DAC 60

WHERE INNOVATION BEGINS

DESIGN AUTOMATION CONFERENCE

Accelerating Sparse LU Factorization with Density-Aware Adaptive Matrix Multiplication for Circuit Simulation

Tengcheng Wang, Wenhao Li, Haojie Pei, Yuying Sun, Zhou Jin and Weifeng Liu

Super Scientific Software Laboratory, China University of Petroleum-Beijing, China

Email: jinzhou@cup.edu.cn

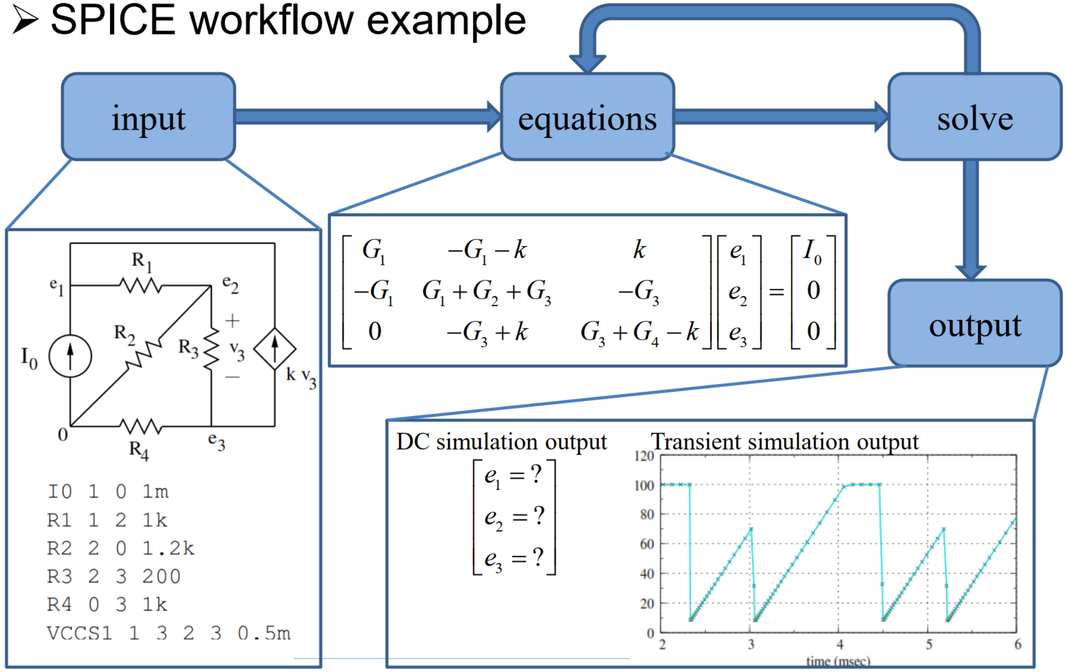
Outline

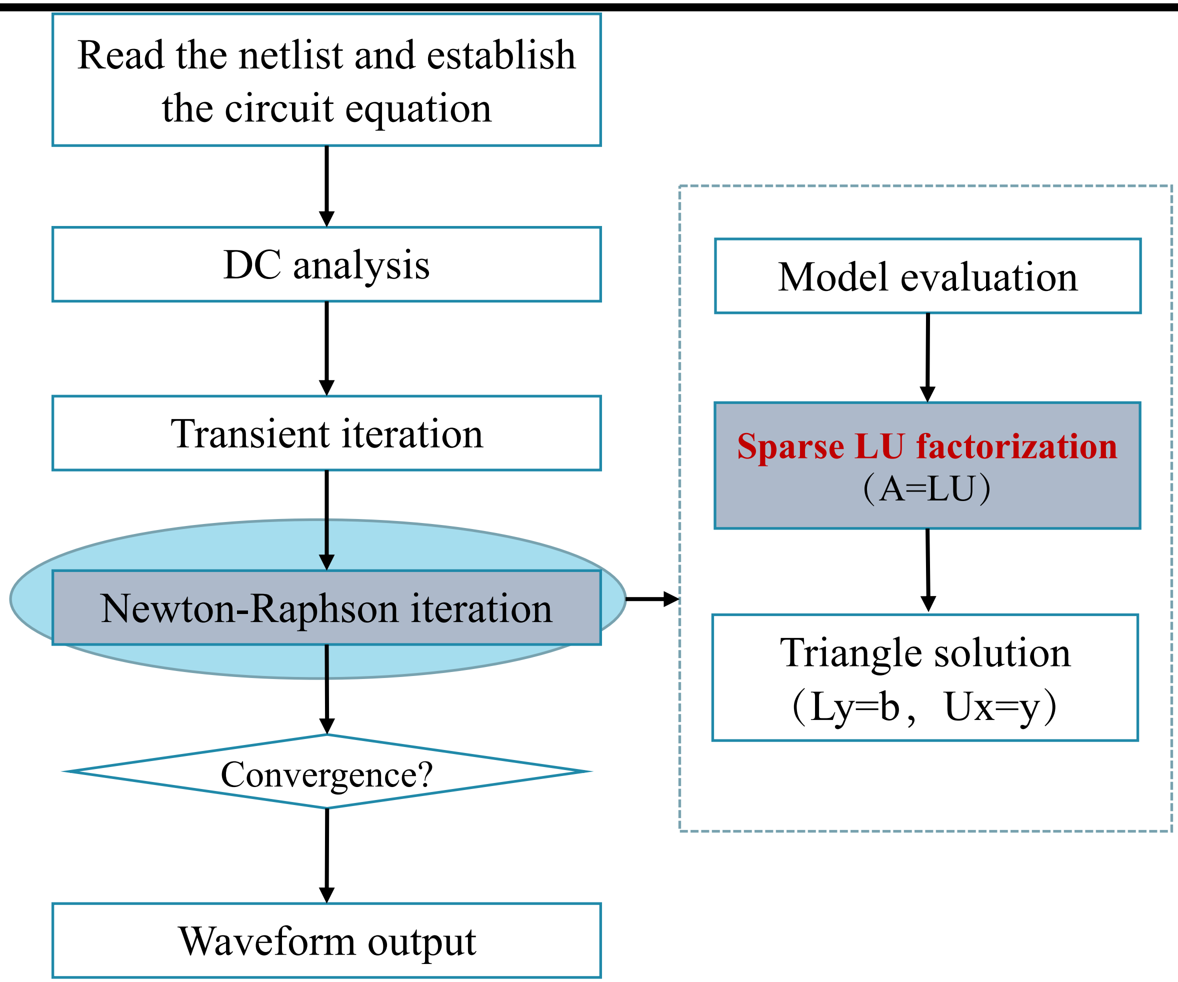
- Background
- Motivation
- Density-aware matrix multiplication
- Machine learning driven adaptive acceleration
- Experimental results
- Conclusions

01 Background

With the development of integrated circuit processes, **the device feature size decreases rapidly, the circuit size grows**, and the **post-layout SPICE simulation** considering parasitic effects is significantly time consuming.

- SPICE workflow example



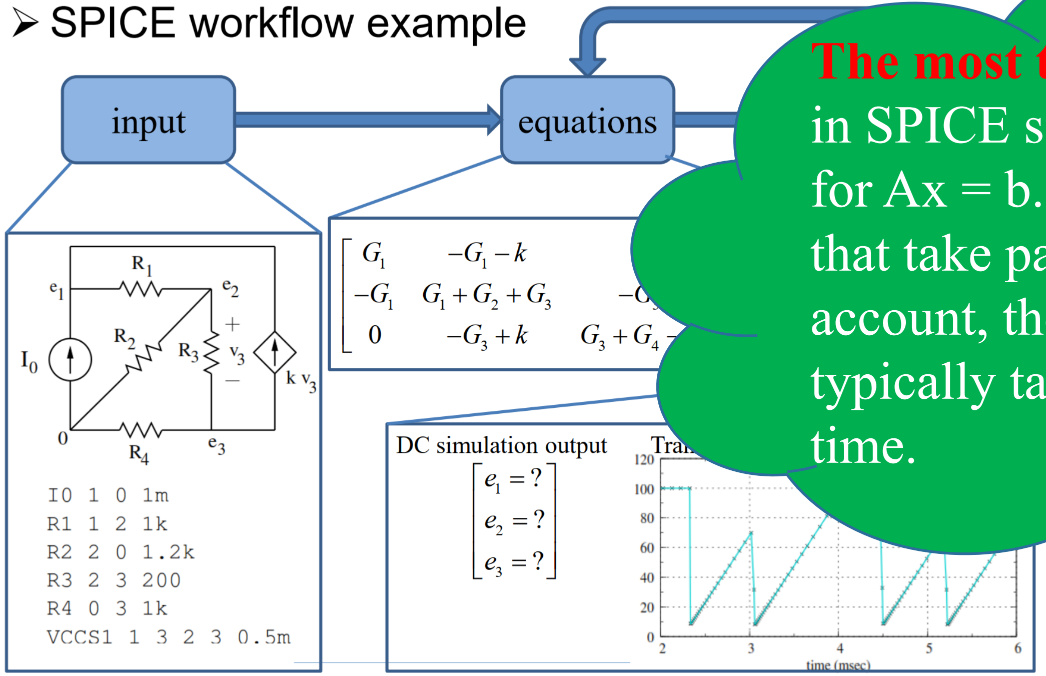


[1] Tien-Hsiung Weng, Ruey-Kuen Perng, Barbara Chapman. OpenMP implementation of SPICE3 circuit simulator. International journal of parallel programming. 2007.

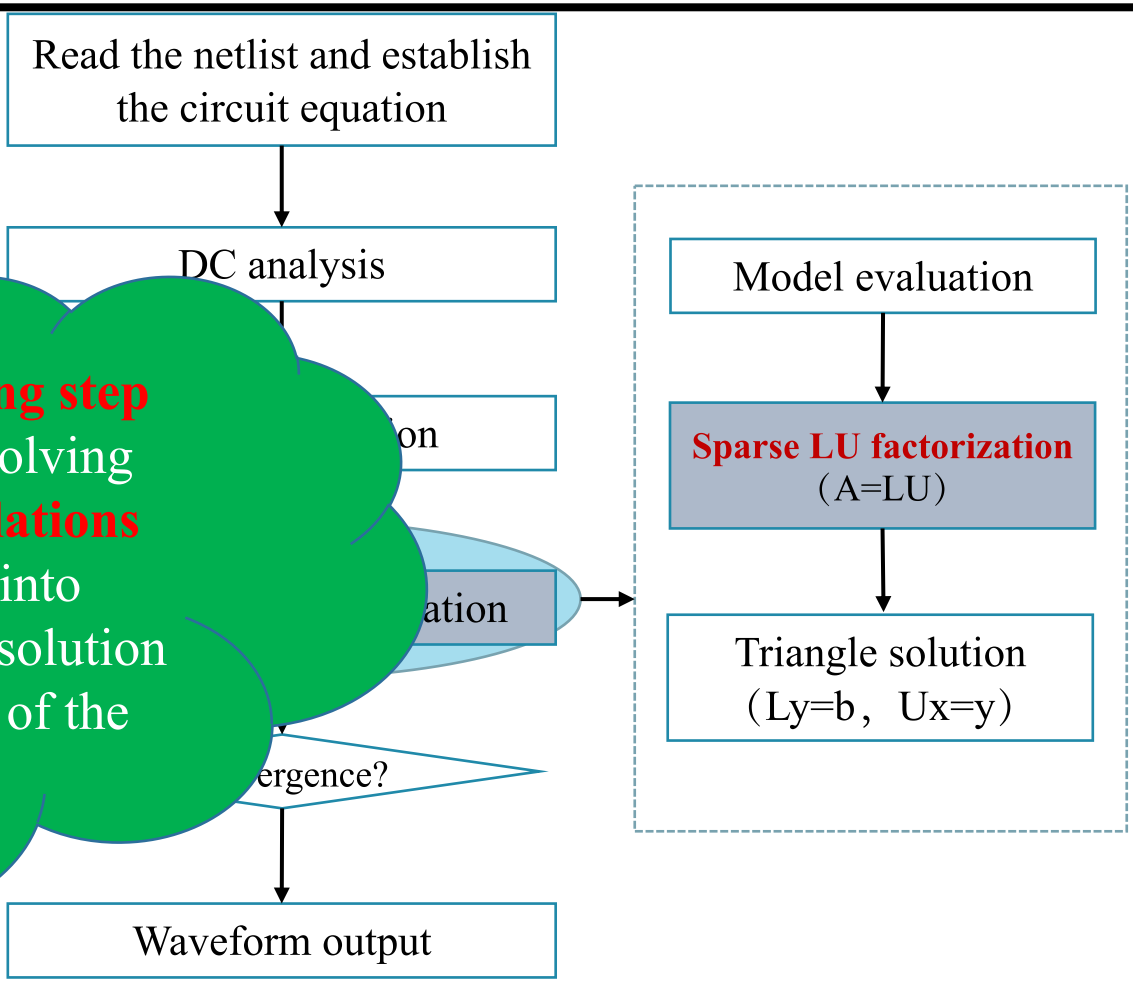
01 Background

With the development of integrated circuit processes, **the device feature size decreases rapidly, the circuit size grows**, and the **post-layout SPICE simulation** considering parasitic effects is significantly time consuming.

- SPICE workflow example



The most time-consuming step in SPICE simulations is solving for Ax = b. **In post-simulations** that take parasitic effects into account, the linear direct solution typically takes **60-90%**[1] of the time.



[1] Tien-Hsiung Weng, Ruey-Kuen Perng, Barbara Chapman. OpenMP implementation of SPICE3 circuit simulator. International journal of parallel programming. 2007.

01 Background

The sparse LU factorization is that factorize the square matrix A into the product of the sparse **lower triangular matrix L** and the **upper triangular matrix U**.

$$\begin{bmatrix} a_{11} & & & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ a_{31} & & a_{33} & \\ & a_{42} & & a_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ & 1 & 0 & 0 \\ l_{31} & & 1 & 0 \\ & l_{42} & l_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & & & u_{14} \\ 0 & u_{22} & u_{23} & u_{44} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

- The matrix is **sparse** (there are a large number of zero elements).
- **Non-zero elements will be added** during the solution process.

Sparse LU factorization process

preprocessing — Reduce fill-ins.

↓

Symbolic — Identify the locations of these fill-in elements.

↓

Numeric — Compute the final numerical results

↙ ↘

Left-looking | Right-looking

01 Background

The sparse LU factorization is that factorize the square matrix A into the product of the sparse **lower triangular matrix L** and the **upper triangular matrix U**.

$$\begin{bmatrix} a_{11} & & & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ a_{31} & & a_{33} & \\ & a_{42} & & \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ & 1 & 0 & 0 \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} u_{11} & & & u_{14} \\ 0 & & & u_{44} \\ & & & \\ & & & \end{bmatrix}$$

- The matrix i... zero element...
- **Non-zero elemen...** process.

The numeric factorization contains **a large number of floating-point calculations**, which is generally the most time-consuming step and is **the object of optimization** in our work.

Sparse LU factorization process

preprocessing — Reduce fill-ins.

Symbolic — Identify the locations of these fill-in elements.

Numeric — Compute the final numerical results

Left-looking

Right-looking

01 Background

MUMPS is a solver for solving large sparse linear systems using the **multifrontal method**. MUMPS exploits dense submatrices in matrices and invokes a Level-3 BLAS to achieve LU factorization acceleration[1].

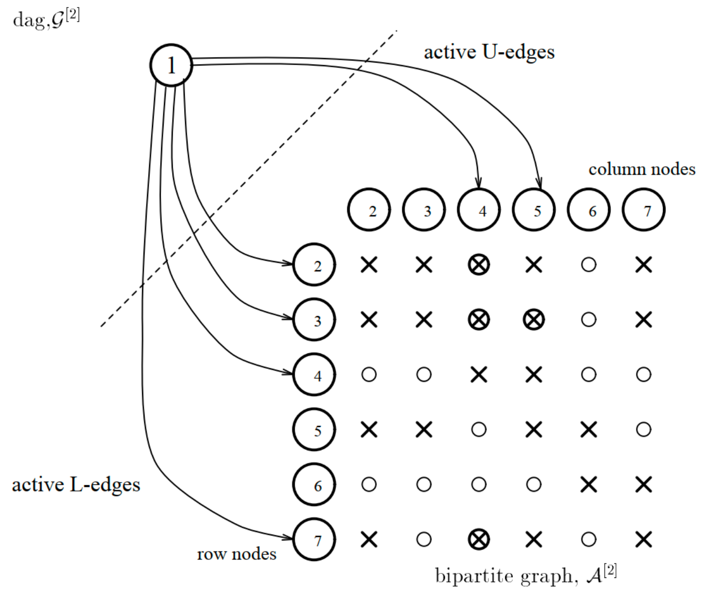


Figure 2: Assembly graph $\mathcal{D}^{[2]} = (\mathcal{A}^{[2]}, \mathcal{G}^{[2]}, \mathcal{F}^{[2]})$ for matrix A in Equation 6

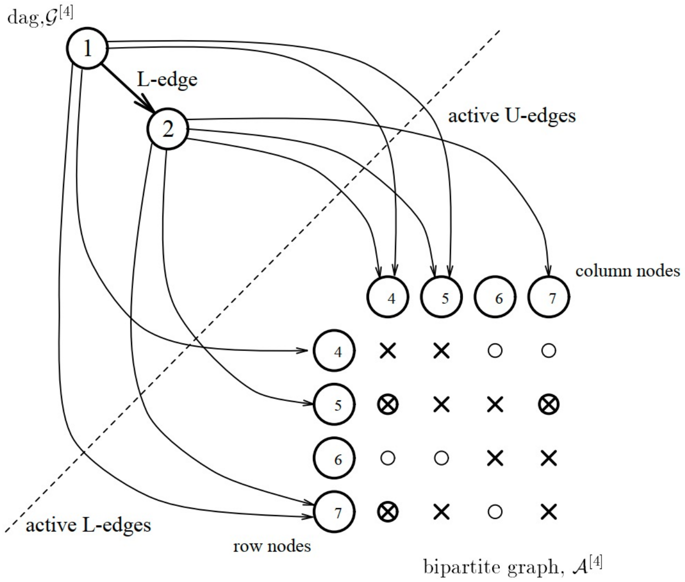


Figure 3: Assembly graph $\mathcal{D}^{[4]} = (\mathcal{A}^{[4]}, \mathcal{G}^{[4]}, \mathcal{F}^{[4]})$ for matrix A in Equation 6

[2]

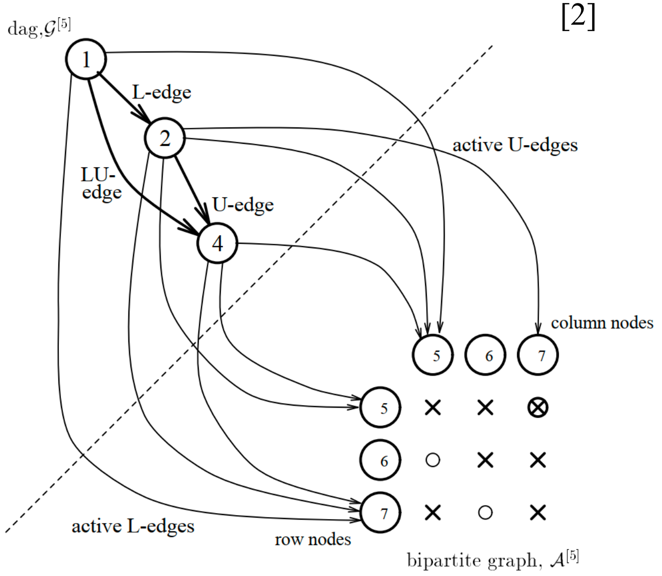
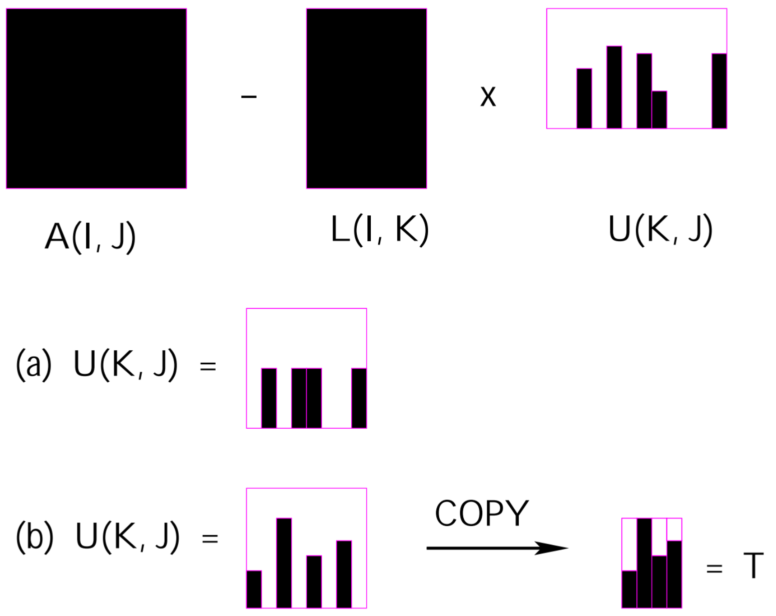


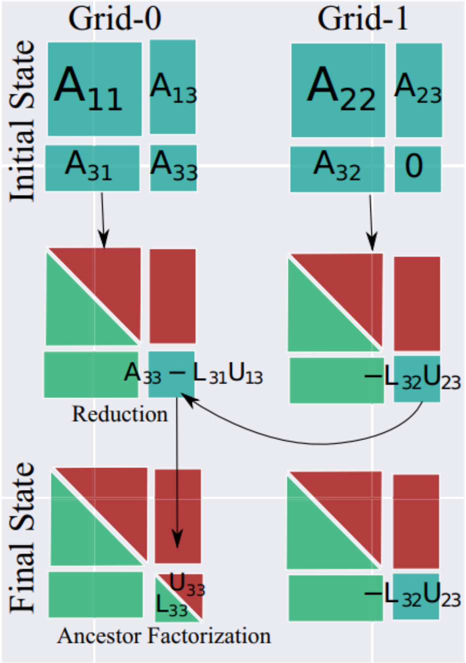
Figure 4: Assembly graph $\mathcal{D}^{[5]} = (\mathcal{A}^{[5]}, \mathcal{G}^{[5]}, \mathcal{F}^{[5]})$ for matrix A in Equation 6

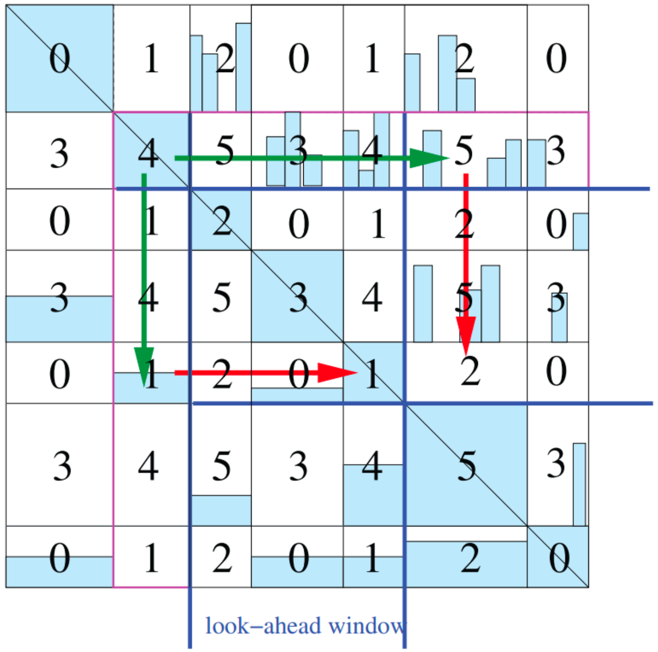
[1] Patrick Amestoy, Iain Duff, Jean-Yves L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. Computer methods in applied mechanics and engineering. 2000.
[2] Iain Duff and John Reid. The multifrontal solution of indefinite sparse symmetric linear. TOMS '83.

01 Background

SuperLU is a solver that introduces a supernode strategy into the LU factorization of unsymmetric matrices, further utilizes a Level-3 BLAS, and solves using a right-looking algorithm[1][2][3].



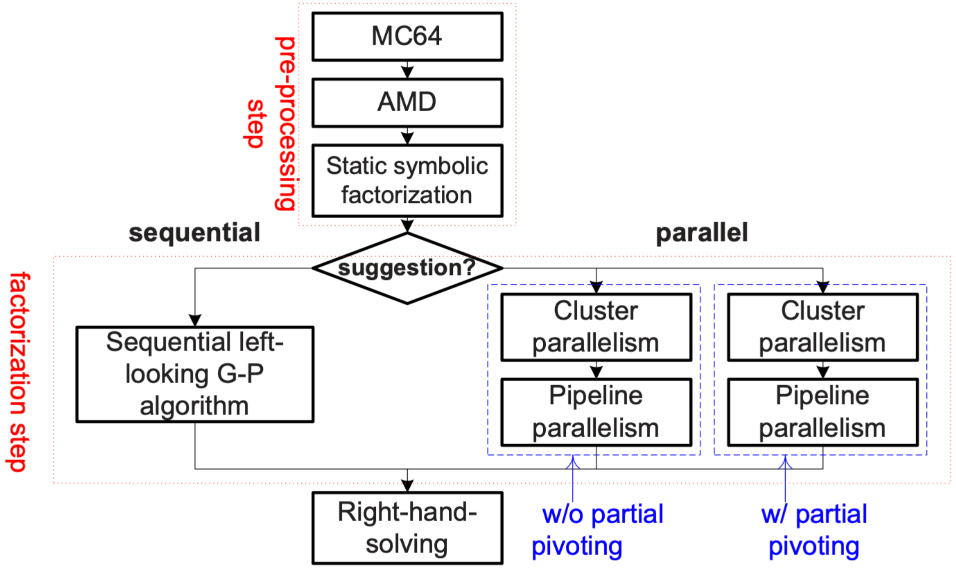




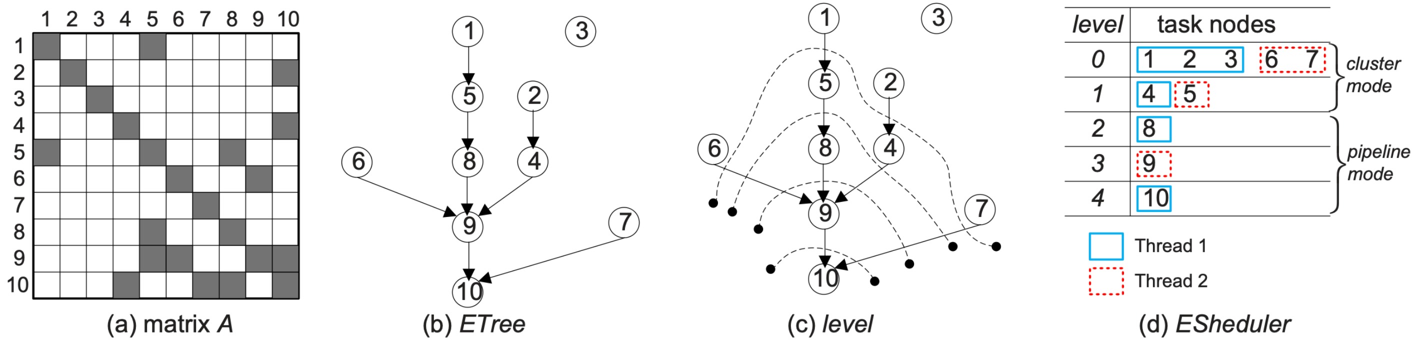
[1] Piyush Sao, Xiaoye Li, Richard Vuduc. A communication-avoiding 3D algorithm for sparse LU factorization on heterogeneous systems. Journal of parallel and distributed computing. 2019.
[2] Ichitaro Yamazaki, Xiaoye Li. New scheduling strategies and hybrid programming for a parallel right-looking sparse LU factorization algorithm on multicore cluster systems. ISPA '12.
[3] Xiaoye Li, James Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. TOMS '03.

01 Background

NICSLU is a solver that uses a division of **Level-sets,** incorporates **supernodes**, and uses parallel/serial algorithms to achieve acceleration in the numeric factorization phase based on matrix properties predictions[1][2].



The overall flow of the proposed adaptive solver

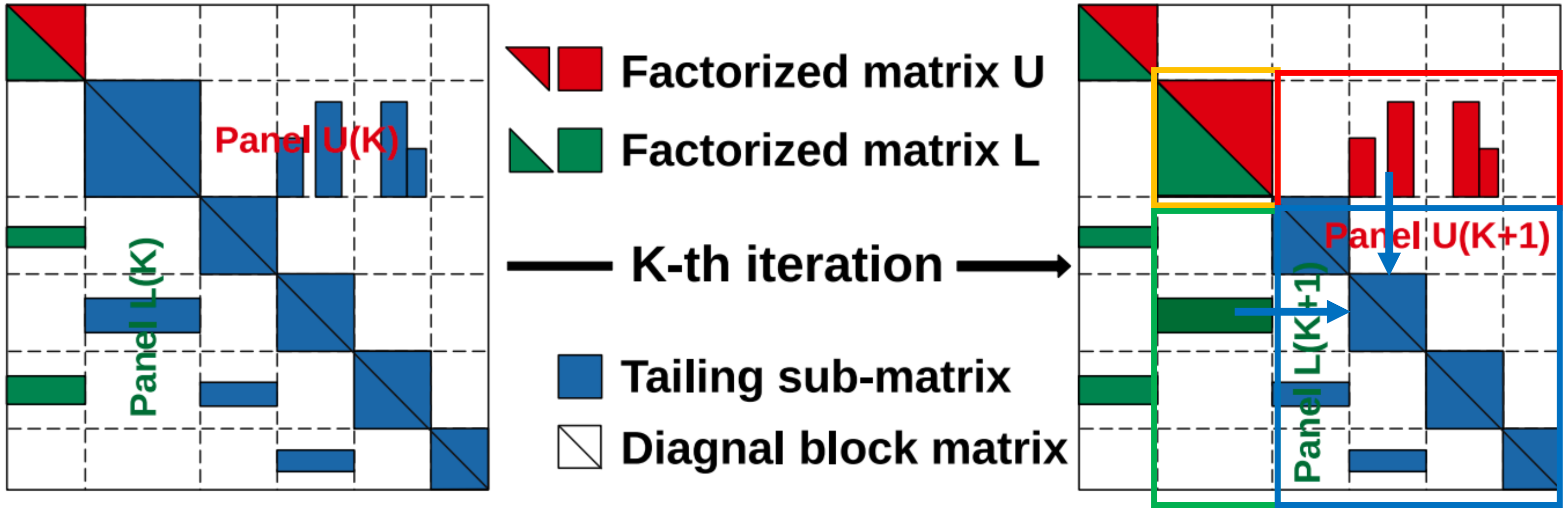


An example to illustrate ETree, level, and EScheduler.

[1] Xiaoming Chen, Yu Wang, Huazhong Yang. An adaptive LU factorization algorithm for parallel circuit simulation. ASP-DAC '12.
[2] Xiaoming Chen, Yu Wang and Huazhong Yang. NICSLU: an adaptive sparse matrix solver for parallel circuit simulation. IEEE transactions on computer-aided design of integrated circuits and systems. 2013.

02 Motivation

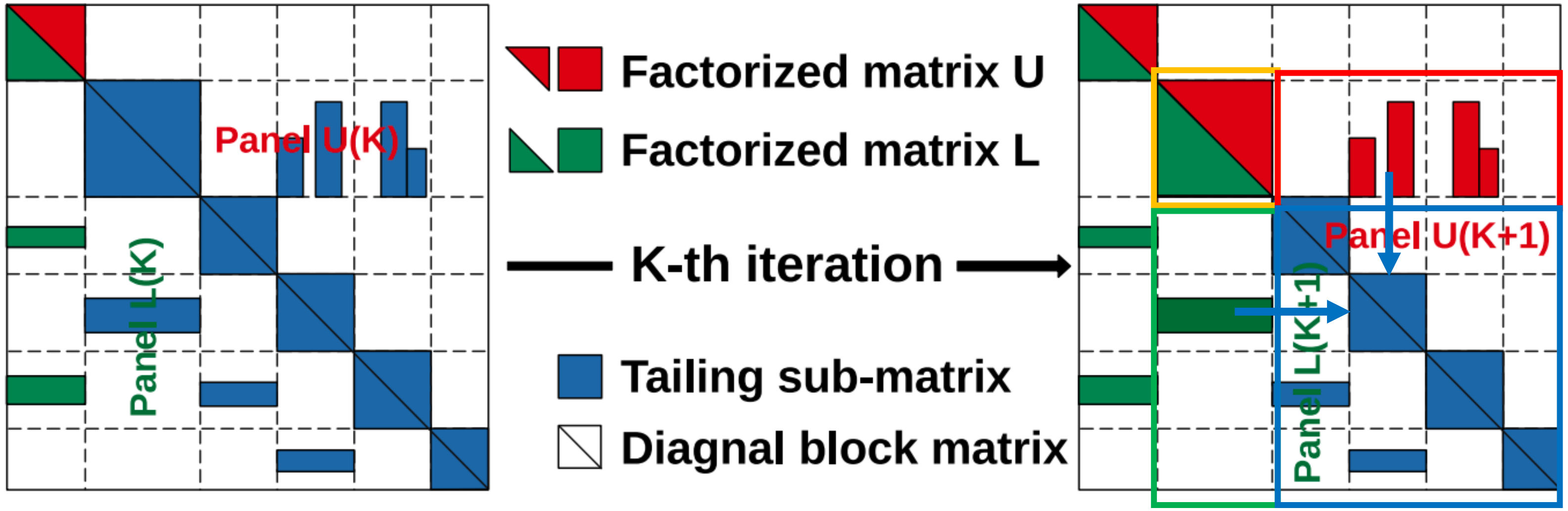
The numeric factorization in supernodal LU factorization follows four steps， where **K signifies the K-th iteration and N denotes the number of matrix blocks on the diagonal**.



- Factorize the diagonal block;
- Factorize the sub-matrices in L panel: L(K : N, K);
- Factorize the sub-matrices in U panel: U(K, K +1 : N);
- Perform the Schur-complement for all the tailing sub-matrices **by using A = A−L×U.**

02 Motivation

The numeric factorization in supernodal LU factorization follows four steps， where **K signifies the K-th iteration and N denotes the number of matrix blocks on the diagonal**.

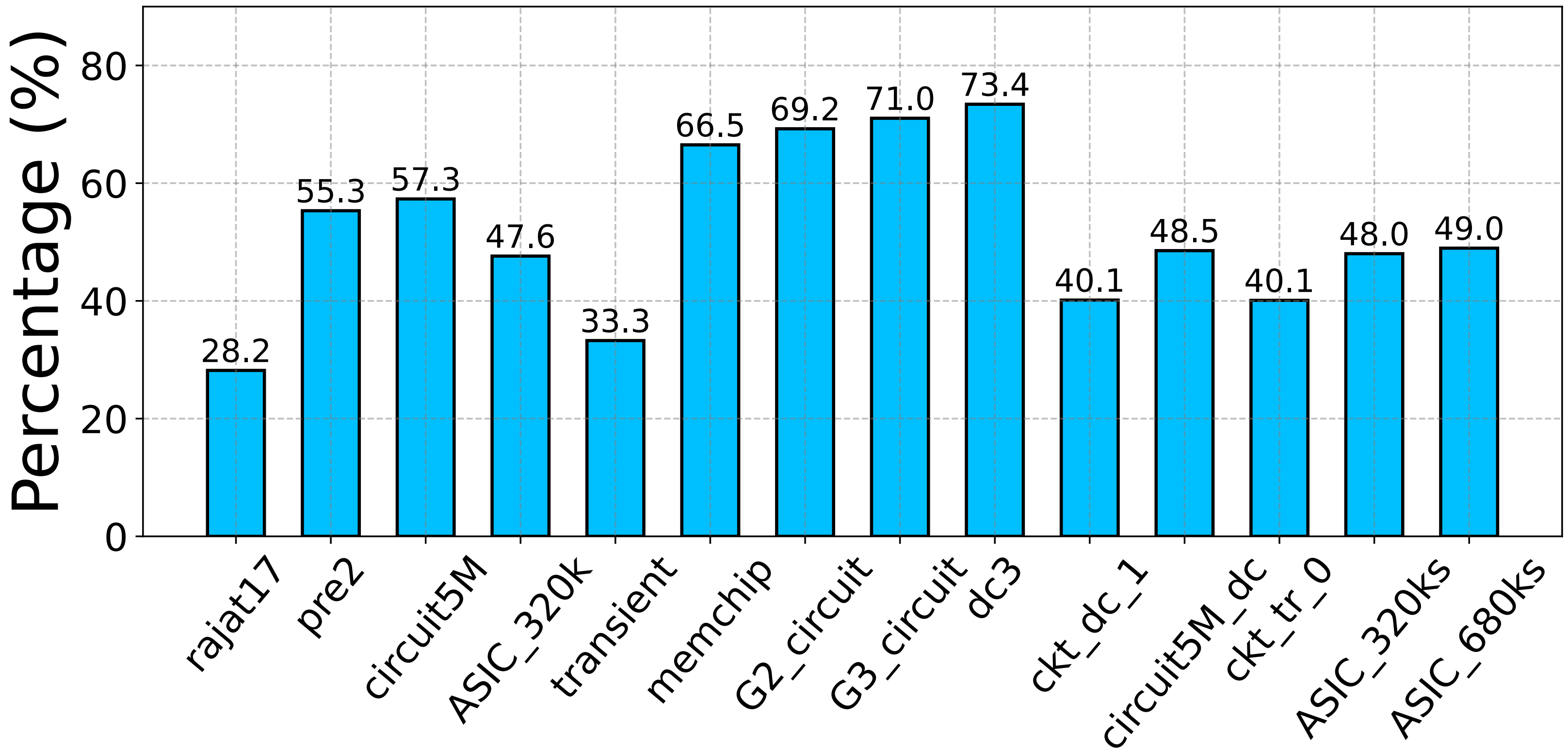


- Factorize the diagonal block;
- Factorize the sub-matrices in L panel: L(K : N, K);
- Factorize the sub-matrices in U panel: U(K, K +1 : N);
- Perform the Schur-complement for all the tailing sub-matrices **by using A = A−L×U.**

The Schur-complement phase contains **a large number of GEMM operations.**

02 Motivation

GEMM takes up **much of the time** for numeric factoriztaion.

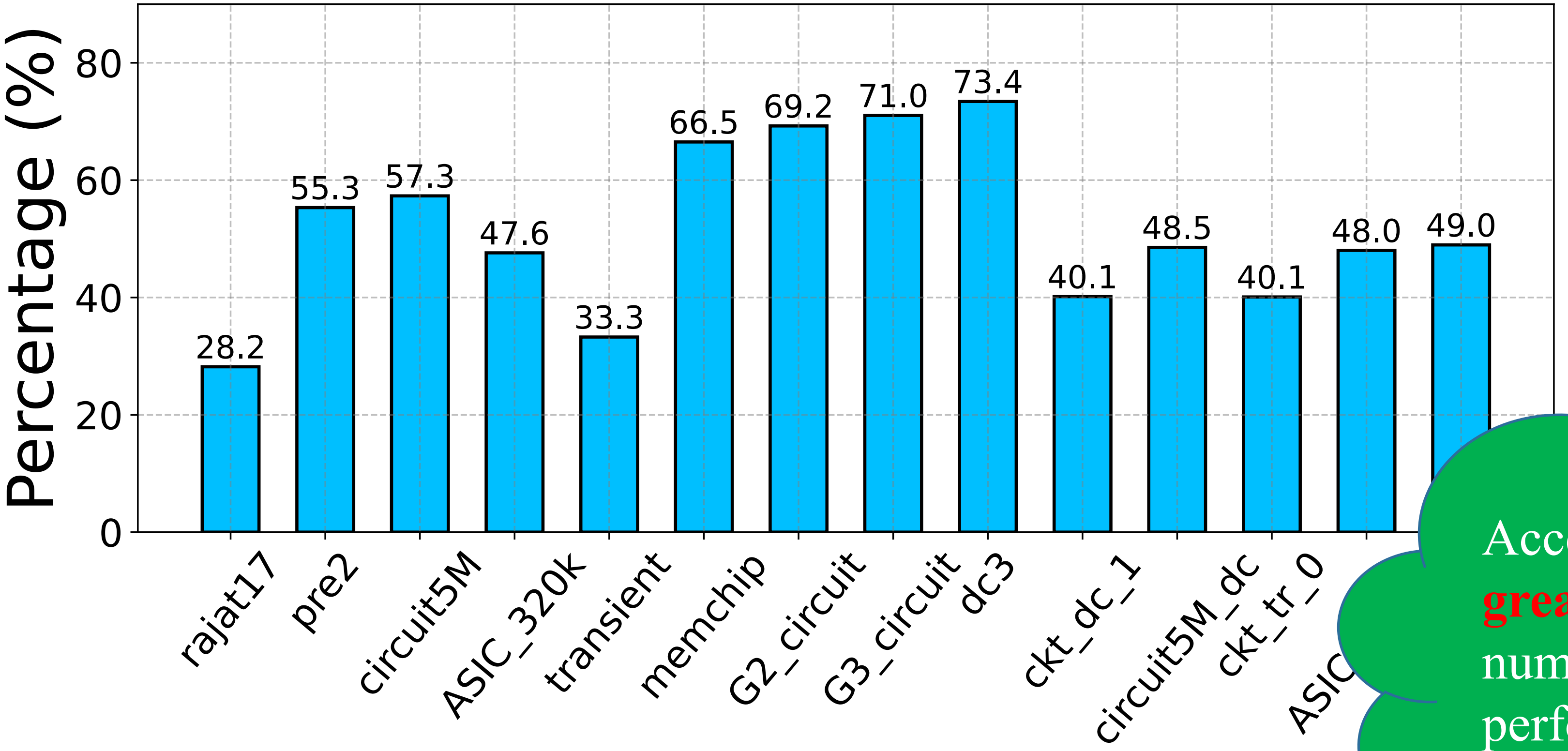


The time proportion of GEMM in numeric factorization.

We tested some circuit matrices. such as the G3_circuit matrix in the figure, which accounts for **as much as 73.4% of the time**, and most of the other matrices tend to account for **40%-60%** of the GEMM time.

02 Motivation

GEMM takes up **much of the time** for numeric factoriztaion.



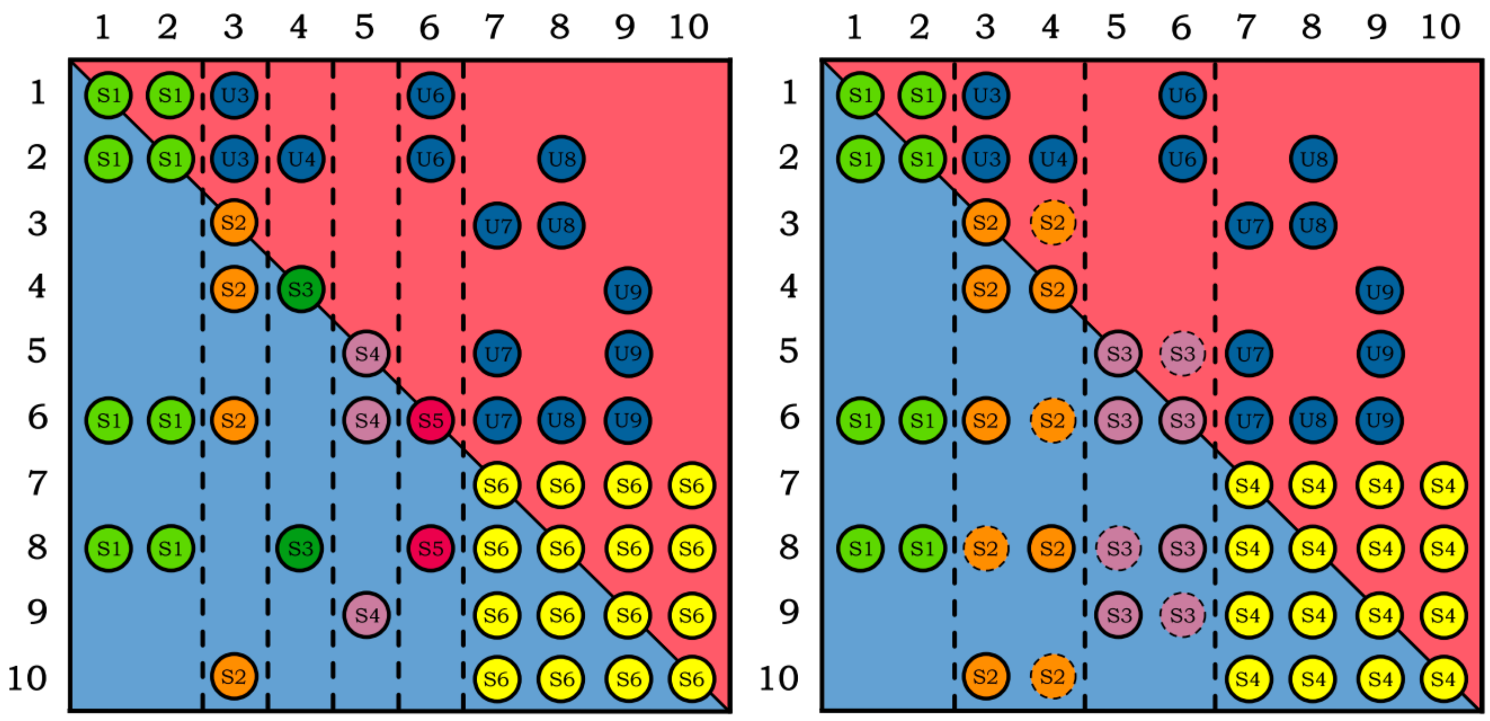
The time proportion of GEMM in numeric factorization.

We tested some circuit matrices. such as the G3_circuit matrix in the figure, which accounts for **as much as 73.4% of the time**, and most of the other matrices te **%**

Accelerating GEMM is **of great significance** for numeric factorization performance improvement.

02 Motivation

Supernodal LU factorization will divide the sparse matrix into **supernodes.**

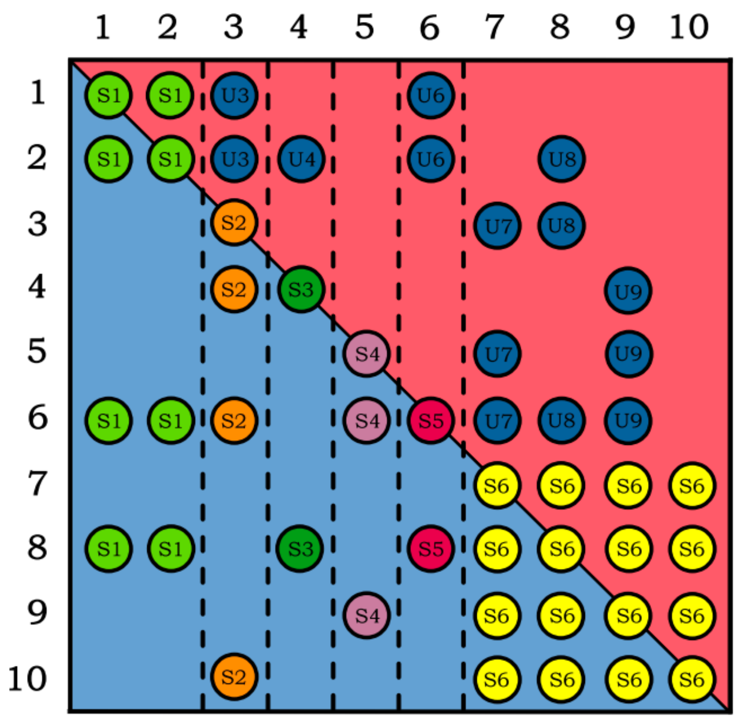


(a) Original supernodes

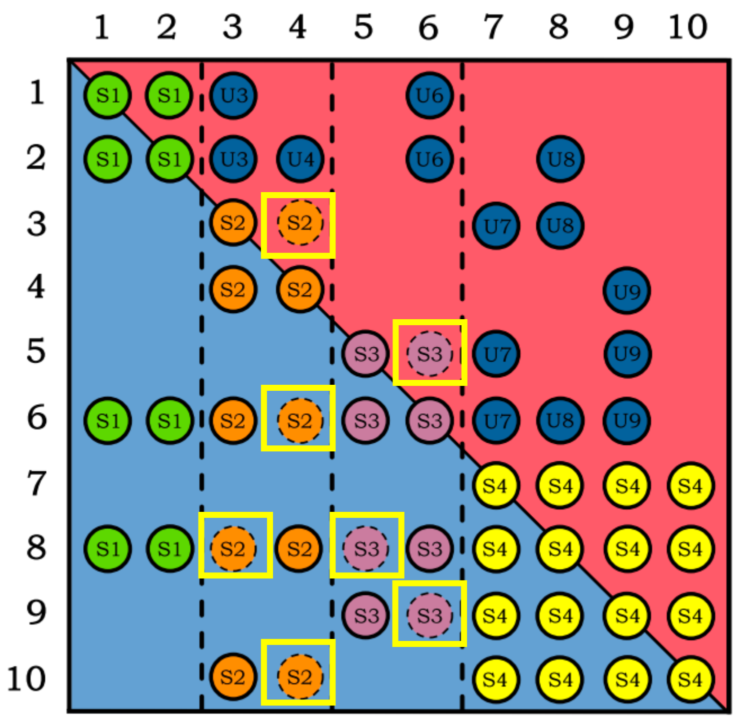
(b) Supernodes after merging

02 Motivation

Supernodal LU factorization will divide the sparse matrix into **supernodes.**



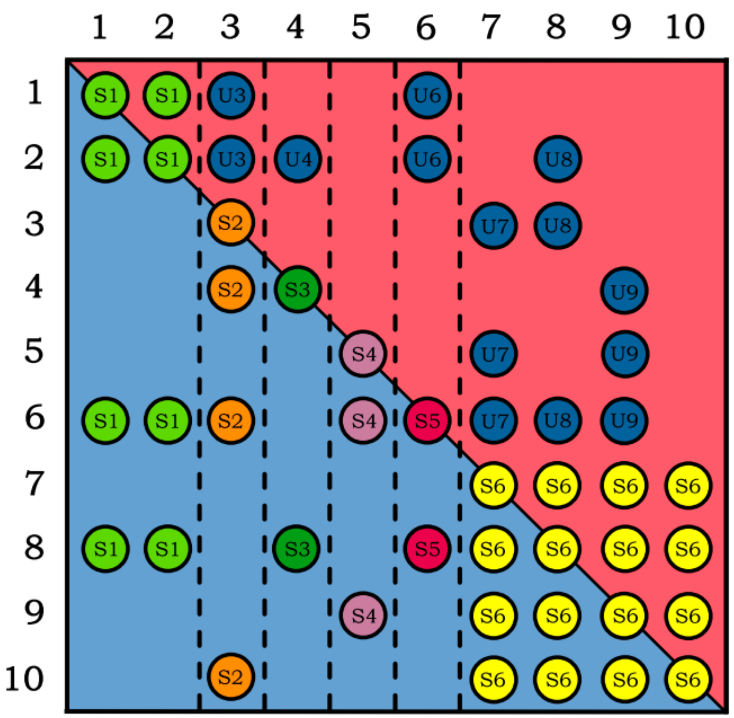
(a) Original supernodes



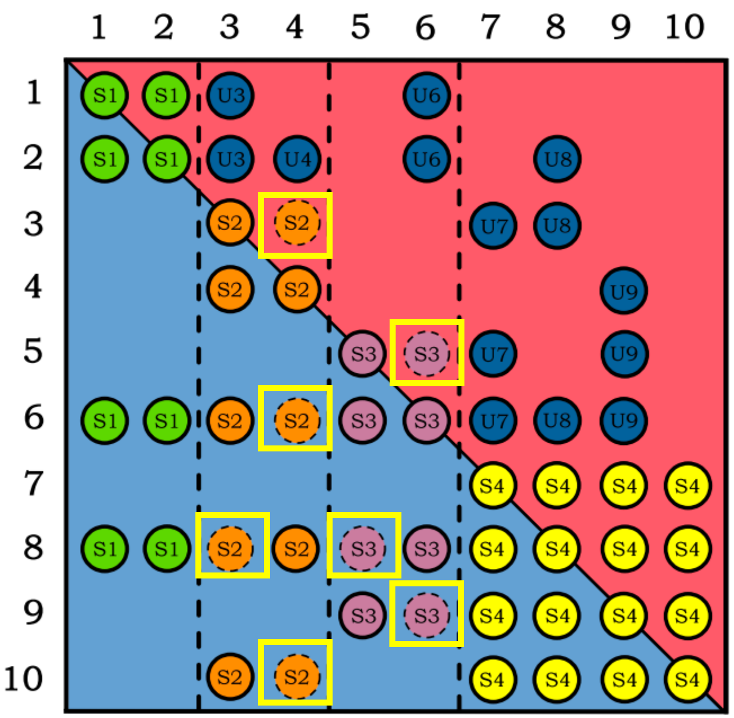
(b) Supernodes after merging

02 Motivation

Supernodal LU factorization will divide the sparse matrix into **supernodes.**



(a) Original supernodes

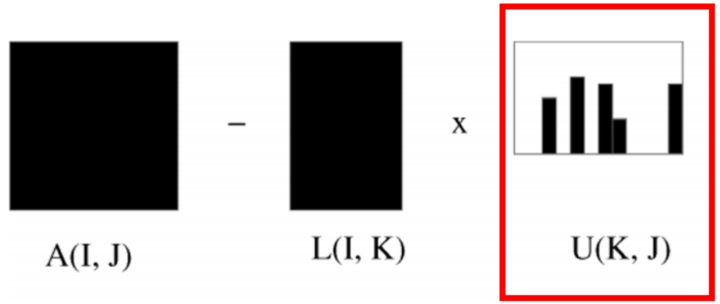


(b) Supernodes after merging

Schur-complement

$$A(I,J) \leftarrow A(I,J) - \boxed{L(I,K) \times U(K,J);}$$

It may have **some sparsity**, and GEMM operations will **add unnecessary operations.**



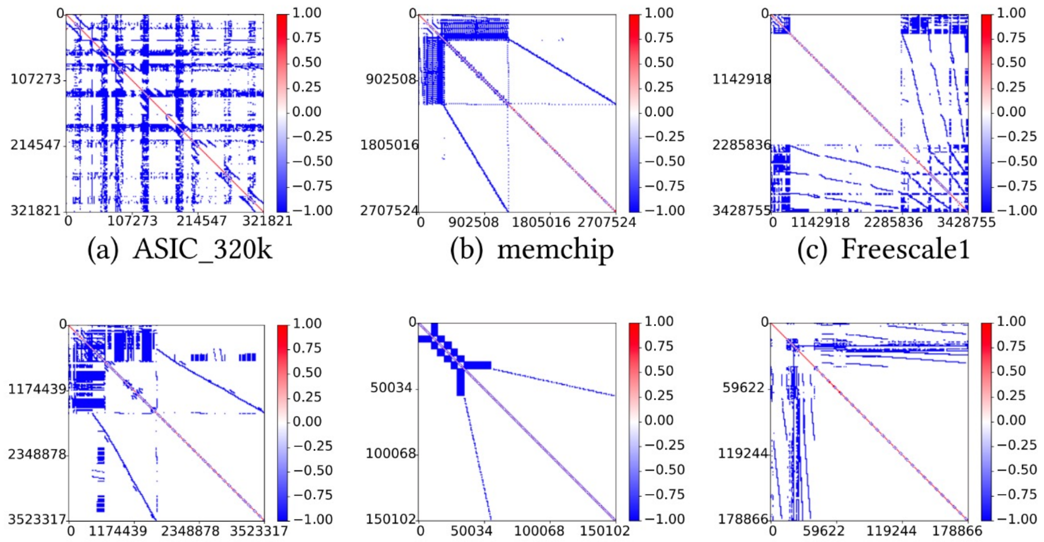
02 Motivation

In circuit simulation, the matrix usually has the following properties:

(1) Circuit matrices are usually **unsymmetric.**

(2) Circuit matrices may have some **dense rows and columns** (e.g., power supplies are usually connected to a larger number of devices).

(3) Circuit matrices are often **very sparse** and the distribution of non-zero elements is **extremely irregular.**



(a) ASIC_320k (b) memchip (c) Freescale1

(d) circuit5M_dc (e) G2_circuit (f) transient

Circuit sparse matrix analysis

Circuit Matrix	N	Entries per row				Symmetry
		max	min	average	variation	
ASIC_320k	321,821	203,800	1	8.2	502.95	100.00%
memchip	2,707,524	27	2	5.5	2.06	0.32%
Freescale1	3,428,755	27	1	5.5	2.07	7.67%
circuit5M	3,523,317	27	1	10.7	1356.61	55.99%
G2_circuit	150,102	4	1	2.9	0.52	0.0005%
transient	178,866	60423	1	5.4	147.2	68.99%

02 Motivation

In circuit simulation, the matrix usually has the following properties:

(1) Circuit

(2) Ci

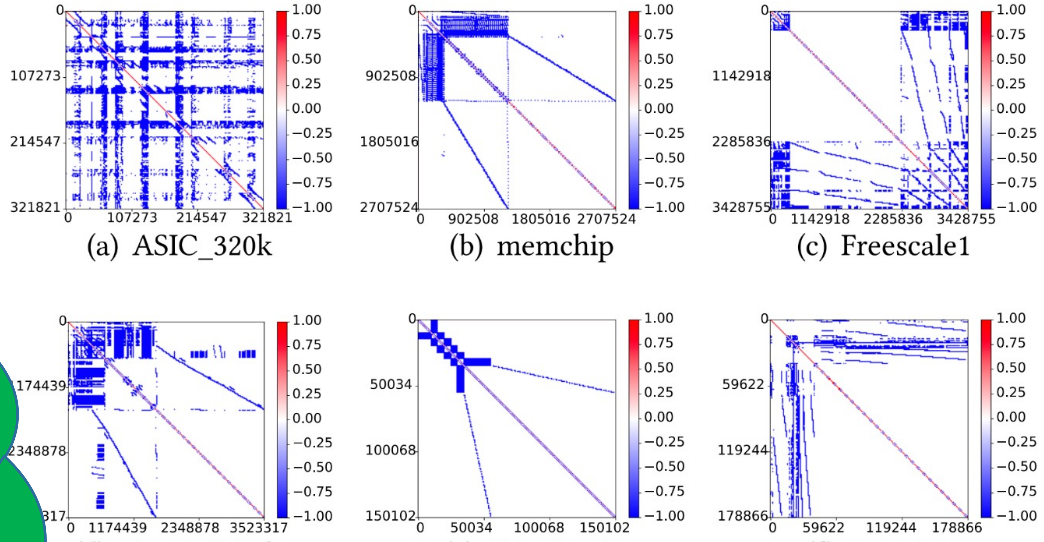
co

(3

distributi

irregular.

When solving circuit matrices by supernodal LU factorization, it is often **difficult to form supernodes or the formed supernodes are sparse.** It will **bring additional computational effort and time** in numeric factorization.



(a) ASIC_320k (b) memchip (c) Freescale1

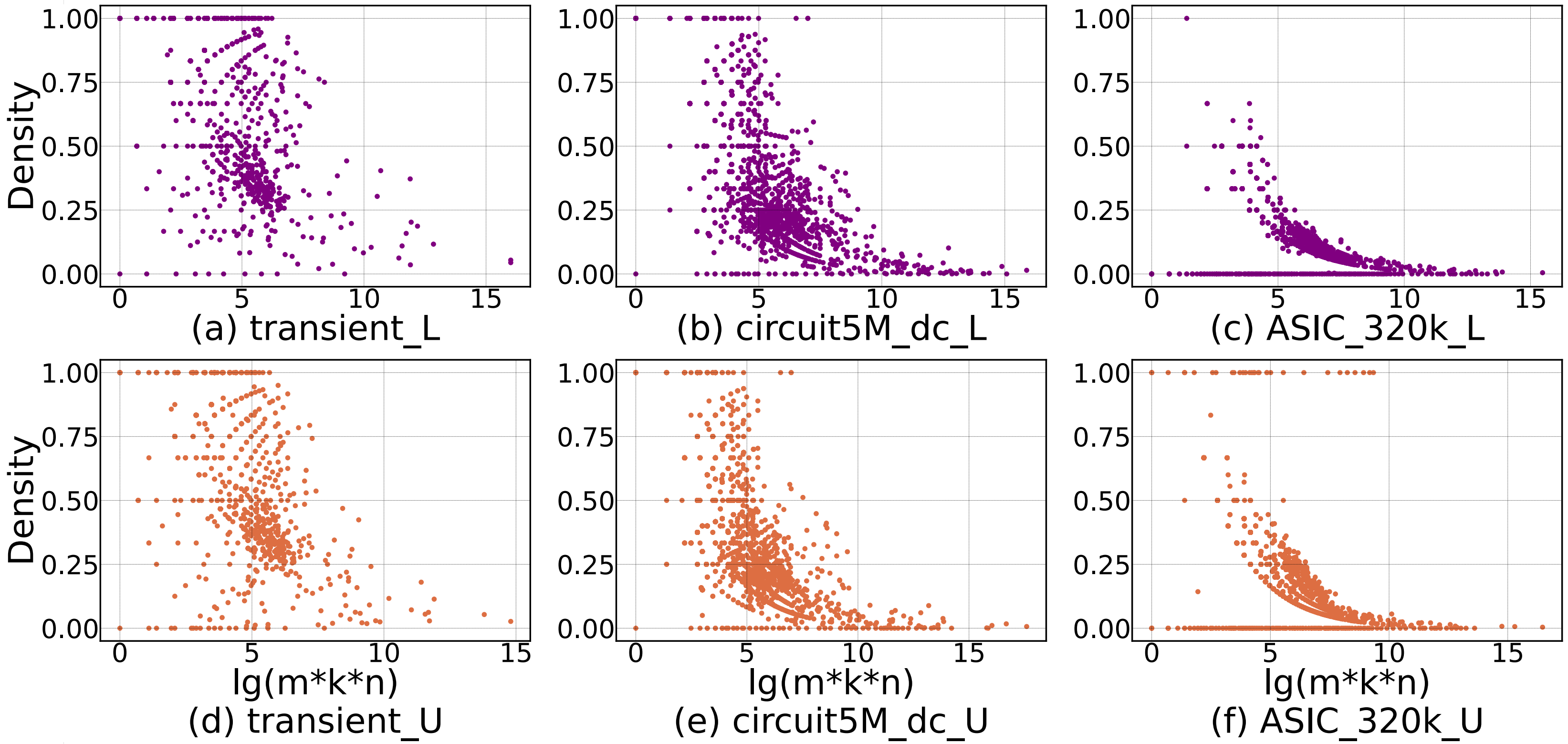
(d) circuit5M_dc (e) G2_circuit (f) transient

Circuit sparse matrix analysis

Circuit Matrix	N	Entries per row: max	min	average	variation	Symmetry
ASIC_320k	321,821	203,800	1	8.2	502.95	100.00%
memchip	2,707,524	27	2	5.5	2.06	0.32%
Freescale1	3,428,755	27	1	5.5	2.07	7.67%
circuit5M	3,523,317	27	1	10.7	1356.61	55.99%
G2_circuit	150,102	4	1	2.9	0.52	0.0005%
transient	178,866	60423	1	5.4	147.2	68.99%

02 Motivation

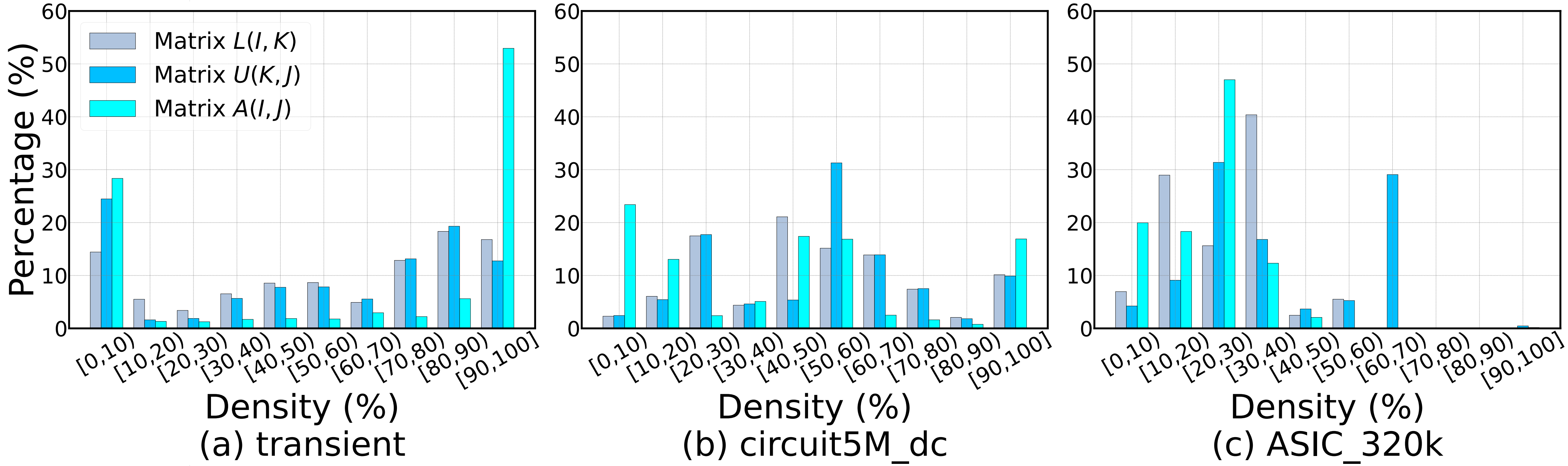
We have selected three representative circuit matrices and can see that **the matrix factors (L and U blocks) involved in GEMM are generally sparse.**



Density distribution of matrix factors (L and U blocks) participating in GEMM.

02 Motivation

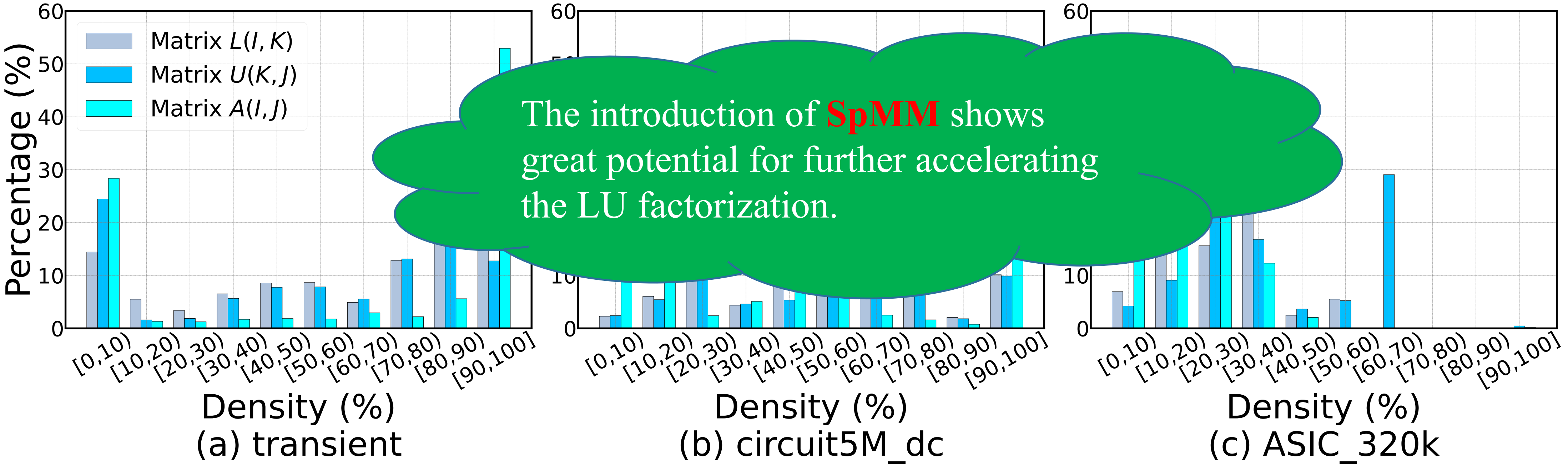
We have selected three representative circuit matrices and can see that **the matrix factors (L and U blocks) involved in GEMM are generally sparse.**



Density distribution percentage of matrix factors (L and U blocks) participating in GEMM.

02 Motivation

We have selected three representative circuit matrices and can see that **the matrix factors (L and U blocks) involved in GEMM are generally sparse.**



The introduction of **SpMM** shows great potential for further accelerating the LU factorization.

Density distribution percentage of matrix factors (L and U blocks) participating in GEMM.

03 Density-aware matrix multiplication

We analyze three cases for computing matrix multiplication time: using **GEMM**, using **SpMM**, and using **the oracle combination of GEMM and SpMM** in the supernodal LU factorization.

Circuit matrix	nnz (A)	GEMM (s)	SpMM (s)	Oracle (s)	Speedup1	Speedup2
ASIC_320k	1,931,828	3.5809	0.4543	0.3653	9.80x	1.24x
Freescale1	17,052,626	8.8363	35.9429	8.5388	1.03x	4.21x
ckt11752_dc_1	333,029	0.0279	0.0433	0.0225	1.24x	1.92x
pre2	5,834,044	57.2954	10.9046	7.2531	7.90x	1.50x
meg4	58,142	0.0037	0.0027	0.0022	1.68x	1.23x
G2_circuit	726,674	7.1145	26.0853	6.1415	1.16x	**4.25x**
Freescale2	14,313,235	2.3253	3.8100	1.9497	1.19x	1.95x
FullChip	26,621,983	510.416	480.202	344.1850	1.48x	1.40x
ASIC_320ks	1,316,085	2.9155	0.322	0.2846	**10.24x**	1.13x
ASIC_680ks	1,693,767	2.6196	1.3393	0.9320	2.81x	1.44x
circuit5M_dc	14,865,409	7.5022	1.2420	1.0230	6.04x	1.21x
transient	961,368	0.5721	0.2710	0.2100	2.69x	1.29x

The "Speedup1" and "Speedup2" show that "Oracle" has a performance improvement potential of **1.03x-10.24x** and **1.13x-4.25x**, respectively, compared to **GEMM** and **SpMM**.

- **SpMM**

Based on the column principal order structure in supernodal LU factorization, we choose dense **matrix*CSC** as SpMM method.

- **Oracle**

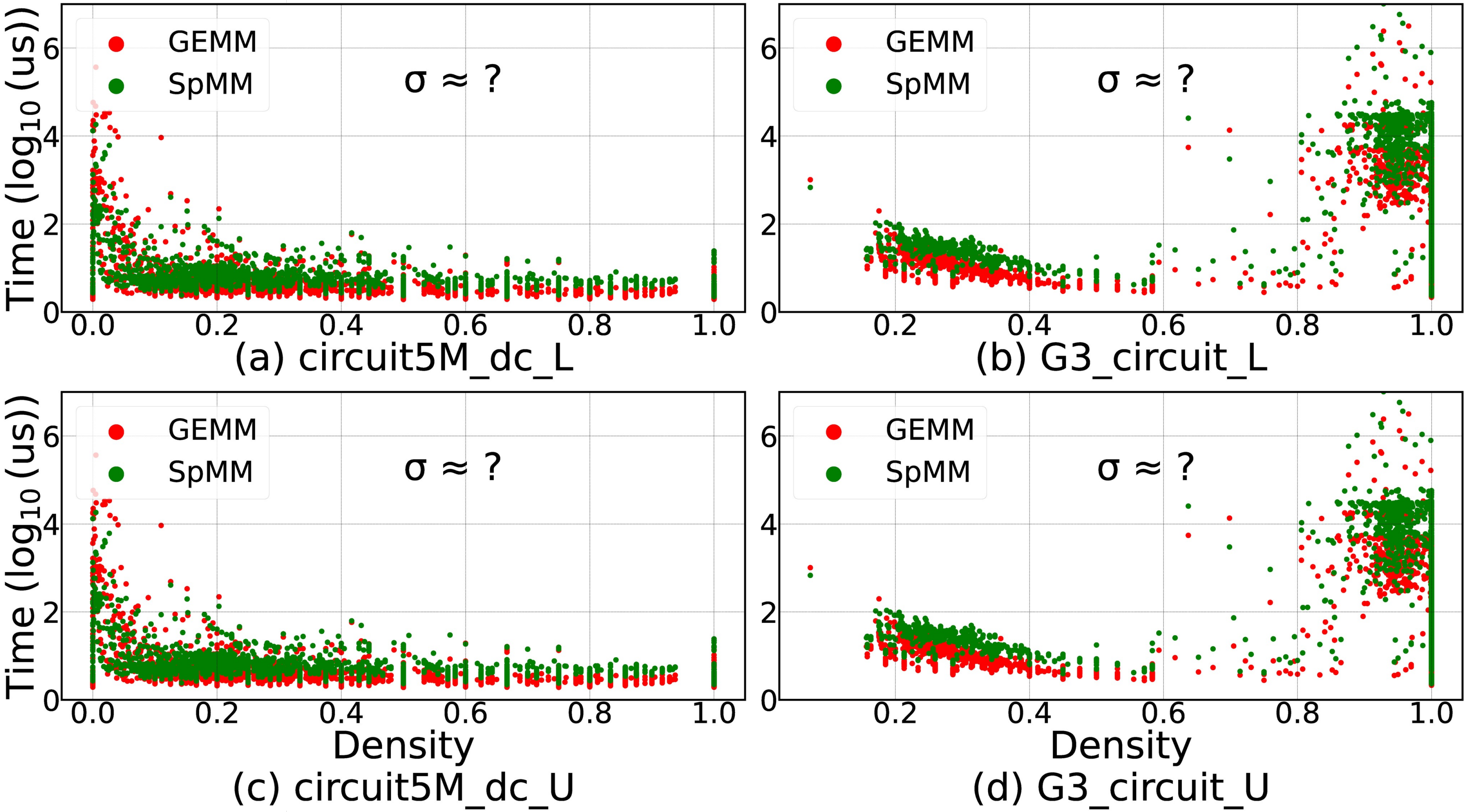
Oracle is selecting the optimal case of SpMM and GEMM at each matrix multiplication.

Which method is **best** ?
GEMM ? SpMM ?

Can a **single threshold** (density, matrix size, etc.) determine the optimal method?

03 Density-aware matrix multiplication

Is it possible to select GEMM or SpMM based on the **matrix density**?

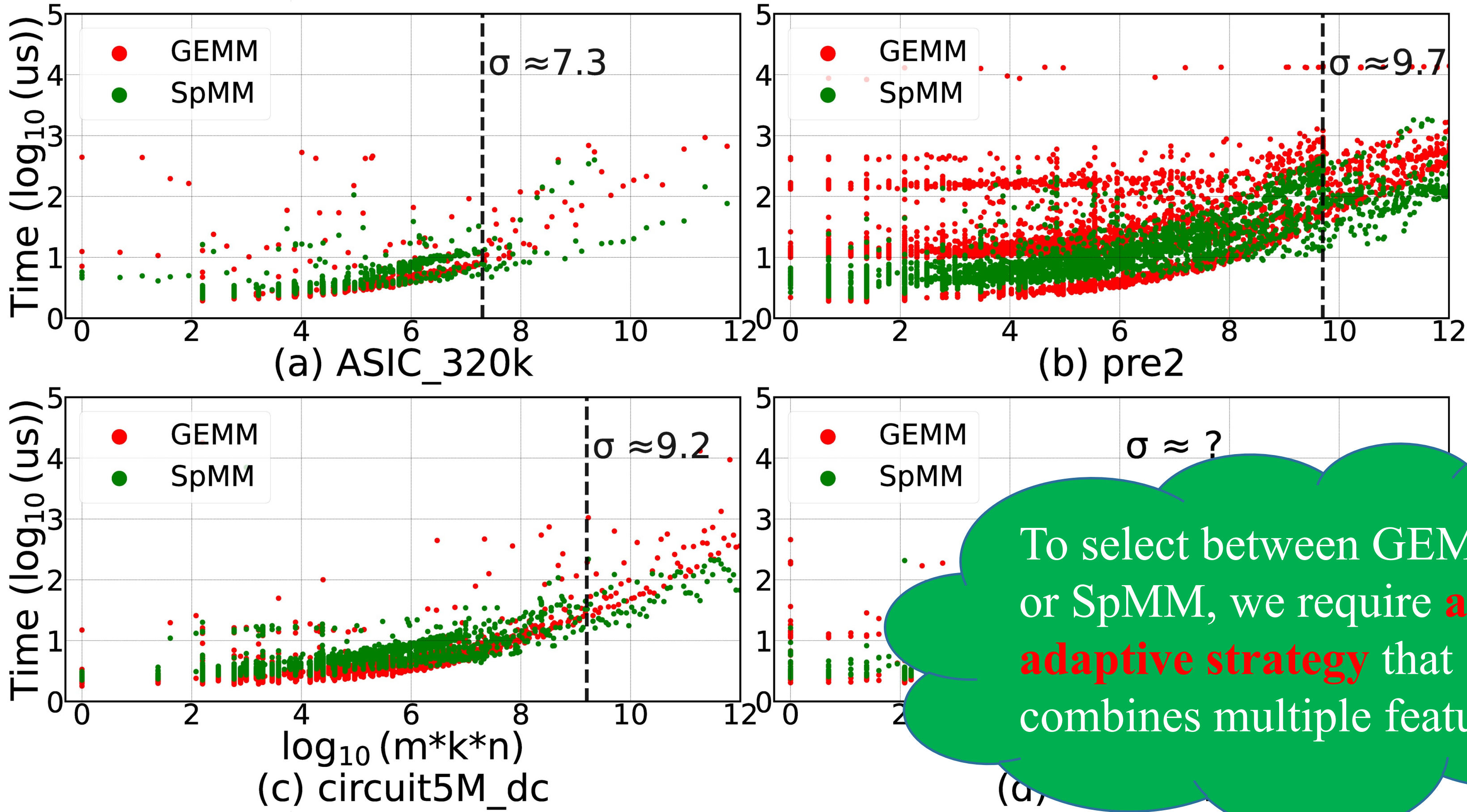


- x：Density of L and U block
- y： Computation time （ log_{10} ） (green：SpMM； red：GEMM)

It is difficult to define the selection of SpMM and GEMM by the **density alone**.

03 Density-aware matrix multiplication

Is it possible to select GEMM or SpMM based on the **matrix size**?



- x：Size of L and U block
- y： Computation time
  （ log_{10} ）
  (green：SpMM；
  red：GEMM)

There exists a **definite** ...**shold *σ*** on some matrices, ...e matrices do not have ... threshold ...s transient).

To select between GEMM or SpMM, we require **an adaptive strategy** that combines multiple features.

04 Machine learning driven adaptive acceleration

We need a suitable method to select the better matrix multiplication algorithm **for complex and variable matrices.**

~~Single threshold determination?~~

We require combining multiple matrix features to select the optimal algorithm.

04 Machine learning driven adaptive acceleration

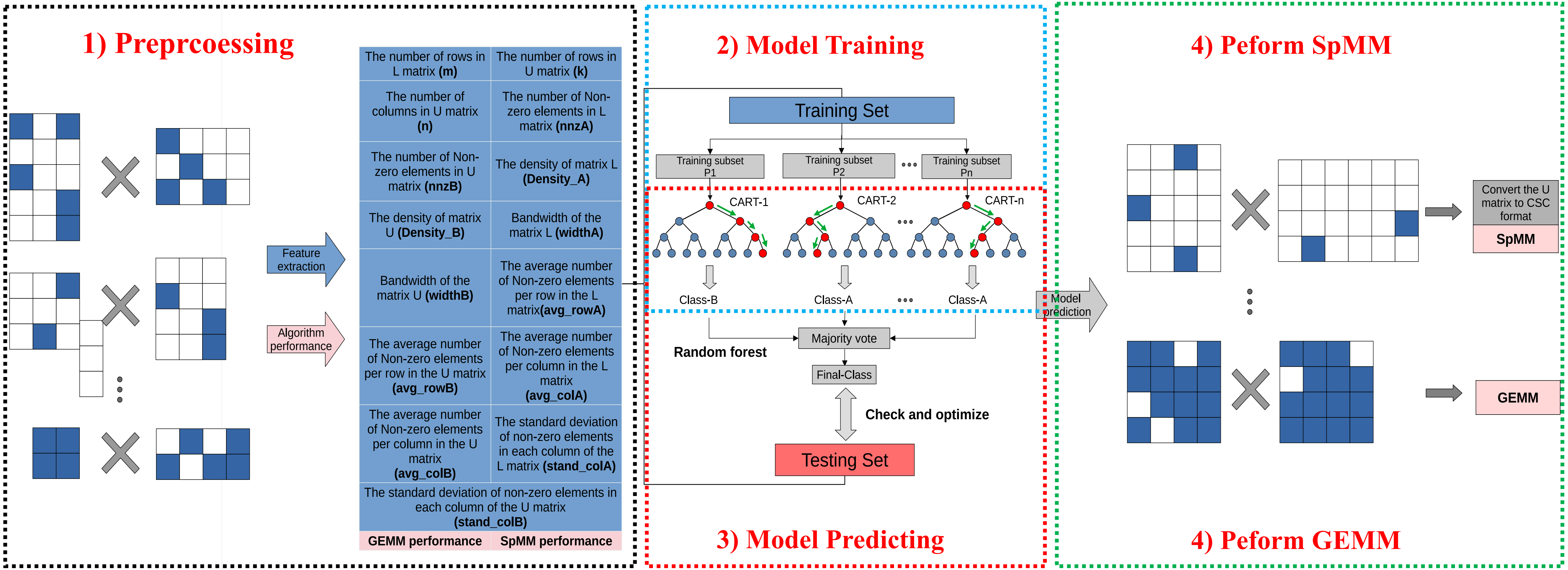
We need a suitable method to select the better matrix multiplication algorithm **for complex and variable matrices.**

Machine learning classification algorithm?

Combining machine learning algorithm with matrix features to construct a dataset and train the model to select the better algorithm. The above problems can be avoided.

04 Machine learning driven adaptive acceleration

We select **the random forest algorithm** as machine learning classification algorithm.

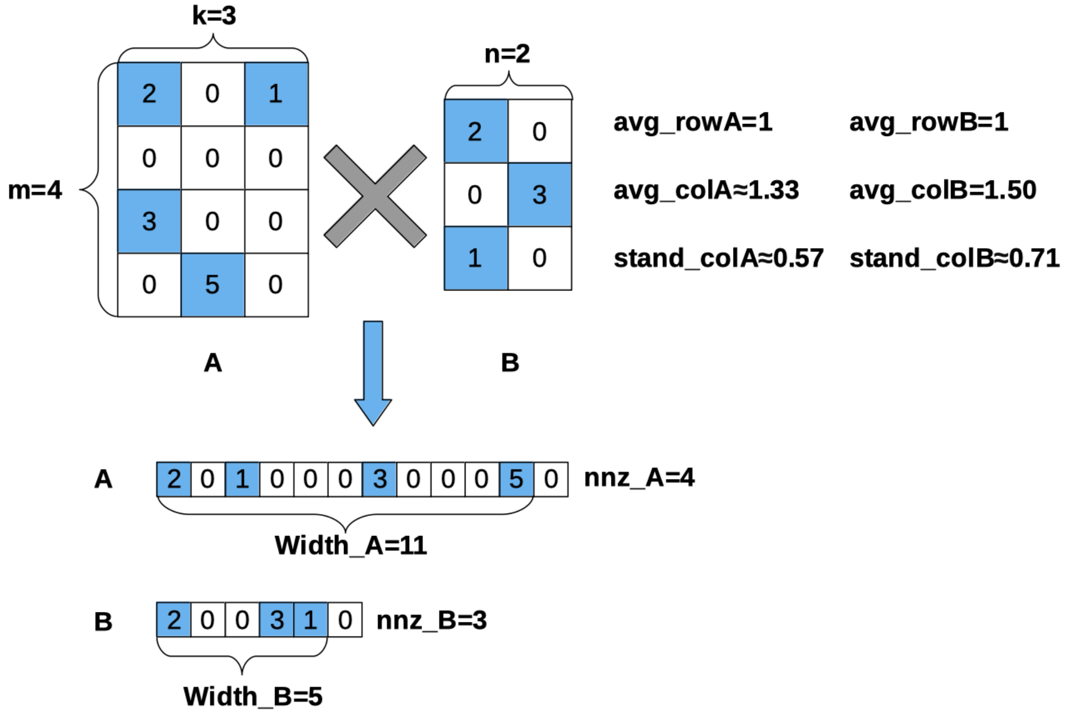


The random forest algorithm has the advantages of being able to **handle higher dimensional data, high generalisation ability** of the model**, can balance the errors,** and the training **can be easy to parallelize**, etc., so it's a better choice.

04 Machine learning driven adaptive acceleration

- **Dataset Source:** The large-scale circuit matrices in SuiteSparse and exported by SuperLU_DIST 8.0.0.
- **Samples in the dataset:** Each sample contains **15 matrix features** (F1-F15) and a label P, where P=1 indicates that using GEMM is better than that of SpMM, and P=0 is vice versa.
- **Data preprocessing:**
  - **Z-score normalization**
  - **Sample equalization**
- **Number of samples in the dataset ：**

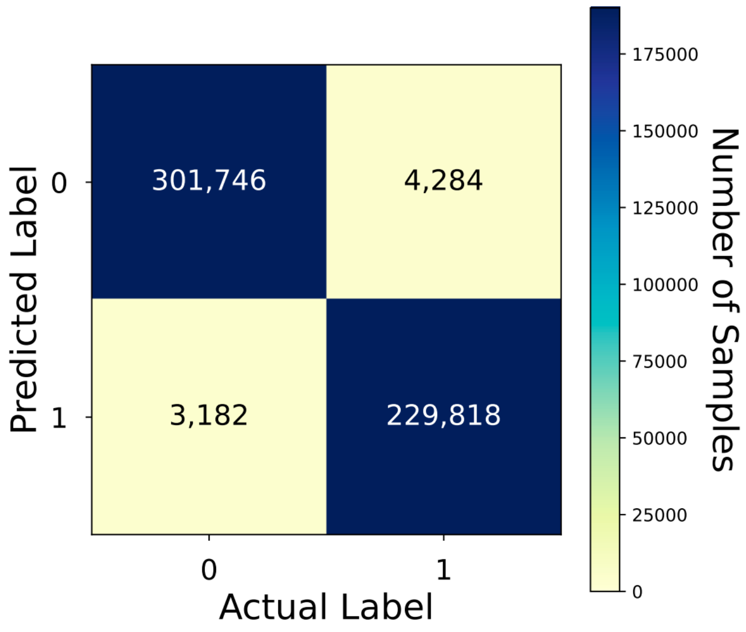
Dataset	GEMM	SpMM
Total sample set(D1)	456,000	383,000
Training set (D2)	150,000	150,000
Testing set(D3)	306000	233000



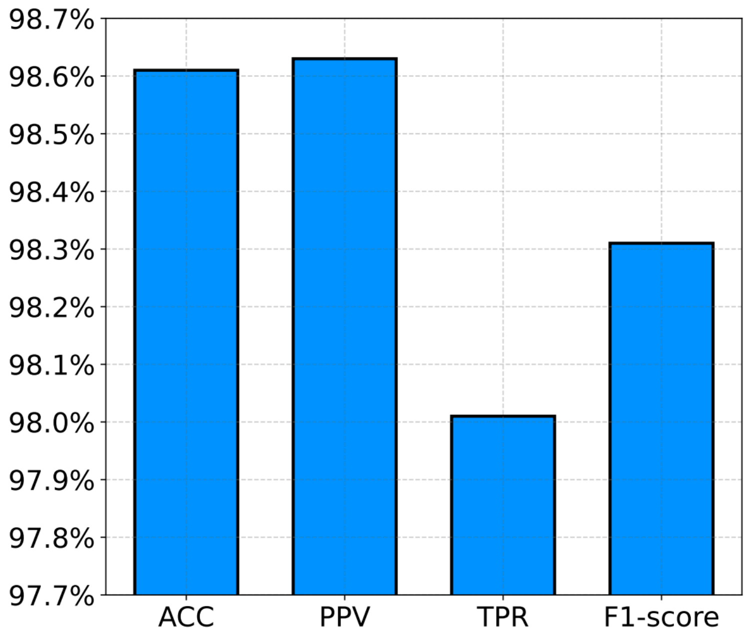
An example of 15 matirx features (F1-F15).

05 Experimental results

Experimental platform：2 * Intel Xeon Silver 4210 CPU @ 2.20GHz, 512GB DDR4



(a) Confusion matrix



(b) Performance of the binary classifiers

We tested the model based on the training set against the test set in the sample set, **and the confusion matrix and correlation performance** were excellent. ACC, PPV, TPR and F1-score are all exceed 90%.

05 Experimental results

Performance evaluation on circuit matrices.

circuit matrix	nnz	Matrix multiplication time (s)						Numeric factorization time (s)		
		GEMM	SpMM	Oracle	AI	Speedup (AI vs GEMM)	Speedup (AI vs SpMM)	SuperLU	Our work	Speedup
ASIC_320k	1,931,828	3.5809	0.4543	0.3653	0.5045	7.10x	0.90x	7.5201	4.5770	1.64x
ASIC_320ks	1,316,085	2.9155	0.3223	0.2846	0.3117	**9.35x**	1.03x	6.0602	3.5947	1.69x
ASIC_680ks	1,693,767	2.6196	1.3393	0.9323	1.0085	2.60x	1.33x	5.3502	3.8139	1.40x
circuit5M_dc	14,865,409	7.5022	1.2418	1.0231	2.0084	6.04x	0.62x	19.2301	14.5605	1.32x
pre2	5,834,044	57.2954	10.9046	7.2531	8.1021	7.07x	**1.35x**	103.5721	58.7290	**1.76x**
transient	961,368	0.5721	0.2709	0.2121	0.2532	2.26x	1.07x	1.7202	1.4517	1.18x
Average	-	-	-	-	-	5.35x	1.05x	-	-	1.50x

Performance evaluation on non-circuit matrices.

non-circuit matrix	nnz	Matrix multiplication time (s)						Numeric factorization time (s)		
		GEMM	SpMM	Oracle	AI	Speedup (AI vs GEMM)	Speedup (AI vs SpMM)	SuperLU	Our work	Speedup
sinc12	283,992	9.7211	2.7060	2.0120	2.2491	**4.32x**	1.20x	14.7541	7.4277	2.04x
psmigr_3	543,160	16.3840	7.8951	5.8241	6.2611	2.62x	1.26x	28.4921	19.8387	1.54x
psmigr_2	540,022	30.8151	12.0731	8.6251	9.1721	3.36x	1.32x	45.9061	24.5057	**2.08x**
epb2	175,027	0.1291	0.10021	0.05631	0.07621	1.69x	1.31x	0.4351	0.3937	1.08x
benzene	242,669	8.1778	9.6211	6.2315	7.2724	1.13x	**1.32x**	17.6351	16.9551	1.04x
Average	-	-	-	-	-	2.62x	1.28x	-	-	1.55x

It can be seen that there are **different degrees of acceleration** on the non-regular matrices.

Among the six circuit matrices.

- In the matrix multiplication phase, our work has a maximum of **9.35x** and an average of **5.35x** acceleration.
- In the numeric factorization phase, our work has a maximum of **1.76x** and an average of **1.50x** acceleration.

Among the five non-circuit matrices.

- In the matrix multiplication phase, our work has a maximum of **4.32x** and an average of **2.63x** acceleration.
- In the numeric factorization phase, our work has a maximum of **2.08x** and an average of **1.55x** acceleration.

06 Conclusions

- We propose a **density-aware sparse LU factorization acceleration method**, leveraging sparse matrix multiplication in the large amount of Schur-complement updates.
- Sampling method based on the sample proportion of the unit matrix in total dataset **improves the inference accuracy and model generality**.
- Our method shows an **average 5.35× (maximum 9.35×)** speedup on 6 benchmark circuit matrice and an average **2.62×(maximum 4.32×)** speedup on 5 non-circuit matrices.

WHERE **INNOVATION** BEGINS

DESIGN AUTOMATION CONFERENCE

Thanks for your listening!

Any questions?

JULY 9–13, 2023

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA