MASC: A Memory-Efficient Adjoint Sensitivity Analysis through Compression Using Novel Spatiotemporal Prediction

Chenxi Li, Boyuan Zhang, Yongqiang Duan, Yang Li, Zuochang Ye, Weifeng Liu, Dingwen Tao, Zhou Jin

MASC: A Memory-Efficient Adjoint Sensitivity Analysis through Compression Using Novel Spatiotemporal Prediction

Chenxi Li[1], Boyuan Zhang[2], Yongqiang Duan[1], Yang Li[1], Zuochang Ye[3],
Weifeng Liu[1], Dingwen Tao[2] and Zhou Jin[1]
1. Super Scientific Software Laboratory, China University of Petroleum-Beijing, Beijing, China
2. Indiana University, Bloomington, IN, USA 3. Tsinghua University, Beijing, China

ABSTRACT

Adjoint sensitivity analysis is critical in modern integrated circuit design and verification, but its computational intensity grows significantly with the size of the circuit, the number of objective functions, and the accumulation of time points. This growth can impede its wider application. The intimate link between the forward integration in transient analysis and the reverse integration in adjoint sensitivity analysis allows for the retention of Jacobian matrices from transient analysis, thereby speeding up sensitivity analysis. However, Jacobian matrices across multiple timesteps are often so large that they cannot be stored in memory during the forward integration process, necessitating disk storage and incurring significant I/O overhead. To address this, we develop a memory-efficient sensitivity analysis method that utilizes data compression to minimize memory overhead during simulation and enhance analysis efficiency. Our compression method can efficiently compress the sparse tensor that contains the Jacobian matrices over time by exploiting the spatiotemporal characteristics of the data and circuit attributes. It also introduces a shared-indices technique, a cutting-edge spatiotemporal prediction model, and robust residual encoding. We evaluate our compression method on 7 datasets from real-world simulations and demonstrate that it can reduce the memory requirements for storing Jacobian matrices by more than 16× on average, which is significantly more efficient than other state-of-the-art compression techniques.

1 INTRODUCTION

In contemporary integrated circuit design, transient sensitivity analysis plays a vital role in various domains, including circuit optimization [1], performance modeling [2], and yield estimation [3]. The working performance of a circuit is influenced by numerous critical parameters, such as transistor dimensions, parasitic resistances, capacitances, and more. Sensitivity analysis is a valuable tool for examining the impact of these factors on system output. However, the conventional direct method falls short when dealing with a large number of parameters [4]; as a result, the adjoint method has become the standard in modern circuit simulations.

In large-scale circuit simulations, transient adjoint sensitivity analysis often results in significant computational overhead. This arises from the discretization of the dynamic system through numerical integration methods, which typically require solving large-scale differential equations at numerous time points – a process that can be quite time-consuming. This challenge is particularly prevalent in modern integrated circuit design. A major portion of the time spent in the adjoint method is devoted to computing Jacobian matrices [5]. Therefore, any approach that enables the adjoint method to bypass these computations could lead to considerable time savings. Fortunately, the adjoint system and the original system usually share the same Jacobian matrices. By storing and reusing these matrices instead of recomputing them, significant reductions in simulation time can be achieved.

Nonetheless, efforts to speed up sensitivity analysis by retaining Jacobians during transient analysis result in excessive memory overhead. Consequently, an alternative approach is to save this data on disk storage. However, this introduces frequent file I/O operations and data retrieval processes, leading to considerable time overhead. Such an approach significantly diminishes the simulator's efficiency while also placing substantial demands on disk storage.

To address the aforementioned issues, the introduction of in-memory compression is crucial. However, existing compression methods do not adequately exploit the intrinsic characteristics of data during the simulation process, resulting in suboptimal compression ratios and speeds. It is noteworthy that these matrices form a tensor along the time dimension, characterized by the following features: Firstly, each matrix is stored in a sparse storage format, with row indices and column indices represented as integer arrays, and non-zero values as floating-point arrays. Secondly, due to the brief step lengths in numerical integration, there is a significant correlation between temporally adjacent matrices, essentially making these matrices a form of time series data. Furthermore, these Jacobians are intimately linked to the circuit's structure.

To this end, we propose a novel lossless compression method that employs spatiotemporal prediction, effectively enhancing compression ratios for Jacobians across timesteps. This algorithm skillfully utilizes the spatiotemporal attributes of Jacobian matrices, enabling efficient compression of the sparse tensor formed by these matrices. As a result, it significantly reduces memory overhead in simulations without sacrificing accuracy. To our knowledge, this represents the first instance of a data compression-based approach to adjoint sensitivity analysis. It also constitutes the first thorough exploration of sparse Jacobian matrix compression in the context of SPICE simulations. Our contributions are summarized as follows:

- We conduct a comprehensive analysis of data characteristics in circuit simulation and investigate different floating-point compression methods from various fields.
- We develop a lossless floating-point compression method based on a novel spatiotemporal prediction model for Jacobian matrices. We adeptly employ the shared indices technique to address the storage and computational overhead associated with the Jacobian matrix indices.
- We evaluate our compression method on 7 real-world datasets, showcasing its superior performance compared with other leading compression methods.

- We further employ the proposed compression algorithm into the sensitivity analysis in SPICE simulator, and demonstrate memory-efficient acceleration with end-to-end comparison.

The rest of this paper is organized as follows. Section 2 provides an outline of the research background, and Section 3 discusses the motivation. In Section 4, we present our compression method, followed by the MASC design in Section 5. The evaluation results are detailed in Section 6. We conclude the paper in Section 7.

2 BACKGROUND

2.1 ADJOINT SENSITIVITY ANALYSIS

For sensitivity analysis, the circuit we are interested in can be represented by a set of Differential Algebraic Equations (DAE):

$$\boldsymbol{g}(\boldsymbol{x}, t, \boldsymbol{p}) = \frac{d}{dt}\boldsymbol{q}(\boldsymbol{x}, \boldsymbol{p}) + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{p}) + \boldsymbol{b}(t, \boldsymbol{p}) = \boldsymbol{0}, \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^D$ is the state vector, t is time, $\boldsymbol{p} \in \mathbb{R}^M$ is the parameter vector with respect to which we need sensitivities, $\boldsymbol{q} \in \mathbb{R}^D$ and $\boldsymbol{f} \in \mathbb{R}^D$ respectively represent dynamic and static elements, $\boldsymbol{b} \in \mathbb{R}^D$ is the input. It is common to discretize the DAE system using numerical integration methods such as backward Euler, and then employ the Newton-Raphson iteration to solve at each timestep.

The objective function is typically a function of the state vector $\vec{x}$ at one or multiple time points:

$$\boldsymbol{O} = \zeta(\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_n). \tag{2}$$

Then the sensitivity is the derivative of the objective function with respect to the system parameters $\mathbf{p}$:

$$\frac{d\boldsymbol{O}}{d\boldsymbol{p}} = \sum_{i=0}^{n}\left(\frac{d\boldsymbol{O}}{d\boldsymbol{x}}\right)_i\left(\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{p}}\right)_i. \tag{3}$$

Adjoint method is shown as:

$$\frac{dO}{dp} = \sum_{i=0}^{n}\sum_{k=i}^{n}(\frac{\mathrm{d}\boldsymbol{O}}{\mathrm{d}\boldsymbol{x}})_k(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}})_k^{-1}\left(\prod_{j=k-1}^{i}(\frac{1}{h_t}\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{x}})_j(\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}})_j^{-1}\right)(\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{p}})_i. \tag{4}$$

It can be considered as a backward propagation of transient analysis, with h_t as the step size for numerical integration and $\boldsymbol{\varphi}$ representing terms related to the system parameter $\boldsymbol{p}$:

$$\left(\frac{\partial \boldsymbol{\varphi}}{\partial \boldsymbol{p}}\right)_{i+1} = \frac{1}{h_t}[(\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{p}})_{i+1} - (\frac{\partial \boldsymbol{q}}{\partial \boldsymbol{p}})_i] + \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}}\right)_{i+1} + \left(\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{p}}\right)_{i+1} \tag{5}$$

Based on (4), it is evident that during the adjoint process, repetitive computations of Jacobian matrices at multiple time points are required, leading to a significant additional time overhead.

2.2 FLOATING-POINT DATA COMPRESSION

Jacobian matrices, which are usually sparse, are often stored in Compressed Sparse Row (CSR) format, which utilizes integer indices and floating-point non-zero values. To further reduce the size of these CSR sparse matrices, compression techniques can be applied to both the non-zero floating-point values and the integer indices.

General-purpose lossless compression can be divided into two main categories: (1) Dictionary-based compression, which aims to reduce spatial redundancy in the input data stream. Examples include LZ77 and LZW . LZ77 is designed to efficiently encode locally repeated sequences, while LZW assembles a lexicon of recurring data strings, allowing for the substitution of long repeated patterns with shorter dictionary indices. (2) Entropy-based compression, which utilizes the entropy/frequency information of the input for encoding. This category includes methods such as Asymmetric Numeral Systems (ANS) encoding [6] and Huffman encoding . ANS efficiently compresses data by assigning shorter codes to more frequent symbols and longer codes to less frequent ones, while Huffman encoding uses a different encoding strategy which is building a Huffman tree based on the frequency to generate the prefix code. While Huffman encoding is simpler and quite effective for many applications, ANS encoding offers higher efficiency and adaptability at the cost of increased complexity.

Table 1: **Time comparison between transient analysis and sensitivity analysis and the proportion of Jacobian matrix calculation time in sensitivity analysis.**

Circuits	Type	#Elem	#Param	#Obj	#Steps	Tran (s)	Sens (s)	T_{Sens}/T_{Tran}	T_{Jac}/T_{Sens}
CHIP_01	BJT	40221	126	8	3738	144.6	595.6	4.1	63.3%
CHIP_02	BJT	58946	183	12	5524	242.4	1969.3	8.1	62.4%
CHIP_03	BJT	100445	351	21	2837	258.8	3279.1	12.7	57.9%
CHIP_04	BJT	132097	398	27	1278	358.5	2449.0	6.8	57.4%
CHIP_05	BJT	165669	421	32	682	89.3	1668.3	18.7	61.6%
CHIP_06	BJT	162440	409	30	552	73.4	148.1	20.2	57.0%
CHIP_07	BJT	226583	512	38	2625	431.5	9413.9	21.8	58.3%
CHIP_08	BJT	289669	602	40	3521	801.6	16032.7	20.0	59.4%
CHIP_09	BJT	316556	728	48	6678	1855.9	38887.5	21.0	60.4%
ram2k	MOS	14059	109	12	2330	172.6	438.4	2.5	65.3%
smult20	MOS	46075	254	52	6276	867.7	5107.5	5.9	50.6%
RC_01	RC	117843	324	42	5204	55.9	1959.5	35.1	46.1%
RC_02	RC	153338	387	52	1106	14.7	668.3	45.4	45.9%

Several significant endeavors, including FPZIP [7], NDZIP [8], and ZFP [9], have proposed new general-purpose lossless compressors specifically for floating-point data. Each algorithm employs distinct methodologies to achieve decorrelation of floating-point data, which is a prerequisite step before the application of various encoding techniques, including the aforementioned ones or alternative, more straightforward encoding methods. However, these compressors do not consider the specific characteristics of the data such as Jacobian matrices during decorrelation.

3 MOTIVATION

3.1 TIME-CONSUMING COMPUTATION

Table 1 presents a time comparison between transient analysis and ajoint sensitivity analysis for circuits of varying scales. This comparison is made under different numbers of objective functions, given parameters, and at various time points. Due to the impact of these factors, the time overhead associated with sensitivity analysis is substantial, often being several times or even tens of times that of transient simulation, which is unacceptable.

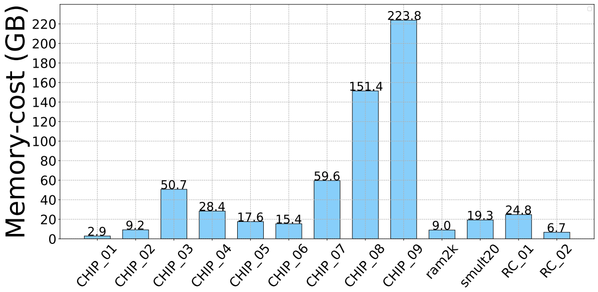


Figure 1: **Memory cost for storing Jacobians.**

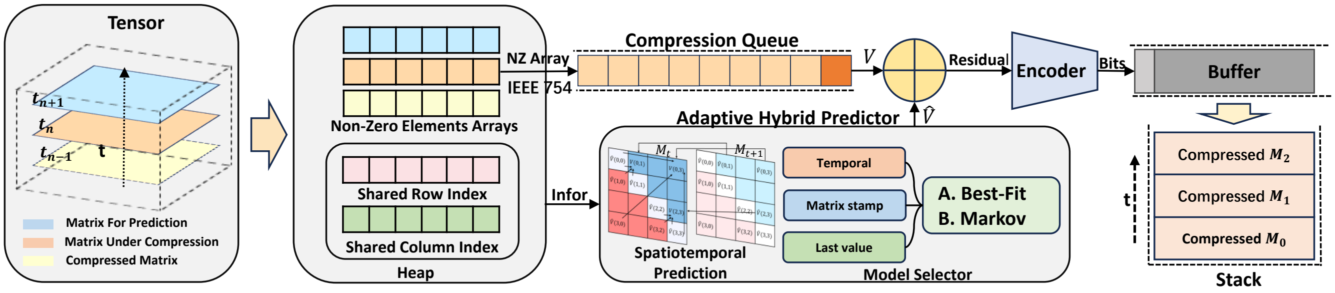


Figure 2: **Overall framework of sparse Jacobian matrices compression technology.**

Furthermore, in adjoint sensitivity analysis, the computation of the Jacobian matrix is notably time-consuming. For example, as shown in Table 1, the time spent on calculating the Jacobian matrix accounts for about 50%, and in some cases, it can go as high as about 65%. If the calculation of the Jacobian matrix can be avoided in the sensitivity analysis process, the efficiency of the sensitivity analysis can be significantly enhanced.

3.2 HIGH MEMORY COST

By reusing Jacobian from transient analysis, we can significantly accelerate the sensitivity analysis process. However, this strategy leads to considerable memory overhead. As illustrated in Figure 1, in adjoint sensitivity analysis, the storage overhead for Jacobians can surpass several hundred gigabytes as the scale of the circuit increases. Consequently, developing a compression method that offers a high compression ratio for Jacobians is essential to effectively reduce the memory burden during the simulation process.

Furthermore, to enhance the accuracy of simulation results, it is preferable to employ efficient lossless compression during the simulation process rather than lossy compression. The use of lossy compression could lead to significant cumulative errors.

4 SPATIOTEMPORAL PREDICTION BASED COMPRESSION FRAMEWORK

To minimize memory overhead during simulation, we propose an innovative approach for effectively compressing sparse Jacobians. The complete workflow of our compression method is depicted in Figure 2. We employ sparse storage formats, such as CSR, to store the Jacobian matrix. Considering that the values in the index array are integers and the non-zero element array comprises floating-point numbers, we discuss their compression separately.

4.1 SHARED INDICES

First, let us briefly discuss the compression and storage of indices. By observing the stamping process of the Jacobian matrix, we can notice that the structure of all matrices is identical. Therefore, we have ingeniously employed a technique called "Shared Indices" to effectively reduce the computational and storage overhead of the index arrays. Since all Jacobian matrices share a common set of row and column index, we place them in heap memory to ensure a longer lifecycle. If further compressing index arrays is desired, delta encoding can first be utilized to convert large integers into smaller ones, and then variable-length encoding can be applied to achieve efficient compression of indices.

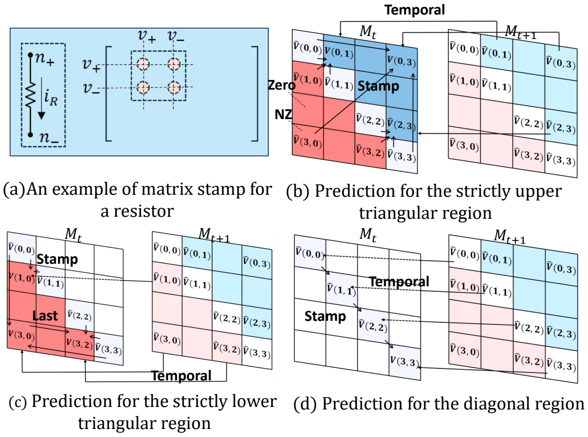
4.2 SPATIOTEMPORAL PREDICTION MODEL

The primary challenge in compressing Jacobians lies in the efficient compression of non-zero element arrays. In this section, we introduce an efficient lossless floating-point compression method based on a novel hybrid prediction model to address this challenge.

Temporal Prediction Model. Due to the significant temporal correlations present in Jacobian matrices, our predictor employs the temporal prediction model to effectively capture and address these temporal dependencies.

In time-series compression, the typical approach is to compute the predictive value using historical data points[10]. We expand this concept to the spatial dimension by employing a neighboring time point matrix to predict the current matrix $\boldsymbol{M}_t$, where $\hat{\boldsymbol{M}}_t = \boldsymbol{M}_{t+1}$. However, using multiple adjacent matrices to predict the current matrix is not feasible, as storing the historical information of several matrices would result in a substantial increase in memory overhead. Therefore, during the forward integration process, we store the Jacobian matrix at time t_n for use at time point t_{n+1}, relying solely on Jacobian matrices from adjacent time points for predictive modeling. Subsequent difference computation can eliminate the data redundancy introduced by linear elements.

Spatial Prediction Model. In the spatial dimension, we establish the prediction model, termed the "stamping-based prediction model", aligning with the Jacobian matrix construction methodolog, i.e. MNA, as illustrated in Figure 3(a). The Jacobian matrix is essentially an aggregate of the stamps from all the elements in the circuit [11].



(a)An example of matrix stamp for a resistor

(b) Prediction for the strictly upper triangular region

(c) Prediction for the strictly lower triangular region

(d) Prediction for the diagonal region

Figure 3: **Matrix stamp and proposed spatiotemporal prediction model.**

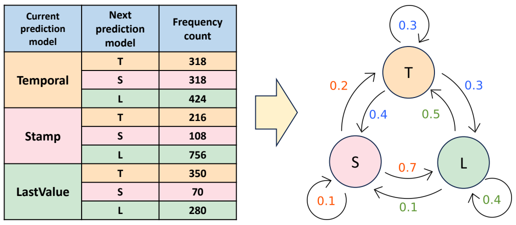
A typical matrix stamp can be characterized by the set $\{S(i,i), S(i,j), S(j,i), S(j,j)\}$. The four elements in this set are derived from the same element's contribution to the matrix, demonstrating a significant interdependence among them. For example, in linear resistors and capacitors, there exists a relationship such that $S(i,i) = S(j,j) = -S(i,j) = -S(j,i)$.

Similarly, we examine four non-zero elements at corresponding positions in the matrix, forming a pattern described by the set $\{V(i,i), V(i,j), V(j,i), V(j,j)\}$. One of the values in this set can be predicted using the other values. The design of our prediction model is structured as follows:

$$\begin{cases} \hat{V}_p(i,j) = eval(\hat{V}(j,i), -\hat{V}(i,i), -\hat{V}(j,j)) \\ \hat{V}_p(j,i) = eval(-\hat{V}(i,i), -\hat{V}(j,j)) \\ \hat{V}_p(j,j) = \hat{V}(i,i) \end{cases} \tag{6}$$

Due to the unique nature of the values in the matrix stamp of different elements, we do not employ the Lorenz predictor scheme, in contrast to the approaches used in NDZIP and FPZIP. Instead, we independently evaluate the similarity between the true value and the other values in the matrix stamp. We use an evaluation function to select the value that is closest to the true value as the prediction result for this model. Notably, the non-zero elements on the main diagonal of the Jacobian matrix in simulations often have sign bits opposite to those in other regions. To capitalize on this, we invert the sign bits of the elements on the main diagonal, leading to a higher compression ratio during the encoding phase. Figure 3(b)(c)(d) outlines the workflow of our predictor. A more detailed explanation will be provided in the next section.

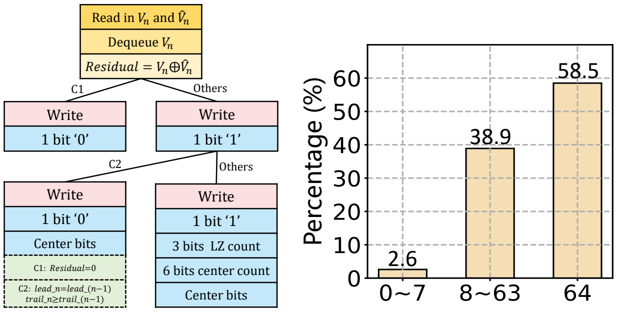
Markov predictor. To minimize the time overhead involved in selecting the best-fit prediction model, we innovatively employ Markov models. This approach is based on our observation of regularity in the selection of prediction models, due to strong spatiotemporal dependencies in the data. Specifically, we record the predictor's decision history during the best-fit phase in a historical table, estimate probabilities using frequency, and then utilize a Markov predictor to directly make predictions for the choice of prediction models (as shown in Figure 4). This design not only reduces the compression time overhead but also significantly lessens the extra space overhead incurred by recording the selection of prediction models, though there may be a slight decrease in prediction accuracy (see detail in Table 3).



Current prediction model	Next prediction model	Frequency count
Temporal	T	318
	S	318
	L	424
Stamp	T	216
	S	108
	L	756
LastValue	T	350
	S	70
	L	280

(a) Using the history table to record historical decisions **(b) Prediction model selection using the Markov chain**

Figure 4: **Matrix stamp and proposed spatiotemporal prediction model.**



(a) XOR operator and residual coding (b) Statistics on the number of leading zeros in the Jacobian matrix

Figure 5: **(a) Our encoding method and (b) Distribution of leading zeros.**

4.3 EFFICIENT RESIDUAL CODING

Figure 5 shows the encoding process for non-zero elements. The compressor reads the actual value from the front of the compression queue and the predicted value from the hybrid predictor, then performs an XOR operation between them to generate the residual. Subsequently, the encoder encodes these residuals. Since the actual and predicted values are usually very close, the residuals often contain numerous leading and trailing zeros. Hence, effectively encoding these zero bits can result in a significant compression ratio for the floating-point data.

We perform a statistical analysis of the residual characteristics across multiple datasets and developed an efficient residual encoding scheme, as shown in Figure 5(a). Figure 5(b) displays the distribution of leading zeros in the residuals, indicating that approximately 60% of the residuals comprise 64 consecutive zero bits, this also robustly validates the effectiveness of our spatiotemporal prediction model. Consequently, we utilize just 1 bit to represent this scenario. Given that fewer than 2% of the residuals have fewer than 8 leading zeros, we use 3 bits for encoding the number of leading zeros, treating it as 0 if the count of leading zero bits is between 0 and 7. Moreover, if the meaningful bits of the current residual are within the range of the previous residual's meaningful bits, the encoding of leading zeros can be shared. Once the encoding process is completed, these bits are concatenated and stored in a buffer.

Algorithm 1: Compressing a Jacobian Matrix $\boldsymbol{M_t}$.

```
Input: $M_t$ and $M_{t+1}$
Output: Compressed $M_t$
Partition $M_t$, obtain $U$, $L$, and $D$, initiate compression task for these three sequentially,
  involving $N$ floating-point values.
for ($V_i,\ i = 0, 1 \cdots N-1$) do
    if $sol = bestfit\_sol$ then
        $\hat{V}_i^{(T)} \leftarrow Temporal,\ \hat{V}_i^{(S)} \leftarrow MatrixStamp.$
        if $V_i \in L$ then
            $\hat{V}_i^{(L)} \leftarrow LastValue.$
        end if
        $sol = \arg\min_{P=\{(T),(S),(L)\}} (|\hat{V}_i^P - V_i|)$
        if $V_i \in D$ then
            Use 1 bit to record the bestfit prediction model.
        else
            Use 2 bits to record the bestfit prediction model.
        end if
    else if $sol = Markov\_sol$ then
        $\pi^* \leftarrow MarkovPredictor,\ sol = \pi^*.$
    end if
    $Res_i = \hat{V}_i^{sol} \oplus V_i$, encode the residual,write it into the buffer.
end for
```

Overview of our proposed compression method. Finally, our complete spatiotemporal compression method is presented in Algorithm 1. Specifically, we compress $\boldsymbol{M}_t$ at the $t+1$ time point during the transient simulation process, using $\boldsymbol{M}_{t+1}$ as the prediction matrix. Due to the strong symmetry of the Jacobian matrix, we divide the original matrix into three regions: the strictly upper triangular region, the strictly lower triangular region, and the diagonal region. The non-zero elements in these regions correspond to three sets U, L, and D, respectively. Following the matrix stamp-based prediction model, we sequentially compress the floating-point numbers in these sets. To minimize bit wastage in encoding prediction results in set L and enhance prediction accuracy, we employ a last value predictor [12]. This predictor selects the neighboring non-zero element in the same row as the matrix's true value for the predicted value. Ultimately, it chooses the prediction closest to the true value from multiple prediction models as the final prediction value.

5 ACCELERATING ADJOINT SENSITIVITY SIMULATION USING COMPRESSION

Integrating the SPICE simulation process with efficient data compression techniques substantially improves the efficiency of sensitivity analysis and adeptly tackles the issue of memory overhead. It is important to note that in scenarios with ample computational resources, the proposed design offers potential for parallel execution.

The complete simulation workflow is depicted in Algorithm 2. In the transient analysis, data compression and storage are implemented. Initially, shared indices are obtained during the calculation of the circuit's DC operating point. During transient simulation, the Jacobian matrix at t_n is stored to t_{n+1}, and compression of $\boldsymbol{M}_n$ occurs at t_{n+1}. This ensures that during the reverse integration process, data decompression is conducted in a reverse manner. Data decompression and memory release are performed during sensitivity analysis. Assuming the objective function is a function of the system's final state, we decompress $\boldsymbol{M}_{n-1}$ at t_n for subsequent adjoint sensitivity analysis at t_{n-1}. At this point, $\boldsymbol{M}_n$ can be released, thereby reducing memory overhead.

Algorithm 2: Proposed MASC sensitivity simulation.

```
Input: Input netlist
Output: Sensitivity analysis results $\frac{dO}{dx}$
Parse netlist and begin transient simulation.
while $t_n \leq t_f$ do
    if $t_n = t_0$ then
        Compute DC operating points; get shared indices; store $\boldsymbol{M}_0$.
    else
        Transient analysis at $t_n$; compress $\boldsymbol{M}_{n-1}$ using $\boldsymbol{M}_n$; store $\boldsymbol{M}_n$.
    end if
end while
End transient simulation and begin sensitivity simulation.
while $t_n \geq t_0$ do
    Adjoint sensitivity analysis at $t_n$.
    Decompress $\boldsymbol{M}_{n-1}$ using $\boldsymbol{M}_n$; free memory for $\boldsymbol{M}_n$.
end while
```

6 EXPERIMENTAL EVALUATION

In this section, we present our evaluation results. We start by describing our experimental setup. Next, we examine the compression ratio and time of our proposed compression method. We then demonstrate the effectiveness of our prediction models and conclude by showcasing the overall end-to-end performance of MASC in sensitivity analysis simulations with integrated compression.

Table 2: Detail of our test datasets.

Dataset	#CirElem	#Steps	S_{CSR}(GB)	S_{NZ}(GB)	CR (gzip)	T_{comp}(gzip)
add20	5091	42799	9.09	5.80	2.52	164.60s
smult20	46075	9048	22.99	14.37	5.85	211.17s
mem_plus	217431	13124	18.38	11.56	13.04	114.64s
MOS_T5	902631	8680	40.13	24.32	2.00	736.68s
MOS_T7	175487	25280	163.68	99.20	1.81	3076.71s
MOS_T8	494077	12740	177.48	111.62	1.97	3366.62s
MOS_T10	224398	20592	208.08	129.24	1.98	3827.76s

6.1 EXPERIMENTAL SETUP

Platform and software. Experiments are conducted on a Linux server with an AMD EPYC 7702 CPU at 2.0 GHz, 512 GB of RAM, and a Micron 5100 SSD, running Ubuntu 20.04. We implement our proposed compressor in C++ and develop its parallel version using OpenMP. It is available at [13]. We implement MASC based on Xyce [14], a high-performance, SPICE-compatible circuit simulator.

Datasets. We evaluate the performance of our compression on seven datasets, as shown in Table 2, all generated from actual simulations. Specifically, #CirElem and #Steps represent the number of circuit elements and discrete time points, respectively; S_{CSR} denotes the space required to store all Jacobian matrices in CSR format; and S_{NZ} indicates the space occupied by all non-zero elements in the matrices, which is the primary focus of our compression evaluation.

Baselines. We compare the performance of our proposed compressor with that of the general compressor GZIP , two advanced floating-point compressors (FPZIP [7] and NDZIP [8]), and the lossy compressor SpiceMate [15] from the EDA domain.

6.2 COMPRESSION PERFORMANCE

First, we evaluate the performance of our proposed compression method and compare it with the aforementioned baselines. Table 3 demonstrates that our spatiotemporal compression significantly surpasses other compression methods, achieving a compression ratio 4.95× higher than SpiceMate and 2.24× higher than FPZIP. This superior performance can be attributed to the effective exploitation of the spatiotemporal characteristics of sparse tensors. In terms of compression time, the average compression time of our spatiotemporal compression method is about 25% of that of GZIP, although it is slower than FPZIP. This slower speed is due to the matrix partitioning step in our spatiotemporal compression method. Despite this, the compression efficiency is remarkably high. Overall, the performance of our spatiotemporal compression method satisfactorily meets the demands of simulation processes.

6.3 EFFECTIVENESS OF PREDICTION MODELS

We then evaluate the effectiveness of three prediction models, as illustrated in Figure 6. Specifically, the temporal prediction model was selected with a frequency exceeding 60% in certain datasets, indicating that temporal dependencies are more significant than spatial dependencies in these datasets. In datasets exhibiting regularity in the spatial dimension, the matrix stamping-based prediction model was chosen with a frequency close to 60%, confirming the effectiveness of this model. Concurrently, the last value prediction, introduced to improve prediction accuracy, was selected with a frequency of around 10%, validating its necessity in the model.

6.4 END-TO-END PERFORMANCE OF MASC

Finally, we assess the end-to-end performance of our proposed simulator, MASC, and compare it with Xyce [14]. As Figure 7 illustrates,

Table 3: Comparison of compression ratio, compression time (in seconds), and decompression time (in seconds) for different compression methods.

Dataset	FPZIP [7]			NDZIP [8]			SpiceMate [15]			MASC w/o Markov			MASC w/ Markov		
	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}
add20	4.92	34.02	39.18	1.05	179.33	176.31	2.24	113.07	60.79	**11.90**	44.54	29.93	10.59	40.09	**27.24**
smult20	4.66	86.97	95.85	1.04	215.22	211.95	5.01	127.17	67.56	**11.07**	105.96	74.46	10.04	87.83	**70.74**
mem_plus	5.31	57.94	66.24	1.01	168.35	163.47	8.68	86.72	56.48	**11.51**	83.70	61.54	10.27	65.40	**59.52**
MOS_T5	14.53	86.16	107.92	1.10	341.78	342.76	1.96	422.36	91.60	**35.32**	161.40	142.80	32.32	124.29	135.16
MOS_T7	9.78	407.06	484.42	1.14	1454.23	1443.15	1.88	1740.08	444.01	**20.65**	739.97	594.45	19.62	643.77	563.20
MOS_T8	4.80	646.84	726.27	1.06	1603.85	1505.99	1.97	1998.04	574.19	**9.91**	880.75	676.13	9.02	702.21	631.28
MOS_T10	8.46	542.52	624.24	1.12	1845.00	1848.24	2.00	2103.76	497.17	**17.19**	967.68	820.08	16.16	872.85	796.54
Average	7.49	265.93	306.30	1.07	829.68	826.64	3.39	941.60	255.97	**16.79**	426.28	342.77	15.43	362.34	326.24

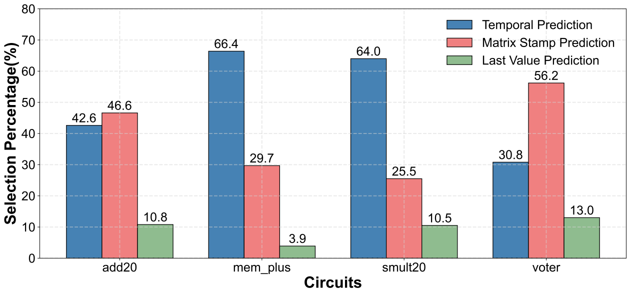


Figure 6: **Statistics analysis of selection rate for three prediction models.**

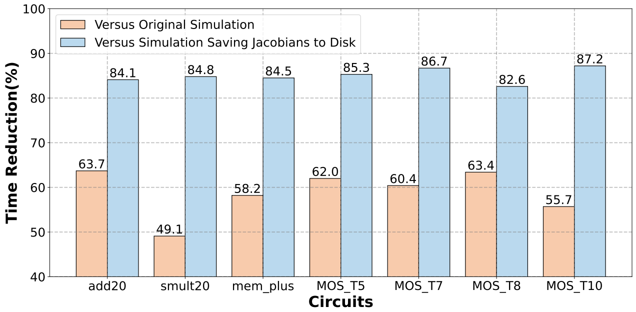


Figure 7: **End-to-end simulation time reduction of MASC compared to baselines.**

MASC achieves approximately a 50% reduction in time overhead for sensitivity analysis compared to the original simulation Xyce. This improvement is primarily due to MASC’s in-memory compression scheme, which efficiently circumvents the significant time overhead involved in repeatedly computing the Jacobian matrix during the reverse process. When comparing MASC to simulations that directly save Jacobian matrices to disk rather than using in-memory compression, MASC demonstrates 6× end-to-end performance improvement. This notable performance boost is attributed to two factors: (1) our compression method leads to over an 80% reduction in the overhead associated with storing Jacobian matrices on disk, and (2) the speed of our parallel compressor is considerably high (~2.3 GB/s), far surpassing the SSD’s bandwidth (~0.5 GB/s), which ensures minimal additional time overhead in simulations. Note that the speed of our parallel (de)compression increases rapidly with the number of threads, peaking at around 16 threads. At this point, the parallel speed is more than 8× of the serial speed.

7 CONCLUSION

In this paper, we seamlessly integrate data compression into simulation processes, utilizing Jacobian matrices stored during forward integration to expedite the reverse adjoint sensitivity solving process. Concurrently, we develop a novel lossless compression method that significantly reduces memory overhead during the simulation. The time overhead of our MASC is approximately half of that with Xyce. Furthermore, our compression method exhibits remarkable enhancements compared to general compression methods and state-of-the-art floating-point compression methods, effectively minimizing memory overhead during the simulation process.

ACKNOWLEDGMENTS

Corresponding authors: Zhou Jin and Dingwen Tao. This work was supported by the NSFC under Grant No. 62204265, 62234010, the National Key R&D Program of China under Grant No. 2022YFB4400400, and U.S. NSF under Grant No. 2312673, 2311876, 2247080, and 2303064.

REFERENCES

[1] B. Agrawal, F. Liu, and S. Nassif. Circuit optimization using scale based sensitivities. In *IEEE Custom Integrated Circuits Conference*, 2006.

[2] I. Stevanovic and C. McAndrew. Quadratic backward propagation of variance for nonlinear statistical circuit modeling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[3] C.J. Gu and J. Roychowdhury. An efficient, fully nonlinear, variability-aware non-monte-carlo yield estimation procedure with applications to sram cells and ring oscillators. In *Asia and South Pacific Design Automation Conference*, 2008.

[4] N. Sagan and J. Roychowdhury. Transient adjoint dae sensitivities: a complete, rigorous, and numerically accurate formulation. In *27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022.

[5] W.F. Hu, Z.C. Ye, and Y. Wang. Adjoint transient sensitivity analysis for objective functions associated to many time points. In *57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

[6] J. Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

[7] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE transactions on visualization and computer graphics*, 2006.

[8] F. Knorr, P. Thoman, and T. Fahringer. ndzip: A high-throughput parallel lossless compressor for scientific data. In *Data Compression Conference (DCC)*. IEEE, 2021.

[9] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics*, 2014.

[10] P.Liakos, P.Katia, and Y.Kotidis. Chimp: efficient lossless floating point compression for time series databases. *Proceedings of the VLDB Endowment*, 2022.

[11] J. Roychowdhury. *Numerical Simulation and Modelling of Electronic and Biochemical Systems*. 2009.

[12] Y. Sazeides and J.E. Smith. The predictability of data values. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, 1997.

[13] https://github.com/tensorzip/Spatiotemporal-Compression.

[14] J. Verley, E.R. Keiter, and H.K. Thornquist. Xyce: Open source simulation for large-scale circuits. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

[15] L.J. Li and W.J. Yu. Efficient and accuracy-ensured waveform compression for transient circuit simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.