5					x	x		x		
6					x	x		x	x	
7							x	x		x
8					x	x	x	x	x	
9		x				x		x	x	
10		x		x			x			x



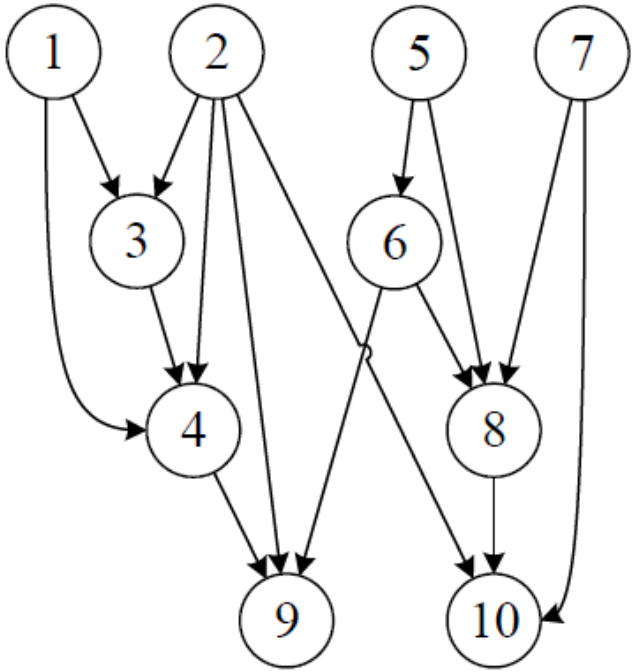
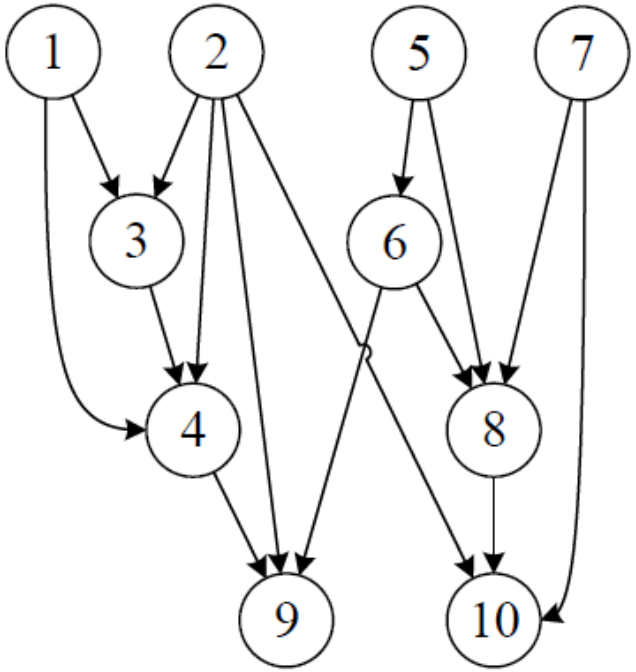


Figure 1: The dependency DAG of a matrix with given lower triangular structure. $k \rightarrow i$ means row i depends on row k.

Related work

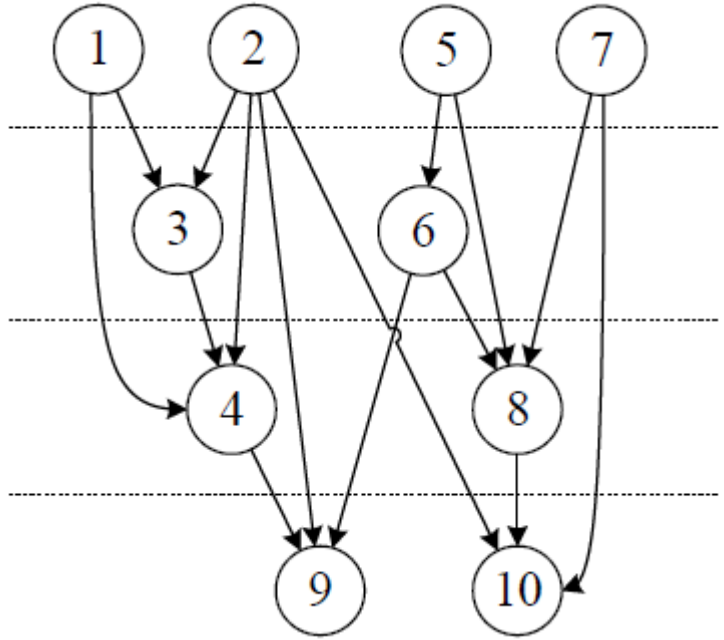
How to schedule?

- Dynamic task pool
  - Too much overhead
- Static scheduling
  - Fixed-structure
  - Pre-analyze the dependency at *setup* stage
  - Overhead can be amortized



Related work

- Level-scheduling [Dutto, IJNMF 99]
  - No dependencies in same level
  - Barriers between levels
- NISCLU [Chen, TCAD 13]
  - Dual-mode level scheduling
  - Same for upper levels
  - Pipeline mode for bottom levels with very few nodes



Related work

- Pipeline mode
  - Remove barriers between levels
  - Add dependency handling before *row-update*

Algorithm 1 Conventional ILU(0) factorization

Input: $A \in \mathbb{R}^{N \times N}$.

```
1: for i = 2, ..., N do                          ▷ up-looking(i)
2:     for k = 1, ..., i − 1 do                  ▷ row-update(i, k)
3:         if a_ik ∈ S then
4:             a_ik /= a_kk.
5:             for j = k + 1, ..., N do
6:                 if a_ij ∈ S and a_kj ∈ S then
7:                     a_ij −= a_ik a_kj.
```

Wait until row k finished

Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

Proposed Parallel ILU based Iterative Solver

Algorithm 3 ILU-GMRES solver (sketch)

Input: Equation $\boldsymbol{Ax} = \boldsymbol{b}$, initial guess $\boldsymbol{x}^{(0)}$, I_{restart}.

1: Incomplete LU factorization $\boldsymbol{A} \approx \boldsymbol{LU}$. ▷ *set up preconditioner*
2: **repeat**
3: $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}^{(0)}$. ▷ *compute initial residue*
4: $\boldsymbol{v}^{(1)} = \boldsymbol{r}/\|\boldsymbol{r}\|_2, s_1 = \|\boldsymbol{r}\|_2$.
5: **for** $i = 1, 2, \ldots, I_{\text{restart}}$ **do**
6: $\boldsymbol{z}^{(i)} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{v}^{(i)}$. ▷ *right preconditioning*
7: $\boldsymbol{w} = \boldsymbol{Az}^{(i)}$. ▷ *sparse matrix-vector multiplication*
8: **for** $k = 1, \ldots, i$ **do** ▷ *modified Gram-Schmidt*
9: $h_{ki} = (\boldsymbol{w}, \boldsymbol{v}^{(k)}), \boldsymbol{w} = \boldsymbol{w} - h_{ki}\boldsymbol{v}^{(k)}$.
10: $h_{i+1,i} = \|\boldsymbol{w}\|, \boldsymbol{v}^{(i+1)} = \boldsymbol{w}/h_{i+1,i}$.
11: Apply $\boldsymbol{J}_1, \ldots, \boldsymbol{J}_{i-1}$ on $\boldsymbol{h}_{:,i}$, and construct Givens rotation $\boldsymbol{J}_i$ acting on h_{ii} and $h_{i+1,i}$ so that $(\boldsymbol{J}_i\boldsymbol{h}_{:,i})_{i+1} = 0. \boldsymbol{s} = \boldsymbol{J}_i\boldsymbol{s}$.
12: Solve triangular system $\boldsymbol{Hy} = \boldsymbol{s}$. $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(0)} + \sum_i y_i \boldsymbol{z}^{(i)}$.
13: **until** residual tolerence satisfied

Parallel ILU factorization

Parallel forward/backward substitutions

Others: Parallel BLAS

- E.g. Intel MKL

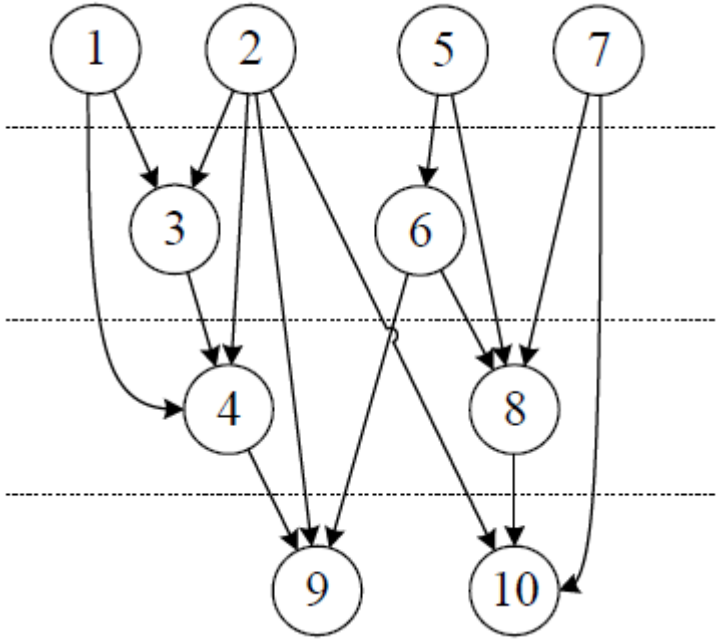
Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

Subtree-based Parallel ILU Factorization

Problems of level-based scheduling

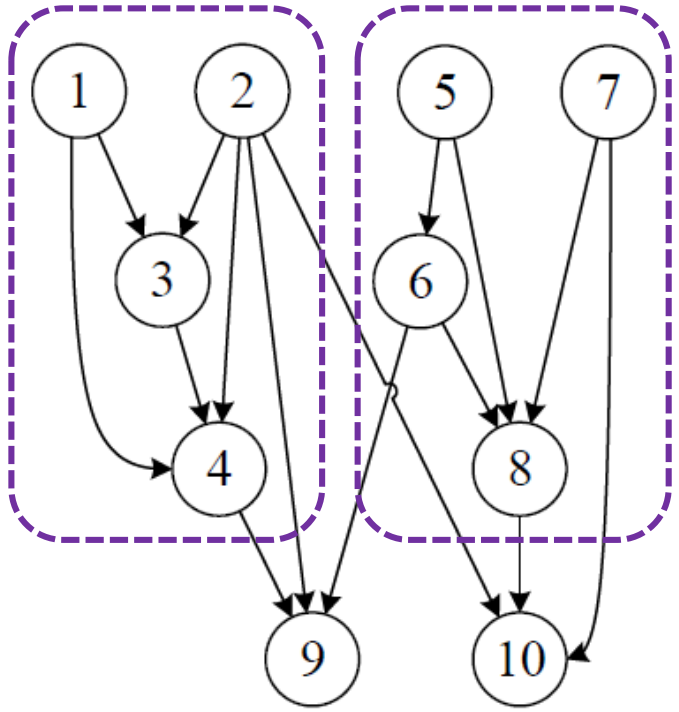
- Load-balancing
  - Barriers between levels
  - More levels to pipeline mode?
    - Overhead for dependency handling
- Data locality
  - Memory access order: 1 2 5 7 | 3 6 | 4 8 | 9 10
  - Larger matrix -> poorer cache efficiency



Subtree-based Parallel ILU Factorization

Assume we have partitioned the rows into P balanced private queue and 1 small shared queue

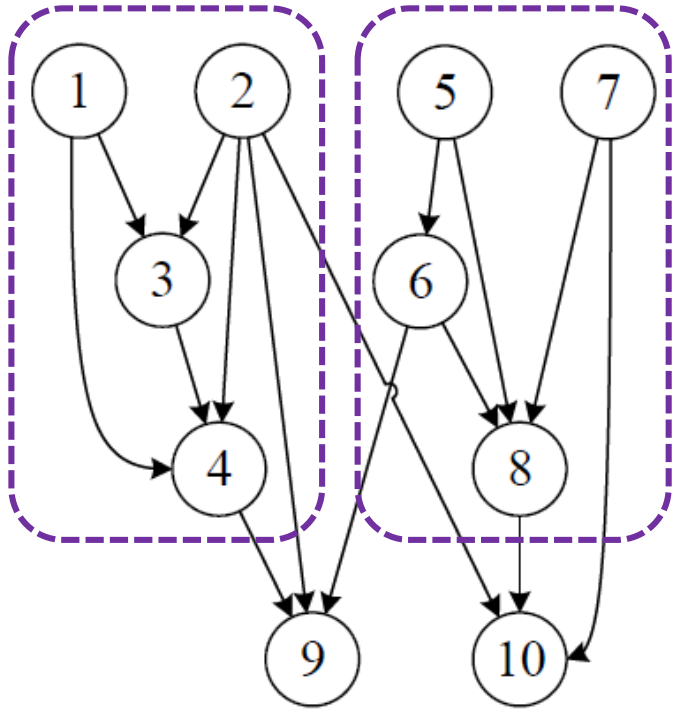
- P : number of threads
- All tasks in one private queue have no dependencies on tasks in any other queue



private queue 1	1 2 3 4
private queue 2	5 6 7 8
shared queue	9 10

Subtree-based Parallel ILU Factorization

- Each thread works on its private queue
  - In natural order: Better data locality
  - No inter-thread communication
- After one finishing its private queue
  - Work on shared queue in pipeline mode
  - No barrier before shared queue
  - Better load-balancing

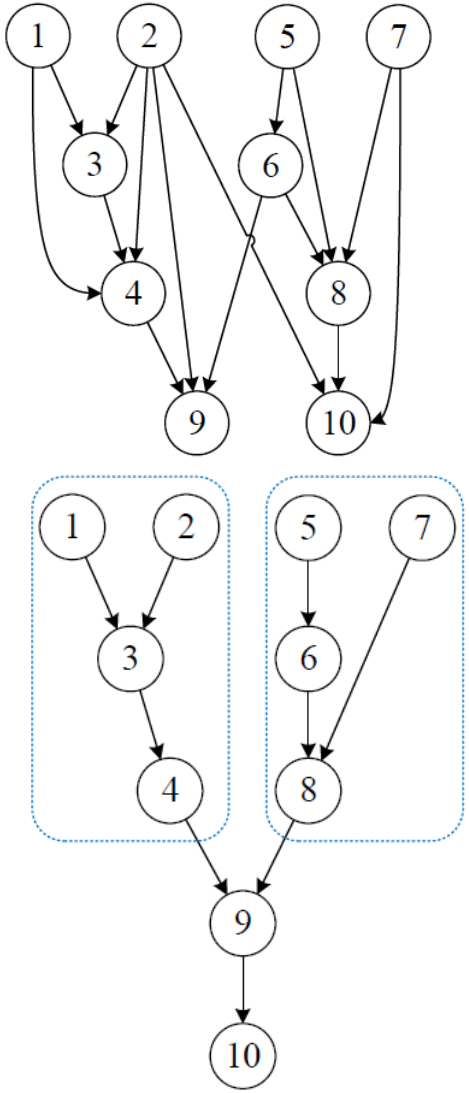


private queue 1	1 2 3 4
private queue 2	5 6 7 8
shared queue	9 10

Subtree-based Parallel ILU Factorization

How to obtain the partition?

- Hard on arbitrary DAG, but easy on tree
- Elimination tree (ETree) [Gilbert, SIMAX 93]
  - Generated from DAG
  - The dependencies of LU factorization for $\boldsymbol{L} + \boldsymbol{L}^{\mathrm{T}}$
  - Overestimates the dependencies
- Subtree partitioning



Subtree-based Parallel ILU Factorization

Algorithm 5 Build private queues based on ETree

Input: Dependency DAG G, P threads, relaxing parameter α.

1: Build ETree T based on G.
2: Subtree set $S = \{T\}$.
3: Let w_S denote the total weight of subtrees in S.
4: Let l_S denote the root of the subtree with largest weight in S.
5: ▷ *subtree partitioning* ◁
6: **while** $S \neq \emptyset$ and weight(subtree(l_S)) $\times P\alpha > w_S$ **do**
7: $\quad u = l_S$.
8: $\quad S = S \setminus \{\text{subtree}(u)\}$.
9: $\quad$ **for all** $v \in$ children of u **do**
10: $\quad\quad S = S \cup \{\text{subtree}(v)\}$.
11: Evenly assign each subtree in S to one of the P threads based on their weights.
12: Build P private queues according to the subtree assignment, and sort each queue in natural order.
13: Build the shared queue with the remaining nodes, and sort the queue in level order of G.

Setup

$O(N \log N)$

Divide subtree

In most cases:
shared queue < 5%

Assign task

Build private queues

Build shared queues

Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

Parallel Forward/backward Substitutions

- Very similar row-level parallelism to the ILU factorization
  - Forward substitution: structure of $\boldsymbol{L}$
  - Backward substitution: structure of $\boldsymbol{U}$
- Similar static scheduling
- Fewer FLOPs
  - Synchronization and memory access affects more
  - Subtree-based > level-based
- Parallel efficiency in shared queue remains to be a problem

Parallel Forward/backward Substitutions

- Task granularity: improve the data locality of the shared queue
  - At most g adjacent tasks are aggregated into a *task pack*
  - each thread takes a pack of tasks instead of one task
- To minimize the loss of parallelism: 2-round execution

Algorithm 1 Conventional ILU(0) factorization

Input: $A \in \mathbb{R}^{N \times N}$.

```
1: for i = 2, ..., N do                          ▷ up-looking(i)
2:     for k = 1, ..., i − 1 do                  ▷ row-update(i, k)
3:         if a_ik ∈ S then
4:             a_ik /= a_kk.
5:             for j = k + 1, ..., N do
6:                 if a_ij ∈ S and a_kj ∈ S then
7:                     a_ij −= a_ik a_kj.
```

If row k not finished
1st round: skip to next task
2nd round: wait

Parallel Forward/backward Substitutions

- Node stripping: minimize the size of shared queue
  - When reordering matrix to get a better ETree of the factorization, ETree of $\boldsymbol{U}$ tends to be badly degenerated

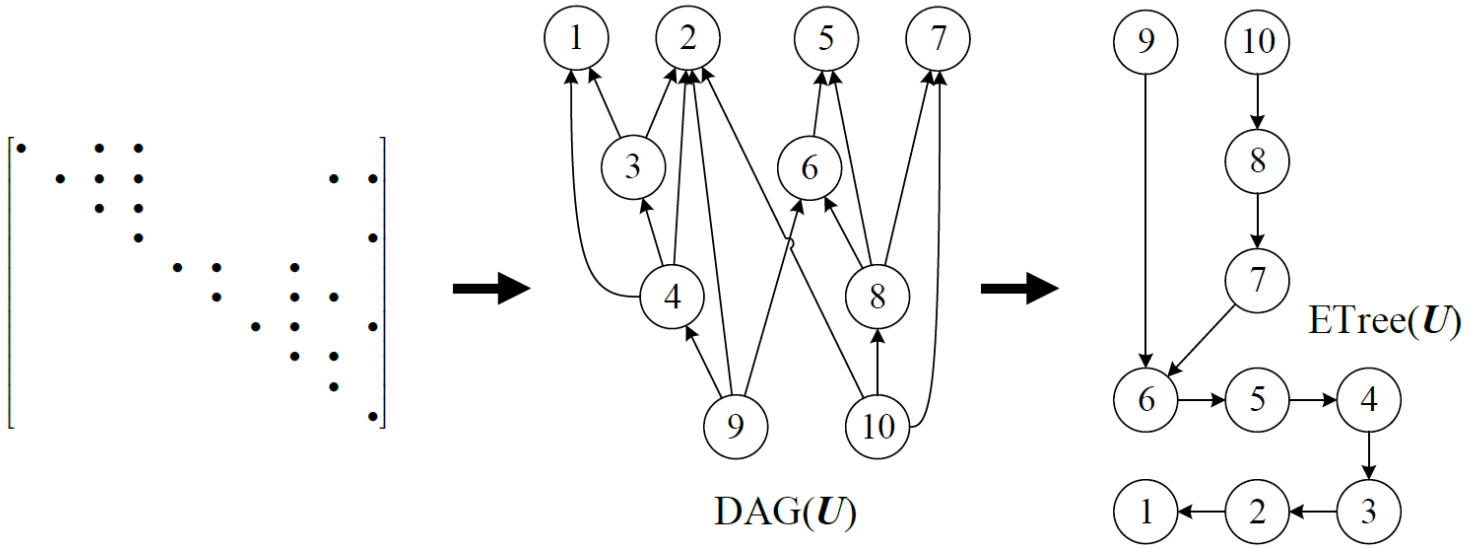
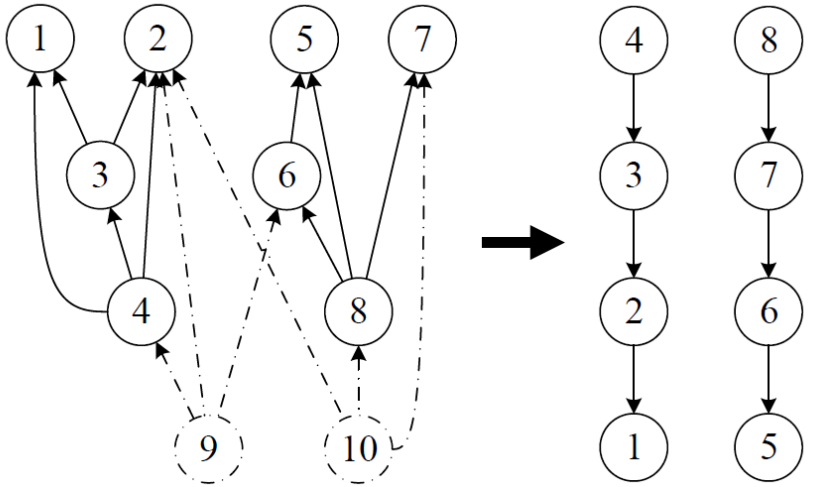


Figure 4: The dependency DAG and ETree for U in backward substitution. Note that ETree(U) is badly degenerated.

Parallel Forward/backward Substitutions

- Node stripping: minimize the size of the shared queue
  - At most $\beta \times N$ nodes are stripped from the top levels of DAG
  - Top levels: level-based scheduling
  - Rest of DAG: subtree-based scheduling



cluster mode	10 9
private queue 1	4 3 2 1
private queue 2	8 7 6 5
shared queue	*empty*

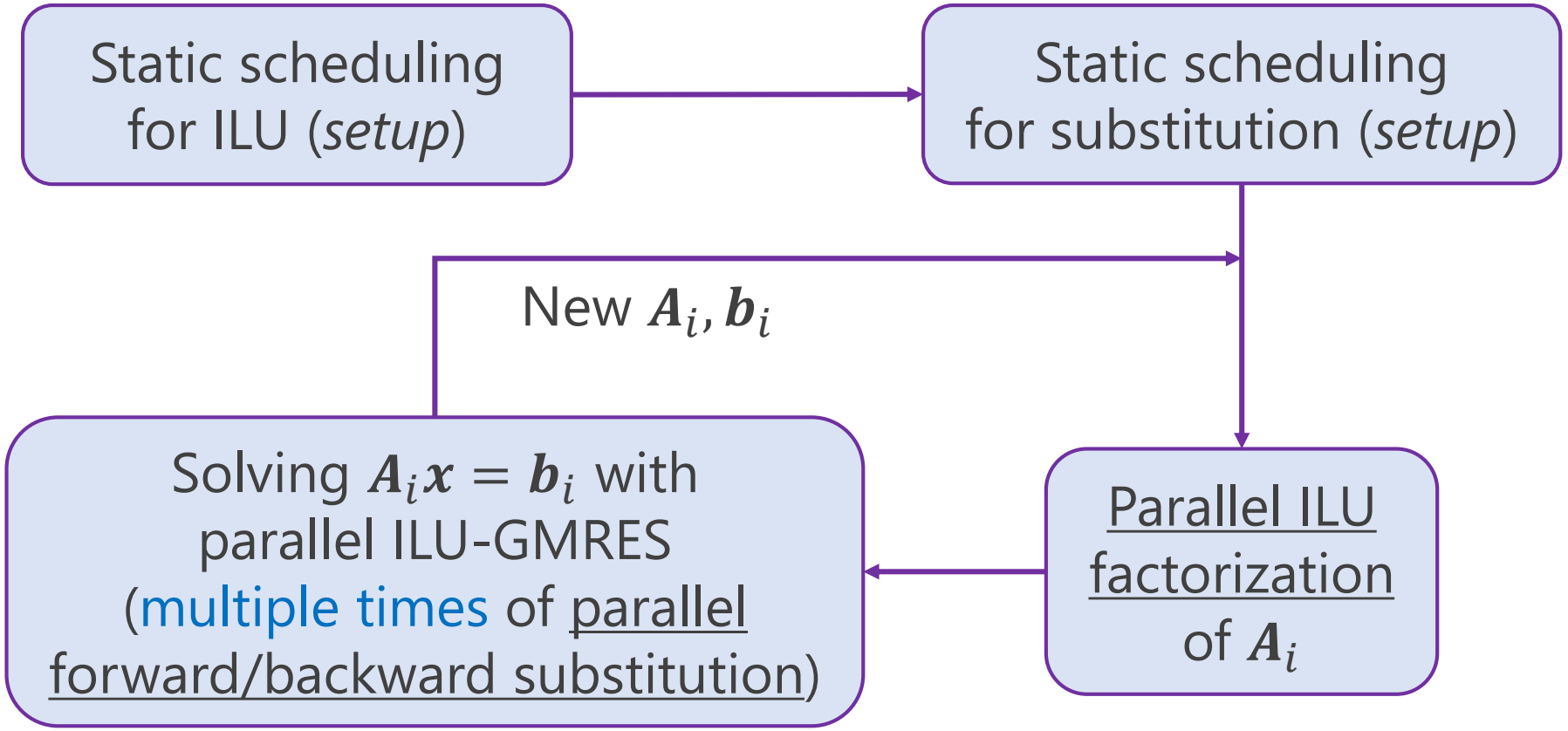
β_1 = 20%, β_2 = 50%
shared queue < 2%

Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

Numerical Experiments

- Overview



Numerical Experiments

- 1 setup + 100 factorization
  - S: sequential version
  - L: dual-mode level scheduling (NISCLU)
  - P: proposed subtree-based scheduling
  - 8 threads
- Better speedup for larger/denser matrices

Table 1: Runtime of performing ILU(k) factorization for 100 times (including the static schedule for "L" and "P")

matrix	N	$\frac{nnz}{N}$	T_{ILU} (s)			Sp.	
			S	L	P	P/S	P/L
case9	3993	878	75.8	37.5	14.5	5.23	**2.58**
case10	12155	691	156	45.5	30.3	5.15	**1.50**
mult_dcop_01	25187	7.7	0.149	0.0824	0.0751	1.98	**1.10**
rajat15	37261	20.2	1.07	0.288	0.232	4.62	**1.24**
ckt11752_tr_0	49702	18.8	1.62	0.468	0.392	4.14	**1.20**
ASIC_100ks	99190	5.8	0.372	0.105	0.0958	3.88	**1.10**
ASIC_100k	99340	9.6	0.686	0.296	0.266	2.58	**1.11**
trans4	116835	6.6	0.40	0.238	0.208	1.90	**1.14**
transient	178866	6.0	0.636	0.276	0.242	2.63	**1.16**
Raj1	263743	11.4	2.95	1.41	0.800	3.69	**1.77**
nxp1	414604	10.7	4.42	2.46	1.13	3.90	**2.17**

Level of fill (i.e. k) in ILU(k) varies among matrices for GMRES convergence. transient, rajat15, nxp1: ILU(1); Raj1, ckt11752_tr_0: ILU(5); the others: ILU(0).

Numerical Experiments

- GMRES until convergence
  - relative tolerance: 10^{-4}
- Higher speedup for substitution than factorization
- Around 4X (up to 4.5X) overall speedup with 8 cores

Table 2: Time for executing GMRES iterations with the ILU(k) preconditioners (same k as the experiment for Table 1)

matrix	#iter	T_{subs} (ms)			Sp.		T_{GMRES} (ms)			Sp.	
		S	L	P	P/S	P/L	S	L	P	P/S	P/L
case9	109	725	505	163	4.4	**3.1**	1107	649	309	**3.6**	2.1
case10	25	420	288	106	4.0	**2.7**	641	375	192	**3.3**	2.0
mult_dcop_01	17	10.1	4.78	3.24	3.1	**1.5**	20.8	8.10	6.51	**3.2**	1.2
rajat15	168	268	135	78.4	3.4	**1.7**	632	217	160	**3.9**	1.4
ckt11752_tr_0	51	111	52.4	24.2	4.6	**2.2**	237	81.5	53.2	**4.5**	1.5
ASIC_100ks	5	9.51	3.16	2.54	3.7	**1.2**	20.5	5.51	5.03	**4.1**	1.1
ASIC_100k	5	12.9	5.09	4.17	3.1	**1.2**	27.1	9.61	8.78	**3.1**	1.1
trans4	45	97.4	48.4	35.9	2.7	**1.4**	343	105	91.1	**3.8**	1.2
transient	180	561	286	187	3.0	**1.5**	2290	674	566	**4.0**	1.2
Raj1	427	3300	1955	1024	3.2	**1.9**	9096	3210	2273	**4.0**	1.4
nxp1	339	4087	2503	1372	3.0	**1.8**	11694	4078	2958	**4.0**	1.4

Numerical Experiments

- Results on power grid transient simulation

Table 3: Computational time for power grid transient simulation (unit in second)

circuit	KLU	S	L	P			Sp.		
	T_{tr}	T_{tr}	T_{tr}	#iter	Err(mV)	T_{tr}	P/K	P/S	P/L
ibmpg3t	115	172	79.7	14.7	2.0/1.0	38.9	**3.0**	4.4	**2.1**
ibmpg4t	138	124	56.7	15.1	0.20/0.07	35.7	**3.9**	3.5	**1.6**
ibmpg5t	96.9	177	89.8	11.2	0.84/0.56	46.0	**2.1**	3.8	**2.0**
ibmpg6t	117	247	123	11.2	1.4/0.82	61.4	**1.9**	4.0	**2.0**

T_{tr}: time for linear equation solutions in simulation; #iter: average GMRES iterations; Err(mV): maximum/average errors of the solutions.

Numerical Experiments

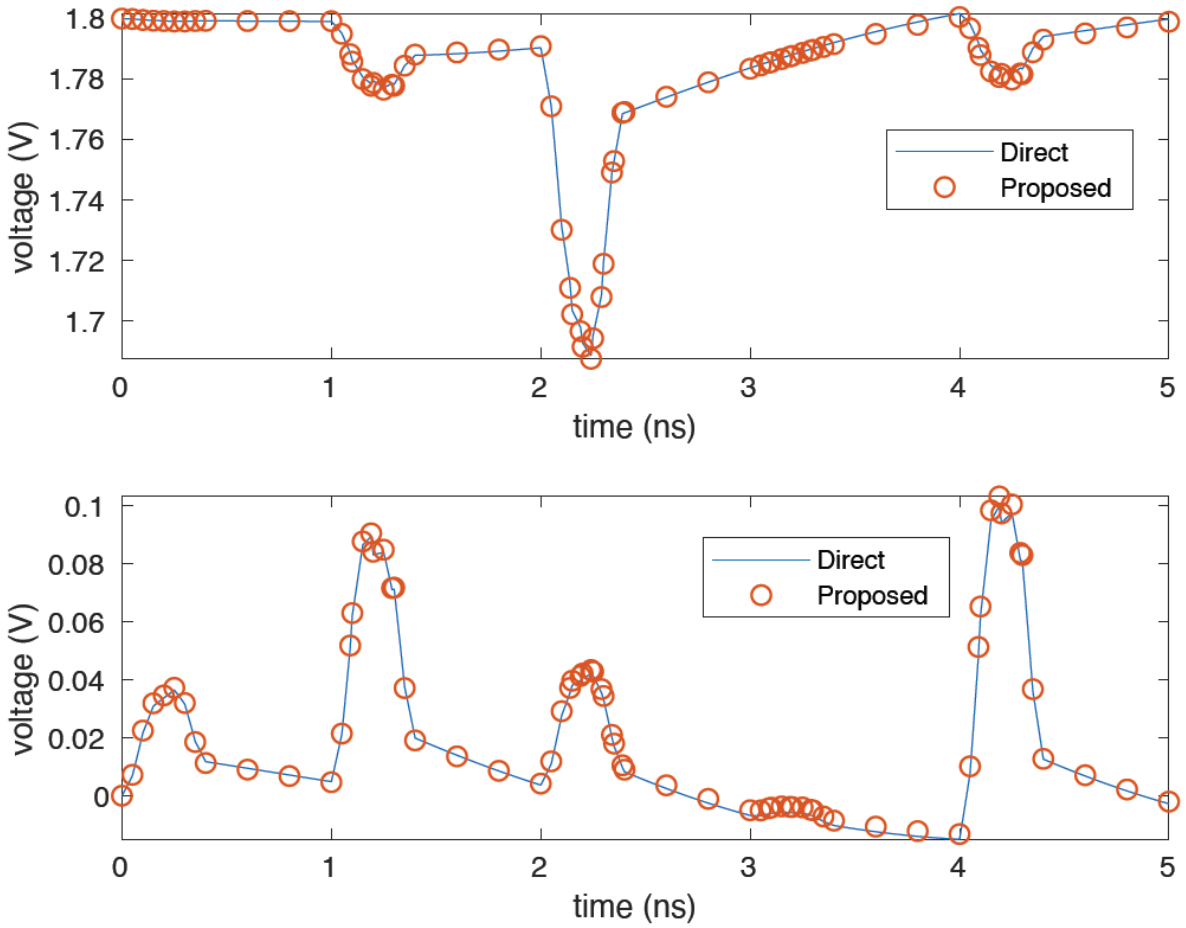


Figure 6: The transient simulation results at a VDD (up) and a GND (down) node in case "ibmpg3t", obtained with direct equation solver and the proposed ILU-GMRES solver.

Numerical Experiments

- Results on nonlinear circuit simulation
  - ngspice*(P): modified ngspice that uses proposed parallel iterative solver

Table 4: Time for DC analysis and each time-point advancing for transient simulation of nonlinear circuits (unit in second)

circuit	N	#Dev.	HSPICE		ngspice		ngspice*(P)	
			T_{DC}	T_{trans}	T_{DC}	T_{trans}	T_{DC}	T_{trans}
ckt1	164899	47614	32.8	0.741	945	45.8	2.9	0.190
ckt2	1043446	201056	38.4	2.25	fail	fail	25.9	1.90
ckt3	1214290	265946	128	15.3	fail	fail	29.8	2.03

#Dev: the number of nonlinear devices; T_{DC}: time for DC analysis including setup time; T_{trans}: average time for a time step.

Numerical Experiments

- Results on nonlinear circuit simulation

Table 5: The total time for equation solutions with "S", "L" and "P" in the transient simulation based on ngspice*

circuit	ILU-GMRES time (s)			Sp.	
	S	L	P	P/S	P/L
ckt1	10.0	3.08	2.55	**3.92**	**1.21**
ckt2	170	79.5	44.7	**3.80**	**1.78**
ckt3	209	96.8	52.7	**3.97**	**1.84**

Outline

- Background
- Related work
- Proposed Parallel ILU based Iterative Solver
  - Subtree-based parallel ILU factorization
  - Parallel forward/backward substitutions
- Numerical Experiments
- Conclusions

Conclusions

- In this work, we have presented a subtree-based parallel ILU factorization and forward/backward substitution algorithm for fixed-structure matrices in circuit simulation, and an ILU-GMRES solver incorporating them.
- With matrices from circuit simulation, the proposed ILU-GMRES solver using 8 threads is up to 5.23X faster than the sequential baseline for solving fixed-structure equations.
- Experiments involving transient simulation of linear and nonlinear circuits show that the proposed parallel algorithms are faster than the parallel baselines derived from the dual-mode level scheduling in NICSLU with up to 2.1X speedup.
- The large benefit over HSPICE for solving large nonlinear circuits is also demonstrated.

References

[1] R. Akhunov, S. Kuksenko, V. Salov, and T. Gazizov. Optimization of the ILU(0) factorization algorithm with the use of compressed sparse row format. *Journal of Mathematical Sciences*, 191(1):19–27, 2013.

[2] E. Anderson and Y. Saad. Solving sparse triangular linear systems on parallel computers. *International Journal of High Speed Computing*, 1(01):73–95, 1989.

[3] M. Benzi, W. Joubert, et al. Numerical experiments with parallel orderings for ILU preconditioners. *Electronic Trans. Numeric Analysis*, 8:88–114, 1999.

[4] L. Blackford, A. Petitet, et al. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans.s on Mathematical Software*, 28(2):135–151, 2002.

[5] X. Chen, Y. Wang, and H. Yang. NICSLU: An adaptive sparse matrix solver for parallel circuit simulation. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 32(2):261–274, 2013.

[6] X. Chen, Y. Wang, and H. Yang. *Parallel Sparse Direct Solver for Integrated Circuit Simulation*. Springer, 2017.

[7] E. Chow and A. Patel. Fine-grained parallel incomplete LU factorization. *SIAM J. Scientific Comput.*, 37(2):C169–C193, 2015.

[8] T. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1–25, 2011.

[9] T. A. Davis and E. Palamadai Natarajan. Algorithm 907: KLU, A Direct Sparse Solver for Circuit Simulation Problems. *ACM Trans. Math. Softw.*, sep 2010.

[10] L. Dutto and W. Habashi. Parallelization of the ILU(0) preconditioner for CFD problems on shared-memory computers. *International J. numerical methods in fluids*, 30(8):995–1008, 1999.

[11] J. Gilbert and J. Liu. Elimination structures for unsymmetric sparse LU factors. *SIAM J. Matrix Analysis and Applications*, 14(2):334–352, 1993.

[12] M. Heroux, P. Vu, and C. Yang. A parallel preconditioned conjugate gradient package for solving sparse linear systems on a Cray Y-MP. *Applied Numerical Mathematics*, 8(2):93–115, 1991.

[13] D. Hysom and A. Pothen. Efficient parallel computation of ilu (k) preconditioners. In *Proc. ACM/IEEE conference on Supercomputing*, pages 29–es, 1999.

[14] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. *SIAM J. Scientific Comput.*, 22(6):2194–2215, 2001.

[15] Intel Corporation. Developer Reference for Intel® oneAPI Math Kernel Library - C. https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html, 2022.

[16] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Comput.*, 20(1):359–392, 1998.

[17] S. Nassif. IBM power grid benchmarks. https://web.ece.ucsb.edu/~lip/PGBenchmarks/ibmpgbench.html, 2008.

[18] Ngspice. Open source spice simulator. http://ngspice.sourceforge.net/, 2022.

[19] Y. Saad. ILUT: A dual threshold incomplete LU factorization. *Numerical linear algebra with applications*, 1(4):387–402, 1994.

[20] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific and Statistical Computing*, 7(3):856–869, 1986.

[21] K. Shen, X. Jiao, and T. Yang. Elimination forest guided 2D sparse LU factorization. In *Proc. ACM SPAA*, pages 5–15, 1998.

[22] Synopsys, Inc. PrimeSim HSPICE. https://www.synopsys.com/implementation-and-signoff/ams-simulation/primesim-hspice.html, 2022.

[23] H. Thornquist, E. Keiter, R. Hoekstra, D. Day, and E. Boman. A parallel preconditioning strategy for efficient transistor-level circuit simulation. In *Proc. ICCAD*, pages 410–417, 2009.

[24] A. Vladimirescu. *The SPICE Book*. Wiley New York, 1994.

[25] X. Xu. OpenMP parallel implementation of stiffly stable time-stepping projection/GMRES(ILU(0)) implicit simulation of incompressible fluid flows on shared-memory, multicore architecture. *Applied Mathematics and Computation*, 355:238–252, 2019.

[26] J. Zhao, Y. Wen, et al. SFLU: Synchronization-free sparse LU factorization for fast circuit simulation on GPUs. In *Proc. ACM/IEEE DAC*, pages 37–42, 2021.

[27] X. Zhao and Z. Feng. GPSCP: A general-purpose support-circuit preconditioning approach to large-scale SPICE-accurate nonlinear circuit simulations. In *Proc. IEEE/ACM ICCAD*, pages 429–435, 2012.

Thank You !