Accelerating Relative-error Bounded Lossy Compression for HPC datasets with Precomputation-Based Mechanisms

Xiangyu Zou[1,2], Tao Lu[3], Wen Xia[1,2], Xuan Wang[1], Weizhe Zhang[1,2],
Sheng Di[4], Dingwen Tao[5], and Franck Cappello[4,6]

[1] Harbin Institute of Technology, Shenzhen, China
[2] Peng Cheng Laboratory, Shenzhen, China
[3] Marvell Technology Group, USA
[4] Argonne National Laboratory, IL, USA
[5] University of Alabama, AL, USA
[6] University of Illinois at Urbana-Champaign, IL, USA

***Abstract*—Scientific simulations in high-performance computing (HPC) environments are producing vast volume of data, which may cause a severe I/O bottleneck at runtime and a huge burden on storage space for post-analysis. Unlike the traditional data reduction schemes (such as deduplication or lossless compression), not only can error-controlled lossy compression significantly reduce the data size but it can also hold the promise to satisfy user demand on error control. Point-wise relative error bounds (i.e., compression errors depends on the data values) are widely used by many scientific applications in the lossy compression, since error control can adapt to the precision in the dataset automatically. Point-wise relative error bounded compression is complicated and time consuming. In this work, we develop efficient precomputation-based mechanisms in the SZ lossy compression framework. Our mechanisms can avoid costly logarithmic transformation and identify quantization factor values via a fast table lookup, greatly accelerating the relative-error bounded compression with excellent compression ratios. In addition, our mechanisms also help reduce traversing operations for Huffman decoding, and thus significantly accelerate the decompression process in SZ. Experiments with four well-known real-world scientific simulation datasets show that our solution can improve the compression rate by about 30% and decompression rate by about 70% in most of cases, making our designed lossy compression strategy the best choice in class in most cases.**

***Index Terms*—Lossy compression, high-performance computing, scientific data, compression rate**

I. Introduction

Cutting-edge computational research in various domains relies on high-performance computing (HPC) systems to accelerate the time to insights. Data generated during such simulations enable domain scientists to validate theories and investigate new microscopic phenomena in a scale that was not possible in the past. Because of the fidelity requirements in both spatial and temporal dimensions, terabytes or even petabytes of analysis data would be produced easily by scientific simulations per run [1]–[3], when trying to capture the time evolution of physics phenomena in a fine spatiotemporal scale. For instance, there are 260 TB of data generated across one ensemble every 16 seconds, when estimating even one ensemble member per simulated day [4]. The data volume and data movement rate are imposing unprecedented pressure on storage and interconnects [5], [6], for both writing data to persistent storage and retrieving them for post-analysis. As HPC storage infrastructure is being pushed to the scalability limits in terms of both throughput and capacity [7], the communities are striving to find new approaches to lower the storage cost. Data reduction, among others, is deemed to be a promising candidate by reducing the amount of data moved to storage systems.

Data deduplication and lossless compression have been widely used in general-purpose systems to reduce data redundancy. In particular, deduplication [8] eliminates redundant data at the file or chunk level, which can result in a high reduction ratio if there are a large number of identical chunks at the granularity of tens of kilobytes. For scientific data, this rarely occurs. It was reported that deduplication typically reduces dataset size by only 20% to 30% [9], which is far from being useful in production. On the other hand, lossless compression in HPC was designed to reduce the storage footprint of applications, primarily for checkpoint/restart. Shannon entropy [10] provides a theoretical upper limit on the data compressibility; simulation data often exhibit high entropy, and as a result, lossless compression usually achieves a modest reduction ratio of less than two [11]. With growing disparity between compute and I/O, more aggressive data reduction schemes are needed to further reduce data by an order of magnitude or more [4], so the focus has shifted towards lossy compression recently.

In this paper, we mainly focus on the point-wise relative error bounded lossy compression due to the fact that point-wise relative error bound (or relative error bound for short) is a critical error controlling way broadly adopted by many scientific applications in the lossy compression community. Unlike the absolute error control adopting a fixed error bound for each data point, pointwise relative-error bound indicates that the compression error on each data point should be restricted within a constant percentage of its data value. In

This research was conducted completely using publicly accessible academic resources. It does not reflect viewpoints of Marvell Technology Group.

other words, the smaller the data value, the lower the absolute error bound on the data point. Accordingly, more details can be preserved in the regions with small values under the pointwise relative error bound than with the absolute error bound. In addition, the pointwise relative error bound is also demanded by some applications in particular. According to cosmologists, for example, the lower a particle's velocity is, the smaller the compression error it should tolerate. Many other studies [12], [13] also focus on point-wise relative error bound in lossy compression.

ZFP and SZ have been widely recognized as the top two error-controlled lossy compressors with respect to both compression ratio and rate [13]–[16]. They have been adopted on thousands of computing nodes. For example, the Adaptable IO System (ADIOS) [17] deployed on the Titan (OLCF-3) supercomputer at Oak Ridge National Laboratory has integrated both ZFP and SZ for data compression. Although being the two best compressors in class, ZFP and SZ still have their own pros and cons because of distinct design principles. Motivated by fixed-rate encoding and random access, ZFP follows the principle of classic transform-based compression for image data. SZ adopts a prediction-based compression model, which involves four key steps: data prediction, linear-scaling quantization, Huffman encoding and lossless compression. In general, SZ outperforms ZFP in compression ratio (about 2X or more), but SZ is often 20-30% slower than ZFP [13], bringing up a dilemma for users to choose an appropriate compressor in between. In fact, the compression/decompression rate is the same important as the compression ratio, in that many of applications particularly require fast compression at runtime because of extremely fast data production rate such as the X-ray analysis data generated by APS/LCLS [18], [19]. A straight-forward question is can we significantly improve the compression and decompression rates based on SZ lossy compression framework, leading to an optimal lossy compressor for users.

In this paper, we focus on how to significantly improve the compression and decompression rates for SZ, while still keeping a high compression ratio and strictly respecting the user-required error bound. This research is non-trivial because of the following two reasons. ① logarithmic transformation, which plays an important role in pointwise relative error bounded compression, is a complicated and time-consuming process. How to speedup logorithmic transform and improve the overall compression rate is challenging. ② To this end, we develop a precomputation-based mechanism with fast table lookup method and prove its effectiveness in theory. However, our initial design of the precomputation-based model can lead to non-error-bounded cases. How to guarantee the user-required error bound has to be considered carefully.

The key contributions of our work are four-fold.

- We identify the performance bottleneck in SZ, by providing a complete performance profiling for SZ compression. Specifically, our in-depth performance analysis shows that the online logarithmic transformation and the calculation of quantization factor values are the main source of the low compression performance. In absolute numbers, they occupy more than 60% of the aggregate compression time, limiting SZ compression rate.
- We propose a very efficient precomputation-based mechanism that can significantly increase the compression rate of the relative error bounded compression in SZ. Our solution eliminates the time cost of logarithmic transformation, replacing it with a fast table-lookup method in the point-by-point processing stage. Specifically, we construct two tables, a precision table *T2* and a look-up table *T1*, respectively. The inputs of *T2* include a user-specified point-wise relative error bound ϵ, interval capacity L, and a user-specified grid size reduce coefficient p. With a few simple arithmetic operations on such parameters, we construct a table *T2*, which would be further used to construct table *T1* for data compression. The table *T1* can quickly determine whether a data point is a predictable or not, and return its quantization factor value if yes. For a predictable data point, a quick *T1* lookup along with a simple bit shifting operation can decide its exponent and mantissa indices to restore it into floating-point data. With detailed analysis on the complicated mathematical relations among quotient values, quantization factor values, and error bounds, our optimized precomputation-based mechanism can strictly respect specified error bounds to achieve accurate error control.
- We also develop a precomputation-based mechanism for Huffman decoding in SZ by constructing three precomputed tables. We replace the costly repeated tree traversing operations during Huffman decoding with efficient table lookup operations, significantly accelerating the decompression process.
- We perform a comprehensive evaluation using four well-known real-world simulation datasets across different scientific domains. Experiments show that our solution would not degrade any compression quality metric, including maximum relative error, peak signal-to-noise ratio (PSNR), and visual quality. Our precomputation-based mechanism improve the compression rate by about 30% and decompression rate by about 70% in most of cases. Our codes are available at https://github.com/Borelset/SZ, which will be merged into the SZ package in the future.

The remainder of this paper is organized as follows. In Section II, we discuss related work. In Section III, we present the time cost of logarithmic transformation and quantization factor calculation, which motivates us to conduct this research. In Section IV, we present the design and implementation of our precomputation and table lookup based logarithmic transformation, quantization factor calculation, and Huffman decoding schemes. In Section V, we evaluate our new schemes with multiple real-world scientific HPC application datasets across different domains, comparing our scheme with the latest SZ and ZFP. In Section VI, we conclude our work, and discuss the future work in this research domain.

II. Related Work

Lossless compression fully maintains data fidelity, but it depends on the repetition of symbols in the data sources. Even for slightly variant floating-point values, their binary representations may hardly contain identical symbols (or exactly duplicated chunks). As such, lossless compression suffers from very low compression ratio [2], [9], [13], [20], [21] on scientific data.

General acceptance of precision loss provides an opportunity to drastically improve data compression ratio; ISABELA [22], SZ [2], [23], and ZFP [20] are three state-of-the-art lossy compressors supporting point-wise relative error bounds.

ZFP [20] follows the classic texture compression for image data. Working in 4^d (where d is the number of dimensions) sized blocks, ZFP first aligns the floating-point data points within each block to a common exponent and encodes exponents. Then, a reversible orthogonal block transform (e.g., discrete cosine transform) is applied to the signed integers in each block. Such a transform is carefully designed to mitigate the spatial correlation between data points, with the intent of generating near-zero coefficients that can be compressed efficiently. Finally, embedded coding [24] is used to encode the coefficients, producing a stream of bits that is roughly ordered by their impact significance on error, and the stream can be truncated to satisfy any user-specified error bound.

Motivated by the reduction potential of spline functions [25], [26], ISABELA [22] uses B-spline based curve-fitting to compress the incompressible scientific data. Intuitively fitting a monotonic curve can provide a model that is more accurate than fitting random data. Based on this, ISABELA first sorts data to convert highly irregular data to a monotonic curve. Its biggest weakness is pretty slow compression/decompression because of its expensive sorting operation.

SZ compressor has experienced multiple revolutions since the very first version 0.1 [23] was released in 2016. SZ 0.1 employed multiple curve-fitting models to compress data streams, with the goal of accurately approximating the original data. SZ 0.1 encoded the bestfit curve-fitting type for each data point or mark the data point as unpredictable data if its value is too far away from any curve-fitted value. SZ 1.4 [2] significantly enhanced the compression ratios by improving the prediction accuracy with a multi-dimensional prediction method plus a linear-scaling quantization method. SZ 2.0 [27] further improved the compression quality for the high-compression cases by leveraging an adaptive method facilitated with two main candidate predictors (Lorenzo and linear regression). Note that the classic SZ framework [2] did not support point-wise relative error bound. As such, Liang et al. [21] proposed an efficient logarithmic transformation to convert the pointwise relative-error-bounded compression problem to an absolute-error-bounded compression problem. However, as we have confirmed in our performance profiling, this will significantly slow down the compression and decompression because of its costly logarithmic transformation operations.

There are also some existing studies working on combining different lossy compressors to obtain better compression quality. Lu et al. [13] conducted a comprehensive evaluation based on SZ and ZFP, and proposed a simple sampling method to select the best compressor with higher compression ratio in between. Tao et al. [15] proposed an efficient online, low-cost selection algorithm that can predict the compression quality accurately for SZ and ZFP in early processing stages and selects the best-fit compression based an important statistical quality metric (PSNR) for each data field. Their work, however, relies on the compression performance of SZ and ZFP, so their compression result would never go beyond the best choice from between SZ and ZFP.

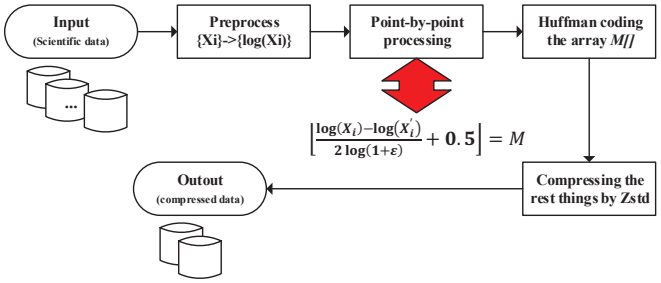


Fig. 1. The generate workflow of SZ Compression with log transformation.

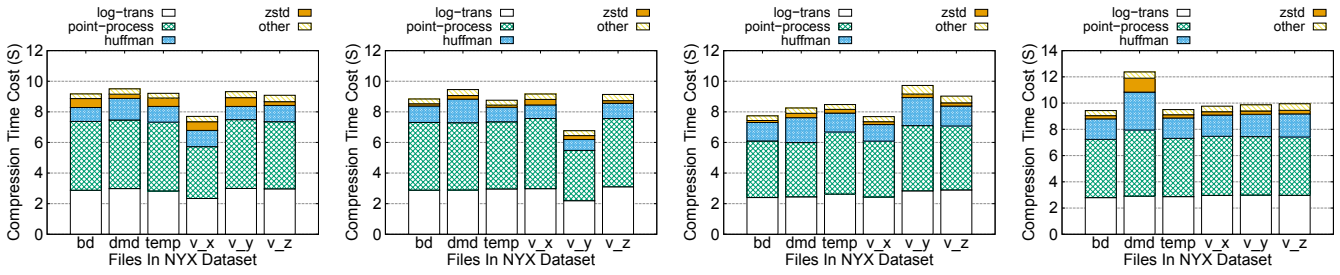
In this work, we propose a precomputaion-based mechanism that can avoid the costly online logarithmic transformation while maintaining the high compression ratios of SZ in point-wise relative error bounded compression. Our optimization accelerates SZ compression rate by up to 70%, and decompression rate by up to 100%, leading to an optimal compressor with both the best compression/decompression rate and compression ratios in the relative error bounded compression.

III. Motivation

Compared with other lossy compression schemes, SZ usually achieves the highest compression ratio [13], making it one of the best compressors for HPC scientific data. Generally, given a user-predefined absolute error bound, SZ performs the compression based on the following four steps:

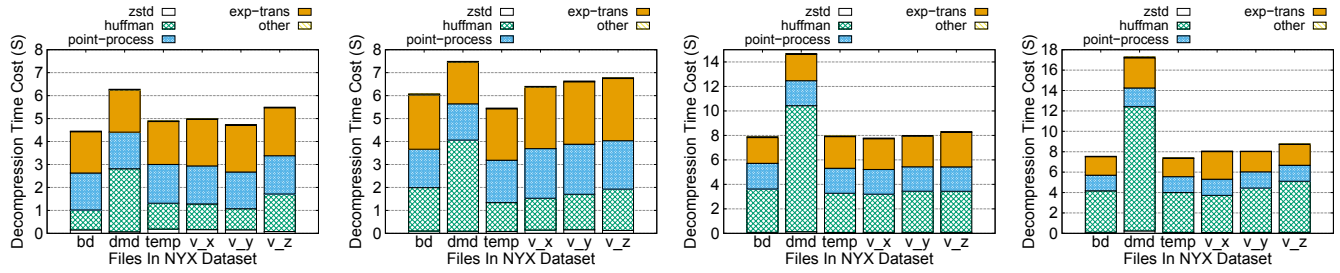
- Applying prediction to the given dataset based on user-set error bound: all the floating-point data values are mapped to an array of quantization factors (integer values), with an accuracy loss restricted within the error bound.
- Constructing a Huffman tree of the quantization factors, and encoding the quantization factors.
- Compressing the unpredictable data points (i.e., the data points whose values cannot be approximated by the first step) by binary-representation analysis.
- The compressed data generated by the above three steps would be further compressed with lossless compressors such as GZip [28] or Zstandard [29] (or called Zstd).

The above SZ compression framework works particularly effective on absolute error bounded compression, but cannot be directly applied to relative-error bounded compression, in which the absolute error bounds are actually dependent on



(a) 1.00E-01 (b) 1.00E-02 (c) 1.00E-03 (d) 1.00E-04

Fig. 2. Break-down of the compression time for SZ with logarithmic transformation (using relative error bound on NYX dataset).



(a) 1.00E-01 (b) 1.00E-02 (c) 1.00E-03 (d) 1.00E-04

Fig. 3. Break-down of the decompression time for SZ with logarithmic transformation (using relative error bound on NYX dataset).

the data values. To address this problem, the solution is to convert the relative error bounded compression problem to an absolute error bounded compression. To this end, the SZ team explored two transformation strategies. The strategy 1 [30] (called block-wise transform) is to split the whole dataset into multiple small blocks and compute the minimum absolute error bound based on user-required relative error bound for each block. Such a strategy may over-preserve the compression errors especially when the data exhibits spiky changes in local regions. To address this issue, the SZ team further developed another strategy [21] - a logarithmic transformation to replace the block-wise transform, which can significantly improve the compression ratio for point-wise relative error bound. As shown in Figure 1, the logarithmic transformation maps all the original values to the logarithmic domain, such that the point-wise relative error bounded compression is converted to an absolute error bounded compression. After that, the transformed data would be compressed by an absolute error bounded compression. After that, the transformed data would be compressed by an absolute error bound derived based on the user-required relative error bound.

A serious weakness of the logarithmic transformation based strategy is its high transformation cost, which may significantly lower the compression/decompression rate. As Figure 2 demonstrates, the logarithmic transformation consumes about 30% of the overall compression time, and the point-by-point quantization factor calculation takes another 30% of the total time during compression. Similarly, the inverse logarithmic transformation is also very time-consuming during decompression, as shown in Figure 3. To avoid such high transformation cost, we propose an efficient precomputation-based mechanism with a fast table-lookup method, dramatically accelerating SZ compression and decompression.

IV. A Precomputation-based Transformation Scheme for Lossy Data Compression

In this section, we present the theories and practices of the precomputation-based transformation scheme. First, we prove the feasibility of replacing logarithmic transformation and floating-point quantization procedures in previous SZ design (denoted as SZ_T) with an efficient precomputation-based table lookup procedure (denoted as SZ_P). Second, we discuss how to construct the tables for the precomputation-based transformation. Finally, we provide a detailed algorithm description of our approach. Table I lists and describes some frequently used symbols in this section.

A. Theories of Precomputation-based Transformation Scheme

As shown in Figure 1, in the latest SZ method with logarithmic transformation (SZ_T), the first step is to convert each data X_i to $\log X_i$ for the purpose of transforming a point-wise relative-error-bounded lossy compression problem to an absolute-error-bounded lossy compression problem. Next, in the point-by-point processing stage, SZ_T processes each transformed data point $\log X_i$ using the following equation.

$$\lfloor \frac{\log X_i - \log X_i'}{2\log(1+\varepsilon)} + 0.5 \rfloor = M, \tag{1}$$

In this equation, ε is a user-predefined relative error bound, M is an derived integer called quantization code or factor,

TABLE I
IMPORTANT AND FREQUENTLY USED SYMBOLS AND THEIR MATHEMATICAL NOTATIONS.

Terms	Explanations
f	float $f = X_i/X_i^{'}$, X_i is an original value, $X_i^{'}$ is its predicted value
M	The quantization code, an integer, which is located in a fixed range L
L	The range of quantization code M specified by users
V	The range covered by all of $PI(M)$ or $PI'(M)$
ε	The error bound (i.e., precision) specified by users
PI(M)	An interval where any float x located in, and x could be presented by $(1+\varepsilon)^{2M}$ with a relative error smaller than ε. This is used for Model A in SZ_P′
θ	A parameter used in Model B to control the intersection size
p	A parameter specified by users, to define θ as 2^{-p}
PI′(M)	An interval where any float x located in, and x could be presented by $(1+\varepsilon)^{2(M-\theta)}$ with a relative error smaller than ε. This is used for Model B in SZ_P
$T1$	Store the mapping relation of $f \rightarrow M$ for compression, consisting of several sub-tables
$T2$	Store the mapping relation of $M \rightarrow f$ for decompression
sub-table	Divide V into several segments by the exponent part of floating-point values in V, each segment corresponds to a sub-table.
grid	Divide each segment into several equal-sized girds, each grid maps to a sub-table entry
Δ	The intersection of two neighboring PI(M). We ensure $\Delta >$ size of the grid, for error control

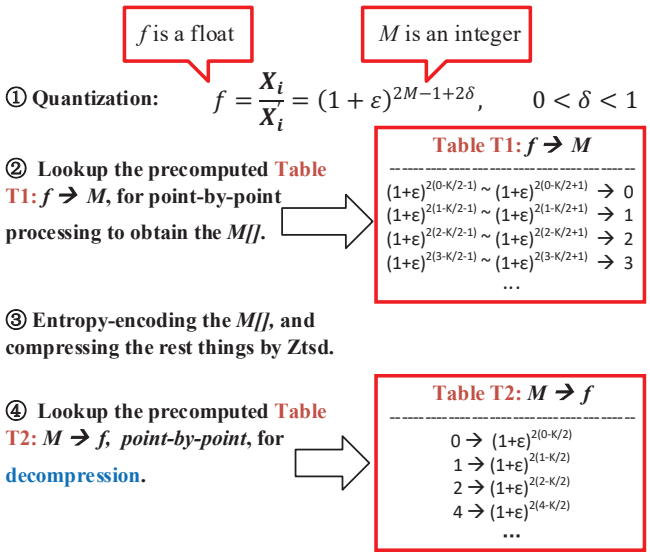


Fig. 4. Our approach uses two precomputed tables to directly transform the error-controlled quantization codes f[] int integers M[].

which can be further encoded using Huffman coding to reduce storage space. M is also used in decompression. $X_i^{'}$ is the predicted value of X_i (using one-dimensional 1D data as an example). For calculating $X_i^{'}$ on 2D and 3D data, readers can refer to the work by Tao et. al. [2] for more details.

Figure 4 demonstrates how the precomputation-based table lookup in SZ_P can avoid the time-consuming logarithmic transformation and quantization factor calculation in SZ_T. We provide a detailed theoretical analysis of our SZ_P method.

Based on Equation (1), we can deduce Equation (2), and subsequently Equation (3) as follows.

$$\frac{\log X_i - \log X_i^{'} + \log(1+\varepsilon)}{2\log(1+\varepsilon)} = M + \delta, \quad 0 < \delta < 1 \tag{2}$$

$$\Longrightarrow \quad f = \frac{X_i}{X_i^{'}} = (1+\varepsilon)^{2M-1+2\delta}. \quad 0 < \delta < 1 \tag{3}$$

Equation (3) indicates that for any floating-point f, we can get an integer M (M belonging to a fixed and predefined range, namely, interval capacity $\boldsymbol{L}$). Because $0 < \varepsilon < 1$, we can get $2M - 1 < 2M - 1 + 2\varepsilon < 2M + 1$. From Equation (3), we can deduce

$$(1+\varepsilon)^{2M-1} < f < (1+\varepsilon)^{2M+1}. \tag{4}$$

That's for any floating-point data f, there must be an integer M that satisfies Equation (4). This is how SZ_T works in the stage of point-by-point processing. We propose a new model to optimize this procedure.

Basic model (Model A) of SZ_P

Inspired by Equation (4), the new model calculate the difference of two neighboring data points as

$$f = \frac{X_i}{X_i^{'}} \approx (1+\varepsilon)^{2M}, \tag{5}$$

We ensure $1-\varepsilon < \frac{(1+\varepsilon)^{2M}}{f} < 1+\varepsilon$ for error control purpose. For decompression, we can rebuild X_i from $X_i^{'}$ and M based on $X_i = X_i^{'} \times (1+\varepsilon)^{2M}$.

This provides us a chance to avoid logarithmic transformation and complicated quantization factor calculation while achieving the same compression efficiency. The last challenge to implement our model is to find the $(1+\varepsilon)^{2M}$ (where M is an integer) that is close enough to f as shown in Equation (7). We solve the problem with a precomputed *table* (i.e., **T1** as shown in Figure 4) as discussed below.

For the precomputed *table* **T1**, what we require is that for any f we can get a corresponding integer M (M belongs to $\boldsymbol{L}$) that satisfies the following requirement in order to meet the error bound ε requirement after decompression.

$$1-\varepsilon \leq \frac{(1+\varepsilon)^{2M}}{f} \leq 1+\varepsilon \tag{6}$$

From Equation (6) we can further deduce that

$$(1+\varepsilon)^{2M-1} \leq f \leq \frac{(1+\varepsilon)^{2M}}{1-\varepsilon}. \tag{7}$$

We call this interval $[(1+\varepsilon)^{(2M-1)}, \frac{(1+\varepsilon)^{2M}}{1-\varepsilon}]$ as M's present interval, denoted by **PI(M)**. For any pair **PI(M)** and **PI(M+1)**, we can get their top and bottom boundaries (see the example shown in Figure 5) as follows.

$$\begin{aligned} PI(M)_{top} &= \frac{(1+\varepsilon)^{2M}}{1-\varepsilon} \\ PI(M+1)_{bottom} &= (1+\varepsilon)^{2M-1} \end{aligned} \tag{8}$$

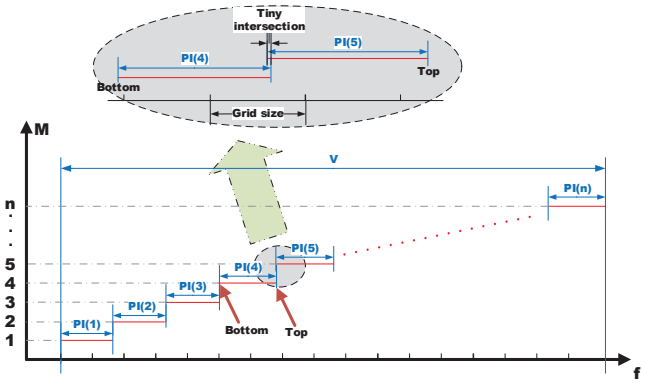


Fig. 5. A general description about model A of SZ_P.

Also, we can get

$$\frac{PI(M)_{top}}{PI(M+1)_{bottom}} = \frac{\frac{(1+\varepsilon)^{2M}}{1-\varepsilon}}{(1+\varepsilon)^{2M-1}} = \frac{1}{1-\varepsilon^2} > 1. \quad (9)$$

Further, we can get the following relations.

$$PI(M)_{bottom} < PI(M+1)_{bottom} < PI(M)_{top} < PI(M+1)_{top}$$

It indicates that two neighboring **PI**s are intersecting (see the example as shown in Figure 5), although the intersections are extremely narrow. Also, $(1+\varepsilon)^{2M}$ belongs to a fixed range because M belongs to a fixed and predefined range.

Based on the above facts, we can learn that **PIs** cover bounded continuous intervals, called ***V***. We can then rasterize ***V*** to build the *table* **T1**, where each grid corresponds to a table entry. Thus, we can build a mapping relation between f (i.e., **PI**(n)) and M (i.e., n). During compression, for each input floating-point f (based on Equation 3), we can calculate in which grid it locates, then getting the value M from *table* **T1**. As a result, f can be represented by $(1+\varepsilon)^{2M}$, which can restore the value of f during decompression respecting the predefined error bound.

Figure 5 summarizes the above mentioned method, namely, building the *table* **T1** to replace the log transform and quantization factor calculation in SZ_T, at the same time achieving an acceptable error level. We call this basic method Model **A** in the paper.

The challenge of adopting Model **A** is that the mapping relation between f (i.e., the grid) and M (i.e., n) is $N \rightarrow N$, which results in higher relative errors compared with the user-required error bound. The reason is that there exist some grids divided by two **PI**s (as shown in Figure 5). Thus, no matter how we decide these grids belongs to which **PI**s, it always leads to higher relative errors than the error bound ε, which makes our compressor *non-error-bounded*. Model **A** can be further optimized to strictly respect the error bound. We continue discussing it.

Advanced model (Model B) of SZ_P

To address the not strictly respecting error bound issue of Model **A**, we propose a new design of **PI**′ (namely, Model **B**) that has more intersecting parts to ensure data are strictly error-bounded after decompression.

In Model **B**, we shift **PI**s (from Equation 7) and enlarge their intersections. Specifically, we define the new f' and **PI**′ as

$$(1+\varepsilon)^{M(2-\theta)-1} \leq f' \leq \frac{(1+\varepsilon)^{M(2-\theta)}}{1-\varepsilon}, \quad 0 < \theta < 1, \quad (10)$$

where θ is introduced for enlarging the intersecting parts. Due to **PI**′**(M)**$_{top}$ > **PI**′**(M)**$_{bottom}$, **PI**′s are intersecting, so they can also cover bounded continuous intervals ***V*** (as shown in Figure 6). About the relation between a grid and a **PI**′, we present the following lemma.

Lemma 1. *If a grid size* **G** *is smaller than the size of any intersecting part of* ***PI***′*, a* ***PI***′ *completely including the grid always exists.*

Proof. Considering the intersecting size of two neighboring **PI**′s:

$$\begin{aligned} PI'(M)_{top} &= \frac{(1+\varepsilon)^{M(2-\theta)}}{1-\varepsilon} \\ PI'(M+1)_{bottom} &= \frac{(1+\varepsilon)^{(M+1)(2-\theta)}}{1+\varepsilon} \end{aligned} \quad (11)$$

$$\begin{aligned} \Delta &= PI'(M)_{top} - PI'(M+1)_{bottom} \\ &\geq (1+\varepsilon)^{M(2-\theta)+1}(1-(1+\varepsilon)^{-\theta}) \end{aligned} \quad (12)$$

As the condition provided by Lemma 1 (as shown in Figure 6), we assume that all the grid sizes are smaller than any of the Δ. We can further prove the lemma with all of the three possible cases of the grids as follows.

1) A grid, in which there is no **PI**′'s boundary, is completely included in a **PI**′. Due to **PI**′s cover a bounded continuously interval, obviously this kind of grid is completely included in a **PI**′.
2) A grid, in which there is only one of the **PI**′'s boundaries, is completely included in a **PI**′. Without loss of generality, we assume that it is **PI**′(n+1)'s bottom boundary and it is a float number K, and the grid is ***G***. So **PI**′(n)'s top boundary is $K+\Delta$. Because the size of grid G is smaller than Δ, so G is completely included in **PI**′(n). Thus, this kind of grid is also completely included in a **PI**′. Same conclusion can be made assuming that the boundary is **PI**′(n-1)'s top boundary.
3) A grid, in which there are two or more **PI**′'s boundaries, does not exist. Because the size of **PI**′ is bigger than Δ and the Δ is bigger than the grid size, so there does not exist a grid which includes two or more **PI**′'s boundaries.

□

Therefore, Lemma 1 demonstrates that the mapping relation between f (i.e., the grid) and M (i.e., n) in Model **B** is $N \rightarrow 1$, and Model **B** can avoid the problem appearing in Model **A**, that is, relative errors higher than predefined bound.

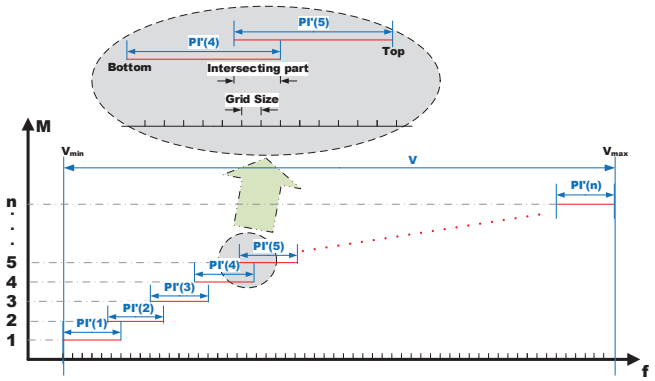


Fig. 6. A general description about model B of SZ_P.

Compared with **PI**, tight arrangement of **PI**′ can lead to a better precision, but also result in a smaller size of cover range V. It depends on the error bound ε, a smaller ε can lead to a similar compression ratio to Model **A**, but also a larger total grid count, which means higher memory cost. Empirically, θ should be set 1, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, and so on, which makes it easier to decide grid size in Model **B**. We finally adopt Model **B** in SZ_P, which brings us a high-precision error control but a lightly decrease in compression ratio. It is worth noting that in both of Model **A** and Model **B**, once the table is generated, it can be saved for repeated future use, because the table is only related to the user-set error bound ε, instead of the values of input datasets.

B. Tables Construction with Model B

We further discuss how to build the *table* **T1** with Model **B**, namely, building the mapping relation between f (i.e., $\frac{X_i}{X_i'}$) and M (i.e., the error-controlled quantization code). We mainly discuss the construction of *table* **T1** here since *table* **T2** can be easily calculated and constructed as Algorithm 1 demonstrates.

In *table* **T1**, for a given error bound ε and an interval capacity ***L***), we divide the range **V** into many equal-size grids. As discussed in Lemma 1, due to the grid size must be smaller than the intersecting size of two smallest neighboring **PI**′s, we assume the size of smallest **PI**′ is S. Therefore, the grid size should be designed based on half of S, so that θ is 1, as shown in Equation 10. Meanwhile, θ decides the size of intersecting parts of **PI**′s. If θ is 1, it means any **PI**′(n) should be covered by two neighbors (i.e., **PI**′(n+1) and **PI**′(n-1)), so grid size should be set smaller than $\frac{S}{2}$. If θ is $\frac{1}{2}$, the grid size should be set smaller than $\frac{S}{4}$, and so on.

Furthermore, the index number of f in *table* **T1** can be calculated as $\frac{f-V_{min}}{gridsize}$. Consider the cost of division operations, in order to save time on getting the index number of f as calculated above, the grid size should be simple in binary system, hence, we just keep one valid number for the grid size in IEEE 754 format [31]. For example, we can transform the $\frac{f-V_{min}}{1.101*2^{-5}}$ to $\frac{f-V_{min}}{1.000*2^{-5}}$ (in order to make the grid size smaller and also satisfy the Lemma 1), which is equal to $(f - V_{min}) * 2^5$. Therefore, we replace costly division operations with more efficient bitwise operations.

The exponential model such as $(1+\varepsilon)^{M(2-\theta)}$ in Equation (11) and (12) has a huge growth rate, and the grid size is defined based on this, controlling the size of the table is another challenge. We discuss how we overcome this.

Since floating-point data in computer systems are saved in IEEE 754 format [31], we divide range V into segments according to f's exponent, and build sub-tables for each segment separately. For the segment where f's exponent is k and the smallest $(1+\varepsilon)^{M(2-\theta)}$ in Seg_k is S, we define the grid size ***G*** as

$$G = 1.0 \times 2^k \times \varepsilon \times \theta.$$

Note that we only keep one valid number in IEEE 754.

Because exponent part of any $(1+\varepsilon)^{M(2-\theta)}$ in Seg_k is k, so any S is larger than 1.0×2^k. And for Seg_k, we can get

$$\begin{aligned}\Delta &= (1+\varepsilon)^{(M(2-\theta)+1)}(1-(1+\varepsilon)^{(-\theta)}) \\ &= S(1+\varepsilon)(1-(1+\varepsilon)^{(-\theta)}).\end{aligned}$$

Due to $S > 1.0{\cdot}2^k$ and $(1+\varepsilon)(1-(1+\varepsilon)^{(-\theta)}) \geq \varepsilon{\cdot}\theta$, we can know $\Delta >$ **G**, so ***G*** is a reasonable grid size that satisfies the requirement discussed in Lemma 1. And, according to Lemma 1, if the grid size is ***G***, any grid in Seg_k always could find a **PI**′ that completely include grid. We can figure out that the size of sub-table of Seg_k is $\frac{the\ size\ of\ Seg_k}{G}$. To better illustrate this, we calculate them separately as follows.

1) The size of Seg_k is $1.111... \times 2^k - 1.0 \times 2^k = 1.0 \times 2^k$.
2) Convert the ε to binary representation, and keep one valid number (e.g. if $\varepsilon = 0.01$ in decimal, $\varepsilon_c = 0.0000001$ in binary).
3) Choose a power of 2 that less than 1.0 as θ, such as 0.5, 0.25, 0.125 etc., denoted as 2^{-p}.

Thus, $\boldsymbol{G} = 1.0 \times 2^k \times \varepsilon_c \times 2^{-p}$ and the size of sub-table for *table* **T1** is $2^p/\varepsilon_c$. It is obviously that the size is totally determined by θ and ε. For example, if $\theta = 0.25$ and $\varepsilon = 0.01$, the size of sub-table is $2^2 \times 2^7 = 2^9 = 512$. So the size of sub-table is in control. On the other hand, the count of sub-table in *table* **T1** is determined by **L** (i.e., the range of quantization code **M**), which is about 10 in general.

Overall, with setting different grid size for different segments (i.e., the sub-tables), we could control the total size of *table* **T1** in an acceptable level. Generally, in our final implementation and evaluation in Section V, **T1** sizes are only about 131 KB, 884 KB, 2,848 KB, 4,608 KB for the ε = 0.1, 0.01, 0.001, and 0.0001, respectively. These memory footprints are ignorable for HPC servers.

C. Implementation Details With Model B

In this subsection, we provide the implementation details of SZ_P in Algorithm 1 and 2 with pseudo code descriptions.

First, we should build a precision *table* **T2**, saving all the $(1+\varepsilon)^{M(2-\theta)}$, and an inverse *table* **T1** which is used to find a M for a float f, as described in last subsection. To save sub-tables from different segment for **T1**, we design the following data structures:

```
struct TopTable{
  uint16_t topIdx; /*the biggest exponent in V*/
  uint16_t btmIdx; /*the smallest exponent in V*/
  int bits; /*define the relative size of grid size*/
  struct SubTable * subTablePtr; /*sub-table array*/
}
struct SubTable{
  uint32_t * grids; /*array of grid to M*/
}
```

For the 'TopTable' in **T1**, it should save the biggest and smallest exponent of $(1+\varepsilon)^{M(2-\theta)}$ as 'topIdx' and 'btmIdx', as well as the pointer 'subTablePtr' to its sub-tables. The exponent of error bound ε required is also saved in top-level table as an important information 'bits'.

Based on the above data structures and as shown in Algorithm 1, we first build the *table* **T2**, and then build build *table* **T1** according **T2**. The size of **T1** is determined by T2[L-1]'s exponent - T1[0]'s exponent, and the size of sub-tables of **T1** is decided by error bound ε's exponent and a user-specified parameter p, as: $2^{-(\varepsilon's\ exponent)+p}$. After that, we traverse the grids in T1, and calculate each grid's bottom and top boundaries to determine which **PI**$'$ the processed grid completely belongs to. After processing all the grids in **T1**, building *table* **T1** is done.

Algorithm 2 describes the point-by-point processing stage in SZ_P. We get the error-controlled quantization code M by looking up table T1, to determine the data Ds[i] whether is a predictable value. Thus, SZ_P avoids many logarithmic transformation operations in SZ_T while achieving nearly the same compression efficiency on HPC scientific data.

Decompression with SZ_P is an inverse process of compression. First, SZ_P decompresses data using Zstd as well as Huffman decoding (the same as SZ_T) to get the quantization factor arrary *M[]*. Then, SZ_P builds table *T2* to restore data. SZ_P uses quantization factors as indices to look up the *T2* to get floating-point values, which have been pre-computed using Equation 5. These floating-point values are not the values of original datasets, but the quotients of neighboring points, which can be further used to restore the original data with simple multiply operations.

D. Optimizing Huffman Decoding

As discussed in Section III, Huffman encoding/decoding is a critical step in SZ. In SZ_P, we optimize the Huffman decoding performance in particular. Generally, Huffman decoding parses the bit stream according to a Huffman tree, which means the decoding process acts as a state machine (bit by bit processing). Meanwhile, in the situation of high precision, quantization factor values will no longer converge, but scatter, leading to a longer average Huffman code length, thus longer decoding time (see results in Figure 3). Based on our observation, there are many repeated calculations in Huffman decoding. Traversing always begins from the root of the Huffman tree, and different Huffman codes with the same prefix will have the same or partially overlapped traverse path. Many repeated traversing operations can be avoided, if we can design a precomputation-based mechanism for them.

Algorithm 1 Building tables *T1* and *T2* with model B.

Input: point-wise relative error bound, ε, interval capacity, L, and the parameter p to define θ;
Output: table $T2$ ($M \rightarrow f$) and table $T1$ ($f \rightarrow M$);

```
T2={0};
θ=2∧(-p);
for i=0 to L-1 do
    T2[i]=(1+ε)∧((i-L/2)*(2-θ));
end for
TopTable T1;
T1.btmIdx = T2[0]'s exponent; T1.topIdx = T2[L-1]'s exponent;
T1.bits = -(ε's exponent) + p;
index = 0; state = false;
subBoundary = (1<<T1.bits) - 1;  /*to define the last index of sub-table*/
for  i=0 to T1.topIdx - T1.btmIdx  do
    for  j=0 to subBoundary  do
        PI'top = T2[index]/(1-ε);
        PI'btm = T2[index]/(1+ε);
        gridBtm =((i+T1.btmIdx) ≪23) + (j≪(23-T1.bits));
        gridTop = ((i+T1.btmIdx) ≪23) + ((j+1)≪(23-T1.bits));
        /*In IEEE 754 format, 32bit float has 23 bits mantissa, i + T1.btmIndex is gridTop's exponent and j is its mantissa.*/
        if gridTop < PI'top && gridBtm > PI'btm then
            T1.subTablePtr[i].grids[j] = index;
            state = true;
        else if index < L-1 and state==true then
            index ++;
            T1.subTablePtr[i].grids[j]=index;
        else
            T1.subTablePtr[i].grids[j]=0;
        end if
    end for
end for
```

Algorithm 2 Point-by-point processing stage in SZ_P.

Input: the dataset $Ds\{\}$, a user-specified point-wise relative error bound ε and interval capacity L;
Output: compressed data stream M and the unpredicted bytes;

```
Build Tables T1 and T2;
M = {0};
pred = 0;
Process Ds[0] as an unpredicted float in SZ, pred = Ds[0]'; /*Ds[0]' is truncated from Ds[0] according to ε as SZ does [23].*/
length = size of Ds;
for i=1 to length-1 do
    ratio = Ds[i]/pred;
    index = 0;
    expoIdx = ratio's exponent;
    if expoIdx > T1.btmIdx && expoIndex < T1.topIdx then;
        mantiIdx = ratio's mantissa ≫ (23-T1.bits);
        index = T1.subTablePtr[expoIdx-T1.btmIdx].grids[mantiIdx];
    end if
    if index != 0  then
        M[i] = index;
        pred = pred * T2[index];
    else
        M[i] = 0;
        process Ds[i] as unpredicted in SZ, pred = Ds[i]'; /*Ds[0]' is truncated from Ds[0] according to ε as SZ does [23].*/
    end if
end for
```

Inspired by our aforementioned precomputed tables for logarithmic transformation, we process multiple bits as a prefix at one time, instead of bit by bit. As a result, we also design the precomputation-based tables for Huffman decoding in SZ_P as shown in Figure 7. We describe the tables in details as below.

- ***Node Table***: It records *prefix bits* $\rightarrow$ *Huffman node address* (or called sub-tree). Here we configure the length

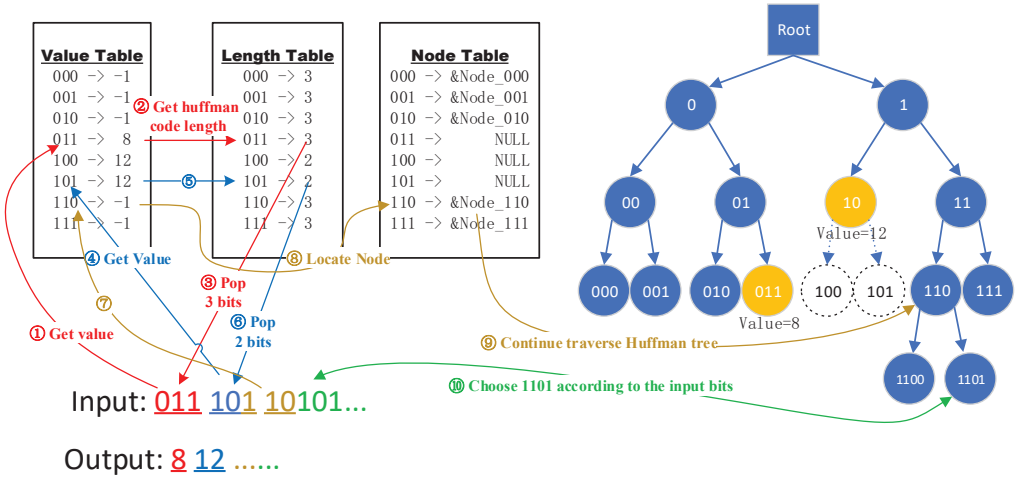


Fig. 7. An example of building precomputation-based tables to accelerate Huffman decoding in SZ_P.

TABLE II
COMPARISONS OF TRADITIONAL SZ METHODS AND OURS (SZ_P). HERE M IS THE ERROR-CONTROLLED QUANTIZATION CODE, ε IS THE ERROR BOUND (I.E., 10^{-2}, 10^{-3}, ETC.), X_i IS THE PROCESSED DATA AND $X_i^{'}$ IS THE PREDICTED VALUE OF X_i IN SZ COMPRESSION.

Approaches	General descriptions	Compression rate	Compression ratio	Data quality
SZ_PWR [30]	$\frac{X_i - X_i^{'}}{2\varepsilon} = M$	Low	Low	High
SZ_T [21]	$\frac{\log X_i - \log X_i^{'}}{2\log(1+\varepsilon)} = M$	Low	High	High
SZ_P	$\frac{X_i}{X_i^{'}} = (1+\varepsilon)^{2M}$	High	High	High

of prefix bits as a fixed number y and we construct the *Node Table* for all y prefix bits in the Huffman tree. Figure 7 provides an example of this *Node Table* (y=3), which records all possibilities of nodes named by 3 prefix bits in the Huffman tree. Thus, by looking up this table, we can directly read 3 bits each time and access the corresponding node, avoiding the traversal cost from level 0 to level 3 in the Huffman tree.

- ***Value Table***: It records *prefix bits* $\rightarrow$ *a possible Huffman code*. To build this table, we traverse all the y prefix bits in Huffman tree, and if it is a leaf node, we record the value of the leaf node in the entry mapping to the corresponding prefix, otherwise record *-1*.
- ***Length Table***: It records *prefix bits* $\rightarrow$ *the length of a Huffman code*. The reason we construct this table is that Huffman codes are variable-length such that some Huffman codes could be shorter than y.

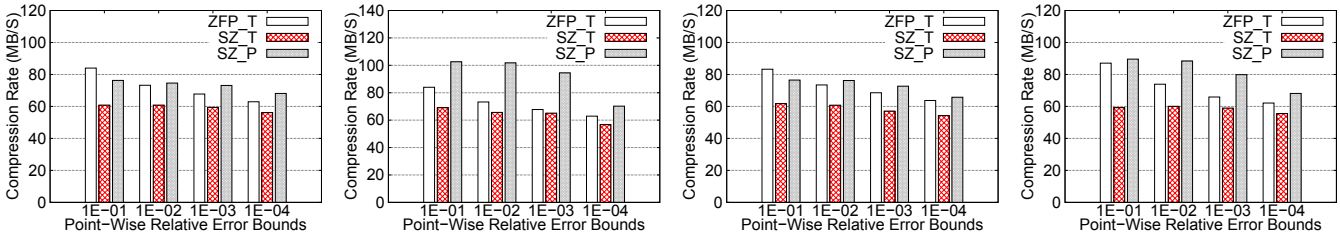
As the example shown in Figure 7, we process 3 bits each time (y=3). For the first 3 bits '011', we first look up the *Value Table* to obtain '011' $\rightarrow$ '8', knowing that it may be a Huffman code and its corresponding value is '8'. We further get its *length*=3 by looking up the *Length Table*. As a result, we output the value '8' for the prefix '011', and move the bit stream pointer 3 bits forward accordingly. For the next 3 bits '101', we look up *Value Table* to obtain '101' $\rightarrow$ '12', and then look up the *Length Table* to obtain *length*=2. Thus, we output the value '12' for the prefix '10'. Because *length*=2, we move the bit stream pointer 2 bits forward. The last '1' (of '101') and its next 2 bits '10' are treated as the next 3 bits '110'. We find '110' is not a Huffman code in the *Value Table* ('110' $\rightarrow$ '-1'), so we get the address of Huffman tree's node '110' from the *Node Table*, and traverse the Huffman tree according to the following (input) bits to get a longer Huffman code.

With these precomputed tables, we can significantly accelerate the decompression process of SZ_P by reducing tree traversing operations during Huffman decoding. Generally, the prefix length y is decided by the number of unique quantization factor values. We set $y \leq 16$, which means each table has 2^{16} entries at most, to get a tradeoff between the memory cost and performance acceleration. 2^{16} is usually large enough since it is a widely used quantization interval value. Therefore, the three tables cost about 851KB at most in our design, which is also ignorable for HPC servers.

E. Discussions

Table II shows the comparison of SZ_P and the traditional SZ compression appoarches. In summary, the key points of our idea in designing SZ_P is:

- Since the predicted values (X_i') are always expected to be close to the real values (X_i)($i.e.$, $\frac{X_i}{X_i'}$ is always distributed in a zone close to 1.0), we can get a new method with a precomputed table, which is mathematically equivalent to



(a) CESM Dataset (b) HACC Dataset (c) Hurricane Dataset (d) NYX Dataset

Fig. 8. Compression rate on given point relative error bound.

the logarithmic transformation used in SZ_T, and in the meahwile the size of precomputed table could be well controlled.

- Through our careful design in Model *B*, we make the error caused by the table look-up be always smaller than error bound ε, which means that the precision of compressing data by SZ_P could be controlled well.

Therefore, our optimization actually includes two parts: the log transformation and calculating quantization factors (i.e., records the relationships of the neighboring floats). We avoid the logarithmic transformation in SZ_T, and optimized the quantization process with table lookup operations. That is the key reason we speed up the point-wise relative error bounded lossy compression significantly.

Because ZFP [32] does not have the quantization factor calculation procedure, our current design cannot be directly applied to ZFP. More specifically, ZFP method saves transformation results of original values instead of the derived values of neighboring points.

V. Evaluation

In this section, we compare our approach (denoted by SZ_P) with two state-of-the-art methods, SZ_T and ZFP_T described in literature [21]. Compression and decompression rate, ratio, and data fidelity are the main metrics to measure compressors. We evaluate these metrics in quantitative ways.

A. Experimental Setup

We conduct our tests on a server with four 2.4GHz Intel Xeon E5-2640 v4 processors and totally 256GB memory. The HPC datasets are from multiple domains, like HACC cosmology simulation (1D), NYX cosmology simulation (3D), Hurricane ISABEL simulation (3D) and CESM-ATM climate simulation (2D). The sizes of these four datasets are 6.3GB, 3.1GB, 1.9GB, and 2.0GB per snapshot for each application, respectively. We respect data dimensions during compression. For example, NYX cosmology simulation dataset is 3D, so we conduct 3D compression on it.

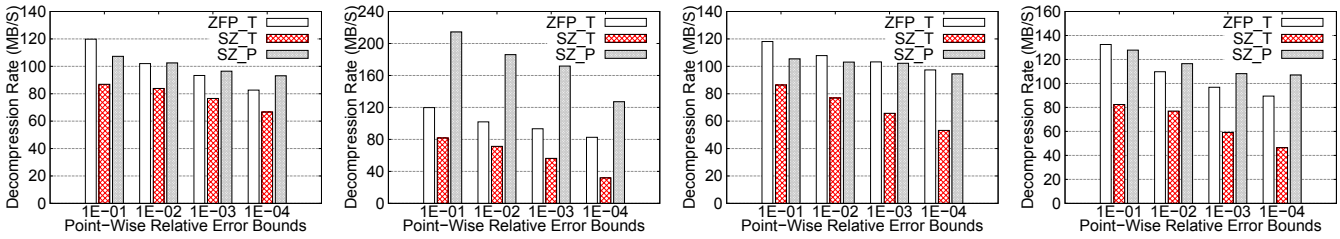
The data fidelity of lossy compression approaches is measured with multiple metrics including the max point-wise relative error (MAX E), which shows whether a method is respecting the error bound, NRMSE (normalized root mean square error, smaller is better), which shows the statistical difference between original values and the values decompressed, and PSNR (peak signal to noise ratio, bigger is better), which measures the statistical errors introduced by lossy compression compared with the value range.

Compression rate and decompression rate indicate the throughput of compression and decompression. For compression and decompression rate, we run each experiment for five times to calculate the average. Since each application involves many fields, each in a data file, we use the aggregated file size to divide by the total compression or decompression time to calculate the rate. Compression ratio is the ratio of the total original data size to the total compressed size. In addition, SZ_P needs a new configuration item named 'plus_bits', which is θ in equations (10) as discussed in Section IV-A, and should be set in the configuration file. A bigger θ will lead to lower quality of data, but a higher compression ratio. According to our observation, plus_bits = 3 is recommended and used in our final experiments as to make a good trade-off between the data fedility and compression ratio.

B. Compression & Decompression Rate

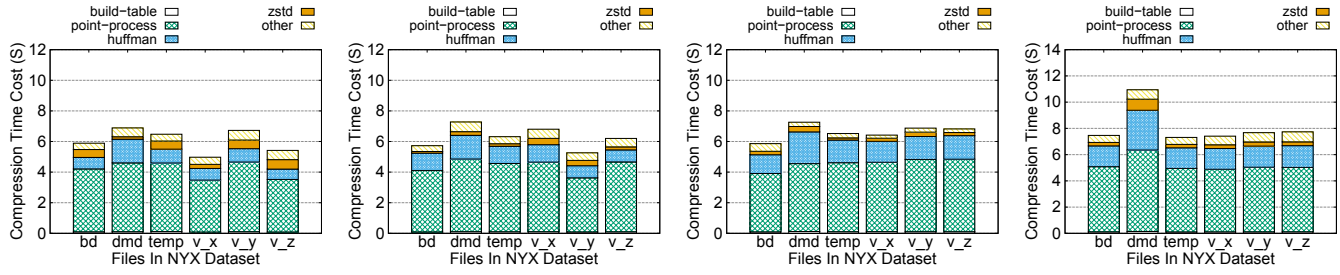
Figures 8 and 9 present the compression rate and decompression rate of SZ_P, SZ_T and ZFP_T on the four datasets, respectively. Generally, SZ_T has the lowest compression rate among the three approaches, because of its logarithmic transformation and floating-point quantization time cost. The compression rate is a particular advantage of ZFP_T, since the implementation of ZFP itself has been optimized for the speed purpose. As observed in the figure, ZFP_T exhibits a 20% to 30% higher compression rate than that of SZ_T. In most cases, SZ_P's speed is about 1.2× to 1.5× of SZ_T on compression, which is attributed to the performance gain of our new table lookup method. As Figure 9 demonstrates, the decompression rate of SZ_P is also about 1.3× to 3.0× as that of SZ_T because of the following two reasons. (1) SZ_P effectively avoids the costly exponential transformation, which is the inverse transformation of logarithmic transformation during decompression, (2) SZ_P significantly reduces the traversing operations during Huffman decoding by constructing three precomputed tables according to the Huffman tree as discussed in Section IV-D.

From Figure 9, it can also be observed that the decompression performance gain of SZ_P over other lossy compressors differs with the error bound, also depending on datasets. Specifically, SZ_P's decompression rate is equivalent to or



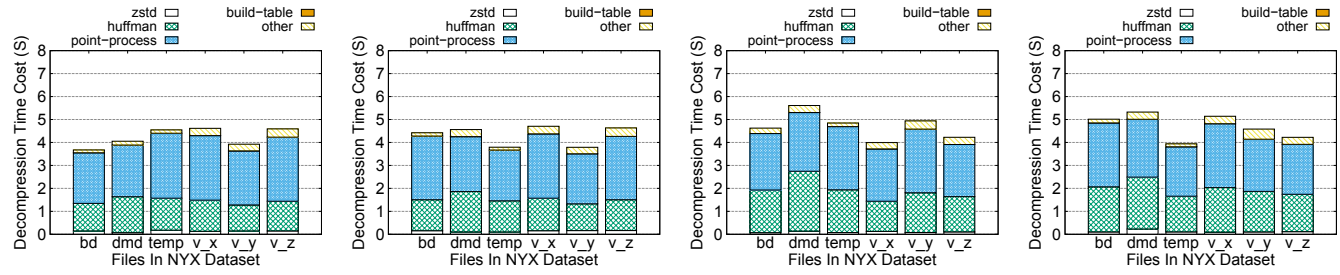
(a) CESM Dataset (b) HACC Dataset (c) Hurricane Dataset (d) NYX Dataset

Fig. 9. Decompression rate on given point relative error bound.



(a) 1.00E-01 (b) 1.00E-02 (c) 1.00E-03 (d) 1.00E-04

Fig. 10. Break-down of the compression time for SZ_P on NYX dataset.



(a) 1.00E-01 (b) 1.00E-02 (c) 1.00E-03 (d) 1.00E-04

Fig. 11. Break-down of the decompression time for SZ_P on NYX dataset.

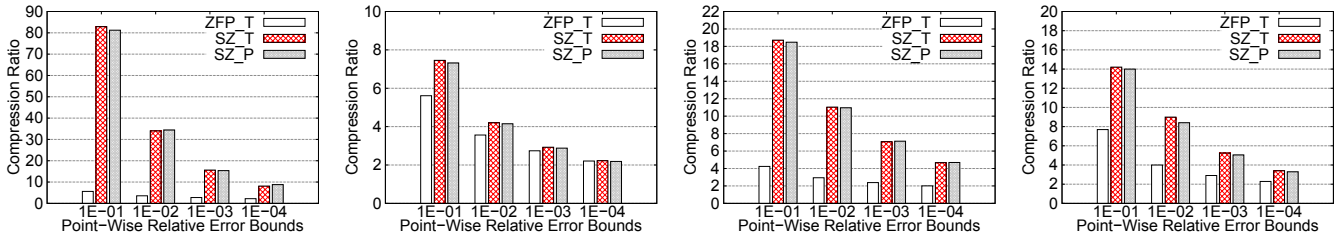
even higher than ZFP_T's decompression rate in most of cases. The key reason is that the exponential transformation and Huffman decoding takes the major portion (more than $\frac{2}{3}$) of the total decompression time for SZ, and SZ_P greatly improve reduce the time cost on these two steps in decompression. This can be confirmed in Figure 3. Note that SZ_P has the highest decompressing rate on HACC dataset. This is because HACC dataset has the lowest compression ratio and thus need longer time for Huffman decoding while SZ_P greatly accelerates Huffman decoding as discussed earlier.

We further present more detailed evaluation of SZ_P time consumption in Figures 10 and 11. From the Figure 10, we can see that SZ_P totally eliminates the time cost on logarithmic transformation in comparison with SZ_T (see Figures 2 and 3 in Section III). This explains why SZ_P achieves much higher compression and decompression rate than SZ_T. Meanwhile, Figures 10 also suggests that the time cost for building tables of our precomputation-based mechanism only occupies less than 5% of the total time. In addition, the building tables in SZ_P are independent of the dataset size, which means this overhead can be amortized, thus ignorable when compressing large datasets. From the Figure 11, we can see that SZ_P also greatly reduces the time cost of Huffman decoding thus achieve much higher decompression rate as show in Figure 9.

C. Compression Ratio

Figure 12 shows the compression ratio of ZFP_T, SZ_T, and SZ_P with the four most widely used point-wise error bounds (0.1, 0.01, 0.001, 0.0001) on the four datasets. Through the figure, we can see that the SZ_P has very similar compression ratios with SZ_T, which is often much higher (even up to one order of magnitude in some cases) than that of ZFP_T. We explain the key reasons as follows.

Unlike SZ_T whose spacing of quantization codes is twice as large as error bound, SZ_P has a smaller spacing of quantization codes - $2*\theta$ times of the error bound ε, which thus slightly decreases the compression ratio. The degree of



(a) CESM Dataset (b) HACC Dataset (c) Hurricane Dataset (d) NYX Dataset

Fig. 12. Compression ratio on given point relative error bound.

decrease depends on the θ value: the case using smaller θ would achieve a smaller decrease but the table size will be increased (because θ determines the grid size). We use θ = 1/8 for SZ_P in Figure 12, which only decreases a little in compression ratio. As shown in the figure, SZ_P achieves a similar compression ratio as SZ_T, and if we use a much smaller θ value, the difference will be further decreased.

Table III shows the compression ratio of SZ_T and SZ_P with high precision (i.e., low error bound) on four datasets. In general, SZ_P achieves about 1.20× compression ratio as large as that of SZ_T. As Liang et. al. [21] suggest, to avoid the round-off error caused by logarithmic transformation for compression and its inverse transformation for decompression, a correction (i.e., do a subtraction) must be introduced for the precision requirement. When the error bound is relatively large such as 0.1, the subtrahend is usually much smaller than the tolerable accuracy loss, the correction on precision will be not obvious, and it does not affect the compression ratio. But in the situation of setting extremely small error bounds, the derived precision value can also be very small, so the round-off value could be at the same order of magnitude as the precision value.

As a result, subtracting the round-off value from the precision value can dramatically reduce the precision value, causing a similar impact as tightening error bounds and thus a lower compression ratio. Our approach SZ_P does not have the preproccessing stage (i.e., X_i are transformed to $\log(X_i)$), thus, it does not need to do the precision correction. Therefore, SZ_P can avoid the compression ratio decrease caused by precision correction, having it achieve better compression ratios than SZ_T in low-error-bound cases. In addition, we also note that the compression and decompression rates of SZ_P and SZ_T with low error bound are nearly the same. This is because compared with SZ_P, the tightening error bounds of SZ_T would lead to lower prediction ratio (or higher amount of unpredictable data) in the point-to-point processing stage of SZ, such that many zero bits ("M[i]=0") would be generated accordingly, as shown in Algorithm 2. This will finally result in less time spent for Huffman coding.

D. Quality of Data from Decompression

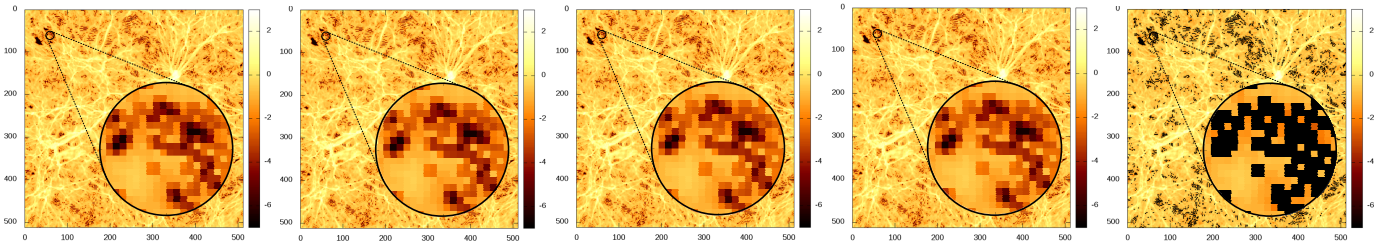
In this subsection, we evaluate the data accuracy loss of SZ_P and compare it with other compressors. As they are evaluated in previous work SZ_T [21], we also select dark_matter_density, velocity_x fields in NYX, as well as a temperature field, to evaluate data fidelity of each approach. Dark_matter_density is a typical use case for point-wise relative error which most value are distributed in [0, 1], and the rest is distributed in [1, 1.378E+4]. The velocity_x includes large values with positive/negative signs indicating directions. Max Point-wise Relative Error, NRMSE, PSNR and Compression Ratio are evaluated. Model A (denoted by SZ_P′) is also evaluated to demonstrate its not strictly respecting error bound limitation.

TABLE III
COMPRESSION RATIO WITH LOW ERROR BOUNDS ϵ (HIGH PRECISION) ON FOUR DATASETS. SZ_P ACHIEVES ABOUT 1.2X HIGHER COMPRESSION RATIO THAN SZ_T

Datasets	CESM		HACC		Hurricane		NYX	
Precision	1e-5	1e-6	1e-5	1e-6	1e-5	1e-6	1e-5	1e-6
SZ_P	1.62	1.24	2.29	1.80	3.52	2.69	5.31	3.61
SZ_T	1.44	1.18	2.25	1.42	2.98	2.25	3.99	2.53
Ratio	1.13	1.05	1.02	1.27	1.18	1.19	1.33	1.43

Table IV shows the data quality results of SZ_P, SZ_T, SZ_P′, ZFP_T with four most widely used point-wise error bounds (0.1, 0.01, 0.001, 0.0001). SZ_P′ does not strictly respect the error bound, achieving about 2 times the error bound in some cases. It also does not perform well on PSNR and NRMSE. Overall, the data accuracy loss of SZ_P′ is higher than either SZ_T′ or SZ_P due to its design limitation in error control.

On the other hand, SZ_P (using model B), an advanced version of SZ_P′ (using Model A), strictly respects the error bound like what SZ_T does, thus works better than SZ_P′ and achieves similar data accuracy as SZ_T. With more detailed analysis, benefiting from its smaller spacing of neighbor quantization code ($2-\theta$ in SZ_P and 2 in SZ_T), SZ_P achieves slightly better accuracy than SZ_T, but SZ_P suffers from a little reduction in compression ratio for the same reason, causing few more unpredictable data points. Although with these slight differences, as Figure 13 shows, in practice SZ_P and SZ_T achieve almost the same decompressed data quality. The data loss of SZ_P′ is about 2 times higher than that of SZ_P, but for accuracy tolerant applications such as visualization, the SZ_P′ performs as well as SZ_P and SZ_T. In comparison, as Figure 13 shows, the visualization effect of ZFP_T decompressed data is obviously different from the



(a) Original (b) SZ_P′ (Model A) (c) SZ_P (Model B) (d) SZ_T (e) ZFP_T

Fig. 13. Visualization of decompressed dark_matter_density dataset (slice 200) at the compression ratio of 2.75.

TABLE IV
POINT-WISE RELATIVE ERROR BOUND ON 3 REPRESENTATIVE FIELDS IN NYX.

pwr_eb	type	dark_matter_density				velocity_x				temperature			
		MAX E	NRMSE	PSNR	CR	MAX E	NRMSE	PSNR	CR	MAX E	NRMSE	PSNR	CR
1E-01	SZ_P′	1.52E-01	3.97E-05	88.02	6.25	1.50E-01	2.27E-04	72.88	23.48	1.52E-01	4.70E-03	46.56	18.83
	SZ_T	1.00E-01	3.49E-05	89.13	6.19	1.00E-01	2.05E-04	73.75	24.97	1.00E-01	4.32E-03	47.29	19.85
	SZ_P	9.99E-02	3.26E-05	89.73	6.03	9.97E-02	1.96E-04	74.15	25.99	9.97E-02	4.10E-03	47.75	20.79
	ZFP_T	5.07E-02	4.64E-06	106.66	3.32	4.80E-02	2.89E-04	70.79	18.40	5.17E-02	2.74E-05	91.25	14.00
1E-02	SZ_P′	1.75E-02	4.05E-06	107.84	3.85	1.70E-02	2.36E-05	92.55	13.46	1.70E-02	5.39E-04	65.37	14.37
	SZ_T	1.00E-02	3.55E-06	108.99	3.85	1.00E-02	2.00E-05	93.97	14.06	1.00E-02	4.50E-04	66.93	13.55
	SZ_P	1.00E-02	3.42E-06	109.31	3.80	9.96E-03	1.82E-05	94.78	12.98	9.96E-03	4.23E-04	67.47	11.93
	ZFP_T	3.02E-03	2.75E-07	131.22	2.35	3.33E-03	3.45E-05	89.24	6.59	3.16E-03	1.75E-06	115.15	5.21
1E-03	SZ_P′	1.96E-03	4.30E-07	127.34	2.75	1.95E-03	2.58E-06	111.76	6.75	1.95E-03	6.70E-05	83.48	8.02
	SZ_T	9.97E-04	3.51E-07	129.10	2.74	9.98E-04	1.98E-06	114.08	6.61	9.98E-04	4.51E-05	86.91	7.63
	SZ_P	1.00E-03	3.44E-07	129.27	2.72	9.99E-04	1.79E-06	114.93	6.49	9.99E-04	4.25E-05	87.44	7.12
	ZFP_T	3.90E-04	3.56E-08	148.98	1.92	3.95E-04	4.63E-06	106.69	4.08	3.97E-04	2.23E-07	133.04	3.50
1E-04	SZ_P′	1.60E-04	3.96E-08	148.04	2.12	1.60E-04	2.10E-07	133.55	3.93	1.60E-04	4.98E-06	106.05	4.39
	SZ_T	9.80E-05	3.48E-08	149.16	2.09	9.90E-05	1.98E-07	134.05	3.92	9.90E-05	4.43E-06	107.08	4.38
	SZ_P	1.00E-04	3.38E-08	149.43	2.01	1.00E-04	1.72E-07	135.29	3.88	1.00E-04	4.25E-06	107.43	4.27
	ZFP_T	5.08E-05	4.49E-09	166.96	1.63	4.99E-05	5.81E-07	124.71	2.95	5.33E-05	2.76E-08	151.19	2.63

original data, thus, the fidelity loss caused by ZFP_T has obvious impact at application level.

For ZFP_T, Table IV suggests it achieves high data quality but low compression ratio. This is because ZFP_T is difficult to control the data quality due to its over-preserved error bound in it design [21] although we fill in its configuration file with the expected error bound. Meanwhile, when we let ZFP_T achieves the same compression ratio as SZ methods, it have obvious low data quality as discussed in the last paragraph and shown in Figure 13.

VI. Conclusion and Future Work

In this paper, we propose an effective approach to accelerate the point-wise relative error bounded lossy compresison of SZ, leading to an optimal lossy compressor for users with respect to both compression ratio and compression rate in most cases. Our optimization strategy originates from an important observation that the logarithmic transformation, Huffman decoding are the performance bottlenecks in SZ. We develop precomputation-based mechanisms with the fast table lookup methods for logarithmic transformation and most of traversing operations in Huffman decoding, which can improve the compression rate by about 30% and decompression rate by about 70% in most of cases. The key findings about our performance evaluation with four well-known application datasets are listed as follows:

- Compression/decompression Rate: SZ_P improve the compression rate by about 30% and decompression rate by about 70% compared with SZ_T. It has comparable or even higher performance than ZFP_T in most of cases.
- Compression ratio: SZ_P and SZ_T have very similar compression ratios, which are always significantly higher (even up to one order of magnitude in some cases) than that of ZFP_T.
- Respecting user-required error control: Similar to SZ_T, SZ_P can always respect user requirements on point-wise relative error bounds. SZ_P also has exactly the same level of data distortion in NRMSE or PSNR with SZ_T.
- Visualization quality: With the same compression ratio, the point-wise relative error bounded compression under both SZ_P and SZ_T exhibit fairly high visual quality on the scientific datasets in our experiments, and they are much higher than that of ZFP_T.

We publish our codes on Github at https://github.com/Borelset/SZ, which are planned to be integrated into SZ main branch in a future release. In our future work, we will also exploit parallelism for SZ_P to further speed up the process of calculating quantization factor values, especially on the 2D and 3D dimensional HPC datasets.

REFERENCES

[1] T. Lu and et al., "Canopus: A paradigm shift towards elastic extreme-scale data analytics on hpc storage," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 58–69.

[2] 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 *IEEE International Parallel and Distributed Processing Symposium*, 2017.

[3] W. Austin, G. Ballard, and T. G. Kolda, "Parallel tensor compression for large-scale scientific data," in *IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2016, pp. 912–922.

[4] I. Foster and et al., "Computing just what you need: Online data analysis and reduction at extreme scales," in *European Conference on Parallel Processing (Euro-Par'17)*, Santiago de Compostela, Spain, 2017.

[5] D. Ghoshal and L. Ramakrishnan, "Madats: Managing data on tiered storage for scientific workflows," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017, pp. 41–52.

[6] B. Dong, K. Wu, S. Byna, J. Liu, W. Zhao, and F. Rusu, "Arrayudf: User-defined scientific data analysis on arrays," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017.

[7] ASCAC Subcommittee, "Top ten exascale research challenges," 2014. [Online]. Available: https://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf

[8] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, 2016.

[9] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A study on data deduplication in hpc storage systems," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2012, pp. 1–11.

[10] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, July 1948.

[11] M. Burtscher and P. Ratanaworabhan, "Fpc: A high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, 2009.

[12] N. Sasaki and et al., "Exploration of lossy compression for application-level checkpoint/restart," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 914–922.

[13] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu *et al.*, "Understanding and modeling lossy compression schemes on hpc scientific data," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 348–357.

[14] J. Kunkel, A. Novikova, E. Betke, and A. Schaare, "Toward decoupling the selection of compression algorithms from quality constraints," in *International Conference on High Performance Computing*. Springer, 2017, pp. 3–14.

[15] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp," *arXiv preprint arXiv:1806.08901*, 2018.

[16] N. J. F. N. B. A. Poppick, A. and D. Hammerling, "A statistical analysis of compressed climate model data," in *The 4th International Workshop on Data Reduction for Big Scientific Data*, 2018.

[17] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield *et al.*, "Hello adios: the challenges and lessons of developing leadership class i/o frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014.

[18] T. E. Fornek, "Advanced photon source upgrade project preliminary design report," 9 2017.

[19] G. Marcus, Y. Ding, P. Emma, Z. Huang, J. Qiang, T. Raubenheimer, M. Venturini, and L. Wang, "High Fidelity Start-to-end Numerical Particle Simulations and Performance Studies for LCLS-II," in *Proceedings, 37th International Free Electron Laser Conference (FEL 2015): Daejeon, Korea, August 23-28, 2015*, 2015.

[20] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, Dec 2014.

[21] X. Liang, S. Di, D. Tao, Z. Chen, and F. Cappello, "An efficient transformation scheme for lossy data compression with point-wise relative error bound," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 179–189.

[22] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," *Euro-Par 2011 Parallel Processing*, pp. 366–379, 2011.

[23] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 730–739.

[24] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on signal processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

[25] S. Wold, "Spline functions in data analysis," *Technometrics*, vol. 16, no. 1, pp. 1–11, 1974.

[26] X. He and P. Shi, "Monotone b-spline smoothing," *Journal of the American statistical Association*, vol. 93, no. 442, pp. 643–650, 1998.

[27] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," 2018.

[28] J.-l. Gailly, "gzip: The data compression program," 2016. [Online]. Available: https://www.gnu.org/software/gzip/manual/gzip.pdf

[29] "Zstandard - fast real-time compression algorithm." [Online]. Available: https://github.com/facebook/zstd

[30] S. Di, D. Tao, X. Liang, and F. Cappello, "Efficient lossy compression for scientific data based on pointwise relative error bound," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[31] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR)*, vol. 23, no. 1, pp. 5–48, 1991.

[32] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.