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

Read the netlist and establish the circuit equation

DC analysis

Transient iteration

Newton-Raphson iteration

Convergence?

Waveform output

Model evaluation

Sparse LU factorization ($A=LU$)

Triangle solution ($Ly=b$, $Ux=y$)

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 $60-90\%$\cite{1} of the time.

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 $U$.

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

- 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

1. **Preprocessing**
   - Reduce fill-ins.
2. **Symbolic**
   - Identify the locations of these fill-in elements.
3. **Numeric**
   - Compute the final numerical results

Left-looking

Right-looking
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** $U$.

$$
\begin{bmatrix}
    a_{11} & a_{14} \\
    a_{21} & a_{22} & a_{23} & a_{24} \\
    a_{31} & a_{32} & a_{33} & \ddots & a_{34} \\
    & \ddots & \ddots & \ddots & \ddots \\
    & & \ddots & a_{42} & a_{44}
\end{bmatrix}
= \begin{bmatrix}
    1 & 0 & 0 & 0 \\
    1 & 1 & 0 & 0 \\
    0 & 1 & 1 & 0 \\
    & \ddots & \ddots & \ddots & \ddots \\
    & & \ddots & 1 & 1
\end{bmatrix} \begin{bmatrix}
    u_{11} & u_{14} \\
    u_{21} & u_{24} \\
    u_{31} & u_{34} & u_{44}
\end{bmatrix}
$$

- 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**

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.
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\[1\].

---


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\([1] [2] [3]\).

\[2\] 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.

---


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 \( A = A - L \times U \).
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 $A = A - LU$.

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

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

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.

The time proportion of GEMM in numeric factorization.
GEMM takes up **much of the time** for 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.

Accelerating GEMM is **of great significance** for numeric factorization performance improvement.
Supernodal LU factorization will divide the sparse matrix into supernodes.
Supernodal LU factorization will divide the sparse matrix into supernodes.
Supernodal LU factorization will divide the sparse matrix into supernodes.

**Schur-complement**

\[ A(I, J) \leftarrow A(I, J) - L(I, K) \times U(K, J); \]

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

<table>
<thead>
<tr>
<th>Circuit Matrix</th>
<th>N</th>
<th>Entries per row</th>
<th></th>
<th>Symmetry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>max</td>
<td>min</td>
<td>average</td>
</tr>
<tr>
<td>ASIC_320k</td>
<td>321,821</td>
<td>203,800</td>
<td>1</td>
<td>8.2</td>
</tr>
<tr>
<td>memchip</td>
<td>2,707,524</td>
<td>27</td>
<td>2</td>
<td>5.5</td>
</tr>
<tr>
<td>Freescale1</td>
<td>3,428,755</td>
<td>27</td>
<td>1</td>
<td>5.5</td>
</tr>
<tr>
<td>circuit5M</td>
<td>3,523,317</td>
<td>27</td>
<td>1</td>
<td>10.7</td>
</tr>
<tr>
<td>G2_circuit</td>
<td>150,102</td>
<td>4</td>
<td>1</td>
<td>2.9</td>
</tr>
<tr>
<td>transient</td>
<td>178,866</td>
<td>60423</td>
<td>1</td>
<td>5.4</td>
</tr>
</tbody>
</table>
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.

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.
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.
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.

<table>
<thead>
<tr>
<th>Circuit matrix</th>
<th>nnz (A)</th>
<th>GEMM (s)</th>
<th>SpMM (s)</th>
<th>Oracle (s)</th>
<th>Speedup1</th>
<th>Speedup2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC_320k</td>
<td>1,931,828</td>
<td>3.5809</td>
<td>0.4543</td>
<td>0.3653</td>
<td>9.80x</td>
<td>1.24x</td>
</tr>
<tr>
<td>Freescale1</td>
<td>17,052,626</td>
<td>8.8363</td>
<td>35.9429</td>
<td>8.5388</td>
<td>1.03x</td>
<td>4.21x</td>
</tr>
<tr>
<td>ckt1752_dc1</td>
<td>333,029</td>
<td>0.0279</td>
<td>0.0433</td>
<td>0.0225</td>
<td>1.24x</td>
<td>1.92x</td>
</tr>
<tr>
<td>pre2</td>
<td>5,834,044</td>
<td>57.2954</td>
<td>10.9046</td>
<td>7.2531</td>
<td>7.90x</td>
<td>1.50x</td>
</tr>
<tr>
<td>meg4</td>
<td>58,142</td>
<td>0.0037</td>
<td>0.0027</td>
<td>0.0022</td>
<td>1.68x</td>
<td>1.23x</td>
</tr>
<tr>
<td>G2_circuit</td>
<td>726,674</td>
<td>7.1145</td>
<td>26.0853</td>
<td>6.1415</td>
<td>1.16x</td>
<td>4.25x</td>
</tr>
<tr>
<td>Freescale2</td>
<td>14,313,235</td>
<td>2.3253</td>
<td>3.8100</td>
<td>1.9497</td>
<td>1.19x</td>
<td>1.95x</td>
</tr>
<tr>
<td>FullChip</td>
<td>26,621,983</td>
<td>510.416</td>
<td>480.202</td>
<td>344.1850</td>
<td>1.48x</td>
<td>1.40x</td>
</tr>
<tr>
<td>ASIC_320ks</td>
<td>1,316,085</td>
<td>2.9155</td>
<td>0.322</td>
<td>0.2846</td>
<td>10.24x</td>
<td>1.13x</td>
</tr>
<tr>
<td>ASIC_680ks</td>
<td>1,693,767</td>
<td>2.6196</td>
<td>1.3393</td>
<td>0.9320</td>
<td>2.81x</td>
<td>1.44x</td>
</tr>
<tr>
<td>circuit5M_dc</td>
<td>14,865,409</td>
<td>7.5022</td>
<td>1.2420</td>
<td>1.0230</td>
<td>6.04x</td>
<td>1.21x</td>
</tr>
<tr>
<td>transient</td>
<td>961,368</td>
<td>0.5721</td>
<td>0.2710</td>
<td>0.2100</td>
<td>2.69x</td>
<td>1.29x</td>
</tr>
</tbody>
</table>

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.
There exists a definite threshold $\sigma$ on some matrices, but some matrices do not have a definite threshold $\sigma$ (such as transient).

To select between GEMM or SpMM, we require an adaptive strategy that combines multiple features.
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.
We need a suitable method to select the better matrix multiplication algorithm for complex and variable matrices.

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.

## 1) Preprocessing

<table>
<thead>
<tr>
<th>Feature extraction</th>
<th>Algorithm performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of rows in L matrix (m)</td>
<td>The number of rows in U matrix (k)</td>
</tr>
<tr>
<td>The number of columns in U matrix (n)</td>
<td>The number of Non-zero elements in L matrix (mnA)</td>
</tr>
<tr>
<td>The number of Non-zero elements in U matrix (mnB)</td>
<td>The density of matrix U (Density_A)</td>
</tr>
<tr>
<td>The density of matrix U (Density_B)</td>
<td>Bandwidth of the matrix U (widthA)</td>
</tr>
<tr>
<td>Bandwidth of the matrix U (widthB)</td>
<td>The average number of Non-zero elements per row in the L matrix (avg_rowA)</td>
</tr>
<tr>
<td>The average number of Non-zero elements per row in the U matrix (avg_rowB)</td>
<td>The average number of Non-zero elements per column in the L matrix (avg_colA)</td>
</tr>
<tr>
<td>The average number of Non-zero elements per column in the U matrix (avg_colB)</td>
<td>The standard deviation of non-zero elements in each column of the L matrix (stand_colA)</td>
</tr>
<tr>
<td>The standard deviation of non-zero elements in each column of the U matrix (stand_colB)</td>
<td>GEMM performance SpMM performance</td>
</tr>
</tbody>
</table>

## 2) Model Training

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:**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>GEMM</th>
<th>SpMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total sample set(D1)</td>
<td>456,000</td>
<td>383,000</td>
</tr>
<tr>
<td>Training set (D2)</td>
<td>150,000</td>
<td>150,000</td>
</tr>
<tr>
<td>Testing set(D3)</td>
<td>306000</td>
<td>233000</td>
</tr>
</tbody>
</table>

An example of 15 matrix features (F1-F15).
Experimental results

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

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%.
## Experimental results

### Performance evaluation on circuit matrices.

<table>
<thead>
<tr>
<th>circuit matrix</th>
<th>nnz</th>
<th>GEMM</th>
<th>SpMM</th>
<th>Oracle</th>
<th>AI</th>
<th>Speedup (AI vs GEMM)</th>
<th>Speedup (AI vs SpMM)</th>
<th>Numeric factorization time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC_320k</td>
<td>1,931,828</td>
<td>3.5809</td>
<td>0.4543</td>
<td>0.3653</td>
<td>0.5045</td>
<td>7.10x</td>
<td>0.90x</td>
<td>7.5201</td>
</tr>
<tr>
<td>ASIC_320ks</td>
<td>1,316,085</td>
<td>2.9155</td>
<td>0.3223</td>
<td>0.2846</td>
<td>0.3117</td>
<td>9.35x</td>
<td>1.03x</td>
<td>6.0602</td>
</tr>
<tr>
<td>ASIC_680ks</td>
<td>1,693,767</td>
<td>2.6196</td>
<td>1.3393</td>
<td>0.9323</td>
<td>1.0085</td>
<td>2.60x</td>
<td>1.33x</td>
<td>5.3502</td>
</tr>
<tr>
<td>circuit5M_dc</td>
<td>14,865,409</td>
<td>7.5022</td>
<td>1.2418</td>
<td>1.0231</td>
<td>2.0084</td>
<td>6.04x</td>
<td>0.62x</td>
<td>19.2301</td>
</tr>
<tr>
<td>pre2 transient</td>
<td>5,834,044</td>
<td>57.2954</td>
<td>10.9046</td>
<td>7.2531</td>
<td>8.1021</td>
<td>7.07x</td>
<td>1.35x</td>
<td>103.5721</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5.35x</td>
<td>1.05x</td>
<td>-</td>
</tr>
</tbody>
</table>

### Performance evaluation on non-circuit matrices.

<table>
<thead>
<tr>
<th>non-circuit matrix</th>
<th>nnz</th>
<th>GEMM</th>
<th>SpMM</th>
<th>Oracle</th>
<th>AI</th>
<th>Speedup (AI vs GEMM)</th>
<th>Speedup (AI vs SpMM)</th>
<th>Numeric factorization time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sine12</td>
<td>283,992</td>
<td>9.7211</td>
<td>2.7060</td>
<td>2.0120</td>
<td>2.2491</td>
<td>4.32x</td>
<td>1.20x</td>
<td>14.7541</td>
</tr>
<tr>
<td>psmigr_3</td>
<td>543,160</td>
<td>16.3140</td>
<td>7.8951</td>
<td>5.8241</td>
<td>6.2611</td>
<td>2.62x</td>
<td>1.26x</td>
<td>28.4921</td>
</tr>
<tr>
<td>psmigr_2</td>
<td>540,022</td>
<td>30.8151</td>
<td>12.0731</td>
<td>8.6251</td>
<td>9.1721</td>
<td>3.36x</td>
<td>1.32x</td>
<td>45.9061</td>
</tr>
<tr>
<td>eph2 benzene</td>
<td>175,027</td>
<td>0.1291</td>
<td>0.10021</td>
<td>0.05631</td>
<td>0.07621</td>
<td>1.69x</td>
<td>1.31x</td>
<td>0.4351</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.62x</td>
<td>1.28x</td>
<td>-</td>
</tr>
</tbody>
</table>

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 $5.35 \times$ (maximum $9.35 \times$) speedup on 6 benchmark circuit matrices and an average $2.62 \times$ (maximum $4.32 \times$) speedup on 5 non-circuit matrices.
Thanks for your listening!
Any questions?

JULY 9–13, 2023
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA