An Adaptive LU Factorization Algorithm for Parallel Circuit Simulation

Xiaoming Chen, Yu Wang, Huazhong Yang

Nano-scaled Integrated Circuits and Systems Lab

Department of Electronic Engineering

Tsinghua National Laboratory for Information Science and Technology

Tsinghua University, Beijing, China

Outline

- Motivation & related work
- LU factorization basics
- Parallel LU methodology
- Experimental results
- Conclusion

Outline

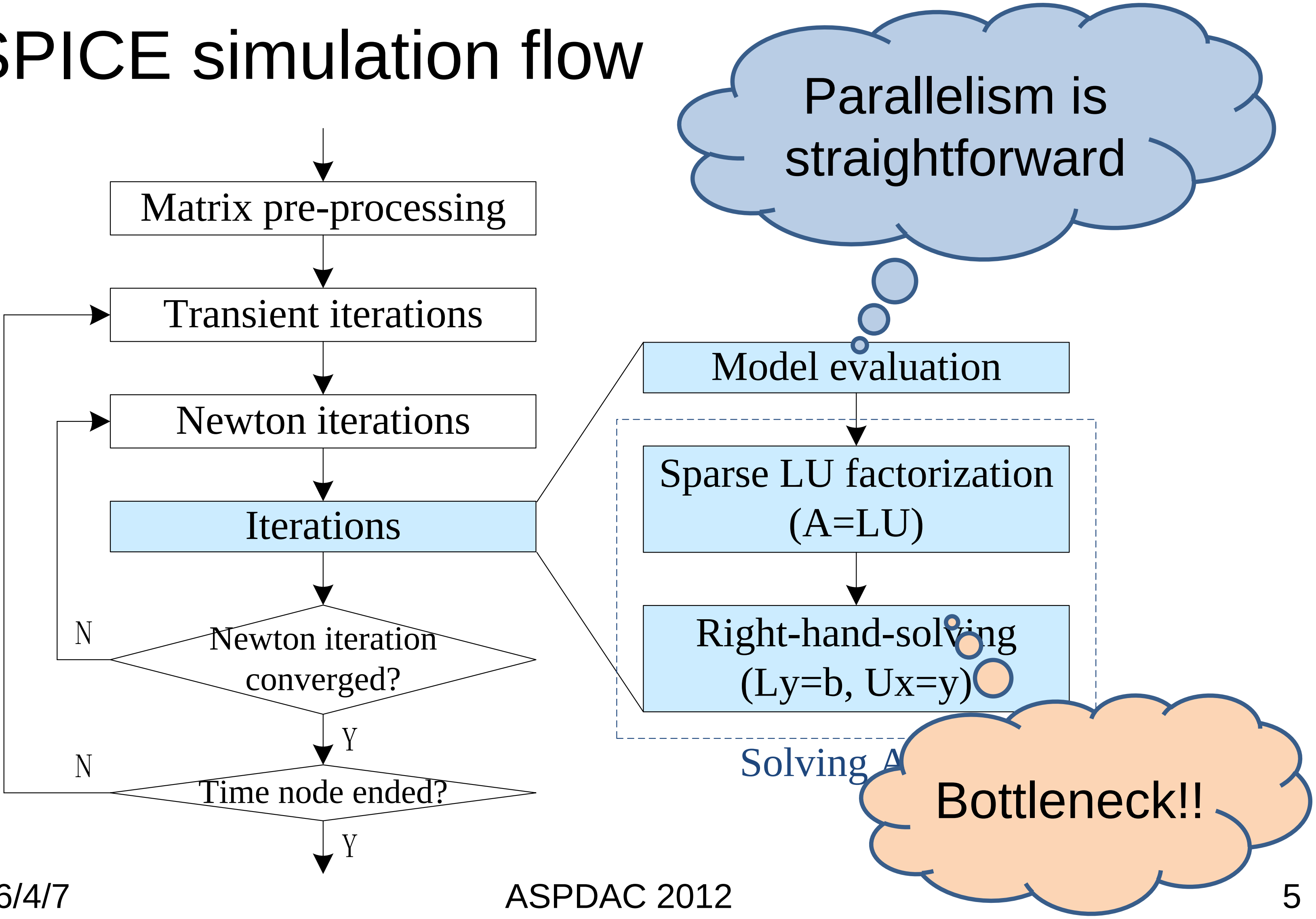
- Motivation & related work
- LU factorization basics
- Parallel LU methodology
- Experimental results
- Conclusion

Motivation

- SPICE: the most widely used circuit simulation engine
  - Challenges: time-consuming, especially in post-layout simulation
  - Requirement: acceleration of SPICE
  - We expect: finish in 12 hours
- Parallel circuit simulation attracts research interests for decades
  - Parallel algorithms by multi-thread are potential solutions

Motivation

- SPICE simulation flow



Motivation

- Features of circuit matrices
  - Extremely sparse
  - Unsymmetric, not positive-definite, usually irregular structure
  - The nonzero pattern remains unchanged during the iterations (no pivoting)
    - The structure of the LU factors are also fixed during the iterations
    - symbolic factorization needs only once
- A special matrix solver for circuit simulation is needed

Related work

- SuperLU(1999)
  - General-purpose matrix solver
  - Sequential/multi-thread/distributed versions
  - Uses Supernodes to handle dense blocks
  - Poor performance for circuit simulation
- Pardiso(2002)
  - General-purpose matrix solver
  - Sequential/multi-thread/distributed versions
  - Also uses Supernodes

•[SuperLU] J. W. Demmel, J. R. Gilbert, and X. S. Li, “An asynchronous parallel supernodal algorithm for sparse gaussian elimination,” *SIAM J. Matrix Analysis and Applications, vol. 20, no. 4, pp. 915–952, 1999.*
•[Pardiso] O. Schenk and K. Gartner, “Solving unsymmetric sparse systems of linear equations with pardiso,” *Computational Science - ICCS 2002,* vol. 2330, pp. 355–363, 2002.

Related work

- KLU(2010)
  - Specially optimized for circuit simulation
  - Only sequential version
- UMFPACK(2004), MUMPS(2006)
  - Multifrontal (dense blocks)

•[KLU] T. A. Davis and E. Palamadai Natarajan, “Algorithm 907: KLU, a direct sparse solver for circuit simulation problems,” *ACM Trans. Math. Softw.,* vol. 37, pp. 36:1–36:17, September 2010
•[UMFPACK] T. A. Davis, “Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method,” *ACM Trans. Math. Softw., vol. 30, pp. 196–199,* June 2004.
•[MUMPS] P. R. Amestoy, A. Guermouche, J.-Y. L’Excellent, and S. Pralet, “Hybrid scheduling for the parallel solution of linear systems,” *Parallel Computing, vol. 32, no. 2, pp. 136–156, 2006.*

Related work

- Among all the public sparse matrix solver implementations, only KLU is specially designed for circuit simulation
  - KLU has no parallel version
  - To our knowledge, currently there’s no research that parallelizes KLU

Outline

- Motivation & related work
- LU factorization basics
- Parallel LU methodology
- Experimental results
- Conclusion

LU factorization basics

- Pre-processing (pre-analysis)
  - performs column/row permutations to increase numeric stability and reduce fill-ins
- Factorization

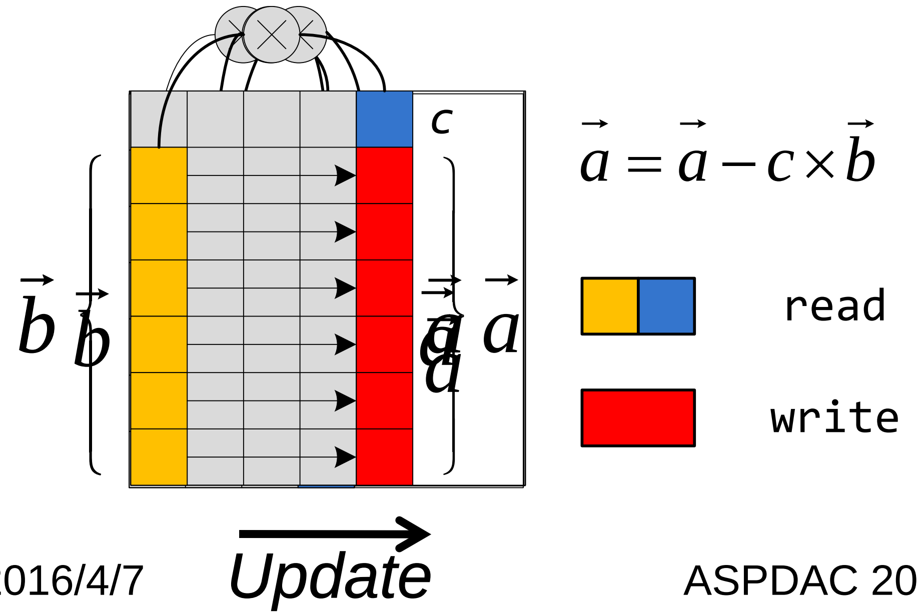
$$A = LU = \begin{bmatrix} l_{11} & & & \\ l_{21} & l_{22} & 0 & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ & 1 & \cdots & u_{2n} \\ & 0 & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

- Right-hand-solving

$$L\overrightarrow{y} = \overrightarrow{b} \qquad U\overrightarrow{x} = \overrightarrow{y}$$

LU factorization basics

- Left-looking algorithm
  - Factorizes the matrix by sequentially processing each column
  - When factorizing each column (say k), it uses all the left columns (1, 2, ..., k-1) to update self



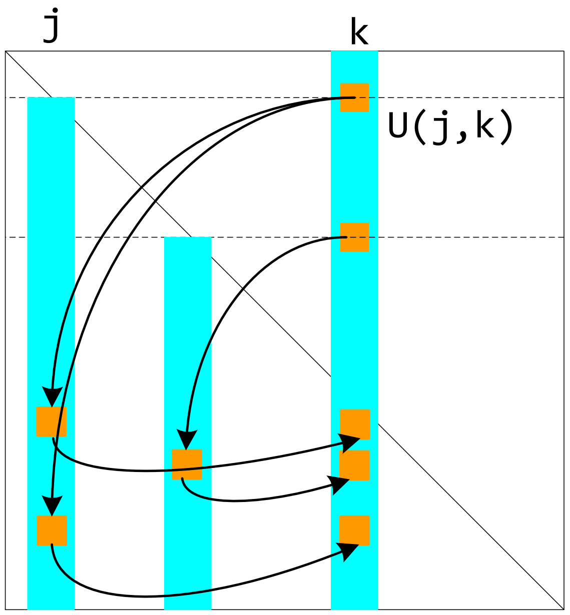
•Suitable for cache based architecture

LU factorization basics

- Left-looking algorithm
  - If the matrix is dense, each column (k) depends on all of its left columns (1, 2, ..., k-1)
  - A complete sequential algorithm, strong data dependency, hard to be parallelized

LU factorization basics

- When the matrix is sparse...
  - Each column only depends on part of its left columns
  - **Column k depends on column j, iff U(j, k) ≠ 0 (j<k)**
  - **The structure of U determines the column-level dependency**
- Sparse left-looking algorithm
  - Gilbert-Peierls (G-P) algorithm
  - KLU is an implementation of the G-P algorithm



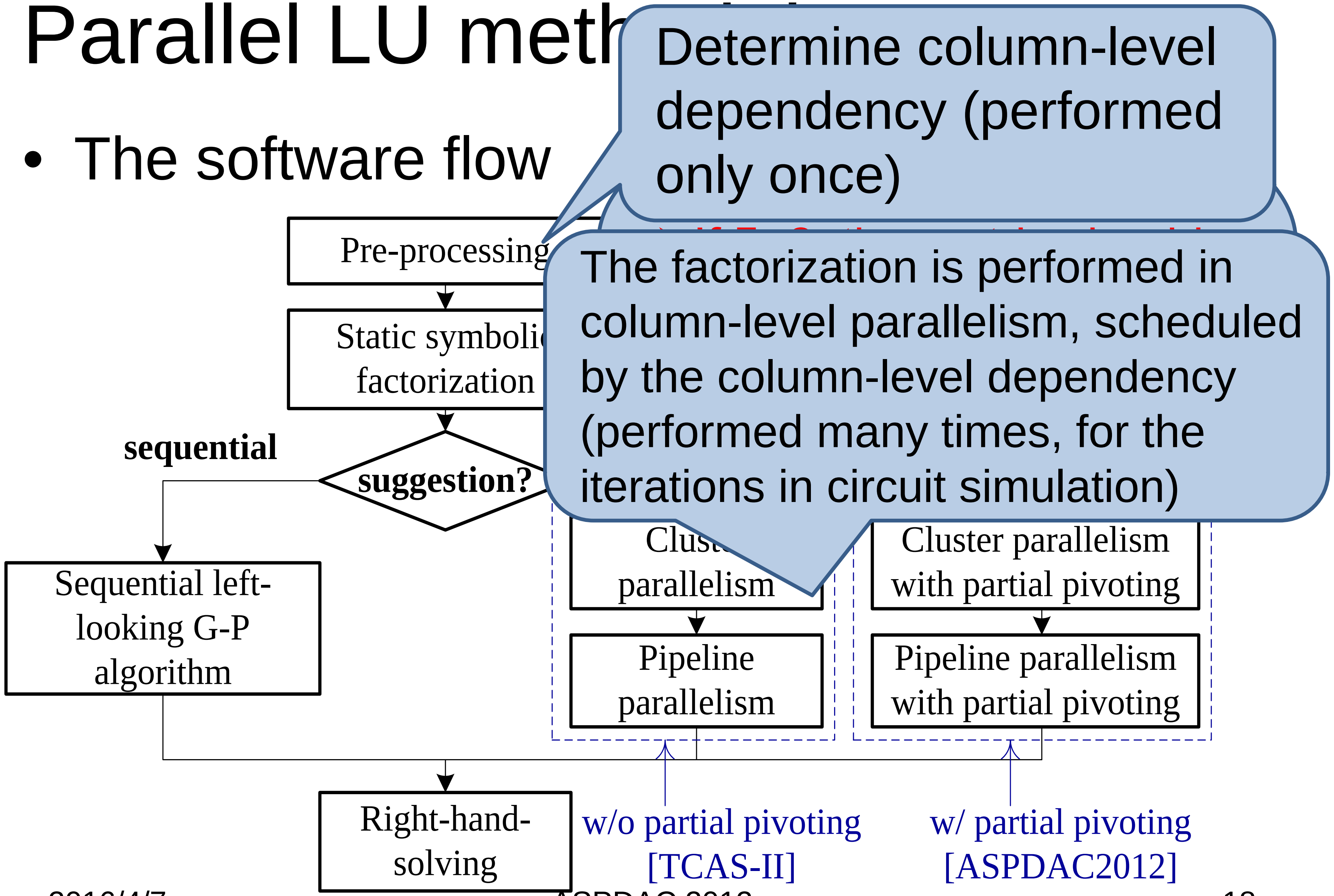
[G-P] J. R. Gilbert and T. Peierls, "Sparse partial pivoting in time proportional to arithmetic operations," SIAM J. Sci. Statist. Comput. , vol. 9, pp. 862–874, 1988.

Outline

- Motivation & related work
- LU factorization basics
- Parallel LU methodology
- Experimental results
- Conclusion

Parallel LU meth

- The software flow


Determine column-level dependency (performed only once)
The factorization is performed in column-level parallelism, scheduled by the column-level dependency (performed many times, for the iterations in circuit simulation)
Pre-processing
Static symboli factorization
sequential
suggestion?
Sequential left-looking G-P algorithm
Clust parallelism
Pipeline parallelism
Cluster parallelism with partial pivoting
Pipeline parallelism with partial pivoting
Right-hand-solving
w/o partial pivoting [TCAS-II]
w/ partial pivoting [ASPDAC2012]

Parallel LU methodology

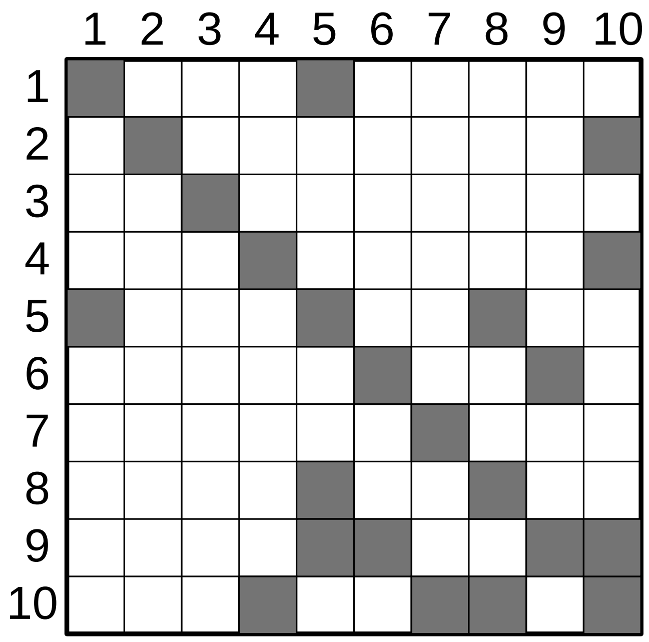
- Partial pivoting
  - During iterations, there may be some 0’s or small values on the diagonal
    - Find the maximum element in each column, and swap it to the diagonal
  - Partial pivoting can interchange the rows
  - The symbolic structure of L and U depends on pivot choices, not fixed during iterations
  - The exact column-level dependency cannot be obtained before factorization (The exact column-level dependency is determined by the structure of U)

Parallel LU methodology

- *Elimination Tree (ETree)* is used to represent the column-level dependency
  - J. W. H. Liu, “The role of elimination trees in sparse factorization,” SIAM J. Matrix Analysis and Applications , vol. 11, pp. 134–172, 1990.
- ETree contains all potential dependency, regardless of the actual pivot choices
  - overestimates the actual dependency

Parallel LU methodology

- Elimination Tree (ETree)



(a) matrix *A*

$$\begin{tikzcd}
& 1 \arrow[d] & & 3 \\
& 5 \arrow[d] & 2 \arrow[d] & \\
6 \arrow[dr] & 8 \arrow[d] & 4 \arrow[dl] & \\
& 9 \arrow[d] & & 7 \arrow[dll] \\
& 10 & &
\end{tikzcd}$$

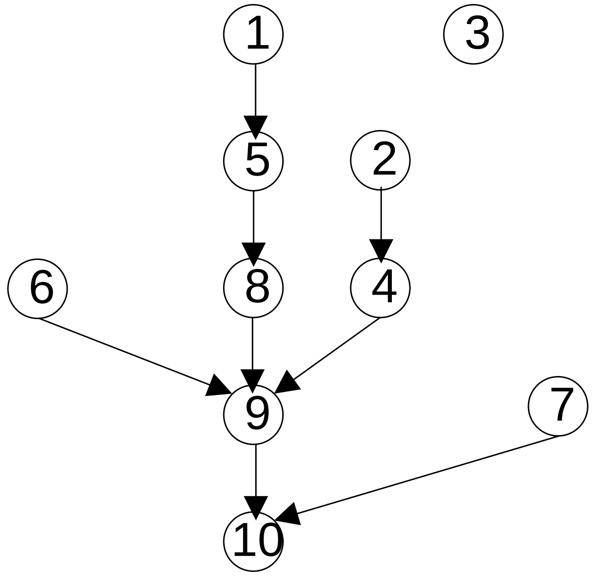
(b) *ETree*

Node ⇔ column

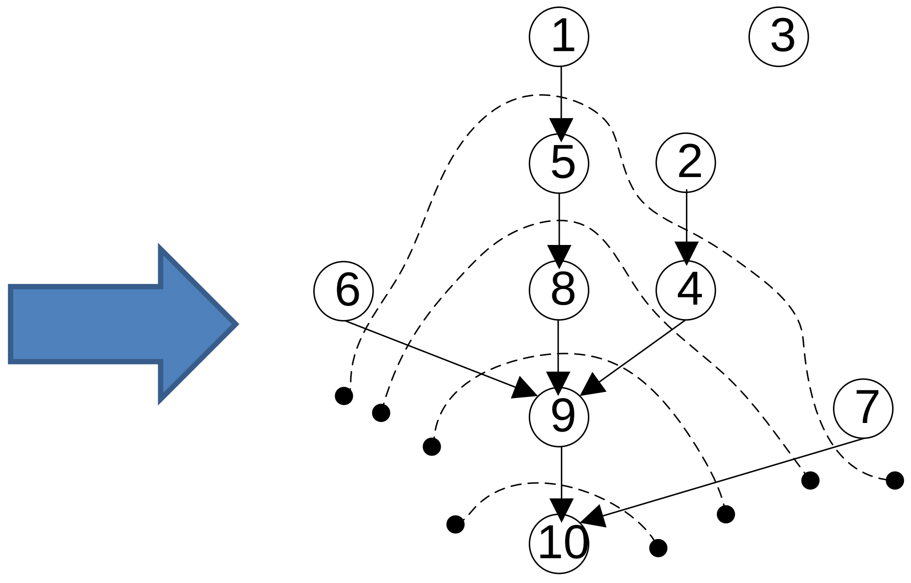
Edge: the potential dependency

Parallel LU methodology

- Level
  - the longest distance from each node to the leaf nodes



(b) *ETree*

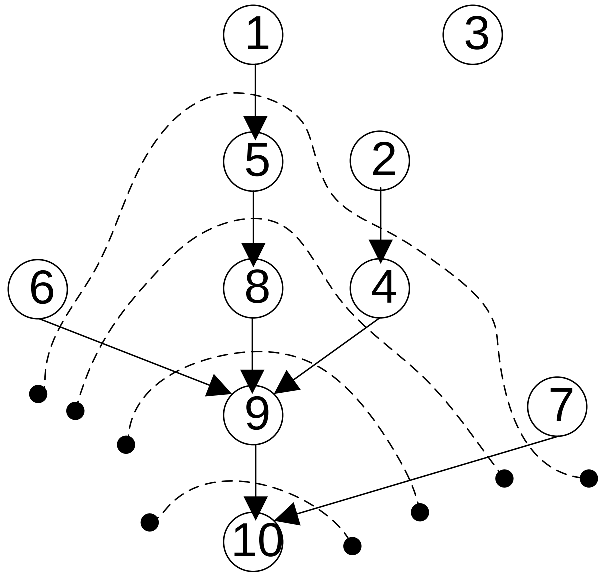


(c) *level*

The nodes in the same level are independent!

Parallel LU methodology

- ***Elimination Scheduler (EScheduler)***
  - EScheduler is the primary scheduler in our solver



(c) *level*

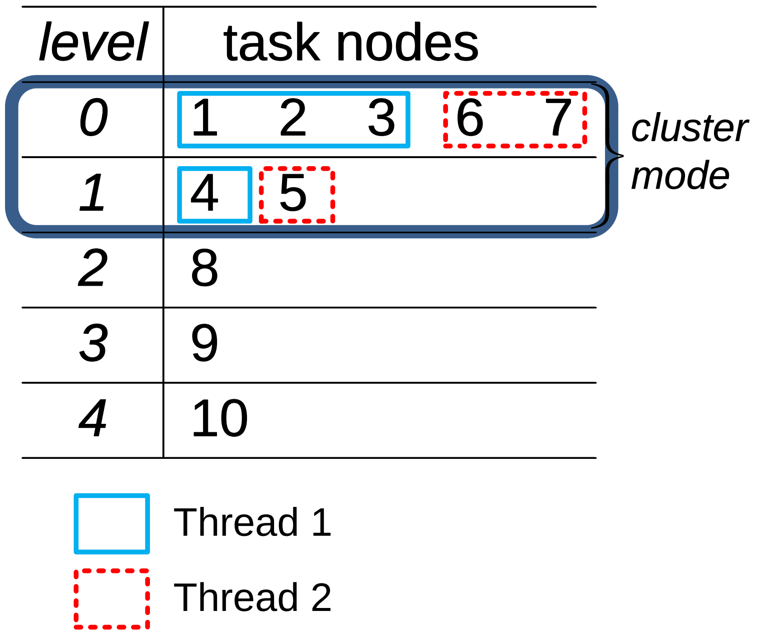
level	task nodes				
0	1	2	3	6	7
1	4	5			
2	8				
3	9				
4	10				

(d) *ESheduler*

The nodes in the same level are independent!

Parallel LU methodology

- Task scheduling
  - Cluster mode



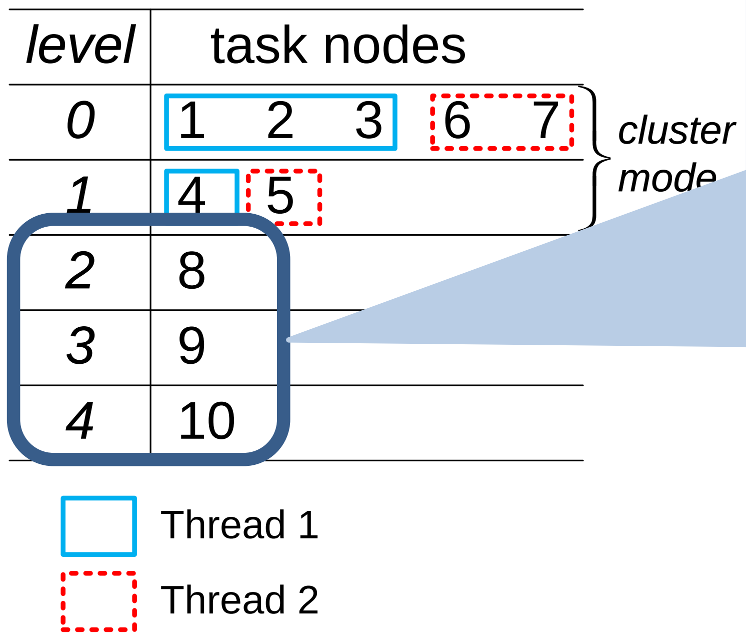
(d) *ESheduler*

➢The nodes in the same level are independent

➢Parallel method 1: cluster mode
➢Level by level
➢For each level, equally assign the nodes to all the cores, the nodes assigned to one core is called a "cluster"

Parallel LU methodology

- Task scheduling
  - Pipeline mode



(d) *ESheduler*

➢Cluster mode cannot achieve effective speedup

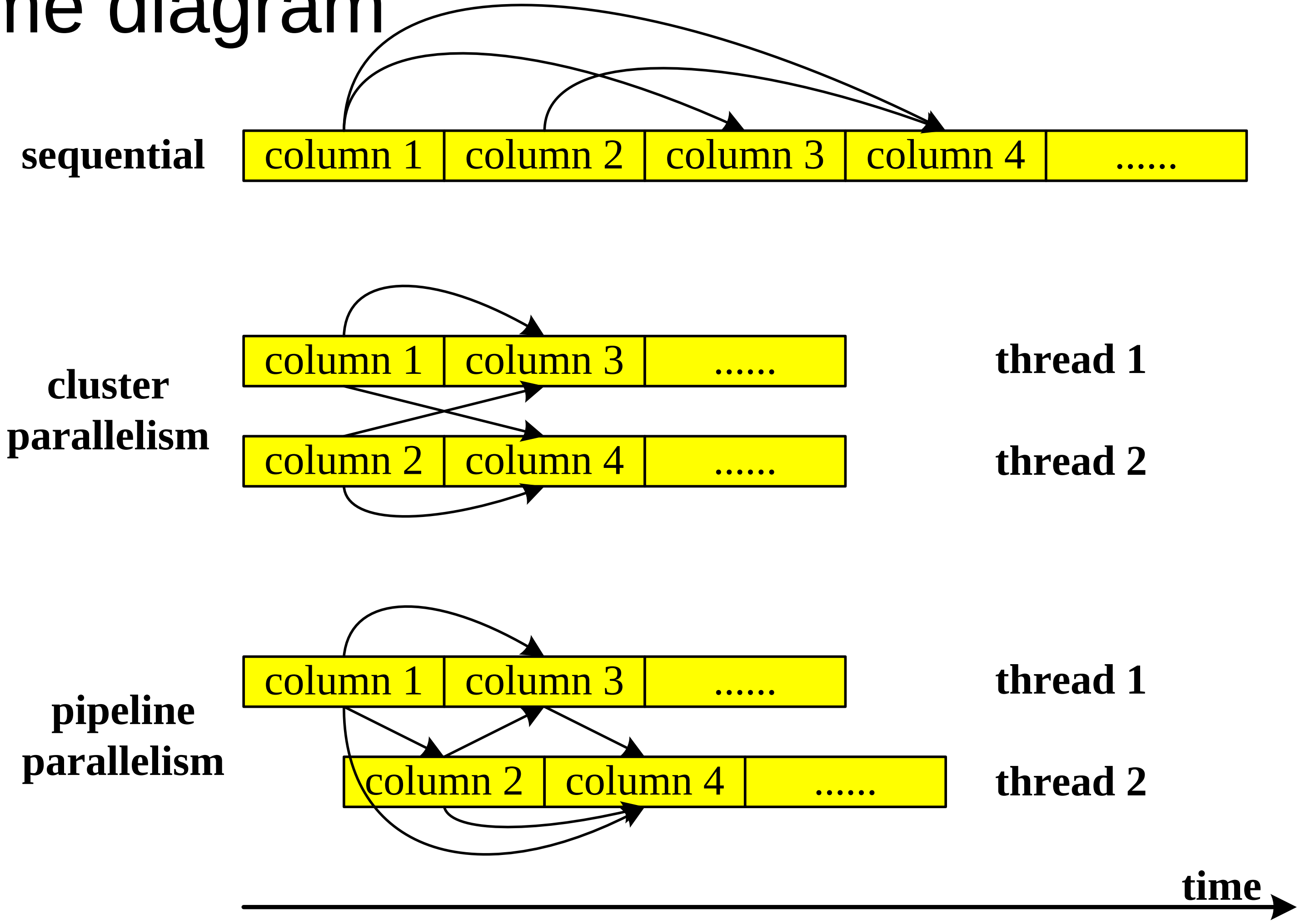
➢The nodes in different levels may be dependent

➢Parallel method 2: pipeline

➢exploits parallelism between dependent levels

Parallel LU methodology

- Time diagram



Outline

- Motivation & related work
- LU factorization basics
- Parallel LU methodology
- Experimental results
- Conclusion

Experimental results

- Linux server
  - 2 Intel Xeon5670 CPUs, 12 cores in total
  - 24GB memory
- Test benchmarks: Tim Davis, University of Florida Sparse Matrix Collection
  - http://www.cise.ufl.edu/research/sparse/matrices/index.html
- Two types of speedups
  - **Speedup**: speedups over KLU
  - **Relative speedup**: speedups over our 1-core performance

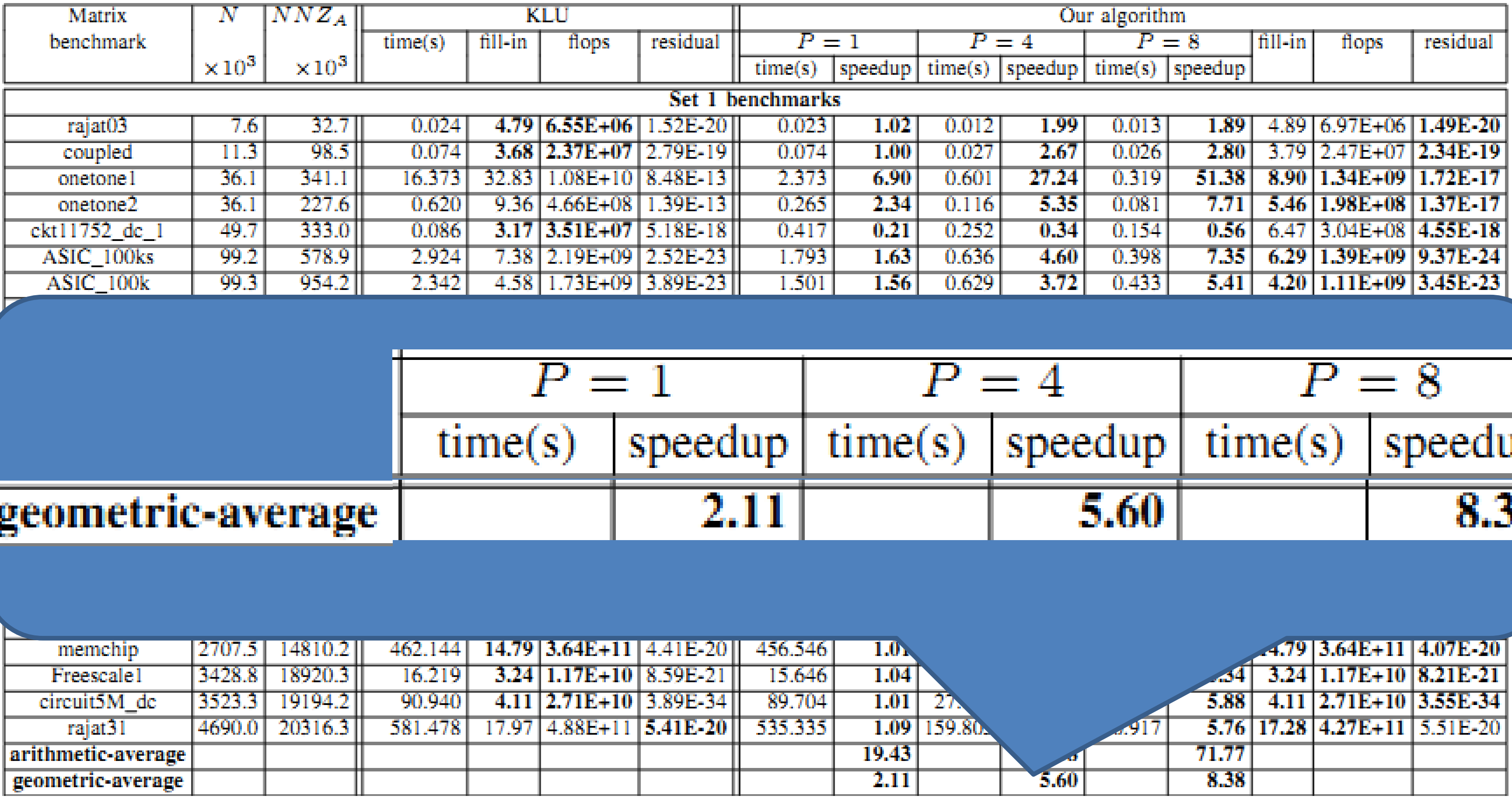
Experimental results

- Benchmark statistics

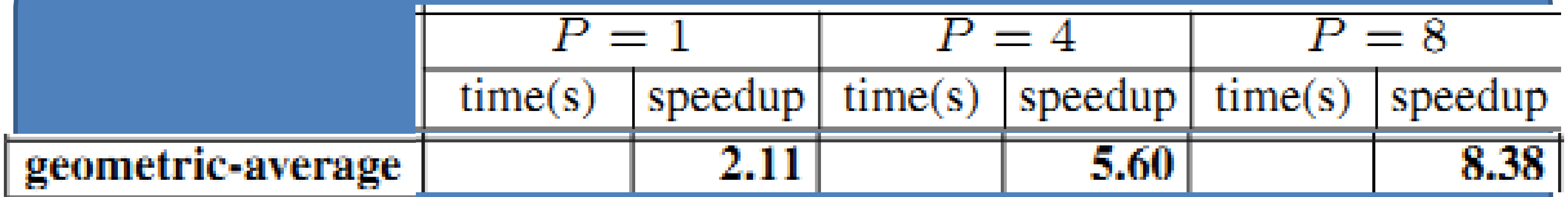
	min	max
#benchmarks	35	
Dimension (N)	2.4k by 2.4k	5.5M by 5.5M
density (NNZ(A)/N/N)	9.2E-7	0.0052
row density (NNZ(A)/N)	3.4	10.1
NNZ(L+U)/NNZ(A)	1	86
condition number	2738	1.79E+21
Average pre-processing time	4.64	
Average factorization time (1-core)	101.72	
Average right-hand-solving time	0.16	

Experimental results

- **Speedup** over KLU, on Set 1 (the matrices which are suitable for parallel factorization)



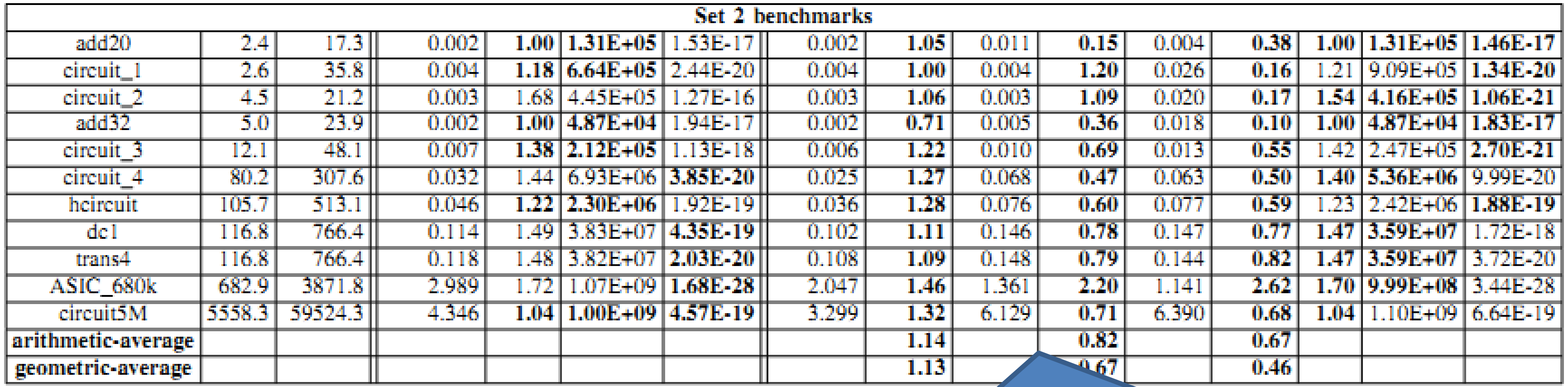
Matrix benchmark	N $\times 10^3$	NNZ_A $\times 10^3$	KLU time(s)	KLU fill-in	KLU flops	KLU residual	Our algorithm $P=1$ time(s)	$P=1$ speedup	$P=4$ time(s)	$P=4$ speedup	$P=8$ time(s)	$P=8$ speedup	fill-in	flops	residual
Set 1 benchmarks															
rajat03	7.6	32.7	0.024	**4.79**	**6.55E+06**	1.52E-20	0.023	**1.02**	0.012	**1.99**	0.013	**1.89**	4.89	6.97E+06	**1.49E-20**
coupled	11.3	98.5	0.074	**3.68**	**2.37E+07**	2.79E-19	0.074	**1.00**	0.027	**2.67**	0.026	**2.80**	3.79	2.47E+07	**2.34E-19**
onetone1	36.1	341.1	16.373	32.83	1.08E+10	8.48E-13	2.373	**6.90**	0.601	**27.24**	0.319	**51.38**	**8.90**	**1.34E+09**	**1.72E-17**
onetone2	36.1	227.6	0.620	9.36	4.66E+08	1.39E-13	0.265	**2.34**	0.116	**5.35**	0.081	**7.71**	**5.46**	**1.98E+08**	**1.37E-17**
ckt11752_dc_1	49.7	333.0	0.086	**3.17**	**3.51E+07**	5.18E-18	0.417	**0.21**	0.252	**0.34**	0.154	**0.56**	6.47	3.04E+08	**4.55E-18**
ASIC_100ks	99.2	578.9	2.924	7.38	2.19E+09	2.52E-23	1.793	**1.63**	0.636	**4.60**	0.398	**7.35**	**6.29**	**1.39E+09**	**9.37E-24**
ASIC_100k	99.3	954.2	2.342	4.58	1.73E+09	3.89E-23	1.501	**1.56**	0.629	**3.72**	0.433	**5.41**	**4.20**	**1.11E+09**	**3.45E-23**
memchip	2707.5	14810.2	462.144	**14.79**	**3.64E+11**	4.41E-20	456.546	**1.0**					**4.79**	**3.64E+11**	**4.07E-20**
Freescale1	3428.8	18920.3	16.219	**3.24**	**1.17E+10**	8.59E-21	15.646	**1.04**				**34**	**3.24**	**1.17E+10**	**8.21E-21**
circuit5M_dc	3523.3	19194.2	90.940	**4.11**	**2.71E+10**	3.89E-34	89.704	**1.01**	27.			**5.88**	**4.11**	**2.71E+10**	**3.55E-34**
rajat31	4690.0	20316.3	581.478	17.97	4.88E+11	**5.41E-20**	535.335	**1.09**	159.80		.917	**5.76**	**17.28**	**4.27E+11**	5.51E-20
arithmetic-average								**19.43**				**71.77**			
geometric-average								**2.11**		**5.60**		**8.38**			



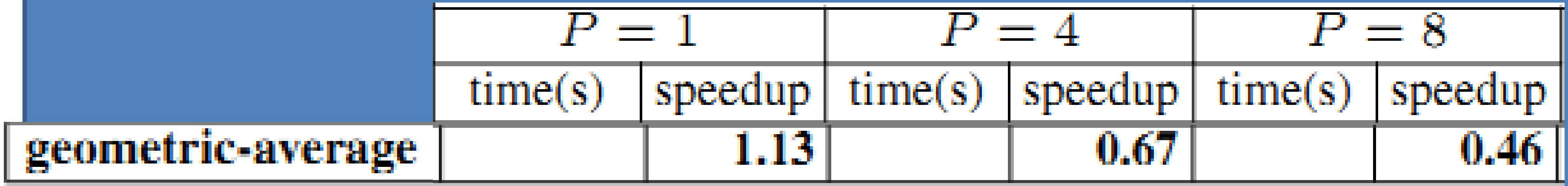
	$P=1$ time(s)	$P=1$ speedup	$P=4$ time(s)	$P=4$ speedup	$P=8$ time(s)	$P=8$ speedup
geometric-average		**2.11**		**5.60**		**8.38**

Experimental results

- **Speedup** over KLU, on Set 2 (the matrices which are suitable for sequential factorization)



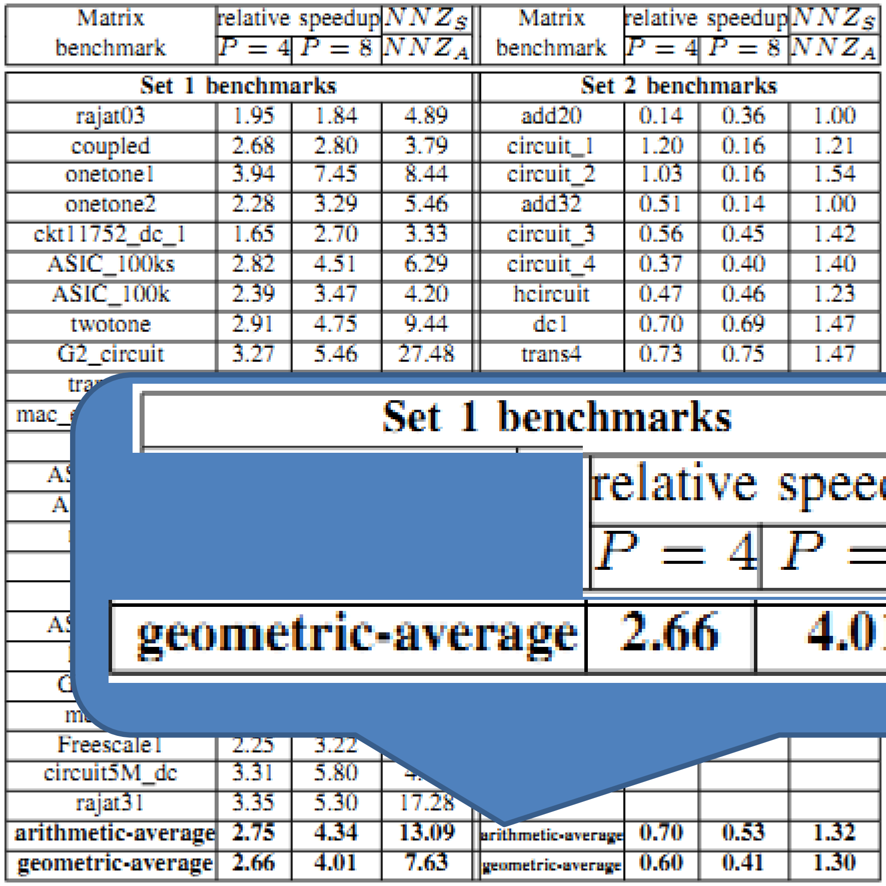
Set 2 benchmarks															
add20	2.4	17.3	0.002	1.00	1.31E+05	1.53E-17	0.002	1.05	0.011	0.15	0.004	0.38	1.00	1.31E+05	1.46E-17
circuit_1	2.6	35.8	0.004	1.18	6.64E+05	2.44E-20	0.004	1.00	0.004	1.20	0.026	0.16	1.21	9.09E+05	1.34E-20
circuit_2	4.5	21.2	0.003	1.68	4.45E+05	1.27E-16	0.003	1.06	0.003	1.09	0.020	0.17	1.54	4.16E+05	1.06E-21
add32	5.0	23.9	0.002	1.00	4.87E+04	1.94E-17	0.002	0.71	0.005	0.36	0.018	0.10	1.00	4.87E+04	1.83E-17
circuit_3	12.1	48.1	0.007	1.38	2.12E+05	1.13E-18	0.006	1.22	0.010	0.69	0.013	0.55	1.42	2.47E+05	2.70E-21
circuit_4	80.2	307.6	0.032	1.44	6.93E+06	3.85E-20	0.025	1.27	0.068	0.47	0.063	0.50	1.40	5.36E+06	9.99E-20
hcircuit	105.7	513.1	0.046	1.22	2.30E+06	1.92E-19	0.036	1.28	0.076	0.60	0.077	0.59	1.23	2.42E+06	1.88E-19
dc1	116.8	766.4	0.114	1.49	3.83E+07	4.35E-19	0.102	1.11	0.146	0.78	0.147	0.77	1.47	3.59E+07	1.72E-18
trans4	116.8	766.4	0.118	1.48	3.82E+07	2.03E-20	0.108	1.09	0.148	0.79	0.144	0.82	1.47	3.59E+07	3.72E-20
ASIC_680k	682.9	3871.8	2.989	1.72	1.07E+09	1.68E-28	2.047	1.46	1.361	2.20	1.141	2.62	1.70	9.99E+08	3.44E-28
circuit5M	5558.3	59524.3	4.346	1.04	1.00E+09	4.57E-19	3.299	1.32	6.129	0.71	6.390	0.68	1.04	1.10E+09	6.64E-19
arithmetic-average								1.14		0.82		0.67			
geometric-average								1.13		0.67		0.46			



	$P = 1$		$P = 4$		$P = 8$	
	time(s)	speedup	time(s)	speedup	time(s)	speedup
geometric-average		1.13		0.67		0.46

Experimental results

- **Relative speedup** over our 1-core solver

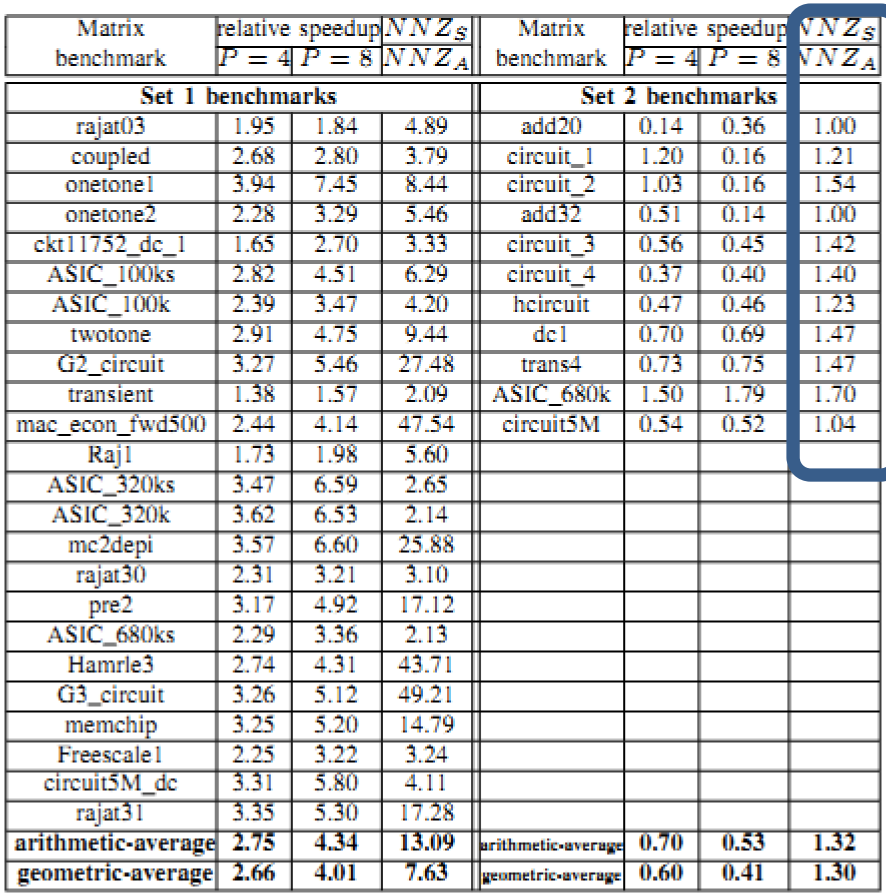


Matrix benchmark	relative speedup $P=4$	relative speedup $P=8$	$\frac{NNZ_S}{NNZ_A}$	Matrix benchmark	relative speedup $P=4$	relative speedup $P=8$	$\frac{NNZ_S}{NNZ_A}$
Set 1 benchmarks				**Set 2 benchmarks**			
rajat03	1.95	1.84	4.89	add20	0.14	0.36	1.00
coupled	2.68	2.80	3.79	circuit_1	1.20	0.16	1.21
onetone1	3.94	7.45	8.44	circuit_2	1.03	0.16	1.54
onetone2	2.28	3.29	5.46	add32	0.51	0.14	1.00
ckt11752_dc_1	1.65	2.70	3.33	circuit_3	0.56	0.45	1.42
ASIC_100ks	2.82	4.51	6.29	circuit_4	0.37	0.40	1.40
ASIC_100k	2.39	3.47	4.20	hcircuit	0.47	0.46	1.23
twotone	2.91	4.75	9.44	dc1	0.70	0.69	1.47
G2_circuit	3.27	5.46	27.48	trans4	0.73	0.75	1.47
tra	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
mac_	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
AS	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
A	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
AS	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
G	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
m	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Freescale1	2.25	3.22	[illegible]	[illegible]	[illegible]		
circuit5M_dc	3.31	5.80	4.	[illegible]	[illegible]		
rajat31	3.35	5.30	17.28	[illegible]			
arithmetic-average	**2.75**	**4.34**	**13.09**	**arithmetic-average**	**0.70**	**0.53**	**1.32**
geometric-average	**2.66**	**4.01**	**7.63**	**geometric-average**	**0.60**	**0.41**	**1.30**

Set 1 benchmarks	relative speedup $P=4$	relative speedup $P=8$	**Set 2 benchmarks**	relative speedup $P=4$	relative speedup $P=8$
geometric-average	**2.66**	**4.01**	**geometric-average**	**0.60**	**0.41**

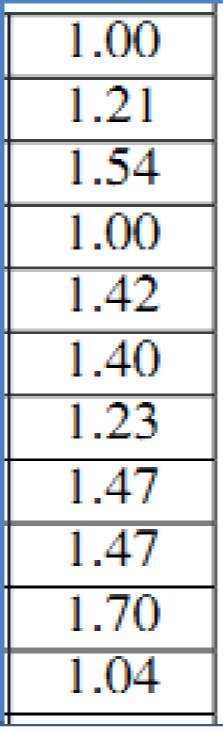
Experimental results

- Estimated relative fill-in



Matrix benchmark	relative speedup $P=4$	$P=8$	$\frac{NNZ_S}{NNZ_A}$	Matrix benchmark	relative speedup $P=4$	$P=8$	$\frac{NNZ_S}{NNZ_A}$
Set 1 benchmarks				**Set 2 benchmarks**			
rajat03	1.95	1.84	4.89	add20	0.14	0.36	1.00
coupled	2.68	2.80	3.79	circuit_1	1.20	0.16	1.21
onetone1	3.94	7.45	8.44	circuit_2	1.03	0.16	1.54
onetone2	2.28	3.29	5.46	add32	0.51	0.14	1.00
ckt11752_dc_1	1.65	2.70	3.33	circuit_3	0.56	0.45	1.42
ASIC_100ks	2.82	4.51	6.29	circuit_4	0.37	0.40	1.40
ASIC_100k	2.39	3.47	4.20	hcircuit	0.47	0.46	1.23
twotone	2.91	4.75	9.44	dc1	0.70	0.69	1.47
G2_circuit	3.27	5.46	27.48	trans4	0.73	0.75	1.47
transient	1.38	1.57	2.09	ASIC_680k	1.50	1.79	1.70
mac_econ_fwd500	2.44	4.14	47.54	circuit5M	0.54	0.52	1.04
Raj1	1.73	1.98	5.60				
ASIC_320ks	3.47	6.59	2.65				
ASIC_320k	3.62	6.53	2.14				
mc2depi	3.57	6.60	25.88				
rajat30	2.31	3.21	3.10				
pre2	3.17	4.92	17.12				
ASIC_680ks	2.29	3.36	2.13				
Hamrle3	2.74	4.31	43.71				
G3_circuit	3.26	5.12	49.21				
memchip	3.25	5.20	14.79				
Freescale1	2.25	3.22	3.24				
circuit5M_dc	3.31	5.80	4.11				
rajat31	3.35	5.30	17.28				
arithmetic-average	**2.75**	**4.34**	**13.09**	**arithmetic-average**	**0.70**	**0.53**	**1.32**
geometric-average	**2.66**	**4.01**	**7.63**	**geometric-average**	**0.60**	**0.41**	**1.30**

Set 2
NNZ(L+U) / NNZ(A)



1.00
1.21
1.54
1.00
1.42
1.40
1.23
1.47
1.47
1.70
1.04

Experimental results

- Not every matrix is suitable for parallel algorithm
  - If the numeric computation time is little, the parallel overheads will dominate in the total runtime, such as: scheduling time, synchronization time, memory/cache conflicts...
  - NNZ(L+U) / NNZ(A) (the relative fill-in) can effectively predict the sequential/parallel decision

Outline

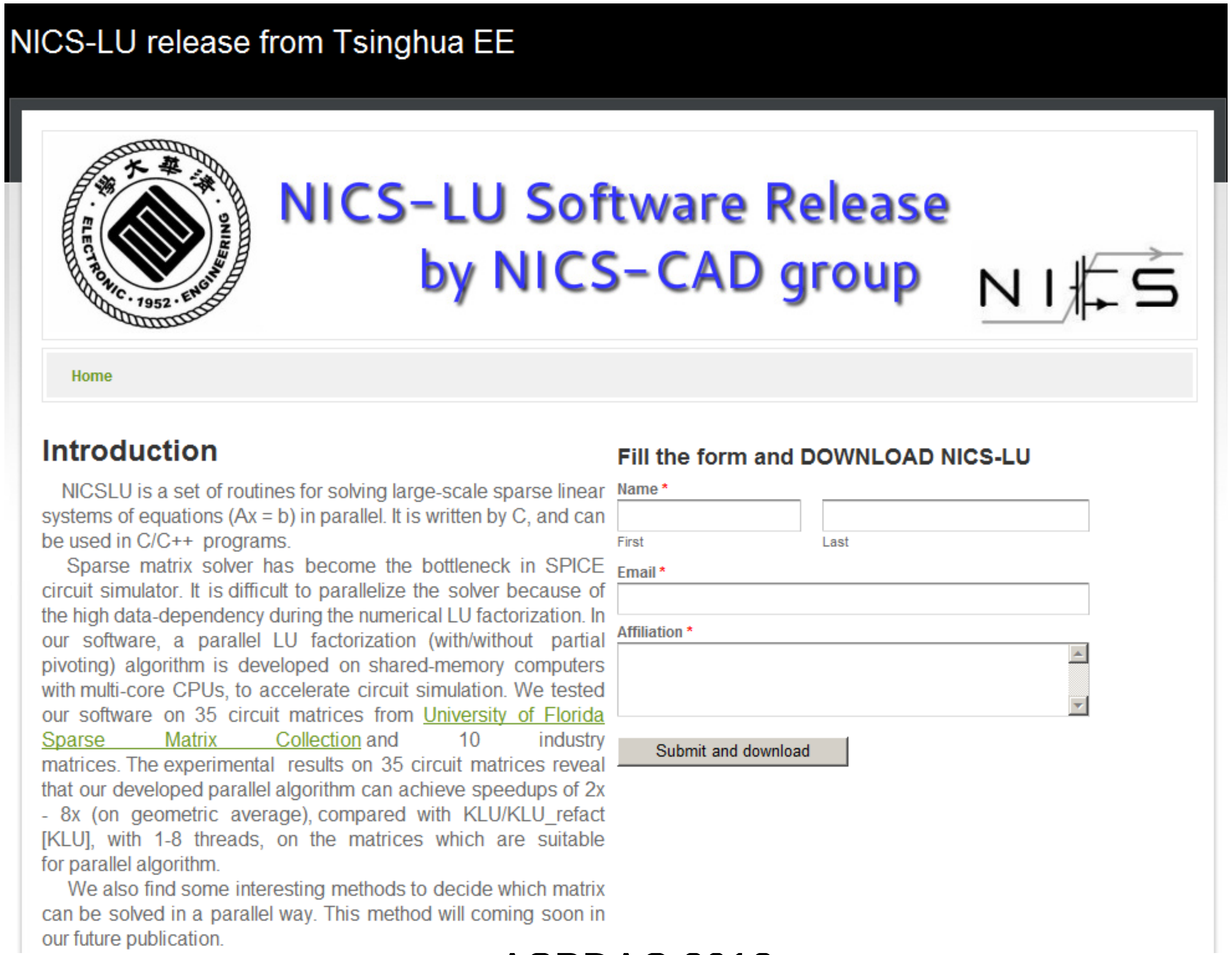
- Motivation & related work
- LU factorization basics
- Parallel LU methodology
- Experimental results
- Conclusion

Conclusion

- Column-level parallel LU factorization algorithm for circuit simulation
  - Two parallel modes to fit the different data-dependency and reduce scheduling overhead
- A simple method to decide whether a matrix should use parallel or sequential algorithm
  - Each matrix can achieve the optimal performance

Website

- http://nicslu.weebly.com/index.html



NICS-LU release from Tsinghua EE

NICS-LU Software Release by NICS-CAD group

Introduction

NICSLU is a set of routines for solving large-scale sparse linear systems of equations (Ax = b) in parallel. It is written by C, and can be used in C/C++ programs.

Sparse matrix solver has become the bottleneck in SPICE circuit simulator. It is difficult to parallelize the solver because of the high data-dependency during the numerical LU factorization. In our software, a parallel LU factorization (with/without partial pivoting) algorithm is developed on shared-memory computers with multi-core CPUs, to accelerate circuit simulation. We tested our software on 35 circuit matrices from University of Florida Sparse Matrix Collection and 10 industry matrices. The experimental results on 35 circuit matrices reveal that our developed parallel algorithm can achieve speedups of 2x - 8x (on geometric average), compared with KLU/KLU_refact [KLU], with 1-8 threads, on the matrices which are suitable for parallel algorithm.

We also find some interesting methods to decide which matrix can be solved in a parallel way. This method will coming soon in our future publication.

Fill the form and DOWNLOAD NICS-LU

Name *

First Last

Email *

Affiliation *

Submit and download

Thanks for your attention