Efficient and Accuracy-Ensured Waveform Compression for Transient Circuit Simulation

Lingjie Li and Wenjian Yu, *Senior Member, IEEE*

***Abstract*—Efficient and accurate waveform compression is essentially important for the application of analog circuit transition simulation nowadays. In this article, an analog waveform compression scheme is proposed which includes two compressed formats for representing small and large signal values, respectively. The compressed formats and the corresponding compression/decompression approaches ensure that both the absolute and relative errors of each signal value restored from the compression are within specified criteria. The formats are integrated in a block-by-block compressing procedure which facilitates a secondary lossless compression and a three-stage pipeline scheme for fast conversion from the simulator's output to the compressed hard-disk file. Theoretical analysis is presented to prove the accuracy-ensured property of our approach. Two schemes are also proposed to incorporate the prediction method. They achieve larger compression ratio for some cases while preserving the accuracy and runtime efficiency. Experiments are carried out with voltage waveforms from industrial circuits. The results validate the accuracy and the efficiency of the proposed techniques. The obtained compression ratio is 2.6X larger than existing work without overhead, even though the latter induces much larger error. Compared with the original double-precision floating number format, the proposed approach achieves the compression ratio of 26 averagely and up to 70, while keeping the relative error less than 10^{-3} and absolute error less than 10^{-5}. And with the pipelined computation, the proposed compression approach hardly increases time cost to the transient simulation.**

***Index Terms*—Data compression, floating-point number, quantization, the prediction method, transient simulation waveform.**

I. Introduction

TRANSIENT circuit simulation has become one of the most important steps in the design of analog or mixed-signal integrated circuits. Its output is voltage waveforms for nodes and/or current waveforms along branches in the circuit. Each waveform consists of voltage/current signal values at discrete time points [1]. The waveforms are subsequently displayed or used for validating performance metrics of the design. With the increase of circuit size, the number of waveforms in the result of transient simulation increases, leading to large data storage for the waveforms. For accurate transient simulation, the number of time points ranges from several hundred thousands to even a billion, and voltage/current values are stored as floating-point numbers. Therefore, the simulator may output a large hard-disk file occupying multiple GBs of disk space. This further leads to considerable time for writing the waveforms into a file and for loading the file into a viewer, and largely worsens the performance of circuit simulator and the customer experience of waveform viewer. Hence, it is greatly demanded to have an efficient waveform compression approach which reduces the data storage of waveforms and ensures an acceptable accuracy as well [2].

Some work on waveform compression or related problems have been reported in the literature. In [3], a lossless adaptive prediction approach using a recursive least squares lattice with arithmetic coding was proposed. However, the compression ratio is unsatisfactory. In [4], an approach was proposed for digital waveform compression, instead of the analog transient waveforms considered in this work. Hatami *et al.* [5], [6] proposed to compress the waveforms by performing principal component analysis (PCA) on signal values. It however means a lot of computation and large memory usage [7]. Later on, Liu *et al.* [8] proposed a stream compression method with waveform prediction and quantization. It reduces the storage of original waveforms in double-precision floating-point numbers by 10×, with about 0.1% error. Nonetheless, it cannot guarantee any accuracy criterion. In [9] and [10], an effective lossy compression technique for high-performance computing (HPC) data is proposed based on multilayer prediction and error-bounded quantization. Yet, in HPC applications, such as climate simulation and hurricane simulation, the data for compression exhibit strong correlation in local regions of a snapshot. The effectiveness of the compressor highly relies on the prediction based on the correlation, which is however not suitable for transient circuit simulation. Moreover, the error control in [9] and [10] does not consider the relative error. Recently, Saurabh and Mittal [11] used a recursive polynomial representation to compress technology libraries. Notice that the problem in [11] allows as large as 5% relative error, which is quite different from the problem considered in this work. Besides, their approach is rather simple which could not produce large compression ratio.

There are commercial tools that support dumping compressed waveform formats, like FSDB [12]. Unfortunately, their formats and algorithms are not disclosed. Some common lossless compression algorithms, such as Huffman coding, Deflate [13], and LZW [14], and some lossless floating-point

Manuscript received May 20, 2020; revised July 28, 2020; accepted August 24, 2020. Date of publication August 31, 2020; date of current version June 18, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61872206; in part by the Beijing National Research Center for Information Science and Technology under Grant BNR2019ZS01001; and in part by the Tsinghua University Initiative Scientific Research Program. This article was recommended by Associate Editor N. Wong. *(Corresponding author: Wenjian Yu.)*

The authors are with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China (e-mail: li-lj18@mails.tsinghua.edu.cn; yu-wj@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TCAD.2020.3020496

compressor [15], [16] may also be used to reduce the data storage of waveforms. However, these algorithms do not consider the correlation of waveform data and thus cannot provide sufficient compression.

In this work, we develop an efficient and accuracy-ensured waveform compression scheme as a companion of transient circuit simulator. It includes two compressed formats for representing small and large signal values, respectively, and the approaches for compressing/decompressing the waveforms. By distinguishing between the small value and large value with a calculated threshold, we can ensure that both the absolute and relative errors of each compressed signal value are within specified criteria. The proposed approach exploits the fact that the signal values at successive time points vary less, and incorporates the lossless compression technique, so as to deliver large compression ratio. It also enables a three-stage pipeline scheme for fast conversion from the simulator's output to the compressed file on the hard disk. Besides, we can combine the proposed scheme with the prediction method, leading to larger compression ratio for some cases. Two combination schemes are proposed, which perform differently for different cases. Experimental results with voltage waveforms from analog circuit simulation have validated the accuracy and the efficiency of the proposed techniques. They demonstrate remarkable advantages over the existing methods, like [8]. The impact of error tolerance on the compression ratio and the convenience of decompressing partial waveforms from the compressed file are also presented.

The main contributions of this article are as follows.

1) A signal-value compression scheme, which discriminates between small value and large value and represents them with two variable-length compressed formats, respectively, is proposed. The scheme is able to efficiently and largely compress sequential signal values, and theoretically ensures the absolute and relative errors of each compressed signal value to be within specified error tolerances.
2) A waveform compression approach incorporating the signal-value compression scheme and the lossless compression technique is proposed, which enables an efficient three-stage pipeline scheme for generating the compressed data file. It does not increase the error to the compressed signal value. Accordingly, a waveform decompression approach is presented which recovers signal values from the compressed file without accuracy loss.
3) Two schemes combining the proposed waveform compression approach with the prediction method are proposed. The combined schemes preserve the accuracy and the efficiency for compressing/decompressing, and result in larger compression ratio in the scenarios without and with the lossless secondary compression.
4) With seven industrial data of waveforms, the proposed compression approach demonstrates the compression ratio up to 67 and averagely 26, while meeting 10^{-5} absolute-error tolerance and 10^{-3} relative-error tolerance. On average, it is 2.6× larger than the compression ratio achieved with the method in [8], even though the latter shows much larger error. The proposed compression/decompression algorithm consumes equal or less time than existing methods. And with the pipelined parallel computing, our approach hardly adds time cost to the transient simulation. Finally, combined with prediction our approaches can achieve the compression ratio up to 70.4 with the secondary lossless compression and 38.2 without the lossless compression.

The remainder of this article is organized as follows. The problem formulation and related work are presented in Section II. Section III includes the proposed storage formats and the compression/decompression algorithms. The combination with the prediction method is discussed in Section IV, and the experimental results are demonstrated in Section V. Finally, conclusions are drawn in Section VI.

II. Background

A. Problem Formulation

The waveforms produced during the transient circuit simulation can be represented as a vector-valued function

$$\boldsymbol{v}(t) = \begin{bmatrix} v_1(t) & v_2(t) & \cdots & v_{N_s}(t) \end{bmatrix}^T \tag{1}$$

where t is the simulation time, N_s is the number of simulated signals, and $v_i(t)$ means the value of the ith signal at time t. We do not store the waveform continuously in practice. Instead, a set of N_t discrete time-points $\boldsymbol{t} = \begin{bmatrix} t_1 & t_2 & \cdots & t_{N_t} \end{bmatrix}$ is stored along with their corresponding signal values $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}(t_1) & \boldsymbol{v}(t_2) & \cdots & \boldsymbol{v}(t_{N_t}) \end{bmatrix}$. The waveform can thus be represent as a matrix

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{t} \\ \boldsymbol{V} \end{bmatrix} = \begin{bmatrix} t_1 & t_2 & \cdots & t_{N_t} \\ v_{11} & v_{12} & \cdots & v_{1N_t} \\ v_{21} & v_{22} & \cdots & v_{2N_t} \\ \vdots & \vdots & \ddots & \vdots \\ v_{N_s1} & v_{N_s2} & \cdots & v_{N_sN_t} \end{bmatrix}. \tag{2}$$

Typically, all elements of matrix $\boldsymbol{W}$ are stored in *IEEE* 754 double-precision floating-point format which encodes each value with 64 bits. Hence, it takes $64(N_s + 1)N_t$ bits storage to store the raw waveform.

Waveform compression can be performed during simulation. The compression method handles streaming data outputted from the simulation and should run fast enough not to encumber the simulation. The compressed waveforms are stored as a file on hard disk. The compression ratio is calculated as

$$\text{compression ratio} = \frac{\text{storage of the raw waveforms}}{\text{size of the compressed file}}. \tag{3}$$

When debugging (or in the occasion that requires the waveform), decompression is performed to restore the waveform $\widehat{\boldsymbol{W}}$ from the compression file. Sometimes, only a fraction of the signals are needed, and therefore the decompression method should be able to restore specified rows of $\boldsymbol{V}$. The decompression should run fast enough compared to directly reading from raw waveform format.

Compression and decompression of waveform may induce error. Otherwise, it is a lossless compression. For any signal

value v and its corresponding restored value $\hat{v}$, we usually consider the absolute error ϵ and the relative error η

$$\epsilon = \left|v - \hat{v}\right| \tag{4}$$

$$\eta = \frac{\epsilon}{|v|} = \left|\frac{v - \hat{v}}{v}\right|. \tag{5}$$

Under normal circumstances, we do not allow any error for time-points $\boldsymbol{t}$, but we do tolerate a bit of error for signal values. More precisely, maximum tolerated absolute error ε_{abs} and relative error ε_{rel} may be specified before compressing. And it is desirable that the compression/decompression algorithm ensure that $\epsilon \leq \varepsilon_{\text{abs}}$ and $\eta \leq \varepsilon_{\text{rel}}$ for each restored signal value. For example, during transient circuit simulation of voltage signals, a common setting of the maximum tolerated errors is $\varepsilon_{\text{abs}} = 10^{-5}$ and $\varepsilon_{\text{rel}} = 10^{-3}$.

B. Related Work

The method in [8] can achieve higher compression ratio compared with other existing work. The experiments in [8] show that it runs much faster than the method in [4] while achieving better accuracy and larger compression ratio. And, for some larger circuits, the method in [4] could not perform the compression due to its excessive memory usage.

The main techniques used in [8] are prediction and quantization. Signal values $\boldsymbol{v}(t_i)$ for time point t_i are regarded as a frame, here denoted by $\boldsymbol{v}_i$. Some frames (including the first frame) are called reference frames to which no compression is applied. If $\boldsymbol{v}_i$ is a reference frame, $\boldsymbol{v}_r$ is picked as the next reference frame which satisfies $r - i \geq \alpha$ or $\|\boldsymbol{v}_r - \boldsymbol{v}_i\|/\|\boldsymbol{v}_i\| > \beta$. α and β are two preset parameters.

In [8], the combination of linear prediction and quantization is used to compress nonreference frames. Suppose $\boldsymbol{v}_j$ is a nonreference frame after reference frame $\boldsymbol{v}_i$. With the linear prediction approach, it is approximated as

$$\boldsymbol{v}_j \approx \hat{\boldsymbol{v}}_j = \boldsymbol{v}_i + \boldsymbol{h}_i\big(t_j - t_i\big) \tag{6}$$

where $\boldsymbol{h}_i = (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)/(t_{i+1} - t_i)$. For each signal value in the frame $\boldsymbol{v}_j$, if the predicted value is accurate enough which is called success of prediction, there is no need to store the signal value. Otherwise, a quantization approach [17] should be used to compress the value.

For performing the quantization, the difference of current frame and the reference frame is first calculated in [8]

$$\boldsymbol{d} = \boldsymbol{v}_j - \boldsymbol{v}_i. \tag{7}$$

$\boldsymbol{d}$ is then rescaled to range $[0, 1]$ by min–max normalization

$$\tilde{d}_k = \frac{d_k - \min(\boldsymbol{d})}{\max(\boldsymbol{d}) - \min(\boldsymbol{d})}, \quad k = 1, 2, \ldots, N_s. \tag{8}$$

After normalization, $\tilde{d}_k$ (originally a 64-bit floating-point number) is uniformly quantized to 8-bit or 16-bit integers.

Consequently, for each nonreference frame, the indices of signals that have failed in prediction are stored, along with the integers representing the quantized signal values. If the ratio of the number of signals failed in prediction to N_s exceeds a threshold, all signals will be stored as quantized integers, therefore omitting the storage of signal indices.

The aforementioned quantization is the uniform quantization which usually results in a large relative error for small values. In order to reduce the relative error, a nonuniform quantization method can be performed to make the quantization resolution smaller for small values and larger for large values. For this aim, a companding function, such as A-law or μ-law algorithm [18], is applied on $\tilde{d}_k$ before the uniform quantization is performed in [8]. For the compressed frames, the lossless compression method (like *zlib* [19]) is further applied to improve the compression ratio.

The experiments in [8] demonstrate that, compared with the prediction, either the quantization, or the lossless compression contributes more to the final compression ratio. Although the results in [8] show the maximum relative error of compressed signal value in tested cases is 0.13%, no error bound can be guaranteed with the method in [8], due to the employed quantization technique.

III. Accuracy-Ensured Waveform Compression Scheme and Corresponding Algorithms

In this section, we first introduce the idea of ensuring both the absolute error and relative error criteria of compression by discriminating between small value and large value. Then, two compressed formats are proposed for small and large signal values, respectively, followed by description of the algorithms for compressing and decompressing a signal value. Finally, we present the scheme for further compression with a data block structure and the lossless compression technique. The overall compressing and decompressing algorithms are described, along with the discussion of gaining acceleration through parallel computing.

A. Basic Idea

To ensure the accuracy of waveform compression, it is desirable that both the absolute error ϵ and relative error η of every signal value satisfy preset criteria

$$\epsilon \leq \varepsilon_{\text{abs}}, \text{ and } \eta \leq \varepsilon_{\text{rel}}. \tag{9}$$

For a larger signal value v, the requirement for absolute error is stricter, while for a smaller v the requirement for relative error is more crucial. Specifically, we find out that [according to (4) and (5)]

$$\begin{cases} \epsilon \leq \varepsilon_{\text{abs}}, & \text{if } |v| \leq \varepsilon_{\text{abs}}/\varepsilon_{\text{rel}} \text{ and } \eta \leq \varepsilon_{\text{rel}} \\ \eta \leq \varepsilon_{\text{rel}}, & \text{if } |v| > \varepsilon_{\text{abs}}/\varepsilon_{\text{rel}} \text{ and } \epsilon \leq \varepsilon_{\text{abs}}. \end{cases} \tag{10}$$

This means we can define a threshold value

$$\tau = \varepsilon_{\text{abs}}/\varepsilon_{\text{rel}}. \tag{11}$$

For signal values satisfying $|v| \leq \tau$ which we call *small values*, we can only care their relative errors η. If the compressed format ensures $\eta \leq \varepsilon_{\text{rel}}$, (9) is satisfied as well. In contrast, for the others (with $|v| > \tau$) which we call *large values*, only the absolute error ϵ is worth of consideration. Based on this observation, we present two different compressed formats for the small-value and large-value signal values, respectively, in the following two sections. They guarantee the relative error or absolute error criterion and achieve as much as possible reduction of storage space.

B. Compressed Format for Small Values

As we want to ensure the relative error criterion for the small values with magnitude not larger than τ, we first consider a customized floating-point number format. The representation of floating-point number (as those in *IEEE* 754) consists of a sign bit S, an exponent E (e bits) and an mantissa M (m bits) [20]. Under normalization, it represents the number

$$(-1)^S\left(1+M\cdot 2^{-m}\right)\cdot 2^{E-b} \tag{12}$$

where b is an exponent bias. This format is determined by three parameters e, m, and b, and its length is $e+m+1$ bits. The floating-point arithmetic ensures the following statement on the relative error [21].

Lemma 1: For any nonzero real number x, the maximum relative error in representing it in the normalized floating-point number system is given by

$$\left|\frac{\hat{x}-x}{x}\right| \le \varepsilon_{\text{mach}} \tag{13}$$

where $\hat{x}$ is the floating-point number of x. The machine epsilon

$$\varepsilon_{\text{mach}} = 2^{-(m+1)} \tag{14}$$

with m being the bit-length of mantissa.

If a small value v is represented as this kind of normalized floating-point number, the following expressions of S, E, and M are derived:

$$S = \begin{cases} 0, & \text{if } v \ge 0 \\ 1, & \text{if } v < 0 \end{cases} \tag{15}$$

$$E = \left\lfloor \log_2 |v| \right\rfloor + b \tag{16}$$

$$M = \text{ROUND}\left(\left(|v|\cdot 2^{b-E}-1\right)\cdot 2^m\right) \tag{17}$$

where $\text{ROUND}(x) = \lfloor x+0.5 \rfloor$. It corresponds to a restored (decompressed) value $\hat{v}$, which can be calculated with formula (12). The relative error of $\hat{v}$ satisfies

$$\eta \le \varepsilon_{\text{mach}} = 2^{-(m+1)} \tag{18}$$

according to Lemma 1.

Same as the UFL and OFL in floating-point arithmetic [21], the minimum and maximum of $\hat{v}$'s absolute value are

$$\left|\hat{v}\right|_{\min} = 2^{-b+1} \tag{19}$$

$$\left|\hat{v}\right|_{\max} = \left(1-2^{-m-1}\right)\cdot 2^{2^e-b}. \tag{20}$$

Notice that $E=0$ is set only when the value is 0. To ensure $\varepsilon_{\text{mach}} \le \varepsilon_{\text{rel}}$ and $|\hat{v}|_{\max} \ge \tau$, we derive the following guidelines for choosing the values of e, m, and b:

$$\begin{cases} m \ge -\log_2 \varepsilon_{\text{rel}} - 1 \\ 2^e - b > \log_2(\varepsilon_{\text{abs}}/\varepsilon_{\text{rel}}). \end{cases} \tag{21}$$

Besides, the value of $|\hat{v}|_{\min}$ is also of concern, which should be not larger than the smallest possible signal value above 0.

This customized floating-point format uses $e+m+2$ bits to store a small value, as a leading bit 0 is needed to indicate this is a small value. However, it does not take advantage of the characteristic of signal waveform to save storage. The value of a signal usually does not change much at adjacent time points, which means v at time t_i only differs a little from the value v' of the same signal at time t_{i-1}. There is a large probability that the exponent E of v and E' of v' are equal or very close to each other. Under this situation, we can use a 2-bit field Δ_E to replace the e-bit integer E. Based on this, we propose to add a Δ_E field to the format whose value is defined as

$$\Delta_E = \begin{cases} 0, & \text{if } E = E'-1 \\ 1, & \text{if } E = E' \\ 2, & \text{if } E = E'+1 \\ 3, & \text{otherwise.} \end{cases} \tag{22}$$

We thus derive a variable-length format for small value as shown in Fig. 1. E is only stored when $\Delta_E = 3$, otherwise the storage is reduced. By the way, a special case is $v=0$, for which M can be omitted as we only need E to differ a zero value.

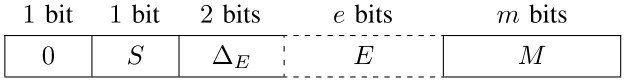

Fig. 1. Variable-length compressed format for a small value. E is omitted when $\Delta_E \ne 3$, and M is omitted when the value is 0.

With this variable-length compressed format, only $m+4$ bits are needed to store a value of small-value signal as long as it does not differ much from its value at the previous time point. For consecutive 0 values, the required bits per value is even reduced to 4 bits.

Example 1: We consider a case of waveform compression where the accuracy criteria are $\varepsilon_{\text{abs}} = 10^{-5}$ and $\varepsilon_{\text{rel}} = 10^{-3}$. The threshold is $\tau = \varepsilon_{\text{rel}}/\varepsilon_{\text{abs}} = 0.01$. To compress the small-value signal in this case, we can choose $e=7$, $m=9$, and $b=128$. They satisfy (21) and result in

$$\left|\hat{v}\right|_{\min} = 2^{-127} \approx 6\times 10^{-39} \tag{23}$$

which is small enough to meet the coverage range requirement. With this parameter setting, the compression ratio for small-value signals would be close to $64/(m+4) \approx 5$, or larger due to consecutive zeros.

C. Compressed Format for Large Values

For the large values, we want to meet the absolute error criterion. Hence, we design a format based on uniform quantization. The quantization resolution is denoted by c. Each large value v is divided by c and then rounded to an integer. In this way, the absolute error of resulted value $\hat{v}$ satisfies

$$\epsilon \le c/2. \tag{24}$$

With this approach, the representation of a large value consists of a sign bit S and an unsigned integer U (u bits).

$$S = \begin{cases} 0, & \text{if } v > 0 \\ 1, & \text{if } v < 0 \end{cases} \tag{25}$$

$$U = \text{ROUND}(|v|/c). \tag{26}$$

And, the restored value is

$$\hat{v} = (-1)^S U c. \tag{27}$$

This storage format is determined by parameters c and u, whose length is $u+2$ bits (with a leading bit 1 denoting a

"large value"). The minimum and maximum of $\hat{v}$'s absolute value are

$$\left|\hat{v}\right|_{\min} = c \tag{28}$$

$$\left|\hat{v}\right|_{\max} = \left(2^u - 1\right)c \tag{29}$$

according to (27).

Similar to the small-value format, we need to guarantee $c/2 \le \varepsilon_{\text{abs}}$ and $|\hat{v}|_{\min} \le \tau$. This derives a guideline for choosing c

$$c \le 2\varepsilon_{\text{abs}} \tag{30}$$

because τ is much larger than ε_{abs} in practice. The other important constraint is that $|\hat{v}|_{\max}$ should be not smaller than the largest possible signal value. This may cause a large value of u, and thus large storage cost.

To improve the above uniform quantization approach, we propose not using a fixed u. Although this induces an additional field storing the value of u, allowing variable-length storage fields and only recording the difference of consecutive signal values can reduce the storage for most scenarios.

Suppose we use l bits to represent u. The maximum of $\hat{v}$ becomes

$$\left|\hat{v}\right|_{\max} = \left(2^{2^l - 1} - 1\right)c. \tag{31}$$

If $c = 2 \times 10^{-5}$ and $l = 7$, this upper bound is as large as $|\hat{v}|_{\max} \approx 3 \times 10^{33}$. However, the length of this format increases to $l+u+2$. We propose two strategies to reduce the bit-length. First, a signal value v at time t_i usually only differs a little from the value v' at time t_{i-1}. If we store their difference instead of v, U will be much smaller. In practice, we store

$$d = v - \hat{v}' \tag{32}$$

where $\hat{v}'$ is the compressed value of v' (last large value). Hence, the absolute error of d equal to the absolute error of v. Second, there is a large probability that U for d and U' for d' can be stored in a similar number of bits. d' denotes the signal value difference at time t_{i-1}, and U' is the corresponding integer in its quantization representation. This means u for d and u' for d' differ little from each other in most occasions. Therefore, like the treatment for the small values, we can use a 2-bit field Δ_u to describe most changes of u to omit the field for storing u.

The final compressed format for a large value is shown as Fig. 2. At the first time point where the signal value is a large value, we store the signal value with $l + u + 4$ bits. Initially

$$u = \lceil \log_2 U \rceil. \tag{33}$$

At other time points, the difference d of adjacent large values is stored in a variable-length format. The values of Δ_u and u depend on the value of U and u' at the previous time point. We use the following rules to update them:

$$\begin{cases} \Delta_u = 0, \ u = u' - 1, & \text{if } \lceil \log_2 U \rceil < u' \\ \Delta_u = 1, \ u = u', & \text{if } \lceil \log_2 U \rceil = u' \\ \Delta_u = 2, \ u = u' + 3, & \text{if } u' < \lceil \log_2 U \rceil \le u' + 3 \\ \Delta_u = 3, \ u = \lceil \log_2 U \rceil, & \text{if } \lceil \log_2 U \rceil > u' + 3. \end{cases} \tag{34}$$

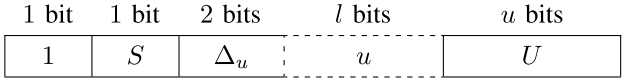

Fig. 2. Variable-length compressed format for a large value. u is omitted when $\Delta_u \neq 3$.

$\lceil \log_2 U \rceil$ is the minimum number of bits to represent U. For the first three situations in (34), the field for u can be omitted in the format because u's value is determined. Only for the last situation which means u is increased for 4 or more, u is stored in the format. Since the last situation rarely happens, a large value is usually stored in $u + 4$ bits, and u is a variable due to the first and third rules in (34).

Example 2: We consider the same accuracy requirement as in Example 1: $\varepsilon_{\text{abs}} = 10^{-5}$ and $\varepsilon_{\text{rel}} = 10^{-3}$. The threshold is still $\tau = 0.01$. To compress the large-value signal, we can choose $c = 2 \times 10^{-5}$, $l = 7$. Obviously, (24) or (30) is satisfied. And based on (31), the maximum value that can be represented is

$$\left|\hat{v}\right|_{\max} = \left(2^{127} - 1\right) \times 2 \times 10^{-5} \approx 3 \times 10^{33} \tag{35}$$

which is sufficiently large. With this parameter setting, if on average $u = 7$, i.e., $|d|$ is not larger than $(2^7 - 1)c \approx 2.5 \times 10^{-3}$, the compression ratio would be about $64/(7 + 4) \approx 5.8$.

The accuracy of the compressed format for large value is stated as Lemma 2.

Lemma 2: If a large value v is stored in the proposed format [see Fig. 2 and formulas (25), (26), and (32)–(34)], the absolute error of restored signal value $\hat{v}$ satisfies

$$\epsilon = \left|\hat{v} - v\right| \le \varepsilon_{\text{abs}} \tag{36}$$

providing that quantization resolution $c \le 2\varepsilon_{\text{abs}}$.

Proof: Based on (32), d's absolute error is the same as that of v, since $\hat{v}'$ is a value stored without approximation. Consequently, whether v or d is stored, the absolute error follows the rule of uniform quantization described as (24). Then, if $c \le 2\varepsilon_{\text{abs}}$, we have the absolute error of v satisfies (36). ■

D. Algorithm Description and Discussion

Algorithm 1 shows the procedure for compressing a single signal value. With it v is transformed to a bit-string b_t. E', $\hat{v}'$, and u' for last small value or large value are like state variables, and useful for compressing a sequence of signal values. At the starting time point, E' is initialized with an invalid value so that Δ_E becomes 3. The initial value of $\hat{v}'$ is 0 while u' can be initialized as -4 making $\Delta_u = 3$.

Algorithm 2 shows the procedure for restoring a signal value $\hat{v}$ from a bit-string b_t. It is the reverse of Algorithm 1. The restored large value $\hat{v}$ in line 27 is the difference of two successive large values. The initial value of s and $\hat{v}'$ are 0, while E' and u' can be initialized with any value. S, Δ_E, E, M, Δ_u, u, and U in the two algorithms are all unsigned integer. Hence, converting them to binary code and the reverse process are straightforward. Notice that the parameters τ, e, m, b, c, l are fixed, determined by preset accuracy criteria and the lower and upper bounds of nonzero signal value. Therefore, we do not list them as the input of the two algorithms.

Algorithm 1 Compressing a Signal Value

Input: Signal value v, exponent E' of last small value, the compressed value $\hat{v}'$ of last large value, the bit-length u' of U for last large value

Output: Compressed bit-string $\mathrm{b_t}$, updated E', $\hat{v}'$, and u'

if $|v| \le \tau$ (meaning this is a small value) **then**
  if $v = 0$ **then**
    $E \coloneqq 0$, $S \coloneqq 0$
    Determine Δ_E with (22)
    if $\Delta_E = 3$ **then**
      Concatenate the binary codes of 0, S, Δ_E and E to form $\mathrm{b_t}$, according to Fig. 1
    else
      Concatenate the binary codes of 0, S and Δ_E to form $\mathrm{b_t}$, according to Fig. 1
    end if
  else
    Compute S, E and M with (15) to (17)
    Determine Δ_E with (22)
    if $\Delta_E = 3$ **then**
      Concatenate the binary codes of 0, S, Δ_E, E and M to form $\mathrm{b_t}$, according to Fig. 1
    else
      Concatenate the binary codes of 0, S, Δ_E and M to form $\mathrm{b_t}$, according to Fig. 1
    end if
  end if
  $E' \coloneqq E$
else
  $v \coloneqq v - \hat{v}'$
  Compute S and U with (25) and (26), respectively
  Determine Δ_u and u with (34)
  if $\Delta_u = 3$ **then**
    Concatenate the binary codes of 1, S, Δ_u, u and U to form $\mathrm{b_t}$, according to Fig. 2
  else
    Concatenate the binary codes of 1, S, Δ_u and U to form $\mathrm{b_t}$, according to Fig. 2
  end if
  Compute $\hat{v}$ with (27)
  $\hat{v}' \coloneqq \hat{v}' + \hat{v}$, $u' \coloneqq u$
end if
return $\mathrm{b_t}$, E', $\hat{v}'$, u'

Algorithm 2 Decompressing a Signal Value From a Bit-String

Input: Bit-string $\mathrm{b_t}$, starting index s of $\mathrm{b_t}$ for a signal value, exponent E' of last small value, last decompressed large value $\hat{v}'$, the bit-length u' of U for last large value

Output: Restored signal value $\hat{v}$, updated s, E', $\hat{v}'$, u'

if $\mathrm{b_t}[s] = 0$ (meaning this is a small value) **then**
  Read S and Δ_E from $\mathrm{b_t}[s{+}1]$ and $\mathrm{b_t}[s{+}2{:}s{+}3]$
  if $\Delta_E = 3$ **then**
    Read E from $\mathrm{b_t}[s{+}4{:}s{+}e{+}3]$
    $s \coloneqq s + 4 + e + m$
  else
    Compute E based on Δ_E, E' and (22)
    $s \coloneqq s + 4 + m$
  end if
  if $E = 0$ **then**
    $s \coloneqq s - m, \quad \hat{v} = 0$
  else
    Read M from $\mathrm{b_t}[s{-}m{:}s{-}1]$
    Compute $\hat{v}$ with (12)
  end if
  $E' \coloneqq E$
else
  Read S and Δ_u from $\mathrm{b_t}[s{+}1]$ and $\mathrm{b_t}[s{+}2{:}s{+}3]$
  if $\Delta_u = 3$ **then**
    Read u from $\mathrm{b_t}[s{+}4{:}s{+}l{+}3]$
    $s \coloneqq s + 4 + l + u$
  else
    Compute u based on Δ_u, u' and (34)
    $s \coloneqq s + 4 + u$
  end if
  Read U from $\mathrm{b_t}[s{-}u{:}s{-}1]$
  Compute $\hat{v}$ with (27)
  $\hat{v} \coloneqq \hat{v} + \hat{v}', \quad \hat{v}' \coloneqq \hat{v}, \quad u' \coloneqq u$
end if
return $\hat{v}$, s, E', $\hat{v}'$, u'

Now, we draw the conclusion on the accuracy of the proposed signal-value compression scheme.

Theorem 1: If a signal value v is compressed/encoded with Algorithm 1 and then decomposed/decoded with Algorithm 2, the errors of restored value $\hat{v}$ satisfy

$$\begin{cases} \epsilon = \left|\hat{v} - v\right| \le \varepsilon_{\mathrm{abs}} \\ \eta = \left|\hat{v} - v\right| / |v| \le \varepsilon_{\mathrm{rel}} \end{cases} \tag{37}$$

providing that the bit-length of mantissa $m \ge -\log_2 \varepsilon_{\mathrm{rel}} - 1$, and the quantization resolution $c \le 2\varepsilon_{\mathrm{abs}}$.

Proof: If $v = 0$, there is no error during the compressing and decompressing procedure. For a nonzero signal value, if $|v| \le \tau = \varepsilon_{\mathrm{abs}}/\varepsilon_{\mathrm{rel}}$, with τ defined as (11), it is a small value. Therefore, its relative error $\eta \le 2^{-(m+1)}$ due to Lemma 1. Substituting $m \ge -\log_2 \varepsilon_{\mathrm{rel}} - 1$, we have $\eta \le \varepsilon_{\mathrm{rel}}$. Since the absolute error $\epsilon = \eta \cdot |v|$, we then derive $\epsilon \le \varepsilon_{\mathrm{abs}}$.

If $|v| > \tau = \varepsilon_{\mathrm{abs}}/\varepsilon_{\mathrm{rel}}$, it is a large value. From Lemma 2, we see its absolute error $\epsilon = |\hat{v} - v| \le \varepsilon_{\mathrm{abs}}$. Then, since the relative error $\eta = \epsilon/|v|$, we get $\eta \le \varepsilon_{\mathrm{rel}}$. This ends the proof. ■

Algorithms 1 and 2 include a few of integer arithmetics, floating-point arithmetics, and the operations on bit-string. The computational complexity for compressing or decompressing a signal value is just $O(1)$.

E. Further Compression and Overall Algorithms

The waveform compression is carried out during transient simulation. Therefore, the signal values are a kind of data stream; the matrix $\boldsymbol{W}$ is dynamically augmented from the right. To control the memory cost of compression, we organize the compressed signal values in blocks. Each block corresponds to a time slice of simulation, i.e., n adjacent time points, or n

block head			compressed signal values		
n	$\hat{\boldsymbol{t}}$	$\boldsymbol{p}$	signal 1	...	signal N_s

Fig. 3. Storage format of a data block for compressed waveforms.

adjacent columns of $\boldsymbol{W}$. The storage format of a data block is shown as Fig. 3.

It consists of a block head (n, $\hat{\boldsymbol{t}}$, and $\boldsymbol{p}$) and N_s sub-blocks of compressed signal values. $\hat{\boldsymbol{t}}$ is the n time points stored in double-precision floating-point numbers. Vector $\boldsymbol{p}$ stores the start positions of the N_s signal sub-blocks in the block, which is required since the bit-length of each compressed signal value varies. The default value for n can be determined by the memory cost constraint for the waveform compression. The separation of different signals in the block also makes it easy to decompress a portion of signals from the block.

During the compression, we can use the general lossless compression algorithms, such as Deflate [13] to perform a secondary compression on the data block. The Deflate algorithm employs a combination of LZSS and Huffman coding and can further reduce 20%–50% size of data.

We present the whole procedure of waveform compression as Algorithm 3. It gets input from the data stream generated by circuit simulator, and writes the compressed waveforms into a hard-disk file. At line 10, Algorithm 1 is used to compress a signal value. n_t denotes the default value of n. "Deflate" denotes the Deflate algorithm for the secondary compression, which is invoked at lines 16 and 20 to compress the N_s sub-blocks of compressed signal values and the block head, respectively. Each time it is invoked, the result's bit-length is attached in front of the result, which is omitted in Algorithm 3 for clarity. At line 17, size() means the bit-length of a bit-string.

Algorithm 4 shows the decompression procedure, whose input is the compressed file obtained with Algorithm 3 and output is the data stream of recovered signal values. In Algorithm 4, Algorithm 2 is used at line 11 to decompress signal values and "Inflate" denotes the reverse process of the Deflate algorithm. In addition to extracting/decompressing all signals stored in the compressed file, Algorithm 4 can perform partial decompression which only extracts/recovers the waveforms of specified signals. Therefore, the input argument of Algorithm 4 includes $\{k_1, k_2, \ldots, k_g\}$ standing for the indices of g signals for recovery. Due to the format of data block in the proposed approach (see Fig. 3), the partial decomposition can be naturally realized.

Because the time complexity of Algorithms 1 and 2 are $O(1)$, the time complexity of Algorithms 3 and 4 are both $O(N_t N_s)$, where N_t and N_s are the number of time points and the number of signals, respectively. The space complexity of them is $O(n_t N_s)$, where n_t is the default number of time points in a block. We can change the value of n_t to control the memory cost for the compressing and decompressing procedure.

Since the Deflate algorithm is a lossless compression technique, we can draw the following conclusion.

Algorithm 3 Waveform Compression for Circuit Simulation

Input: Data stream **D** from circuit simulator, which outputs a time value t, and signal values $[v_1, v_2, \ldots, v_{N_s}]$ each time.
Output: Disk file **F** for the compressed waveforms.

1: Create an empty disk file **F**.
2: Determine the number of time points n_t for each data block, based on N_s and the budget of memory usage.
3: Initialize the input variables $\hat{v}'_j, E'_j, u'_j$ of Algorithm 1 for $j = 1, 2, \ldots, N_s$.
4: **while** The simulation is not completed **do**
5: Reset an empty array $\hat{\boldsymbol{t}}$ for recording time points.
6: Reset empty bit-strings $\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_{N_s}$.
7: Fetch time t and signal values $v_1, v_2, \ldots, v_{N_s}$ from **D**.
8: Append t to $\hat{\boldsymbol{t}}$.
9: **for** j in $1, 2, \ldots, N_s$ **do**
10: Use Algorithm 1 to compress the signal value v_j, obtaining bit-string $\mathrm{b_t}$ and updating $\hat{v}'_j, E'_j, u'_j$.
11: Append $\mathrm{b_t}$ to $\boldsymbol{b}_j$.
12: **end for**
13: **if** $\hat{\boldsymbol{t}}$ has n_t elements or t is the last time point **then**
14: $n \coloneqq$ the number of elements in $\hat{\boldsymbol{t}}$.
15: **for** j in $1, 2, \ldots, N_s$ **do**
16: Use Deflate to compress bit-string $\boldsymbol{b}_j$ to $\boldsymbol{b}'_j$.
17: $p_j \coloneqq \sum_{k=1}^{j-1} \mathrm{size}(\boldsymbol{b}'_k)$.
18: **end for**
19: $\boldsymbol{p} \coloneqq [p_1, p_2, \ldots, p_{N_s}]$
20: Use Deflate to compress $[\hat{\boldsymbol{t}}, \boldsymbol{p}]$ to $\hat{\boldsymbol{t}}'$.
21: Concatenate $n, \hat{\boldsymbol{t}}', \boldsymbol{b}'_1, \boldsymbol{b}'_2, \ldots, \boldsymbol{b}'_{N_s}$ to form a data block, and write it to **F**.
22: **end if**
23: **end while**

Theorem 2: If the signal waveforms outputted from transient simulation are compressed with Algorithm 3 and then decomposed with Algorithm 4, the error of each restored signal value will definitely satisfy the preset accuracy criteria (37), providing that the conditions in Theorem 1 are met.

The runtime of Algorithms 3 and 4 can be further reduced by using parallel computing. We find out that the waveform compression is carried out data block by data block, and the processing of each data block can be briefly regarded as a three-stage pipeline, as shown in Fig. 4. In the first stage a data block is formed with the proposed signal-value compression scheme (i.e., Algorithm 1). Then, the Deflate algorithm is used to compress the data block in the second stage. Finally, the compressed block is written to a disk file. The relationship between two adjacent stages can be described as a typical producer-consumer model in parallel computing [22]. Therefore, Algorithm 3 can be easily parallelized with three threads. Compared with the circuit simulation, the waveform compression consumes much less time (less than the former's 1% as stated in [8]). Therefore, the generation of the signal values (from the simulator) is much slower than executing each of the stages in the pipeline of compression. This means with the parallel compressing approach there is little time added to the whole simulation. Similar parallel decomposing procedure

Algorithm 4 Waveform Decompression

Input: Disk file **F** for the compressed waveforms, indices $k_1, k_2, \ldots, k_g$ of signals to decompress.

Output: Data stream $\widehat{\mathbf{D}}$ which outputs a time value t and restored signal values $[\hat{v}_{k_1}, \hat{v}_{k_2}, \ldots, \hat{v}_{k_g}]$ each time.

1: Open the disk file **F** for reading.
2: Initialize the input variables $\hat{v}'_j, E'_j, u'_j$ of Algorithm 2 for $j = k_1, k_2, \ldots, k_g$.
3: **while** The end of **F** is not reached **do**
4: Read $n, \hat{\boldsymbol{t}}'$ from **F**.
5: Use Inflate to decompress $\hat{\boldsymbol{t}}'$ to $[\hat{\boldsymbol{t}}, \boldsymbol{p}]$.
6: **for** $j = k_1, k_2, \ldots, k_g$ **do**
7: Read the j-th sub-block $\boldsymbol{b}'_j$ from **F** based on $\boldsymbol{p}$.
8: Use Inflate to decompress $\boldsymbol{b}'_j$ to $\boldsymbol{b}_j$.
9: Starting index $s = 0$.
10: **for** $i = 1, 2, \ldots, n$ **do**
11: Use Algorithm 2 to decompress the bit-string $\boldsymbol{b}_j$ at starting index s, obtaining decompressed value $\hat{v}_{ji}$ and updated $s, \hat{v}'_j, E'_j, u'_j$.
12: **end for**
13: **end for**
14: **for** $i = 1, 2, \ldots, n$ **do**
15: Push time $\hat{t}_i$ and values $\hat{v}_{k_1 i}, \hat{v}_{k_2 i}, \ldots, \hat{v}_{k_g i}$ to $\widehat{\mathbf{D}}$.
16: **end for**
17: March the reading pointer of **F** for a block based on $\boldsymbol{p}$.
18: **end while**

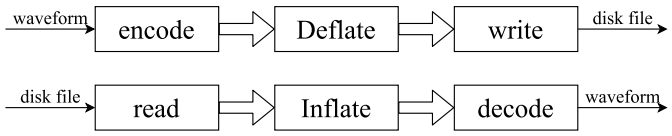

Fig. 4. Three-stage pipelines for parallel compressing/decompressing.

can be devised as shown in Fig. 4, with which the runtime of decompression can be remarkably reduced as well.

IV. Combination With the Prediction Method

With the variable-length compressed formats for small value and large value, we have to some extent considered the correlation between signal values at adjacent time points. However, the prediction method is a bit different on taking advantage of this correlation.

As show in Fig. 5, the idea of prediction is to find out the consecutive signal values lying approximately on a straight line. If we know the slope of the line, we do not need to store most of the signal values on it because they can be calculated with the straight-line equation. Below we propose two combination schemes that combine the prediction with the proposed compressed formats in Section III.

The first scheme detects all the straight-line segments well approximating the consecutive signal values and uses the information of straight-line segment to represent these signal values. Since the compressed values of each signal are organized in sub-blocks, we can analyze each signal sub-block to find the straight-line segments before we actually compress them. Suppose v_γ (for time t_γ) is a possible starting point of a straight-line segment. We first calculate the slope L of the straight line

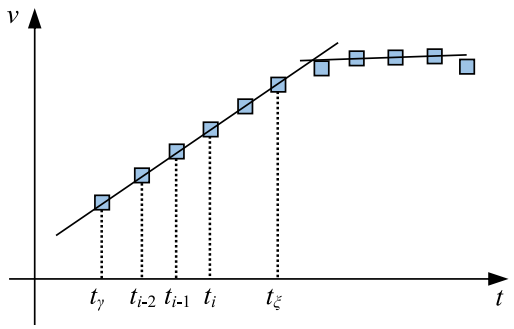

Fig. 5. Illustration of the linear prediction method for compressing signal values. The blue squares denote the signal values at discrete time points.

$$L = \frac{v_{\gamma+1} - v_\gamma}{t_{\gamma+1} - t_\gamma}. \tag{38}$$

Then, for subsequent time points, we try to predict signal value with L. The prediction of v_i is

$$\tilde{v}_i = v_\gamma + \big(t_i - t_\gamma\big)L. \tag{39}$$

If it satisfies the accuracy criteria, this prediction is successful. The consecutive prediction success on signal values indicates a straight-line segment for prediction. Suppose the predictions at time $t_{\gamma+2}$ through t_ξ are all successful, and the prediction fails at time $t_{\xi+1}$. This means $|\tilde{v}_i - v_i| \le \varepsilon_{\text{abs}}$ and $|(\tilde{v}_i - v_i)/v_i| \le \varepsilon_{\text{rel}}$ for all $i \in [\gamma, \xi]$, and the signal values can be accurately recovered with (39) without storing the actual values. The signal values form a subsequence which can be represented by a tuple $(\gamma, \xi, v_\gamma, L)$. If the length of this subsequence $(\xi - \gamma + 1)$ exceeds a threshold τ_p, we call it a predictable subsequence (P-seq), which can replace the original compressed format to make larger compression. Especially, when $L = 0$, which means all values in the P-seq are almost the same, L can be omitted in the tuple and the subsequence is called a same-value subsequence (S-seq).

Supposing the signal values for n time points in a sub-block are: $v_1, v_2, \ldots, v_n$, we start to execute the above procedure with $\gamma = 1$. After one-pass scan detecting the P-seqs in the sequence of n values with a greedy strategy, we obtain the P-seqs and some of them are actually S-seq. For the signal values not in P-seqs or S-seqs, they will be encoded in the small-value/large-value compressed format.

With the information of obtained P-seqs and S-seqs, we add a prediction sub-block, which stores the tuples for the P-seqs and S-seqs, in front of each signal sub-block in Fig. 3. The bit-strings for values in P-seqs or S-seqs are omitted in the following signal sub-block. In Fig. 6, we show this compressed format with signal prediction. The prediction sub-block includes n_p P-seqs and n_s S-seqs. For each tuple, γ and ξ are stored as 16-bit unsigned integer (supposing $n < 65\,536$), while v_γ and L are stored in double-precision floating-point number. Hence, each P-seq takes 160 bits storage and each S-seq takes 96 bits. If we choose τ_p properly, this scheme can save a lot of storage when there are many signal values forming straight-line segments.

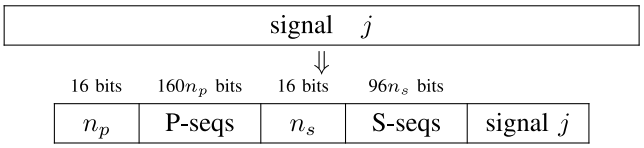

Fig. 6. Compression format of the first combination scheme. Each signal sub-block in Fig. 3 is replaced with one with a prediction sub-block, including n_p P-seqs and n_s S-seqs.

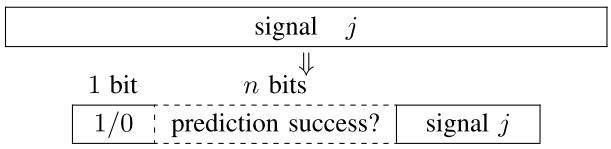

Fig. 7. Compression format of the second combination scheme. A prediction sub-block recording if each signal value is successfully predicted by its two predecessors is added to each signal sub-block.

In the second scheme, we do not pursue sufficiently long straight-line segments. Actually, we can just predict a signal value v_i at time t_i with the restored values of its adjacent predecessors: $\hat{v}_{i-1}$ at time t_{i-1} and $\hat{v}_{i-2}$ at time t_{i-2} (see Fig. 5). Therefore, the prediction of v_i is

$$\tilde{v}_i = \hat{v}_{i-1} + \frac{(t_i - t_{i-1})\big(\hat{v}_{i-1} - \hat{v}_{i-2}\big)}{t_{i-1} - t_{i-2}}. \tag{40}$$

If the prediction is successful, which means $|\tilde{v}_i - v_i| \le \varepsilon_{\text{abs}}$ and $|(\tilde{v}_i - v_i)/v_i| \le \varepsilon_{\text{rel}}$, we have the compressed value $\hat{v}_i = \tilde{v}_i$. Otherwise, $\hat{v}_i$ will be encoded in the small-value/large-value compressed format. Because the restored values are stored without approximation, for the signal value which is predicted successfully the compression errors must satisfy (37).

In order to embed the prediction into the proposed approach, we add N_s prediction sub-blocks which store the prediction information to each data block. The jth prediction sub-block should indicate if the prediction is successful for the n values of signal j. Only for the values that are predicted successfully, the compressed bit-strings for them are omitted. Therefore, if the success rate is low, this may cause even larger storage. To avoid this drawback and to guess the success rate in advance, we monitor the rate of prediction success for all the signals at the beginning time points. If the monitored success rate is above a threshold we determine the signal should be enforced by the prediction. Otherwise, all values of the signal will be processed without prediction and the information indicating if its values are predicted successfully is omitted.

In Fig. 7, we show that the prediction sub-block is added in front of each signal sub-block in Fig. 3. The first bit of the prediction sub-block indicates if this signal is selected to enforce the prediction. If it is "1," the subsequent n bits indicate the success or failure of the prediction for each of the signal values at n time points. Otherwise, the n bits are omitted.

Compared with the first combination scheme, the second one includes more computation as the slope is calculated for each signal value. With the concept of prediction success, the both schemes ensure that each restored signal value will satisfy the preset accuracy criteria. And, since they are just a modification of the block structure in Fig. 3, the lossless compression can also perform upon them to achieve larger compression ratio.

TABLE I
DETAILS OF THE TEST CASES, WITH RESULTS OF GZIP COMPRESSION

Case	N_s	N_t	size	gzip-CR	gzip-T_{comp} (s)
1	6	4.7×10^5	26 MB	2.2	0.53
2	56	6.2×10^5	271 MB	2.0	6.9
3	10	6.9×10^7	5.7 GB	3.8	102
4	216	1.4×10^7	22 GB	2.1	483
5	23	1.0×10^8	19 GB	5.1	180
6	372	2.2×10^6	6.1 GB	1.7	151
7	5360	5.0×10^5	16 GB	1.7	397

gzip-CR is the compression ratio achieved by the gzip command. gzip-T_{comp} means the CPU time for compressing the data file.

V. EXPERIMENTAL RESULTS

We have implemented the proposed waveform compression approach, the method in [8], and compared them with the lossless Deflate algorithm. Our approach includes two versions, with and without a Deflate post-processing. The latter is called *SPICEMate*. The method in [8] employs the prediction method and nonuniform quantization (as described in Section II-B), and a Deflate post-processing as well. We have also implemented the decompression algorithms for these methods. Among them, the implementation of Deflate (and Inflate) algorithm is all provided by *zlib* library [19] with the compression level set to 1 (best speed). All programs are implemented in C++.

Experiments are carried out on a Linux server with Intel Xeon E5-2630 CPU (2.40 GHz). The operating system is Ubuntu 16.04. Unless otherwise stated, the CPU time of different algorithms are measured.

A. Test Cases

The test cases are seven raw waveform files dumped from the transient simulation of analog circuits, provided by our industrial partner. The data in each file are the elements of the matrix in (2), column by column. And, each value is a double-precision floating-point number. The details of these raw waveforms are described in Table I. We have also compressed these data files with the gzip command, and listed the compression ratio, and CPU time in Table I, as a baseline for comparison.

In the following sections, we will first validate the performance of the proposed compression scheme (SPICEMate) in terms of compression ratio, time for compressing and decompressing, and the accuracy. Then, we will show the benefit of our parallel compressing/decompressing scheme with three-stage pipeline, and the application of SPICEMate for partial decompression. Finally, we show the possible benefit brought by combining SPICEMate with the prediction method.

B. Validating Performance of the Proposed Scheme

Suppose the maximum tolerable absolute error of waveform compression is $\varepsilon_{\text{abs}} = 10^{-5}$, and the maximum tolerable

TABLE II
EXPERIMENTAL RESULTS OF THE PROPOSED SCHEME WITH $\varepsilon_{\text{abs}} = 10^{-5}$, $\varepsilon_{\text{rel}} = 10^{-3}$, AND THE COMPARISON WITH THE METHOD IN [8]

Case	Zip			Method in [8]					SpiceMate (w/o Deflate)			SpiceMate				
	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	ϵ_{max}	η_{max}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	ϵ_{max}	η_{max}
1	2.1	0.59	0.24	3.3	0.80	0.80	3.6E-4	2.7E-2	3.0	0.31	0.28	**9.3**	0.53	0.35	1.0E-5	6.8E-4
2	2.0	6.0	2.0	27.0	2.9	2.3	1.1E-3	4.9E+4	9.3	3.0	1.5	**67.1**	3.4	1.7	1.0E-5	9.9E-4
3	3.8	78	31	6.9	138	97	1.2E-3	6.4E-3	6.2	53	60	**34.1**	80	69	1.0E-5	8.4E-4
4	2.1	433	138	8.0	314	262	5.6E-3	2.1E+9	8.3	243	108	**11.5**	322	139	1.0E-5	1.0E-3
5	5.7	166	63	11.2	273	182	3.6E-3	2.4E+6	6.9	189	94	**34.8**	242	116	1.0E-5	9.2E-4
6	1.7	152	46	5.4	99	74	3.2E-3	2.8E+8	5.4	82	30	**10.2**	108	41	1.0E-5	1.0E-3
7	1.6	397	115	8.1	214	182	5.5E-3	2.3E+9	7.3	221	72	**12.6**	294	94	1.0E-5	1.0E-3
avg.	2.7	176	56.5	10.0	149	114	2.9E-3	--	6.6	114	52	**25.7**	150	66	1.0E-5	9.2E-4

CR means the compression ratio. T_{comp} and T_{decomp} are the CPU time for compressing and decompressing, respectively (both in unit of second). ϵ_{max} and η_{max} are the maximum absolute error and maximum relative error, respectively. The row of "avg." shows the average numbers.

relative error is $\varepsilon_{\text{rel}} = 10^{-3}$. This means the threshold for distinguishing small value from large value is $\tau = 0.01$. For SPICEMate, we set parameters $e = 7$, $m = 9$, and $b = 128$ for the small-value format, and $c = 2 \times 10^{-5}$, $l = 7$ for the large-value format. For the method in [8], the parameters for determining reference frames are $\alpha = 2048$, $\beta = 2$, and the quantization width is 16 bits. The A-law algorithm is used for nonuniform quantization. The accuracy criterion for prediction is the relative error no more than 10^{-3}. The threshold for the failure ratio of prediction is 0.67. It means if more than 67% signals are failed to predict, all the signals in a frame will be stored as quantized integers. The parameters for method in [8] have been exhaustively tuned to produce the largest compression ratio.

For each test case, we read it in and imitate the data stream from circuit simulator. Then, Algorithm 3 is used to compress the waveforms. Two versions of our approach, without and with the Deflate steps, are tested. The n_t in Algorithm 3 is set with an empirical formula so that the memory usage does not exceed 1 GB. The experimental results for the test cases are listed in Table II, along with those from the method of [8] and directly applying the Deflate algorithm (denoted by Zip). Algorithm 4 is used for the decompression in our approach. The disk I/O time is excluded from the measurement of CPU time.

From Table II, we see that our SPICEMate always achieves the largest compression ratio (up to 67), whose average number is 25.7. In cases 2, 3, and 5, many signals change linearly or just stay unchanged most of the time, from which all methods can benefit. The compression ratios in these cases are thus significantly larger than others. On average, SPICEMate achieves 2.6× larger compression ratio than the method in [8]. Notice the both methods use *zlib* for secondary compressing. The compression ratios got from Zip are almost the same as those from gzip listed in Table I; it is only 2.7 averagely. As for the CPU time for compressing and decompressing, SPICEMate consumes equal or less time as compared with Zip and method in [8]. The difference in T_{comp} between SPICEMate without Deflate and SPICEMate means the time for using *zlib* to compress the data blocks. The compression times of SPICEMate (w/o Deflate) and SPICEMate are plotted in Fig. 8, in which the trends approximated by straight line

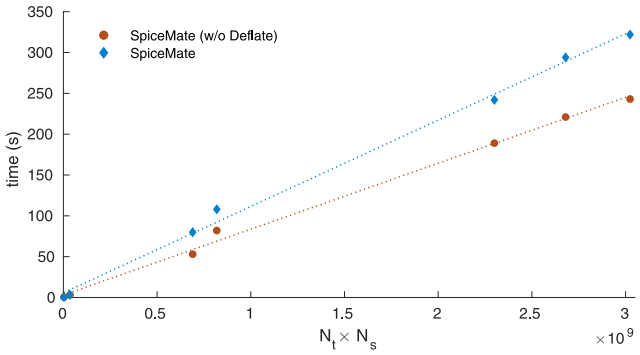

Fig. 8. Compression times of SPICEMate (w/o Deflate) and SPICEMate v.s. the product of N_t and N_s.

verify the $O(N_t N_s)$ time complexity of the proposed approach. Besides, after statistics we see that in each test case small values account for 7%–22% of all signal values.

For the compression accuracy, the results show that our approach (with or without Deflate) always satisfies the preset accuracy criterion, i.e., $\epsilon_{\max} \leq 10^{-5}$ and $\eta_{\max} \leq 10^{-3}$. This verifies our theoretical analysis in Section III. Method in [8], however, cannot guarantee any error bound and usually has much higher relative error. This is due to the fact that it mainly uses a quantization method for compressing. In Fig. 9, we show a part of a waveform in case 1 and the recovered waveforms obtained from the compressed files with our approach and the method in [8]. The three waveforms are indistinguishable. The compression errors (i.e., absolute error) of the two methods are also plotted in Fig. 9, which clearly demonstrates the large error caused by the method in [8]. Fig. 10 shows a case in which method in [8] causes an extremely large relative error (notice the vertical scale is 10^{-4} V). It is a part of a waveform in case 4, where the differences between current frame and the reference frame $\boldsymbol{d}$ in (7) satisfies $\min(\boldsymbol{d}) \approx -4.9$ and $\max(\boldsymbol{d}) \approx 5.0$, while the difference of the plotted signal d_k is about -2×10^{-4}. After the normalization in (8) and the nonuniform quantization with A-law, the quantization resolution near d_k is about 4×10^{-4}. Consequently, we can observe distinct stairs on the recovered waveform in Fig. 10. In such situation, if the original signal values are very closed to 0 (e.g.,

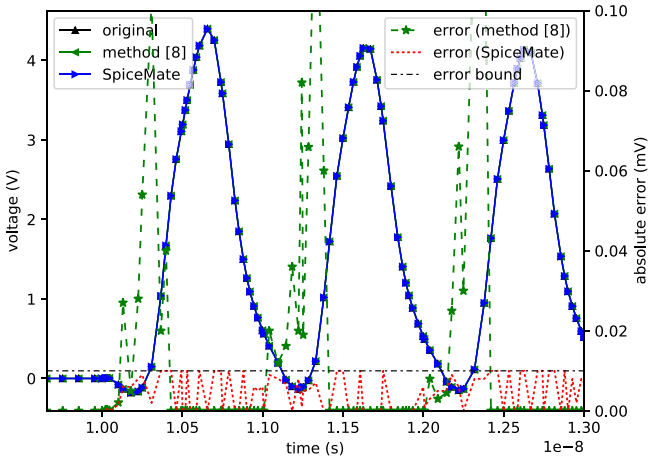

Fig. 9. Part of a waveform in case 1 and the recovered waveforms obtained from the compressed files with our approach (SPICEMate) and the method in [8]. The curves of absolute error (drawn on the secondary y-axis) show the accuracy of SPICEMate and the larger errors of method in [8].

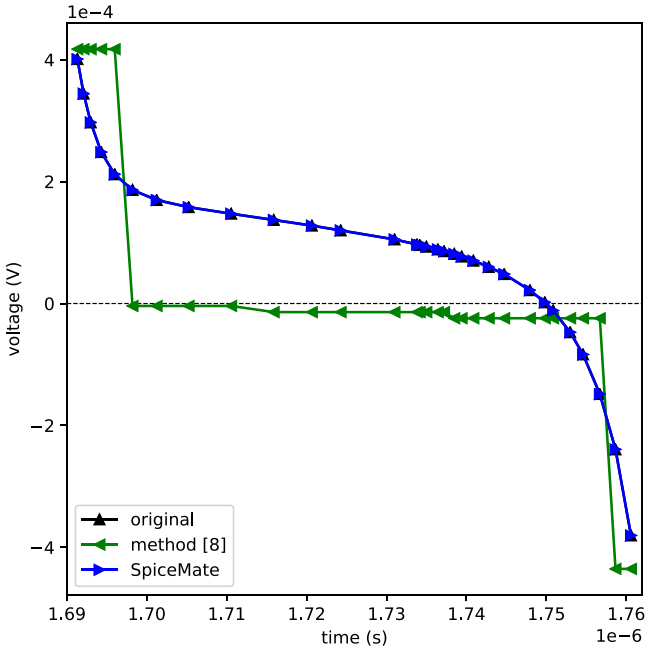

Fig. 10. Part of a waveform in case 4 and the recovered waveforms obtained from the compressed files with our approach (SPICEMate) and the method in [8]. The relative error of method [8] is extremely large around time $t = 1.75 \times 10^{-6}$ s.

around $t = 1.75 \times 10^{-6}$ s), the relative error can be extremely large. This explains the large $\eta_{\max}$ for method in [8] in Table II.

Sometimes smaller error tolerances are needed. This makes the compression ratio decrease. An extra experiment is conducted to show the performance of SPICEMate for this scenario with higher accuracy demand. With ε_{abs} decreasing from 10^{-5} to 10^{-9}, the compression ratios for the seven cases decreases to 8.1, 43.6, 28.5, 6.4, 33.4, 5.6, and 6.7, respectively. Their average is 18.9. The decreasing curve for the worst case, i.e., case 6, is plotted in Fig. 11. We see that even with the absolute error tolerance as low as 10^{-9} our approach can achieve a better compression ratio than the

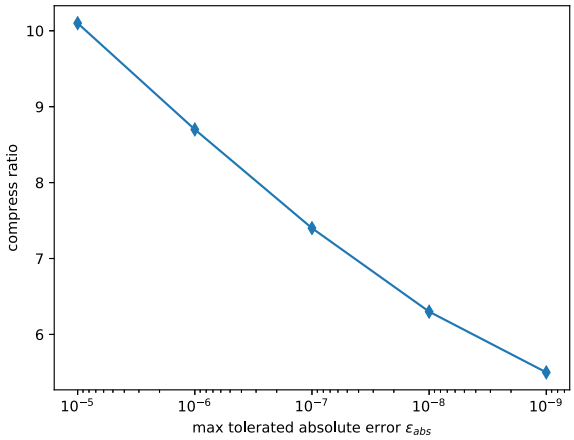

Fig. 11. Compression ratio of our approach (SPICEMate) on case 6 with ε_{abs} decreasing from 10^{-5} to 10^{-9}.

TABLE III
RESULTS OF USING THE THREE-STAGE PIPELINE IN OUR COMPRESSING AND DECOMPRESSING ALGORITHMS

Case	Compression time (s)				Decompression time (s)			
	encode	Deflate	write	T_{par}	read	Inflate	decode	T_{par}
1	0.31	0.22	0.00	0.37	0.00	0.07	0.28	0.29
2	3.0	0.41	0.01	3.1	0.05	0.20	1.5	1.5
3	53	27	0.28	56	0.38	9.8	60	60
4	243	80	2.1	245	4.7	31	108	109
5	189	53	0.70	193	1.6	22	94	94
6	82	26	0.52	84	1.9	11	30	31
7	237	57	2.8	239	2.3	20	74	75

T_{par} means the parallel-computing time with three threads.

TABLE IV
EFFICIENCY OF SPICEMATE FOR PARTIAL DECOMPRESSION

Case	N_s	T_{decomp} (s)	#signal for decomposition	Time for partial decomposition (s)
1	6	0.29	2 (33%)	0.18 (61%)
2	56	1.5	12 (21%)	0.46 (30%)
3	10	60	2 (20%)	23 (39%)
4	216	109	44 (20%)	19 (18%)
5	23	94	5 (22%)	54 (57%)
6	372	31	75 (20%)	7.5 (24%)
7	5360	75	1072 (20%)	16 (21%)

Here T_{decomp} is the runtime of parallel SpiceMate for decompressing all signals.

method in [8] with much worse accuracy (5.6 versus 5.4). We have also tested the cases with the setting of $\varepsilon_{\text{abs}} = 10^{-6}$ and $\varepsilon_{\text{rel}} = 10^{-4}$. The achieved compression ratios are just a little smaller than those in Table II, i.e., 9.3, 62.8, 34.3, 9.1, 34.4, 8.4, and 9.9, respectively.

C. Parallel Compressing/Decompressing

As discussed at the end of Section III-E, SPICEMate can be accelerated by using parallel computing. It follows a three-stage pipeline model and can be easily implemented with

TABLE V
RESULTS OF TWO COMBINATION SCHEMES OF SPICEMATE COMBINED WITH THE PREDICTION METHOD ($\varepsilon_{\text{abs}} = 10^{-5}$, $\varepsilon_{\text{rel}} = 10^{-3}$)

Case	Combination Scheme 1						Combination Scheme 2					
	w/o Deflate			w/ Deflate			w/o Deflate			w/ Deflate		
	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}	CR	T_{comp}	T_{decomp}
1	**3.24**	0.25	0.13	9.28	0.25	0.14	3.18	0.31	0.13	**9.30**	0.53	0.18
2	**38.2**	0.68	1.3	69.6	0.66	1.2	26.5	2.9	1.3	**70.4**	3.4	1.5
3	4.91	38	16	32.8	42	16	**4.96**	51	19	**34.0**	81	23
4	9.53	215	132	11.5	228	134	**10.5**	246	110	**12.1**	325	145
5	**12.9**	73	72	34.8	82	81	12.3	170	71	**37.3**	219	90
6	5.38	94	34	**10.2**	100	36	**5.47**	88	36	10.1	111	45
7	8.96	214	101	12.5	236	89	**9.11**	240	85	**12.8**	320	115
avg.	**11.9**	91	51	25.8	98	51	10.3	114	46	**26.6**	151	60

The number in bold means the larger compression ratio in the two combination schemes. If it is underlined as well, we mean the compression ratio is larger than that of the corresponding version of SpiceMate in Table II.

three threads. We thus have implemented a parallel version of SPICEMate with C++ *Standard Library*. Each stage of the pipeline is implemented as a thread which can be created by `std::async` function. A thread transfers processed data to its next stage using a blocking queue implemented with `std::condition_variable` and `std::mutex`.

This parallel SPICEMate is tested for compressing and decompressing the seven cases. The single-thread runtime of each stage and the overall runtime of the parallel program are listed in Table III. From it we see that the encode/decode stage (i.e., signal-value compressing/decompressing) consumes the most time among the three stages. And, the overall runtime is just a little bit more than it, because of the block-by-block data processing and the parallel computing. This means, by using one more thread (for executing Deflate), our parallel SPICEMate can achieve much larger compression ratio with almost no runtime penalty. And considering that the waveform compression consumes much less time than the transient simulation, we can infer that the pipelined parallel compression hardly increases time cost to the whole transition simulation procedure.

D. Partial Decompression

Partial decompression is a useful function for a waveform compression scheme in practical scenarios like debugging. Due to the block structure (see Fig. 3) in our approach, Algorithm 4 can naturally decompress a fraction of the signals from the compressed file. To demonstrate the capability of our parallel SPICEMate for partial decompression, an experiment is conducted to decompress partial signals of the test cases. The results are listed in Table IV. The ratio of the number of decompressed signals to the total number of signals and the ratio of time for decompressing partial signals to the time for decompressing all signals are given in parenthesis. From the table we see that the ratio of decompressing time is equal or just a little larger than the ratio of number of signal for decompression. Especially for the compressed file with large signal count (N_s), the decompression time of our approach is proportional to the number of signals for decompression.

E. Combination With Prediction Method

To evaluate the two combination schemes proposed in Section IV which incorporate the prediction method, we test them with the seven cases. The results are listed in Table V. For the first scheme, τ_p is set 16 and 10 for the P-seq and S-seq, respectively, in the situation without Deflate. And τ_p is set 32 for the both in the situation with Deflate. For the second scheme, the threshold of prediction success rate for determining if a signal should be encoded with the hybrid scheme is 0.25. The both schemes preserve the accuracy of SPICEMate, and the errors are not listed in the table.

From the table, we see that the first scheme consumes less time for compression and achieves much larger compression ratio (e.g., for case 2 if not using Deflate) than the second scheme. If the Deflate algorithm is applied, the second scheme performs mostly better than the first scheme, in terms of compression ratio. Without the Deflate, the both schemes increases the compression ratio of SPICEMate from 6.6 to more than 10. This is the power of the prediction. However, the resulted storage formats become more irregular, so that the lossless compression cannot bring large benefit. As the result, the combined schemes finally achieve just a little bit larger compression than the SPICEMate in Table II. With the second combination scheme, the average compression ratio is increased from 25.7 to 26.6. In general, it preserves the compression ratio of SPICEMate, and for some cases large improvement on compression ratio is exhibited. For example, the compression ratio is enlarged by 7% for case 5. And, the largest compression ratio reaches 70.4 (for case 2).

VI. CONCLUSION

In this article, we proposed a set of algorithms to compress the waveforms obtained from transient circuit simulation. In order to satisfy both absolute and relative error requirements, we classify the signal values into small values and large values, and propose two different variable-length encoding formats to store them, respectively. The compressed data are organized block by block, and in each data block the data for different signals are stored separately to enable partial

decompression. The block structure also facilitates applying the lossless Deflate algorithm for secondary compression and the efficient pipelined parallel compression/decompression. The latter speeds up the algorithms with multithread computing and makes the compression adding little overhead of runtime to the transient circuit simulation. Two schemes combining the proposed approach with the prediction method are also proposed, which adds the information of prediction to the block structure, to achieve more compression in the scenarios without and with the secondary compression.

We have theoretically proved that the proposed schemes can ensure the preset criteria on absolute error and relative error of compressed signal values. And with waveform data from analog circuit simulation we have validated the advantages of the proposed compression scheme in terms of accuracy, compression ratio and runtime efficiency. Compared with an existing counterpart, the proposed approach achieves $2.6\times$ larger compression ratio on average, while possessing the unique accuracy-ensured advantage. Compared with the raw data stored as double-precision floating-point numbers, the compression ratio achieved by the proposed approach can be as large as 70.4 while keeping the relative error less than 10^{-3} and the absolute error less than 10^{-5}. If the lossless secondary compression cannot be applied somehow, our approach still achieves the compression ratio of 11.9 averagely and up to 38.2.

Although throughout this article, we assume the accuracy criteria are set for all signals, the proposed approach can be easily extended to handle the scenario where several different accuracy criteria are set for different signals. The choice of SPICEMate and the two schemes incorporating the prediction depend on actual engineering concerns. If the compression ratio is not so crucial, the SPICEMate without prediction may be sufficient. And, if the secondary compression cannot be applied and/or the compression time is of concern, the combined schemes without Deflate should be used.

REFERENCES

[1] L. T. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic Circuit & System Simulation Methods*. New York, NY, USA: McGraw-Hill Inc., 1998.

[2] B. Kalloor, "A comparative study of compressing algorithms to reduce the output file size generated from a VHDL-AMS simulator," M.S. thesis, Dept. Comput. Eng., Univ. Cincinnati, Cincinnati, OH, USA, 2006.

[3] F. Livinston, N. Magotra, S. Stearns, and W. McCoy, "Real time implementation concerns for lossless waveform compression," in *Proc. Int. Symp. Circuits Syst. (ISCAS'95)*, Seattle, WA, USA, 1995, pp. 1267–1270.

[4] E. Naroska, S.-J. Ruan, C.-L. Ho, S. Mchaalia, F. Lai, and U. Schwiegelshohn, "A novel approach for digital waveform compression," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2003, pp. 712–715.

[5] S. Hatami, P. Feldmann, S. Abbaspour, and M. Pedram, "Efficient compression and handling of current source model library waveforms," in *Proc. Design Autom. Test Eur. Conf. (DATE)*, Nice, France, 2009, pp. 1178–1183.

[6] S. Hatami and M. Pedram, "Efficient representation, stratification, and compression of variational CSM library waveforms using robust principle component analysis," in *Proc. IEEE Design Autom. Test Eur. Conf. (DATE)*, Dresden, Germany, 2010, pp. 1285–1290.

[7] W. Yu, Y. Gu, and Y. Li, "Efficient randomized algorithms for the fixed-precision low-rank matrix approximation," *SIAM J. Matrix Anal. Appl.*, vol. 39, no. 3, pp. 1339–1359, 2018.

[8] Y. Liu, F. Yang, and X. Zeng, "An efficient compression method for analog waveforms," *J. Fudan Univ. (Nat. Sci.)*, vol. 52, no. 4, pp. 486–491, 2013.

[9] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Chicago, IL, USA, 2016, pp. 730–739.

[10] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Orlando, FL, USA, 2017, pp. 1129–1139.

[11] S. Saurabh and P. Mittal, "A practical methodology to compress technology libraries using recursive polynomial representation," in *Proc. 31st Int. Conf. VLSI Design (VLSID)*, Pune, India, 2018, pp. 301–306.

[12] *CosmosScope*, Synopsys, Mountain View, CA, USA, Apr. 2020. [Online]. Available: https://www.synopsys.com/verification/virtual-prototyping/saber/cosmos-scope.html

[13] P. Deutsch, "DEFLATE compressed data format specification version1.3," Internet Eng. Task Force, RFC 1951, 1996.

[14] A. Yazdanpanah and M. R. Hashemi, "A new compression ratio prediction algorithm for hardware implementations of LZW data compression," in *Proc. IEEE 15th CSI Int. Symp. Comput. Archit. Digit. Syst.*, Tehran, Iran, 2010, pp. 155–156.

[15] M. Burtscher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data," *IEEE Trans. Comput.*, vol. 58, no. 1, pp. 18–31, Jan. 2009.

[16] N. Fout and K.-L. Ma, "An adaptive prediction-based approach to lossless compression of floating-point volume data," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2295–2304, Dec. 2012.

[17] R. Balasubramanian, C. A. Bouman, and J. P. Allebach, "Sequential scalar quantization of vectors: An analysis," *IEEE Trans. Image Process.*, vol. 4, no. 9, pp. 1282–1295, Sep. 1995.

[18] A. Gersho, "Principles of quantization," *IEEE Trans. Circuits Syst.*, vol. 25, no. 7, pp. 427–436, Jul. 1978.

[19] J.-L. Gailly and M. Adler. (Jan. 2017). *Zlib, a Massively Spiffy Yet Delicately Unobtrusive Compression Library*. [Online]. Available: http://www.zlib.net/

[20] C. B. Moler, *Numerical Computing With MATLAB: Revised Reprint*. Philadelphia, PA, USA: SIAM Press, 2008.

[21] M. T. Heath, *Scientific Computing: An Introductory Survey*. New York, NY, USA: McGraw-Hill Inc., 2002.

[22] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. Madison, WI, USA: Arpaci-Dusseau Books LLC, 2018, ch. Condition Variables.

Lingjie Li received the B.S. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018, where he is currently pursuing the Ph.D. degree in computer science and technology.

His research interests include electromagnetic simulation, tensor computation, and machine learning.

Wenjian Yu (Senior Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 1999 and 2003, respectively.

He joined Tsinghua University in 2003, where he is currently a Tenured Associate Professor with the Department of Computer Science and Technology. He was a Visiting Scholar with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, USA, twice during the period from 2005 to 2008. He has authored two books and 150 papers in refereed journals and conferences. His current research interests include physical-level modeling and simulation for IC design, high-performance numerical algorithms, big-data analytics, and machine learning.

Dr. Yu was a recipient of the Distinguished Ph.D. Award from Tsinghua University in 2003, the Excellent Young Scholar Award from the National Science Foundation of China in 2014, the Best Paper Award from the Design, the Automation and Testing in Europe Conference in 2016, the Best Student Paper Award from International Applied Computational Electromagnetics Society Symposium in 2017, and the Best Student Paper Award from International Conference on Tools with Artificial Intelligence in 2019.