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: {tengcheng.wang, wenhao.li, haojie.pei, yuying.sun}@student.cpu.edu.cn,
{jinzhou, weifeng.liu}@cup.edu.cn

***Abstract*—Sparse LU factorization is considered to be one of the most time-consuming components in circuit simulation, particularly when dealing with circuits of considerable size in the advanced process era. Sparse LU factorization can be expedited by utilizing the supernode structure, which partitions the matrix into dense sub-matrices, thereby improving computational performance by utilizing level-3 Basic Linear Algebra Subprograms (BLAS) General Matrix Multiplication (GEMM) operations. The sparse and irregular structure of circuit matrices often impedes the formation of supernodes or results in the formation of supernodes with many zero elements, which in turn poses challenges for exploiting GEMM operations. In this paper, by fully utilizing the density in sub-matrices and combining GEMM with the Dense-Sparse Matrix Multiplication (SpMM), we propose a density-aware adaptive matrix multiplication equipped with machine learning techniques to optimize performance of the most-time consuming matrix multiplication operator so as to accelerate the sparse LU factorization. Numerical experiment results show that among the 6 circuit matrices tested, the average performance of matrix multiplication in our algorithm can be improved by 5.35x (up to 9.35x) compared to the performance of using GEMM directly in Schur-complement updates. Compared with state-of-the-art solver SuperLU_DIST, our method shows a substantial performance improvement.**

***Index Terms*—sparse LU factorization, circuit simulation, matrix multiplication, supernodal LU factorization, machine learning, random forest**

I. Introduction

Solving a set of sparse linear systems $Ax = b$ takes up the majority time of transistor-level SPICE-like circuit simulation, as the underlying computation in most of analysis types [1] (e.g., DC analysis, transient analysis and AC small signal analysis, etc.) is to solve the linear equations established from the modified nodal analysis (MNA). Especially as semiconductor continue to shrink and parasitic effects continue to grow, the size of the matrix to be solved increases and becomes the most time-consuming task in post-layout circuit simulation for solving linear systems. In contrast to iterative solver, direct solver, i.e., LU factorization, which usually shows reliable accuracy and does not suffer from the challenges of choosing a suitable preconditioner, is therefore commonly used in real-world circuit simulations to solve linear systems in most situations. Therefore, accelerating the sparse LU factorization has become significantly important for the transistor-level circuit simulation to help verification in complex large-scale circuit design.

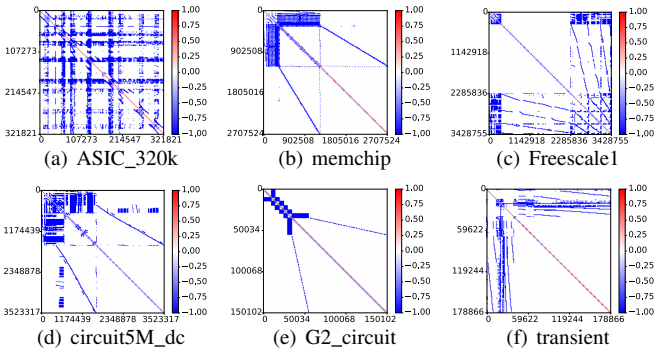


Fig. 1: The structure of the circuit sparse matrices.

Many studies have proposed optimization algorithms for sparse LU factorization, such as the multi-frontal or supernodal methods, which have been applied to packages such as MUMPS, UMFPACK and SuperLU [2]–[5]. The supernodal LU factorization is one of the most successful methods, which is to find or form sub-matrices with the similar dense structure to better invoke the dense matrix multiplication in the level-3 BLAS to achieve performance improvement. However, the circuit matrix is an extremely special class of sparse matrix with irregular non-zero element distribution characteristics, as shown in Fig. 1. Due to this, when performing sparse LU factorization of circuit matrix, it is often difficult to form supernodes or the formed supernodes are always dense and often have a certain sparsity, which makes the dense matrix multiplication [6] method not obtain the best performance. On this basis, in order to solve the problem that it is difficult to achieve the selection of optimal matrix multiplication by a single parameter or threshold, we further proposed an artificial intelligence-based self-tuning strategy, which can select the matrix multiplication operator with optimal performance for

This work was supported by National Key R&D Program of China (Grant No. 2022YFB4400400, 2021YFB0300600), the Key Program of the National Natural Science Foundation of China (Grant No.61972415, 62204265, 62234010), State Key Laboratory of Computer Architecture (ICT, CAS) (Grant No. CARCHA202115).

979-8-3503-2348-1/23/$31.00

each matrix multiplication operation in the solution process by training and learning sparse matrix features.

In this paper, we propose a density-aware acceleration strategy that introduces sparse matrix computation into the sparse LU factorization to improve performance, as well as an AI-based density-aware self-tuning strategy for selecting the optimal matrix multiplication operator for each operation in the factorization process. These strategies address performance bottlenecks in supernodal LU factorization when solving irregular matrix with supernodes.

6 circuit matrices and 5 non-circuit matrices from the SuiteSparse matrix Collection were selected for benchmarking. We highlight the novelty of AI-assisted self-tuning density-aware sparse LU factorization acceleration algorithm as follows.

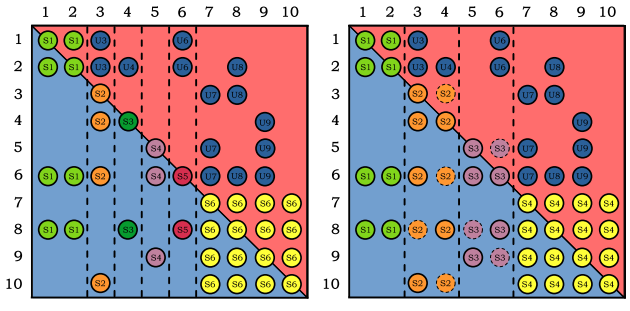
- We propose a density-aware sparse LU factorization acceleration method, leveraging sparse matrix multiplication in the large amount of Schur-complement updates.
- Sampling method based on the sample proportion of the unit matrix in total dataset improves the inference accuracy and model generality.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

II. Background

A. Sparse LU Factorization

Sparse LU factorization involves three steps: preprocessing, symbolic factorization, and numeric factorization [7]. Preprocessing involves the reordering of matrices to minimize the number of fill-in elements [8]. Symbolic factorization aims to identify the locations of these fill-in elements. However, the most crucial and time-consuming step is numeric factorization, which computes the final numerical results based on the determined matrix structure. This step is particularly critical for circuit simulation, where numeric factorization is often performed multiple times, while symbolic factorization is usually only performed once due to the uniform matrix pattern.

B. Supernodal LU Factorization



(a) Original supernodes (b) Supernodes after merging

Fig. 2: Supernodes and merging operation.

As shown in Fig. 2(a), the process of supernodal sparse LU factorization involves the grouping of columns with identical nonzero structure in L to form an unsymmetric supernode [4]. This is done to facilitate storage and computation by treating these columns as dense sub-matrices. As shown in Fig. 2(b), from (a) to (b), columns with different row structures (e.g., column 3 and column 4) are merged to form new supernodes, thus filling in some additional zero elements.

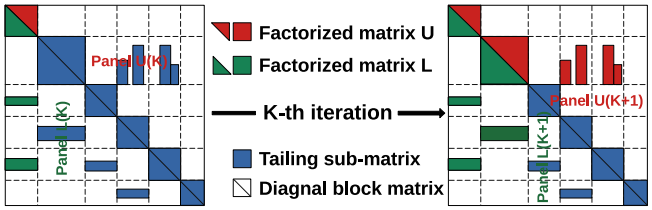


Fig. 3: Supernode-based right-looking update.

Upon formation of supernodes, the sparse LU factorization procedure follows four steps [9], [10]. The process represented in Fig. 3, is conducted in accordance with the following sequence, 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 updates for all the tailing sub-matrices by using $A = A - L \times U$, where L represents $L(K:N, K)$ and U represents $U(K, K+1:N)$.

III. Performance Bottleneck Analysis

Matrix multiplication takes the most proportion of computation time. In the numeric factorization, there are a large number of GEMM operations since we need to update all the tailing matrices for Schur-complement updates in each iteration. Taking the circuit matrices in Fig. 1 as the example, the total number of GEMM can reach 21,361-666,804, respectively. We verified the time proportion of GEMM in numeric factorization, as shown in Fig. 4, up to 73.4% and generally in the range of 40%-60%. This indicates that GEMM has become one of the major performance bottlenecks. Therefore, accelerating the large amounts of matrix multiplication shows significant importance to improve the LU factorization efficiency.

TABLE I: 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%

Irregular non-zero elements distribution pattern in circuit matrix may lead to certain sparsity in supernode

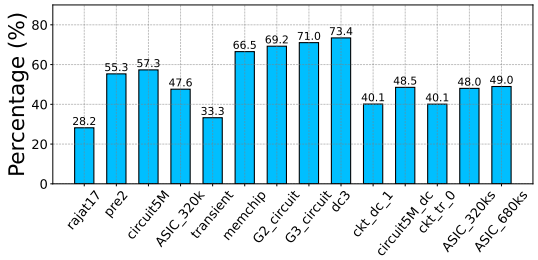


Fig. 4: The time proportion of GEMM in numeric factorization.

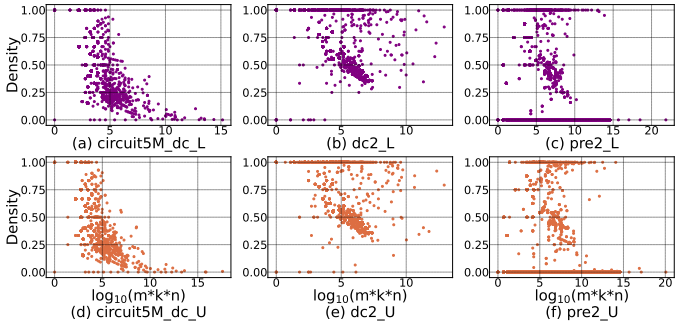


Fig. 5: Density analysis of sub-matrices in supernodal LU factorization.

sub-matrices. Circuit matrices are a type of irregular sparse matrices (Table I), which are often difficult for forming regular supernodes and tend to have extra zero elements that cause sparsity. The density of L-blocks (supernodes) and U-blocks of the typical circuit matrices are shown in Fig. 5 (m and k on the x-coordinate represent the number of rows and columns of the matrix L, respectively, and n is the number of columns of the matrix U), where the density of sub-matrix blocks are irregular. The density of the sub-matrix blocks suggests that GEMM may not always be the best method for computation, hence the introduction of SpMM in suitable scenarios shows great potential for further accelerating the computational efficiency of LU factorization.

IV. Density-aware Matrix Multiplication

From the previous observation, it cames obviously that sparse matrix multiplication may bring some performance improvement opportunities especially on the supernodes which are more sparse. Therefore, in this paper, we propose a density-aware matrix multiplication acceleration algorithm using GEMM and SpMM, and thus bringing the possibility of further accelerating LU factorization.

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. The "Oracle" showcases the potential for performance improvement by selecting the optimal algorithm between GEMM and SpMM for each matrix multiplication. Table II shows the total time of matrix multiplication for 12 circuit matrices in the above three cases. The "Speedup1" and "Speedup2" columns show that for different matrices, 'Oracle' has a performance improvement potential of 1.03x-10.24x and 1.13x-4.25x, respectively, compared to GEMM and SpMM.

TABLE II: Analysis of the performance improvement space of matrix multiplication. The Speedup1 is Oracle vs GEMM. The Speedup2 is Oracle vs SpMM.

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

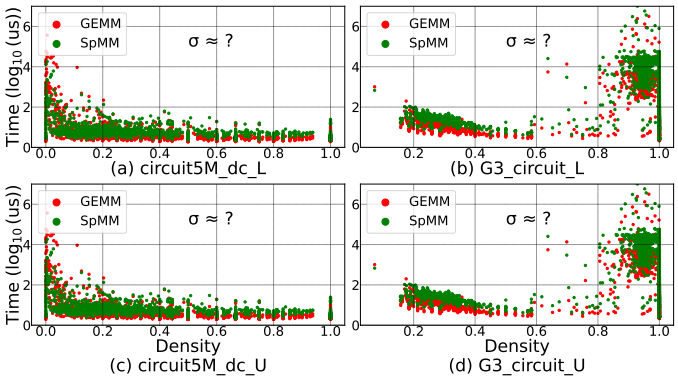


Fig. 6: Performance comparison between GEMM and SpMM based on density.

It is clear that for different matrices, GEMM and SpMM show superior performance in some cases, respectively. Therefore, selecting the appropriate GEMM or SpMM for each matrix multiplication is a crucial issue. Fig. 6 shows the possibility of dividing the GEMM and SpMM thresholds by density, with density as the x-coordinate and matrix multiplication time as the y-coordinate. It is difficult to see the threshold division from the figure, mainly because the matrix multiplication performance depends not only on the consistency, but also on other factors, such as matrix size, sparse structure, etc.

Fig. 7 shows the possibility of dividing the GEMM and SpMM thresholds by matrix size, with $log_{10}(m*k*n)$ as the x-coordinate and matrix multiplication time as the y-coordinate. It can be seen that Fig. 7(a) has threshold value $\sigma \approx 7.3$, Fig. 7(b) has threshold value $\sigma \approx 9.7$ and Fig. 7(c) has threshold value $\sigma \approx 9.2$, but Fig. 7(d) do not. Therefore, a single matrix feature cannot accurately determine whether to use GEMM or SpMM. To select between GEMM or SpMM, we require an adaptive strategy that combines multiple features.

V. Machine learning driven adaptive acceleration

As shown previously, it is difficult for some matrices to intuitively distinguish which algorithm is more efficient

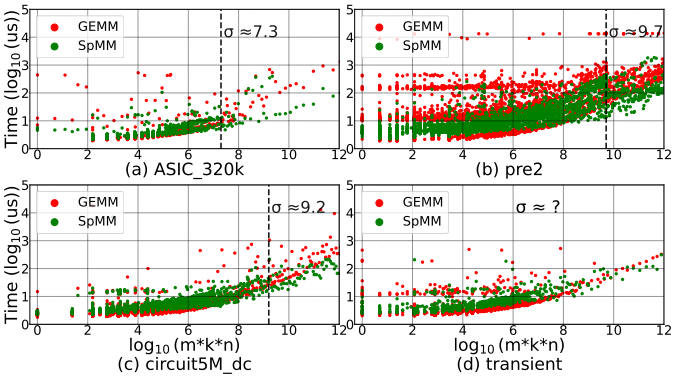


Fig. 7: Performance comparison between GEMM and SpMM based on matrix size.

through one single threshold. Thus, machine learning seems to be a promising alternative to address this situation due to its powerful pattern extraction ability, and we propose an AI self-turning sparse LU factorization acceleration algorithm.

Our core idea is to automatically select the optimal algorithm for the matrix multiplication by a well-trained classifier. Primarily, feature definition and selection determine the performance of trained model. In particular, proper feature selection not only reduces computation of model inference, but also improves accuracy of classification. In addition, once feature definition and selection is completed, we shall test these matrices with different matrix multiplication approaches to get the full dataset including features and labels (which one is the best matrix multiplication algorithm). In summary, the key points of proposed algorithm roughly include the feature selection, the sampling dataset and the technique for classification.

Feature selection. We must select representative features [11] from dataset. Each sample in the dataset includes 15 matrix features ($F1 \sim F15$) of L and U matrices as described in Fig. 8, and a classification label P. The label P represents the best matrix multiplication method for this sample. The following list summarizes a number of important features which can capture the major characteristics of matrix.

(1) Density_A/B. The density of matrix in L or U is calculated by equation $\frac{nnz}{m \times n}$, where nnz, m and n represent the non-zero elements, the rows and the columns in matrix L or U.

(2) Width_A/B. The bandwidth of matrix L or U is the distance between the farthest diagonal with a non-zero element and the main diagonal respectively, which describes matrix structure.

(3) stand_colA/B. The standard deviation of non-zero elements in L and U columns represent their distribution.

Training dataset. The very first stage of any deep learning or machine learning technique deployment is to decide the training dataset. The inference accuracy and model generality then largely depends on the quality of annotation and the diversity of training data. Then, the sampling for training set must be well done. Unlike traditional classification tasks, our dataset is collected from circuit simulations with highly diverse functions and scales, resulting in an unbalanced number of samples contributed by different circuit matrices. To overcome the above problem, each matrix must contribute a certain number of samples to the training set. Therefore, we harness a sampling method based on the sample proportion of the unit matrix in full dataset. The number of samples generated by the unit matrix that can be selected as the training set is shown in follow equation:

$$\#of\ training\ set = \frac{|D|}{|N|}\frac{|T|}{|N|} \tag{1}$$

where $|D|$, $|T|$ and $|N|$ represent numbers of unit matrix, numbers of training set and whole dataset respectively.

Algorithm 1 Random Forest

Require: Training data $\mathcal{D}$
Ensure: The ensemble of trees $\{T_b\}_1^B$
1: **for** $i = 1$ to B **do**
2: 1) Draw a bootstrap sample Z^* of size N from $\mathcal{D}$
3: 2) Grow a random forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
4: a) Select m variables at random from the p variables.
5: b) Pick the best variable/split-point among the m.
6: c) Split the node into daughter nodes.
7: **end for**

Technique for classification. The random forest as described in Algorithm 1, a well-known and widely-used bagging algorithm, is employed to complete AI self-tuning sparse LU factorization acceleration task as shown in Fig. 8. Each decision tree [12] in random forest is independent of each other and is called weak classifier. The information divergence is adopted to implement splitting of nodes in decision tree, which shows the difference between the entropy of the set to be classified and the conditional entropy of the selected feature. As discussed previously, a number of decision trees constitute a random forest, and these trees randomly select samples for training. The result of the random forest, acting as a classifier, is determined by the majority vote among the decision trees. Therefore, it has great advantages over other traditional machine learning algorithms as follows. 1) It can handle higher-dimensional data without requiring feature reduction. 2) Unbiased estimation is used for the generalization error, and the model has strong generalization ability. 3) The training speed is fast, and it is easy to parallelize.

The algorithm we propose, as described in Algorithm 2, involves fitting the implicit classification function between input dynamics (features) and outputs (labels) using a specific data-driven random forest algorithm. The well-trained classifier can predict the label of unseen data.

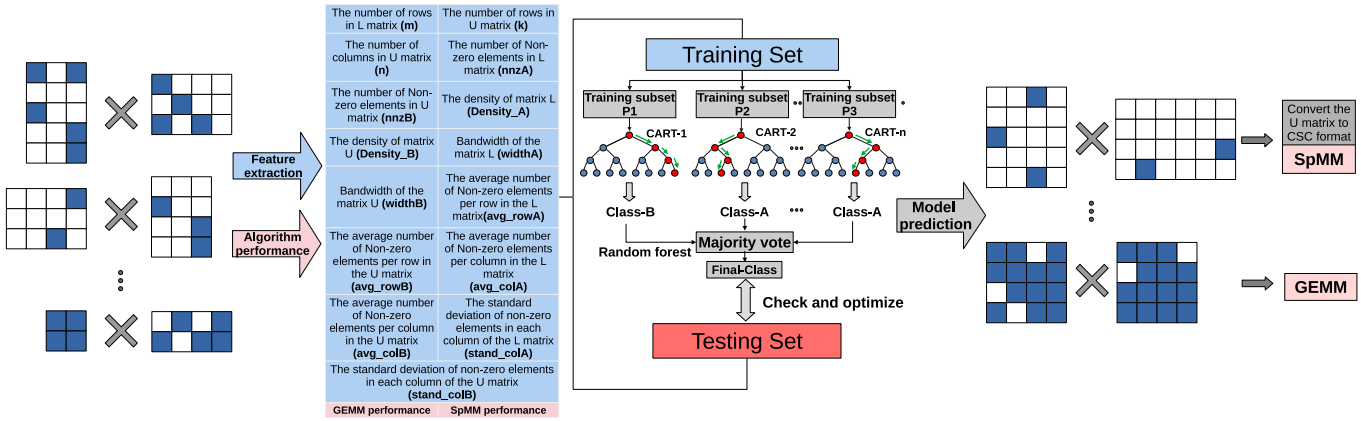


Fig. 8: AI self-tuning sparse LU factorization acceleration based on Random Forest algorithm.

Algorithm 2 AI self-tuning sparse LU factorization acceleration algorithm.

for block $K = 1$ **to** N **do**
  if me $\in PROC_C(K)$ **then**
    Factorize block column $L(K:N, K)$
    Send $L(K:N, K)$ processes in my row who need it
  else
    Receive $L(K:N, K)$ from one process in $PROC_c(K)$
  end if
  if me $\in PROC_C(K)$ **then**
    Factorize block row $U(K:N+1, K)$
    Send $U(K:N+1, K)$ processes in my row who need it
  else
    Receive $U(K:N, K)$ from one process in $PROC_c(K)$
  end if
  for J= K+1 **to** N **do**
    for I= K+1 **to** N **do**
      if me$\in$ $PROC_R(I)$ **and** me$\in$ $PROC_c(J)$ **and** $L(I,K) \neq 0$ **and** $U(K,J) \neq 0$ **then**
        Data Processing
        Calculate features of $L(I,K)$ and $U(K,J)$ as X
        $P = f(X, \theta)$, **Well-trained random forest model**
        if $P = 1$ **then**
          Update $A(I,J) \leftarrow A(I,J) - L(I,K) * U(K,J)$ by **GEMM**
        else
          Update $A(I,J) \leftarrow A(I,J) - L(I,K) * U(K,J)$ by **SpMM**
        end if
      end if
    end for
  end for
end for

VI. Experimental Results

A. Experimental Setup

We apply the ML-driven density-aware adaptive matrix multiplication to the supernodal LU factorization solver (SuperLU_DIST 8.0.0) and run tests on AMD EPYC 7702 CPU 2.0 GHz and 512 GB RAM. All datasets are generated from circuit matrices downloaded from the SuiteSparse Matrix Collection and are completely independent of the matrices tested in this paper. The specific information of three datasets (D1, D2, D3) are shown in Table III.

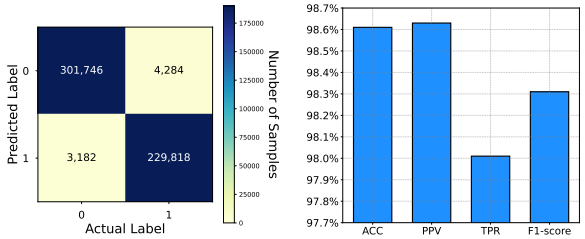
TABLE III: The distribution of dataset.

Dataset	Total sample set(D1)	Training set (D2)	Testing set (D3)
GEMM	456,000	150,000	306,000
SpMM	383,000	150,000	233,000

For all the benchmarks, we record their numeric factorization time and matrix multiplication time when using GEMM, SpMM and density-aware adaptive, respectively.

B. Model Accuracy and Feature Importance Evaluation

We evaluate our trained model with Testing set (D3). The confusion matrix is employed to evaluate the prediction ability of the model. In Fig. 9(a), label "0" represents GEMM and label "1" represents SpMM. We can see that in the sample with better GEMM performance, GEMM and SpMM were selected 301,746 and 4,284 times, respectively. Conversely, in the sample with better SpMM performance, GEMM and SpMM were selected 3,182 and 229,818 times, respectively, indicating that our model has better overall accuracy.



(a) Confusion matrix (b) Performance of the binary classifiers

Fig. 9: Confusion matrix and performance of trained model.

As shown in the Fig. 9(b), where ACC represents the percentage of correctly predicted results to the total samples, PPV represents the percentage of samples that are actually positive among all samples that are predicted to be positive,

TABLE IV: 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

TABLE V: 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

TPR represents the percentage of samples that are predicted to be positive among those that are actually positive, and F1-score represents the weighted summed average of PPV and TPR. The values of ACC, PPV, TPR and F1-Score all exceed 0.9, which indicates that our AI self-tuning model has strong predictive ability.

C. Acceleration for Circuit Matrix

We compare the performance of the following three algorithms, as shown in Table IV. The experimental results show that in the matrix multiplication stage, AI has a maximum speedup of 9.35x and an average speedup of 5.35x compared to GEMM, and a maximum speedup of 1.35x and an average speedup of 1.05x compared to SpMM. In the numeric factorization stage, the maximum speedup is 1.76x and the average speedup is 1.50x. GEMM is part of the numeric factorization process, so the acceleration ratio of numerical factorization will be lower than that of GEMM.

D. Acceleration for Other Irregular Sparse Matrix

Finally, we test 5 non-circuit irregular matrices, as shown in Table V. The results show that in the matrix multiplication stage, AI has a maximum speedup of 4.32x and an average speedup of 2.62x compared to GEMM, and a maximum speedup of 1.32x and an average speedup of 1.28x compared to SpMM. In the numeric factorization stage, the maximum speedup is 2.08x and the average speedup is 1.55x. In cases of irregular matrices, SpMM may outperform GEMM, indicating that this work can also be applied to irregular non-circuit matrices.

VII. Conclusions

In this paper, we study the main performance bottleneck and performance improvement opportunities of sparse LU factorization for circuit matrix. We proposed a ML-driven self-tuning algorithm to accelerate matrix multiplication. The algorithm adaptively selects the appropriate GEMM or SpMM algorithm for each matrix multiplication operation in the Schur-complement updates, based on features such as the density, nnz and bandwidth of the matrix. Numerical experiments show that the new method not only significantly accelerates the solution performance of circuit matrices, but is also effective for the irregular non-circuit matrices. It has good effectiveness and universality.

Acknowledgment

We deeply appreciate the invaluable comments from the reviewers. Zhou Jin and Weifeng Liu are the corresponding authors of this paper.

References

[1] F. N. Najm, *Circuit simulation*, 2010.
[2] P. Amestoy, I. S. Duff, J. Y. L'Excellent, and J. Koster, "Mumps: A general purpose distributed memory sparse solver," in *PARA*, 2000.
[3] T. A. Davis, *Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method*, 2004.
[4] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. Liu, "A supernodal approach to sparse partial pivoting," *SIMAX*, 1999.
[5] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear," *TOMS*, 1983.
[6] L. Bo, K. Per and V. L. Charles, "Gemm-based level 3 blas: high-performance model implementations and performance evaluation benchmark," *TOMS*, 1998.
[7] J. Zhao, Y. Wen, Y. Luo, Z. Jin, W. Liu, and Z. Zhou, "Sflu: Synchronization-free sparse lu factorization for fast circuit simulation on gpus," in *DAC*, 2021.
[8] G. Cui, W. Yu, X. Li, Z. Zeng, and B. Gu, "Machine-learning-driven matrix ordering for power grid analysis," in *DATE*, 2019.
[9] P. Sao, X. S. Li, and R. Vuduc, "A communication-avoiding 3d algorithm for sparse lu factorization on heterogeneous systems," *JPDC*, 2019.
[10] X. S. Li and J. W. Demmel, "Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *TOMS*, 2003.
[11] B. Matthias, S. Olaf, J. Radim, H. Steve, and G. Kiran, "State-of-the-art sparse direct solvers," in *Parallel Algorithms in Computational Science and Engineering*, 2020.
[12] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, "An introduction to decision tree modeling," *Journal of Chemometrics: A Journal of the Chemometrics Society*, 2004.