Efficient Lossy Compression for Scientific Data Based on Pointwise Relative Error Bound

Sheng Di, *Senior Member, IEEE*, Dingwen Tao, *Member, IEEE*, Xin Liang, and Franck Cappello, *Fellow, IEEE*

Abstract—An effective data compressor is becoming increasingly critical to today's scientific research, and many lossy compressors are developed in the context of absolute error bounds. Based on physical/chemical definitions of simulation fields or multiresolution demand, however, many scientific applications need to compress the data with a pointwise relative error bound (i.e., the smaller the data value, the smaller the compression error to tolerate). To this end, we propose two optimized lossy compression strategies under a state-of-the-art three-staged compression framework (prediction + quantization + entropy-encoding). The first strategy (called block-based strategy) splits the data set into many small blocks and computes an absolute error bound for each block, so it is particularly suitable for the data with relatively high consecutiveness in space. The second strategy (called multi-threshold-based strategy) splits the whole value range into multiple groups with exponentially increasing thresholds and performs the compression in each group separately, which is particularly suitable for the data with a relatively large value range and spiky value changes. We implement the two strategies rigorously and evaluate them comprehensively by using two scientific applications which both require lossy compression with point-wise relative error bound. Experiments show that the two strategies exhibit the best compression qualities on different types of data sets respectively. The compression ratio of our lossy compressor is higher than that of other state-of-the-art compressors by 17.2–618 percent on the climate simulation data and 30–210 percent on the N-body simulation data, with the same relative error bound and without degradation of the overall visualization effect of the entire data.

Index Terms—Lossy compression, science data, high performance computing, relative error bound

1 Introduction

An effective data compressor is becoming more and more critical to today's scientific research because of extremely large volume of data produced by scientific simulations. The large volume of data generally are stored in a parallel file system, with a limited capacity of the storage space and limited I/O bandwidth to access. Climate scientists, for example, need to run large ensembles of high-fidelity 1km $\times$ 1km simulations, with each instance simulating 15 years of climate in 24h of computing time. Estimating even one ensemble member per simulated day may generate 260 TB of data every 16s across the ensemble. In the Hardware/Hybrid Accelerated Cosmology Code (HACC) [2] (a well-known cosmology simulation code), the number of particles to simulate could reach up to 3.5 trillion, which may produce 60 petabytes of data to store. One straight-forward data reduction method is decimating data in either time dimension or space, however, this may lose important information for user's analysis.

- S. Di and F. Cappello are with the Mathematics and Computer Science (MCS) Division, Argonne National Laboratory, Lemont, IL 60439. E-mail: sdi1@anl.gov, cappello@mcs.anl.gov.
- D. Tao is with the Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487. E-mail: dingwen.tao@ieee.org.
- X. Liang is with the Computer Science Department, University of California, Riverside, CA 92521. E-mail: xlian007@ucr.edu.

*Manuscript received 8 Mar. 2018; revised 15 June 2018; accepted 22 July 2018. Date of publication 27 July 2018; date of current version 16 Jan. 2019. (Corresponding author: Sheng Di.)
Recommended for acceptance by J. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2859932*

Error-controlled lossy compression techniques have been considered the best trade-off solution compared to lossless compression, because not only can such techniques significantly reduce the data size but they can also keep the data valid after the data decompression based on the error controls. The existing compressors (such as [3], [4], [5]) are basically designed in the context of absolute compression errors (such as the absolute error bound for each data point or the root mean squared errors).

In addition to the absolute error bound, another type of error bound (called *pointwise relative error bound* or *relative error bound* for short) has received increasing attention because it is more reasonable for some applications based on the users' demand on multiresolution or physical definition of simulation fields. As for the pointwise relative error bound, the maximum compression error for each data point is equal to a percentage ratio of the data point's value, such that the larger the value is, the larger the compression error. Hence, such an error bound leads to multiple precisions/resolutions on different data points. As one Argonne Photon Source (APS) researcher pointed out, for instance, he needs to use different levels of precisions to study various regions of an APS generated X-ray image. Moreover, in some N-body applications such as cosmology simulation, thousands of millions of particles are moving in the space with different speeds. According to the corresponding researchers, the faster the particles move, the larger errors their analysis can accept, which means that the compression of the velocity field of particles needs to use point-wise relative error bound.

In this paper, we focus on the optimization of compression quality based on pointwise relative error bound for scientific data with various dimensions. Our exploration is based on a generic compression model - SZ [3], [4], which involves three critical steps: (1) value prediction for each data point, (2) quantization of prediction errors, and (3) entropy encoding of the quantization output. Several challenging issues have to be faced. SZ constructs a set of adjacent quantization bins to transform each original floating-point data value to an integer based on its prediction error. Guaranteeing the absolute error bound in this framework is relatively easy, in that the sizes of quantization bins could be fixed because of the constant absolute error bound. Problems arise, however, when the error bounds are varying with different data points upon the request for a relative error bound.

We propose two effective lossy compression strategies which can adapt to different types of scientific data sets with various data features. The first strategy is particularly suitable for the datasets with relatively consecutive/smooth value changes in space (e.g., 2D or 3D datasets). This strategy (a.k.a., block-based strategy) is to split the whole data set into multiple small blocks in the multidimensional space and adopt a uniform error bound in each block. There are several critical questions to answer: how to determine the error bound in one block? how to minimize the overhead of storing the metadata in each block? how to determine the block size considering the tradeoff between the compression gains with possibly more relaxed error bounds and unavoidable metadata overhead? The other strategy (called multi-threshold-based strategy) is designed particularly for the dataset with spiky value changes on adjacent data points (such as N-body simulation data). In this method, the whole value range will be split into multiple groups with exponentially increasing thresholds and then the data will be compressed according to the groups separately with a common absolute error bound in the same group. As for this strategy, how to determine the absolute error bound for each group and how to encode the group index for each data point would be two serious questions to answer.

Our key contribution is twofold. On the one hand, we answer the above-listed technical questions and propose complete design/implementation methods for the above two strategies based on the requirement of relative error bound. We release the code as an open-source [1]. Further, we validate the effectiveness of the new solution by using real-world scientific data sets. Experiments show that our block-based strategy exhibits 17.2–618 percent higher compression ratios on climate simulation data than other state-of-the-art existing compressors; our multi-threshold-based strategy outperforms other compressors by 31–210 percent on cosmology simulation with the user-required point-wise error bound and comparable compression/decompression time.

The rest of the paper is organized as follows. In Section 2, we discuss related work. We formulate the scientific data compression problem with the requirement of relative error bound in Section 3. In Section 4, we describe the SZ compression framework in details. In Section 5, we propose the block-based lossy compression strategy and multi-threshold-based lossy compression strategy respectively. In Section 6, we describe the experimental setting and present the evaluation results. We conclude in Section 7 with a vision of future work.

2 Related Work

Many data compressors have been developed to significantly reduce the data size. In general, the compressors can be split into two categories, lossless compressor and lossy compressor. Since scientific data are mainly generated/stored in the form of floating-point values each with rather random ending mantissa bits, generic lossless binary-stream compressors such as Gzip [6] and BlosC [7] cannot work effectively. Although there are also some existing lossless compressors [8], [9], [10], [11] designed for floating-point data sets, they still suffer from very limited compression ratios [3], [4].

Lossy compressors have been developed for years and they can be categorized as either error-controlled or not. Many of the existing lossy compressors (such as [12], [13]) are not error controlled, such that the reconstructed data may not be valid from the perspective of data researchers or users. Error-controlled lossy compression techniques have also been developed in the past decade. SZ [3], [4], [14], for example, provides multiple types of error controls, yet all of them are based on a constant/uniform bound (a.k.a., absolute error bound) for all the data points. Another outstanding floating-point lossy compressor, ZFP [5], provides three types of error controls: fixed absolute error bound, fixed rate, and fixed precision (the number of uncompressed bits per value). According to the developer of ZFP, its most efficient mode is fixed absolute error bound, but it still overpreserves the compression errors for a large majority of the data points. Moreover, some specific compression techniques are also customized for particular datasets based on their features. R-index sorting [15], [16], for instance, is adopted to improve the compression ratio for the position fields of N-body simulation; Lee et al. [17] proposed to leverage the statistical features to improve the compression ratio for time-series datasets.

Relative-error-bound based compressors also have been developed, however, they suffer from limited compression quality. ISABELA [18], for example, allows compression with point-wise relative error bounds; but its compression ratio (the ratio of the original data size to the decompressed data size) is usually around 4:1 [3], [4], [18], which is far lower than the demanded level of the extreme-scale application users. Moreover, its compression rate is about 5–10X lower than that of SZ [3], [4] and ZFP [5]. Sasaki et al. proposed a wavelet transform based lossy compressor [12], which splits data into a high-frequency part and low-frequency part by the Wavelet transform and then uses vector quantization and Gzip [6]. Similar to ISABELA, the Wavelet transform based method suffers from lower compression ratio than SZ and ZFP do [3]. Moreover, it cannot deal with the data sets having an odd number of elements at some dimension. FPZIP [19] allows users setting the number of bits (called precision) to maintain for each data point during the compression. For single-precision floating-point data sets, for example, precision = 32 indicates a lossless compression in FPZIP and other settings with lower precisions corresponds to different lossy levels of compression in the sense of point-wise relative error bound approximately.

FPZIP has two drawbacks: on the one hand, it is uneasy for users to control the compression errors based on a specific point-wise error bound. In practice, they have to try compressing the data sets multiple times to estimate the precision required corresponding to the expected error bound. On the other hand, its compression ratio is lower than our proposed solution with the same relative error bound, to be shown in Section 6 in details. All in all, compared with the existing error-controlled lossy compressors, our proposed relative error bound based compression strategies can improve the compression ratio by 17.2–618 percent on CESM-ATM climate simulation data [20] and by 31–210 percent on HACC particular simulation data [2].

3 Problem Formulation

The research objective is to optimize the compression quality for the lossy compression of scientific data with the constraint of the pointwise relative error bound (a.k.a, relative error bound for short). Specifically, given a data set S={d_1, d_2, ..., d_N} with N data points (where d_i refers to the data point i in the data set), the reconstructed data set S'={d'_1, d'_2, ..., d'_N} must satisfy the following inequality:

$$\max_{d_i \in S, d'_i \in S'} \left(\left| \frac{d_i - d'_i}{d_i} \right| \right) \leq \epsilon, \tag{1}$$

where ϵ is a small constant value (i.e., relative error bound) specified by the user.

We present an example to further illustrate the definition of pointwise relative error bound. Suppose we are given a set of data {1.23, 1.65, 2.34, 3.56, 10.0, 20.0} to compress with a relative error bound of 0.01 (or 1 percent). Then the absolute error bounds for the six data points will be 0.0123, 0.0165, 0.0234, 0.0356, 0.1, and 0.2, respectively. That is, as long as the difference of the reconstructed value d'_i and its original value d_i is no greater than 1 percent of d_i, the lossy compression will be thought of as satisfying the error bound.

A typical use-case that requires the relative error bound based compression is the compression of velocity fields in a cosmology simulation such as HACC [2]. In HACC, for example, each particle has two parts of information, position {x,y,z} and velocity {vx,vy,vz}, each being involved in three dimensions. So, there are six 1D arrays - x, y, z, vx, vy, and vz - used to store the position and velocity information for all particles and the index of each array indicates the particle ID. According to the cosmology researchers and HACC developers, the larger velocity a particle has, the larger error it can tolerate in general. In this case, the point-wise relative error bound is required to compress the velocity data.

Our objective is to maximize the compression ratio, subject to the inequality (1), where compression ratio (denoted by γ) is defined

$$\gamma = \frac{original\ data\ size}{compressed\ data\ size}. \tag{2}$$

In addition to the point-wise relative error bound (i.e., Inequality (1)), another important evaluation metric is peak signal-to-noise ratio (PSNR), as defined

TABLE 1
Formulas of 1, 2, 3-Layer Prediction for Two-Dimensional Data Sets

	Prediction Formula
1-Layer	$f(i_0,j_0) = V(i_0,j_0-1) + V(i_0-1,j_0) - V(i_0-1,j_0-1)$
2-Layer	$f(i_0,j_0) = 2V(i_0-1,j_0) + 2V(i_0,j_0-1) - 4V(i_0-1,j_0-1)$ $-V(i_0-2,j_0) - V(i_0,j_0-2) + 2V(i_0-2,j_0-1)$ $+2V(i_0-1,j_0-2) - V(i_0-2,j_0-2)$
3-Layer	$f(i_0,j_0) = 3V(i_0-1,j_0) + 3V(i_0,j_0-1)$ $-9V(i_0-1,j_0-1) - 3V(i_0-2,j_0) - 3V(i_0,j_0-2)$ $+9V(i_0-2,j_0-1) + 9V(i_0-1,j_0-2) - 9V(i_0-2,j_0-2)$ $+V(i_0-3,j_0) + V(i_0,j_0-3) - 3V(i_0-3,j_0-1) - 3V(i_0-1,j_0-3)$ $+3V(i_0-3,j_0-2) + 3V(i_0-2,j_0-3) - V(i_0-3,j_0-3)$

$$psnr = 20 \cdot log_{10}\left(\frac{d_{max} - d_{min}}{rmse}\right), \tag{3}$$

where $rmse = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - d'_i)^2}$, and d_{max} and d_{min} refer to the max and min value respectively.

PSNR is commonly used to assess the overall distortion between the original data and reconstructed data especially in visualization. Logically, the higher the PSNR is, the lower the overall compression error, indicating a better compression quality. In our experiments, we will evaluate both maximum point-wise relative error and PSNR, in order to assess the distortion of data comprehensively.

4 SZ Compression Framework

We designed a novel, effective lossy compression model, namely SZ compression framework [4], which includes three fundamental steps as described in this section. We describe the three steps in the following text.

4.1 Step I: Data Prediction with Reconstructed Values

In our design, we scan the whole data set with an increasing order of dimensions, and the prediction values are generated for each data point based on their preceding processed neighbors. The prediction formulas are derived by constructing a surface using the neighboring data points on one or multiple layers, as presented in Table 1, where (i_0, j_0) refers to the current data point to be predicted, $V(x, y)$ refers to the corresponding value of a data point (x, y), and x=0,1,2,..., y=0,1,2,.... The detailed derivation of the prediction formulas is described in our prior conference publication [4]. In fact, more advanced prediction methods can be customized based on the characteristics of specific scientific data, which will be studied in our future work.

Now, the key question is how many layers we are supposed to use for the data prediction. Before answering the question, we must notice an important constraint: in the compression phase, the data prediction must be conducted based on the previously *decompressed* values instead of the original data values, otherwise the data decompression cannot respect the error bound strictly. In our prior work [4], we presented that the one-layer prediction is the best choice, considering the impact of the inevitable data distortion of the reconstructed data points to the prediction accuracy of

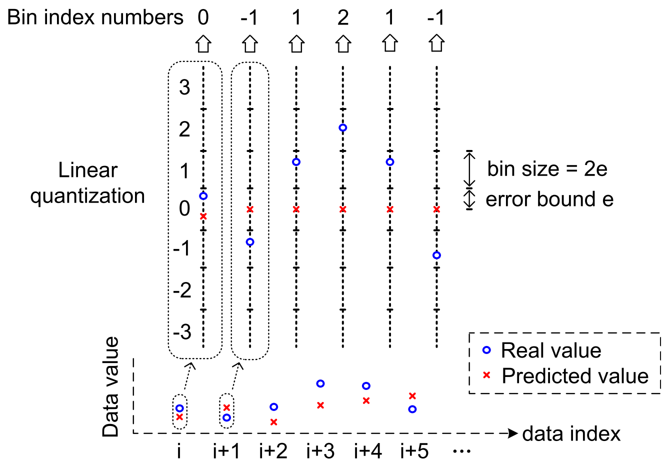


Fig. 1. Illustration of linear-scaling quantization step.

the current data point. We omit the details here because of the space limitation.

4.2 Step II: Error-Bounded Linear-Scaling Quantization

The step II is a critical step for controlling the compression errors. As illustrated in Fig. 1, a set of consecutive bins are constructed based on each predicted data value as the center. The size of each bin is 2X error bound and the bins will be tagged as following indices: $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$. As such, the original raw value of each data point can be transformed to an integer value (i.e., the index of the bin containing the corresponding true value).

In absolute terms, the quantization bin index can be calculated by Formula (4), and the center of each bin will serve as the reconstructed data value of the corresponding data point during the decompression

$$bin\ index = \begin{cases} \left\lfloor \frac{d_i - d_i'}{2e} + \frac{1}{2} \right\rfloor & d_i \geq d_i' \\ -\left\lfloor \frac{d_i' - d_i}{2e} + \frac{1}{2} \right\rfloor & d_i < d_i', \end{cases} \quad (4)$$

where $\lfloor\rfloor$ refers to the floor function. Although step II may introduce distortion of the data, the compression error (i.e., the distance between the center of the located bin and the real value of the data point) must be within the required error bound e, since the bin's size is twice as large as the error bound.

As discussed above, the error-bounded linear-scaling quantization will transform all of the original floating-point data values to another set of integer values, and a large majority of such integer values are expected to be very close to 0, in that most of the raw data values are consecutive in space. Fig. 2 shows an example of the distribution of quantization codes produced by our quantization encoder, which uses 255 quantization bins to represent predictable data. From this figure, we can see that the distribution of quantization codes exhibits a fairly sharp shape and that the shape depends on the accuracy of the data prediction. The higher prediction accuracy, the sharper the distribution exhibits, and then the higher compression ratio we can get by using a variable-length encoding, in which more frequent symbols will be coded using fewer bits. In our implementation, all

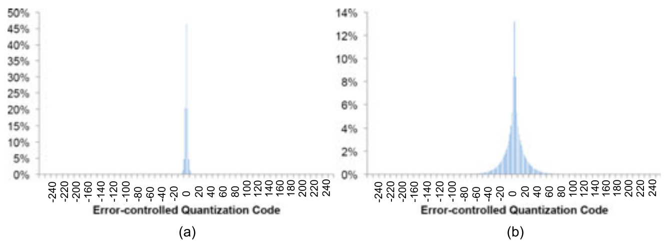


Fig. 2. Distribution produced by error-controlled quantization encoder on ATM data sets of (a) value-range-based relative error bound = 10^{-3} and (b) value-range-based relative error bound = 10^{-4} with 255 quantization intervals ($m = 8$).

the bin-index codes are actually mapped to positive integers by adding a constant offset, because we reserve integer 0 to represent the unpredictable data that cannot be identified by the prediction + quantization method because of limited number of quantization bins for possible spiky data changes. The unpredictable data will be labeled and compressed by keeping only valuable bits in its IEEE 754 binary representation, respecting the specified error bound.

The last critical issue regarding the linear-scaling quantization step is how to set an appropriate number of quantization bins provided a specific data set. To this end, we design an efficient algorithm that can determine the appropriate number of quantization bins. Specifically, we allow users specifying a lower bound of the prediction hitting-rate, based on which our algorithm can estimate the required number of quantization bins accurately. The design idea is first sampling a very small portion (generally only 1 percent) of the data points in the whole data set by a systematic sampling, then performing data prediction and quantization on each of them and finally determining the appropriate number of quantization bins accordingly. We present an example pseudo-code in Algorithm 1. Without loss of generality, we assume the data set to compress is a 3D array with three dimension sizes being equal to M, N, K respectively in this example.

We describe Algorithm 1 as follows. In the beginning, we first allocate a counting buffer (namely *intervals*) in order to count the number of hits for different bin-index codes. The data points will be selected by a systematic sampling (line 2-5) with a sampling distance s, which means that only $\frac{1}{s}$ data will be selected. Then, we compute the prediction value for each sampled data point based on the one-layer prediction (line 7). Note that each bin here involves both cases in Formula (4) (i.e., $d_i \geq d_i'$ and $d_i < d_i'$). Thereafter, we count the total number of hits with increasing number of bins based on the counting buffer against the target hitting count calculated by the lower bound of prediction hitting rate (line 13-20). In the end, the appropriate number of quantization bins will be finalized in the form of power of two (line 22), in order to guarantee enough quantization bins on demand.

Table 2 presents the effectiveness of reaching demanded hitting rate by Algorithm 1 based on two data sets (CESM-ATM and HACC), with a sampling rate of 1 percent (i.e., sampling distance = 100; select one sample point every 100 data points) and two different error bounds (1E-4 and 1E-6 compared to the value range). The column named 'sampled' in the table refers to the percentage of predictable data covered by p quantization bins during the calculation of these bins (i.e., Algorithm 1), and the column tagged 'real' means the percentage of the number of data points covered by the final P quantization bins during the data compression. It is observed that when the lower bound of prediction hitting rate is set to 80$\sim$99 percent, the minimum numbers of bins required (i.e., the value p calculated at line 21 in the algorithm) are in the range of [2, 6515] and [24, 98294] for the two data sets respectively. The appropriate numbers of quantization bins output by our algorithm is always no smaller than the minimum requirement more or less, because of the line 22 (that is, the final number of quantization bins is rounded to 2^h, where $2^{h-1} < p \le 2^h$ and p=2 (r+1)). In practice, we recommend to set the lower bound of prediction hitting rate to 99 percent, which is also the default setting of SZ.

Algorithm 1. Calculating an Appropriate Number of Quantization Bins

Input: user-specified error bound e, lower bound of prediction hitting rate (denoted by η); maximum number of quantization bins (denoted by m); sampling distance s

Output: the appropriate number of quantization bins (denoted by P) based on which the real prediction hitting rate will be greater than η

```
1: Allocate a buffer called intervals, with $M$ elements; /*for
   keeping the counts of different bin-index codes*/.
2: for ($i$ = 1→ $M-1$) do
3:   for ($j$ = 1→ $N-1$) do
4:     for ($k$ = 1→ $K-1$) do
5:       if (($i$+$j$+$k$)% s == 0) then
6:         Perform the one-layer prediction to get $d'_{i,j,k}$.
7:         bin_index = $[\frac{d'_{i,j,k}-d_{i,j,k}}{2e}+\frac{1}{2}]$. /*Formula (4)*/
8:         intervals[bin_index]++.
9:       end if
10:    end for
11:  end for
12: end for
13: target_hit_count = $\eta(M-1)(N-1)(K-1)$.
14: hitting_count = 0.
15: for ($r$ = 0→ $m$) do
16:   hitting_count += intervals[$i$].
17:   if (hitting_count ≥ target_hit_count) then
18:     break.
19:   end if
20: end for
21: $p$ = 2×($r$+1). /*$r$+1 is half of the expected # quantization
    bins.*/
22: $P = 2^h$, where $2^{h-1} < p \le 2^h$, $h$=0,1,2,.... /*if $p$=28, then
    $P$=32*/
```

4.3 Step III: A Customized Variable-Length Encoding

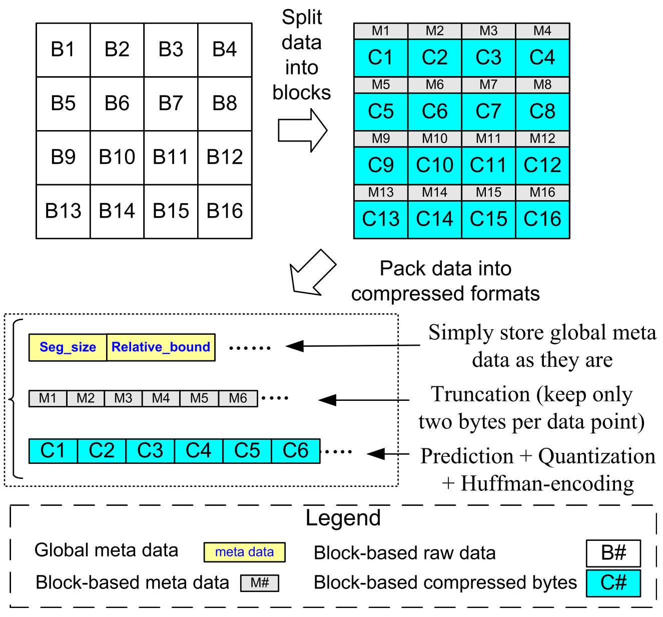
We adopt Huffman encoding method in step III, because Huffman encoding has been proved as an optimal variable-length encoding approach in most of cases (especially when the distribution of codes follows a normal distribution). In order to optimize the compression quality, we improved the Huffman encoding algorithm based on the characteristic of the error-bounded linear scaling quantization bins. As discussed previously, the appropriate number of quantization bins calculated by Algorithm 1 could be very large. Experiments based on HACC data sets (velocity field), for instance, show that the number of quantization bins could reach up to 200k if the required error bound is 10^{-5} while the expected prediction hitting rate is set to 99 percent. However, the traditional Huffman encoding algorithm implemented in existing lossless compression packages such as Zlib [21] treats the stream of data always in the unit of bytes, which will definitely cause a sub-optimal compression effect.

TABLE 2
Effectiveness of Reaching Demanded Hitting Rate Under Algorithm 1 Based on CESM-ATM and HACC Data Set with Different Error Bounds

CESM-ATM data set (err=1E-4)								
Demanded	CLDLOW				FLDSC			
hitting rate	p	P	sampled	real	p	P	sampled	real
80%	14	32	82.77%	96.42%	2	32	85.7%	99.7%
90%	20	32	90.8%	96.42%	4	32	94.3%	99.7%
99%	68	128	99.04%	99.7%	16	32	99.04%	99.7%
HACC data set (err=1E-4)								
Demanded	position x				velocity vx			
hitting rate	p	P	sampled	real	p	P	sampled	real
80%	24	32	80.1%	86.96%	202	256	80.2%	84.8%
90%	40	64	90.7%	95.3%	350	512	90.1%	94.9%
99%	122	128	99.1%	99.3%	984	1,024	99%	99.1%
CESM-ATM data set (err=1E-6)								
Demanded	position x				velocity vx			
hitting rate	p	P	sampled	real	p	P	sampled	real
80%	1,178	2,048	80%	91.6%	70	128	80%	88.3%
90%	1,816	2,048	90%	91.6%	156	256	90%	93.4%
99%	6,518	8,192	99%	99%	1,462	2,048	99%	99.4%
HACC data set (err=1E-6)								
Demanded	position x				velocity vx			
hitting rate	p	P	sampled	real	p	P	sampled	real
80%	2,292	4,096	80%	90.5%	19,904	32,768	80%	89.1%
90%	3,708	4,096	90%	90.5%	34,706	65,536	90%	97%
99%	11,970	16,384	99%	99.15%	98,294	131,072	99%	99.6%

The key difference between our improved Huffman encoding algorithm and the traditional algorithm is that our algorithm treats the sequence of quantization codes (i.e., bin indices generated by Algorithm 1) based on its integer values, which may lead to a large number of nodes in the tree, such that how to store the tree as effectively as possible needs to be handled carefully. We designed a recursive algorithm to convert the Huffman tree to byte stream, as shown in Algorithm 2. The initial node is the root of the Huffman tree and its id is 0. Left node array L and right node array R are used to record the left children and right children respectively. The *tag* array is used to record whether the corresponding node is an internal node or a leaf. The algorithm adopts a pre-order traversal to scan all the nodes and pad the information into four buffer arrays.

The time complexity of the algorithm is $O(N)$, where N is the number of nodes, which is twice the number of quantization bins. The storage overhead of saving the Huffman tree is a constant with respect to the entire compression size, because such an overhead is determined by the number of quantization bins, which is supposed to be a very small constant (such as 65,536 quantization bins) compared



Split data into blocks
Pack data into compressed formats
Seg_size
Relative_bound
Simply store global meta data as they are
Truncation (keep only two bytes per data point)
Prediction + Quantization + Huffman-encoding
Legend
Global meta data
meta data
Block-based meta data
M#
Block-based raw data
B#
Block-based compressed bytes
C#

Fig. 3. Illustration of block-based strategy.

to the number of data points (such as 2^{30} particles in one snapshot of HACC simulation) without loss of generality.

Algorithm 2. PadTree

Input: left node array L, right node array R, tag array tag, and code array C, node's id i, node's object (denoted by $node$)
Output: left node array L, right node array R, tag array, code array C

```
 1: $C[i]$ = $node$.c; /*record the code*/
 2: $tag[i]$ = $node$.tag; /*record whether the node is internal or leaf*/
 3: if ($node$.left_tree $\neq$ null) then
 4:     $j$ ++; /*increment the global node ID $j$*/
 5:     $L[i] = j$; /*append $node_j$ as $node_i$'s left child*/
 6:     PADTREE($L, R, tag, C, j, node$.left_tree); /*recursive call*/
 7: end if
 8: if ($node$.right_tree $\neq$ null) then
 9:     $j$ ++; /*increment the global node ID $j$*/
10:     $R[i] = j$; /*append $node_j$ as $node_i$'s right child*/
11:     PADTREE($L, R, tag, C, j, node$.right_tree); /*recursive call*/
12: end if
```

5 Optimized Lossy Compression Based on Point-Wise Relative Error Bound

In this section, we describe how we design and implement the pointwise-relative-error-based lossy compression under the SZ compression framework. In our solution, we explore two strategies to address this issue, and they are called block-based strategy and multi-threshold-based strategy, respectively.

5.1 Block-Based Strategy

The block-based strategy first splits the whole data set into many small blocks and then computes an absolute error bound for each block based on the relative error bound. The data points inside each block will share the same absolute error bound during the compression, as shown in Fig. 3. In this sense, we just need to store an absolute error bound for each block. In fact, to minimize the overhead, we keep the first two bytes of the IEEE 754 floating-point representation of the absolute error bound because doing so already provides a fairly accurate estimate of absolute error bound for each block. On the other hand, as confirmed by our previous work [3], [4], the data in local regions are generally smooth, such that a common absolute error bound in a small block could be used to approximate the real absolute error bounds calculated based on the individual values.

In our design, users have three options to generate the absolute error bound for each block: calculating it based on the minimum value, average value, or maximum value in each block. If the user chooses the minimum value option, the final compression error must be strictly following the relative error bound for each data point. However, this mode may overreserve the accuracy, especially when the data exhibit spiky changes in local areas, such that the absolute compression errors may be largely smaller than the de facto error bound, causing a poor compression factor. To address this issue, the users can use the average value option to approximate the absolute error bound for each block, which may effectively avoid the issue of over-reserving accuracy. By comparison, the maximum value of the absolute error bound provides the most relaxed error controls with multiple resolutions yet with the highest possible compression ratio.

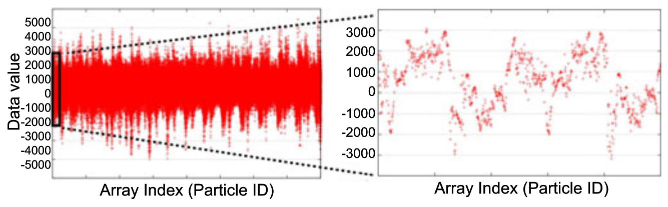


Fig. 4. Illustration of vx data array in a snapshot of HACC simulation.

5.2 Multi-Threshold-Based Strategy

Note that the block-based strategy may work effectively, especially when the data exhibit relatively high coherence in the space. However, this assumption may not always hold. The elements of the velocity array in HACC, for example, represent the velocity values of different particles, thus the adjacent data in the array could be largely different, as presented in Fig. 4.

To address this issue, we propose a multi-threshold-based strategy. The basic idea is splitting the whole value range into multiple groups with exponentially increasing thresholds, as presented in Fig. 5, and then performing the SZ compression (including prediction and quantization) in each group separately. This idea is largely different from the existing multi-level thresholding based image compression

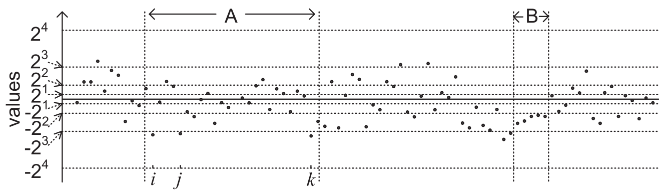


Fig. 5. Illustration of multi-threshold-based strategy.

method [22], [23], which splits the data set into multiple non-overlapping 4×4 blocks and uses two values (called mean-of-high-values and mean-of-low-values) to substitute the pixel values in each block. The threshold determining the high values and low values is calculated based on Shannon entropy. Unfortunately, this method is unable to respect the point-wise relative error bound and also suffers from low compression rate (as confirmed in [23]). Our strategy also has nothing to do with the Wavelet-based thresholding compression method [24] and recursive thresholding technology [25]. The former aims to reduce the noise of the Wavelet-transformed coefficients in the detail subbands (or high resolution subbands) based on a thresholding function such that the mean squared error (MSE) could be minimized. The latter optimized the image segmentation effect, by recursively segmenting the objects in the image data from the lowest intensity until there is only one object left.

In what follows, we will describe our multi-threshold-based strategy. We mainly answer three critical questions: (1) how to construct the multiple groups, (2) how to perform the compression for the data points in each group, and (3) how to encode the group IDs (a.k.a., group indices).

Each data point in the whole data set belongs to only one group, depending on its value. Specifically, the group ID of a data point d_i can be calculated by the following equation:

$$\mathrm{GID}(d_i) = \begin{cases} \mathrm{Expo}_{(2)}(d_i), & d_i \geq 0 \ \&\ \mathrm{Expo}_{(2)}(d_i) \geq \lambda \\ 0, & d_i \geq 0 \ \&\ \mathrm{Expo}_{(2)}(d_i) < \lambda \\ -\mathrm{GID}(-d_i), & d_i < 0 \end{cases}, \quad (5)$$

where $\mathrm{Expo}_{(2)}(d_i)$ refers to the base-2 exponent of the value d_i, which can be quickly received by reading the exponent part of the IEEE 754 representation. The λ is the lower bound of the exponent threshold under which the corresponding values will be compressed by an absolute error bound, in order to avoid too many groups to deal with and over-reservation of the precision during the compression (to be discussed in more details later). For example, if λ is set to 0, then we can get $\mathrm{Expo}_{(2)}(0.123) = 0$, $\mathrm{Expo}_{(2)}(1.23) = 0$, $\mathrm{Expo}_{(2)}(12.3) = 3$ and $\mathrm{Expo}_{(2)}(123) = 6$. In our implementation, λ is set to 0 by default.

The pseudo-code of the multi-threshold-based compression is presented in Algorithm 3. The compression in the multi-threshold-based strategy includes three steps: data prediction, quantization of prediction errors and Entropy-encoding, based on the SZ compression framework. Unlike the original design of SZ compressor, we need to declare a buffer for all groups, and each element of the buffer keeps a preceding value in the corresponding group. Specifically, there are two types of such buffers, called *pos_group_buf* and *neg_group_buf*, to deal with positive values and negative values respectively (see line 1-2). In the main loop, we need to select the corresponding last-preceding-value buffer *g_buf* [$|g|$] first (line 5-10). In the prediction step, the value of a data point is predicted by the preceding data value in its group (line 15). In particular, the size of the quantization bins used in the quantization step is determined based on the group ID (line 17). For instance, if the user expects to do the compression strictly using a relative error bound on each data point, the bin size should be always set to twice as large as the absolute error bound calculated in terms of the bottom threshold line of the data point's group. Specifically, for the data point d_i (it is located in the group $[2^k, 2^{k+1}]$), the absolute error bound should be set to $\varepsilon \cdot 2^k$, where k=GID (d_i) according to Formula (5) and ε is the relative error bound. For instance, if λ is set to 0 in Formula (5), 0.123 and 1.23 will be compressed by the absolute error bound of ε because $2^{GID(0.123)} \cdot \varepsilon = 2^{GID(1.23)} \cdot \varepsilon = \varepsilon$; 12.3 and 123 will be compressed based on the absolute error bounds of $2^3\varepsilon$ and $2^6\varepsilon$ respectively. The setting of λ determines the value range in which the data will be compressed by an absolute error bound to avoid the over-reservation of precision. Table 3 presents the calculated values of the real absolute error bounds for different groups. We can clearly see that such a design realizes a multi-resolution effect to control the compression errors of the data points in various ranges: i.e., the larger the values, the larger the absolute error bounds.

TABLE 3
An Example of Absolute Error Bounds Calculated by Exponential-Increasing Value Ranges (λ=0)

ε	···	**(-64,-32]**	**(-32,-16]**	**(-16,-8]**	**(-8,-4)**	**(-4,-2)**	**(-2,-1]**	**(-1,0)**
0.1	···	3.2	1.6	0.8	0.4	0.2	0.1	0.1
0.01	···	0.32	0.16	0.08	0.04	0.02	0.01	0.01
0.001	···	0.032	0.016	0.008	0.004	0.002	0.001	0.001

ε	**[0,1)**	**[1,2)**	**[2,4)**	**[4,8)**	**[8,16)**	**[16,32)**	**[32,64)**	···
0.1	0.1	0.1	0.2	0.4	0.8	1.6	3.2	···
0.01	0.01	0.01	0.02	0.04	0.08	0.16	0.32	···
0.001	0.001	0.001	0.002	0.004	0.008	0.016	0.032	···

Algorithm 3. Multi-Threshold-Based Design for the Compression with Point-Wise Relative Error Bound

Input: relative error bound ε, lower-bound of threshold λ (=0 by default), maximum group ID G ($2G$ is maximum number of groups), n data points
Output: Byte steam of compressed data

```
1: Construct buffers pos_group_buf for positive data points;
2: Construct buffers neg_group_buf for negative data points;
3: Allocate memory for the array dataGrpCode adn dataQuntCode;
4: for ($i = 1 \rightarrow n-1$) do
5:   if ($d_i \geq 0$) then
6:     g_buf = pos_group_buf;
7:   else
8:     g_buf = neg_group_buf;
9:   end if
10:  Compute group number $g$ = GID($d_i$) based on Formula (5);
11:  if ($|g| > G$ or g_buf[$|g|$] = $\phi$) then
12:    Treat $d_i$ as unpredictable data point;/*binary analysis*/
13:    g_buf[$|g|$] = $d_i'$; /*$d_i'$ here is the decompressed value of $d_i$*/
14:  else
15:    pred = g_buf[$|g|$]; /*$|g|$ is absolute value of $g$*/
16:    perr = $|pred - d_i|$;
17:    Compute real error bound (denoted $\epsilon$) in the group $g$;
18:    dataQuntCode[i] = $\frac{1}{2} \cdot \frac{perr}{\epsilon+1}$;
19:    dataGrpCode[i] = $g$;
20:    g_buf[$|g|$] = $d_i'$; /*Put decompressed value in the buffer.*/
21:  end if
22: end for
23: Compress dataQuntCode by our customized Huffman tree;
24: Compress dataGrpCode by our customized Huffman tree;
```

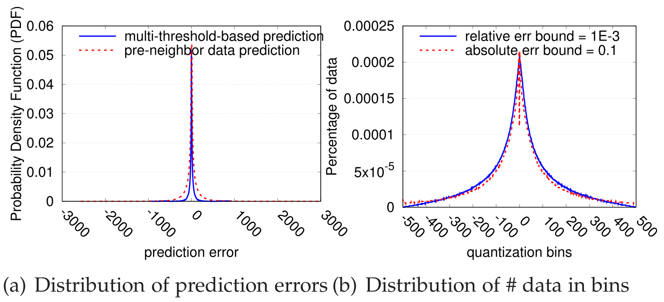


Fig. 6. Distribution analysis of multi-threshold-based design.

There are two significant advantages in the multi-threshold-based design. On the one hand, the data prediction step is not subject to the adjacent neighbor data points in the data set but the preceding data points in the same group, such that the prediction accuracy could be improved significantly especially when the data are not smooth in the array. As shown in Fig. 5, for instance, the segment A includes three data points i, j, and k, whose values are largely different from their previous neighbor values yet are very close with each other in the same group. We compare the distribution of prediction errors (PDF) for the multi-threshold-based design and the traditional neighbor prediction method in Fig. 6a, by using the HACC data set (vx field). We can clearly observe that the former leads to a sharper distribution, which means a higher prediction accuracy. On the other hand, the absolute error bound of each data point is dependent on its value, so the size of each quantization bin would get larger than the fixed bin size used in the traditional compression model with a global absolute error bound. The diverse bin sizes will lead to a more intense clustering of the data points in the quantization bins, i.e., more data points will be represented by fewer integer bin indices after the quantization step. We use Fig. 6b to illustrate the distribution of data points located in different quantization bins ($-500 \sim 500$) based on the relative error bound of *0.001* and absolute error bound of *0.1* respectively. In the figure, the two distributions exhibit very similar shapes. This means that in order to reach the similar quantization effectiveness (or compression ratio), the absolute error bound has to be increased up to two orders of magnitude to the relative error bound, leading to a significant loss of precision on the close-to-zero data points.

We adopt Huffman encoding to encode the difference of adjacent group IDs. The reason is that the data may still have a certain coherence, even for the particle's velocity arrays, as as we observed in Fig. 4. That is, the difference of adjacent group IDs is likely to be around 0 in most of cases, as confirmed in Fig. 7. This figure demonstrates the distribution of the differences of adjacent group IDs when the multi-threshold-based strategy is adopted on the vx array of a snapshot in the HACC cosmology simulation. In particular, there are 50 percent of data points belonging to the group 0, and the three most intensive groups (ID=-1, 0, 1) occupy up to 80 percent of data points. In this situation, Huffman encoding can significantly reduce the storage size for keeping group IDs, in that most frequent group IDs would be represented by very short codes.

Remark:

- The multi-threshold-based strategy is particularly suitable for the scientific data that exhibit a rather low spatial coherence (i.e., spiky changes in local areas). The reason is that unlike the block-based strategy, multi-threshold-based strategy does not closely rely on the smoothness/coherence of the data.
- If the value range of the data set is very small, e.g., if it is in the range $[-2,2]$, there would be only a few groups based on the base-2 thresholds. In this case, the user can amplify the value range by introducing a multiplier θ on each data point such that all the data will be mapped to another space which is θ times as large as the original value range, and we call this transform "θ-mapping". Considering the minimization of the transformation overhead, θ is supposed to be set to 2^c such that we just need to add c onto the exponent part of the IEEE 754 representation of each original data value in the θ-mapping. In the course of decompression, the reconstructed data just need to go through a reverse procedure of θ-mapping (i.e., divide every data point by θ) to get the original data value.
- Based on the Equation (5), all the data points whose values are in the range $[-1,1]$ belong to the group ID 0, hence their absolute error bounds are a constant ε. That is, the reconstructed values for these data points may not follow a strict relative error bound, while such a distortion is tolerable because the user can amplify the value range to an expected space on demand, as discussed above.

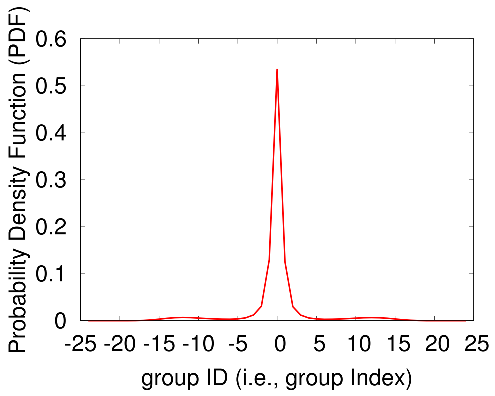


Fig. 7. Distribution of adjacent group ID codes (HACC: vx).

6 Evaluation of Compression Quality

In this section, we evaluate the compression quality of our pointwise relative-error-based compression technique on two real-world HPC simulation data sets, and compare it with other state-of-the-art work.

6.1 Experimental Setting

The state-of-the-art compressors in our evaluation are ZFP [5], FPZIP [19], and ISABELA [18], all of which support pointwise relative-error-based compression to a certain extent. ZFP provides three compression modes: error-bound-based compression, rate-fixed compression, and precision-based compression. The precision-based mode allows users to set the number of bits to keep for the coefficient values in the orthogonal transformed space, so as to achieve an approximate effect of pointwise relative-error based compression. FPZIP also allows users to set a number of bits to

TABLE 4
Comparison of Compression Results Among Different Compressors (CLDLOW Field in CESM-ATM Simulation)

Compressor	Setting	bounded	$\bar{\varepsilon}$	$\max\varepsilon$	PSNR	CR
SZ	**ϵ=1E-2**	**100%**	0.00466	**0.009997**	52.64	**18.6**
(MIN_mode)	**ϵ=1E-4**	**100%**	4.65E-5	**1E-4**	92.56	**4.99**
SZ	ϵ=1E-2	98.6268%	1.03E+7	1.66E+13	52.7	18.67
(AVG_mode)	ϵ=1E-4	98.646%	1.03E+5	1.34E+11	92.7	5.12
SZ	ϵ=1E-2	95.171%	1.5E+8	6.2E+14	52.37	19.63
(MAX_mode)	ϵ=1E-4	95.199%	4.4E+5	1.03E+12	92.4	5.28
	Pc=16	99.96%	4.2E+5	1.26E+12	85.94	4.04
ZFP [5]	Pc=18	99.977%	1.7E+5	4.15E+11	97.86	3.61
	Pc=20	99.984%	2E+5	3.47E+10	109.9	2.98
	Pc=15	87.2515%	0.00566	0.0154	51.32	20.3
FPZIP [19]	**Pc=16**	**100%**	0.00284	**0.0078**	57.32	**15.87**
	Pc=22	97.2724%	4.4E-5	1.22E-4	93.44	4.49
	Pc=23	**100%**	2.2E-5	**6.1E-5**	87.5	**3.95**
ISABELA [18]	ϵ=1E-2	99.9997%	0.00226	1	58.8	2.59
	ϵ=1E-4	99.9952%	5.15E-05	1	92.8	1.39

maintain for each data point, leading to the pointwise relative error controls. Unlike ZFP and FPZIP, ISABELA allows users setting a pointwise relative-error bound explicitly.

In our experiments, the simulation data are from two well-known scientific applications: CESM Atmosphere model (CESM-ATM) and Hardware/Hybrid Accelerated Cosmology Code (HACC). They represent typical meteorological simulation and cosmology simulation respectively. We also evaluate Hurrican simulation data [26], which exhibits the similar compression features with that of CESM-ATM. It is not shown in the paper because of the space limitation.

- CESM is a well-known climate simulation code that may produce large volumes of data every day [20], [27]. The CESM-ATM data comprise 60+ snapshots, each containing 100+ fields. We performed compression on each of them and observed that many fields exhibit similar compression results. Therefore, we illustrate the compression results based on four representative fields (CLDLOW, CLDHGH, FLDSC and PHIS).
- HACC is a large-scale cosmology simulation code developed at Argonne National Laboratory. There are also dozens of snapshots, each of which has 2^{30} particles emulated. Each particle involves six fields (x, y, z, vx, vy, vz) stored as single-precision floating point values, so the total original data size of each snapshot is 24GB in the evaluation.

For the evaluation metric regarding data distortion, we adopt the average relative error, maximum relative error, and peak signal-to-noise ratio (PSNR). PSNR represents the overall distortion of data from the perspective of visualization and is calculated as Formula (3). PSNR is non-negligible even if one chooses the relative error bound to do the compression, because large distortion of overall data visualization is undesired. In addition to PSNR, we will also demonstrate the impact of decompressed data on the visualization of physics-related metrics such as kinetic energy and moving direction of particles in HACC simulation.

We also assess the I/O performance gain when using our lossy compressor against other related work by performing a parallel experiment on a supercomputer [28] using 2,048 cores (i.e., 64 nodes, each with two Intel Xeon E5-2695 v4 processors and 128 GB of memory, and each processor with 16 cores). The storage system uses General Parallel File Systems (GPFS). These file systems are located on a raid array and served by multiple file servers. The I/O and storage systems are typical high-end supercomputer facilities. We use the file-per-process mode with POSIX I/O [29] on each process for reading/writing data in parallel.[1]

6.2 Evaluation Based on CESM-ATM Data

Table 4 presents the compression results generated by different compressors based on the CLDLOW field in the CESM-ATM simulation. Our solution, SZ, has three optional modes—MIN, AVG, and MAX—meaning that the absolute error bound will be calculated based on the minimum, average, and maximum value in each block, respectively. We set the block size in our solution to 5x5 because of its optimality (to be shown later). From the table, we can see that only SZ (MIN_mode) and FPZIP can control the pointwise relative error perfectly. We present the percentage of the data points that meet the relative error bound requirement in the third column. Since ZFP has no explicit relative error bound mode, we compute the bounded percentage for its three compression cases (Pc=16,18,20) based on the relative error bound of 1E-2. Since FPZIP supports only precision setting to control the errors, we have to try different precisions to tune the point-wise relative error bound to a target level. Specifically, Pc=16 and Pc=23 correspond to point-wise relative error bounds of 1E-2 and 1E-4, respectively. In addition, $\bar{\varepsilon}$ and $\max\varepsilon$ denote the average relative error and maximum relative error, respectively; and ϵ refers to the relative error bound setting for SZ. For other compressors, we present the compression results based on a similar maximum relative error or PSNR. SZ(MIN_mode) exhibits the best results, as it not only leads to the highest compression ratio among the compressors studied, but also strictly limits the relative errors. In quantitative terms, when the relative error bound is set to 0.01, SZ(MIN_mode) achieves a compression ratio up to 18.6, which is higher than FPZIP, ZFP, and ISABELA by 17.2, 360, and 618 percent, respectively. SZ(MAX_mode) achieves higher compression ratios because of its more relaxed relative error bound settings. Because of space limits, we cannot present the compression/decompression time here. Basically, we note that SZ, ZFP, and FPZIP lead to similar compression times, while ISABELA is much slower due to its data-sorting step.

We explore the optimal setting of the block size for the relative-error-bound-based compression under the SZ model, as presented in Table 5. We perform the compression evaluation using different block sizes (from 4x4 to 8x8) and observe that the compression qualities are similar. For instance, the compression ratios are in the range of [17.04,18.6] and [4.89,5.02] when the relative error bounds are set to 0.01 and 0.0001, respectively. The reason is that

1. POSIX I/O performance is close to other parallel I/O performance such as MPI-IO [30] when thousands of files are written/read simultaneously on GPFS, as indicated by a recent study [31].

TABLE 5
Compression Results of SZ(MIN) with Different Block Sizes

Compression Ratio					
Setting	**4x4**	**5x5**	**6x6**	**7x7**	**8x8**
1E-2	17.04	18.6	17.13	18.21	17.48
1E-4	4.89	4.99	5.01	5.02	5.02
Maximum Relative Error (i.e., $\max\varepsilon$)					
Setting	**4x4**	**5x5**	**6x6**	**7x7**	**8x8**
1E-2	0.01	0.009997	0.00998	0.009998	0.009998
1E-4	1E-4	1E-4	1E-4	1E-4	1E-4
PSNR					
Setting	**4x4**	**5x5**	**6x6**	**7x7**	**8x8**
1E-2	52.27	52.64	52.92	52.99	53.1
1E-4	92.33	92.56	92.86	92.68	92.83

the climate simulation data exhibit relatively high smoothness in the space, such that the absolute error bound approximated for each block changes little with different block sizes as long as the block size stays small. We can also observe that the maximum relative errors are always close or exactly equal to specified bounds (1E-2 or 1E-4).

We present in Fig. 8 the overall compression ratios based on four representative fields, with various relative error bounds under the four different compressors. We set precisions of FPZIP to 13, 16, 19 and 23, respectively, because such settings can respect the point-wise relative error bound of 1E-1, 1E-2, 1E-3, and 1E-4, respectively. ZFP adopts the precisions of 14, 16, 18, and 20, respectively. We adopt the block-based strategy on all fields except for PHIS which adopts threshold-based strategy because pretty small values are scattered throughout this dataset (leading to severely over-preserved precisions unexpectedly in the block-based strategy). Based on Fig. 8, SZ leads to about 50 percent higher compression ratios than the second best compressor FPZIP based on the point-wise relative error bound.

Table 6 presents the compression rate and decompression rate of compressing CESM-ATM data by the four state-of-the-art compressors respectively. The relative error bound is set to 1E-2 for SZ and ISABLEA; the precision bit-count is set to 16 and 20 for FPZIP and ZFP respectively, such that they will have approximate relative errors with SZ and ISABELA according to our above analysis. The four compressors are all executed on the Argonne Bebop cluster

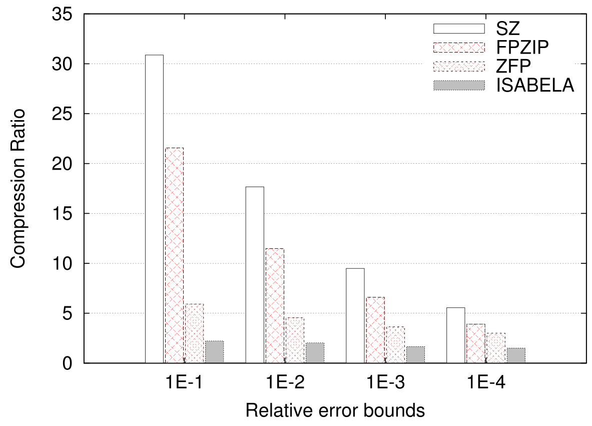


Fig. 8. Compression ratio on CESM-ATM data.

TABLE 6
Compression/Decompression Rate (MB/s): CESM-ATM Data

Compressor	CLDLOW	CLDHGH	FLDSC	PHIS
SZ	53.7/130	54.9/137	65.1/190	49.3/99
FPZIP	95.1/88	85.2/82.4	145.4/117.7	79.7/75
ZFP	103/118	91.6/95	103/145	80/103
ISABELA	4.05/12.5	4.34/13.4	4.86/13.5	1.01/12.6

server [28] and Table 6 demonstrates the single-core processing rates. It is observed that FPZIP has the fastest speed on compression while SZ is the fastest compressor on decompression. Specifically, SZ is slower than FPZIP by 35-56 percent on compression while it is faster than FPZIP by 48-66 percent on decompression. ISABELA exhibits much lower compression rate than others because of its costly sorting operation during the compression. In addition, in our evaluation, we also observe that SZ exhibits similar compression/decompression rate when using the absolute error bound setting and relative error bound setting respectively. The relative error bound based compression is faster sometimes because of the denser distribution of quantization bins leading to a more efficient Huffman encoding operation, while the absolute error bound based compression could be faster because of the cost in calculating statistical error bound for each block in the block-based strategy.

We present the visualization images about absolute-error-bound-based compression and pointwise-relative-error-based compression in Fig. 9. The absolute error bound and relative error bound are set to 1.1E-3 and 1E-2, respectively, such that they have the same compression ratio. We observe that the two figures both look the same as the original image at the full resolution to the naked eyes, indicating that the overall visualization effect under the relative error bound is not degraded at all.

Since the smaller the value the lower the error bound, the relative-error-based compression is supposed to have greater ability to retain the details than does the absolute-error-bound-based compression, in the areas with a lot of small data values. This is confirmed in Fig. 10, which evaluates PSNR with different precisions between the two compression modes using four representative fields in CESM-ATM. We set the relative error bound to 1E-5 and compare the two compression modes under the same compression ratio, with different levels of *visual granularity*. The visual granularity is a threshold to select the areas with small

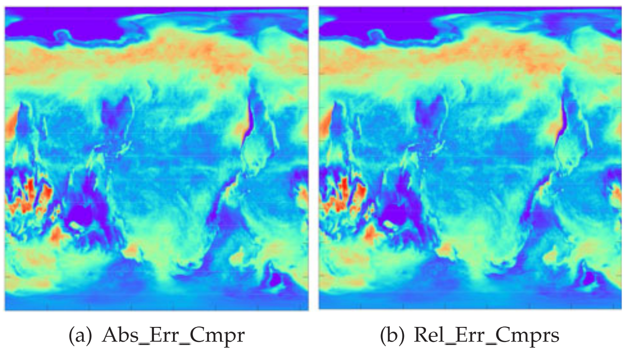


Fig. 9. Visualization of decompressed data (CLDLOW) with different error bounds.

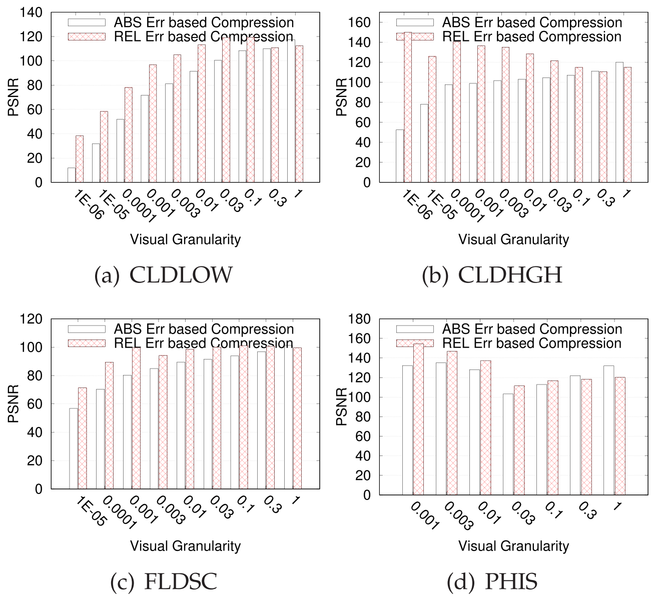


Fig. 10. Evaluation of PSNR with different visual granularity.

values in the data set. For instance, if it is set to 0.01, only the data values that are lower or equal to 0.01 will be selected to check the PSNR. Through the figure, we can clearly see that the PSNR could be retained at a high level with finer visual granularity under the relative error based compression, while it degrades significantly in the absolute-error-based compression.

In Fig. 11, we compare the visualization image of the decompressed data under the two compression modes using the field CLDLOW with the value range [0,0.001]. Because of the space limitation, we do not show the original image, which looks exactly the same as Fig. 11b. The MAX_-mode with ϵ=0.01 is adopted for the relative-error-based compression. We can observe that the image under relative-error-based compression is exactly the same as the original one, which confirms the high effectiveness of our relative-error-based compression technique to maintain details. By comparison, the absolute-error-based compression distorts the visualization considerably in the areas with small values. For instance, some areas that were originally blue/purple exhibit other colors (such as yellow and cyan) because of the changed values after the reconstruction.

6.3 Evaluation Based on HACC Data

In this section, we present the evaluation results by using different lossy compressors on the HACC data set. As discussed

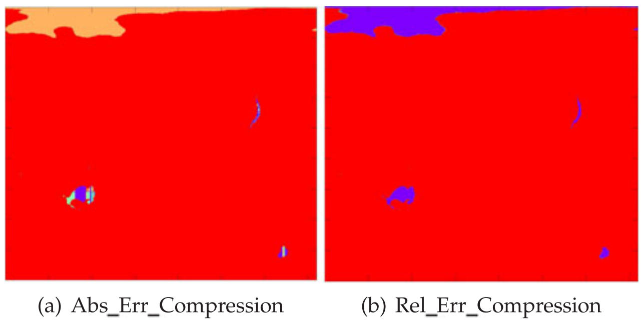


Fig. 11. Visualization of decompressed data (CLDLOW:[0,1E-3]).

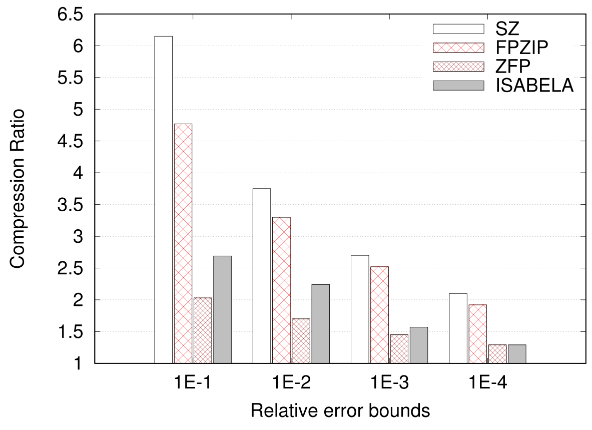


Fig. 12. Compression ratio on HACC velocity data.

previously, each particle involves six fields, which can be split into two categories - position and velocity. According to the HACC developers, the users require to use absolute error bound to compress position values while they hope to compress the velocity values using point-wise relative error bounds. The reason is that the different positions of the particles in the simulation are equivalently significant in space, yet the lower velocity of a particle, the higher demand on the compression precision in general. Accordingly, we mainly focus on the evaluation of the velocity data compression with point-wise relative error bound in our experiment.

In what follows, we first compare the compression quality among different lossy compressors (including SZ, ZFP, FPZIP and ISABELA), in terms of compression ratio and compression/decompression rate. Then, we analyze the impact of decompressed data on particle's kinetic energy and moving direction, based on absolute error bound and relative error bound respectively. We observe that the compression with relative error bound can keep a very satisfactory overall visualization effect on both kinetic energy and moving direction, and also keep the details of low-velocity particles more accurately than the compression with absolute error bound under the same compression ratio.

Fig. 12 presents the point-wise relative error bound based compression ratio of the four compressors, using the HACC velocity data (vx, vy and vz). As for FPZIP and ZFP, we ran them using multiple precisions and select the results with the maximum relative errors closest to target error bound. We can observe that SZ leads to 31-210 percent higher compression ratio than other compressors do. The key reason SZ works more effectively than others on HACC dataset is that we adopt the multi-threshold-based strategy which can effectively reduce the prediction errors (as shown in Fig. 6a) and the group ID could also be compressed effectively because of the sharp distribution of their corresponding codes (as demonstrated in Fig. 7).

Table 7 shows the single-core compression/decompression rates of the four compressors running on Argonne Bebop cluster server [28], using HACC velocity data with the relative error bound of 0.01. It is observed that FPZIP and SZ are the fastest compressors with respect to the compression and decompression respectively. Specifically, FPZIP is faster than SZ by about 50 percent on the compression while SZ is faster than FPZIP by 30 percent on the decompression. By comparison, ISABELA suffers from very

TABLE 7
Compression/Decompression Rate (MB/s): HACC Data

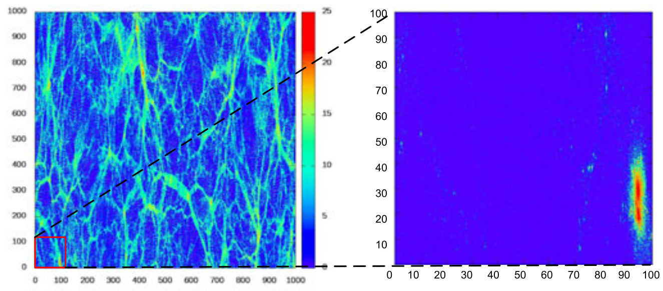
Compressor	vx	vy	vz
SZ	48/88	46.8/77	50/90
FPZIP	71.5/68.7	74.5/69.2	74.2/68.3
ZFP	68.7/60.1	68.2/59.2	66.1/60.2
ISABELA	2.95/14.9	2.85/15.1	2.9/15.3

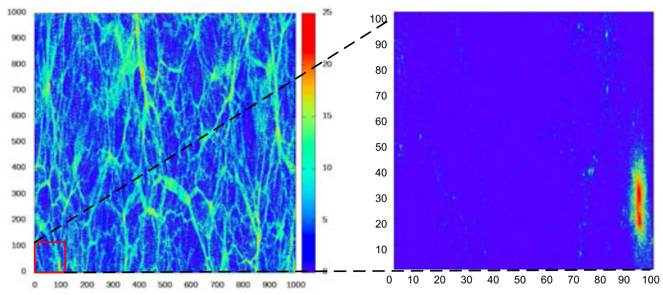
low compression/decompression rate because of its costly sorting operation.

As for the impact of decompressed data in the analysis, we first study particles' kinetic energy in the simulation space. A particle's kinetic energy is defined as follows:

$$E(particle_i) = \frac{1}{2} m_i \cdot |v_i|^2, \tag{6}$$

where m_i is the mass of the particle and $|v_i|$ is the potential of the velocity vector v_i. Since HACC simulation assumes that each particle has the same mass, we set m_i to 1 in our evaluation. In order to compare the compression quality using absolute error bound versus relative error bound, we compress the data with absolute error bound = 6.2 and relative error bound = 0.1 respectively, because they lead to the same compression ratio (about 6.2:1).


(a) Raw data visualization


(b) Decompressed data with ABS_ERR=6.2

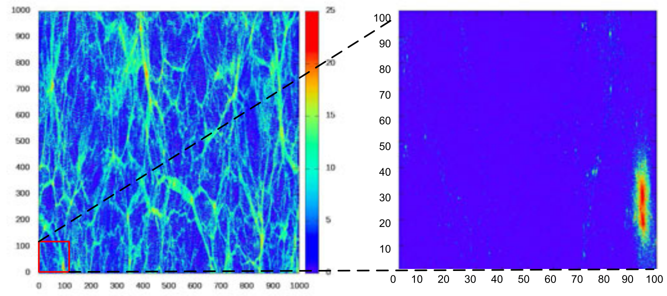
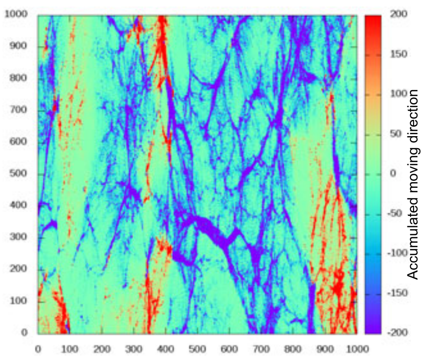
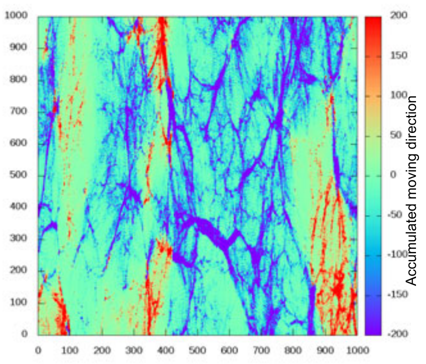

(c) Decompressed data with REL_ERR=0.1

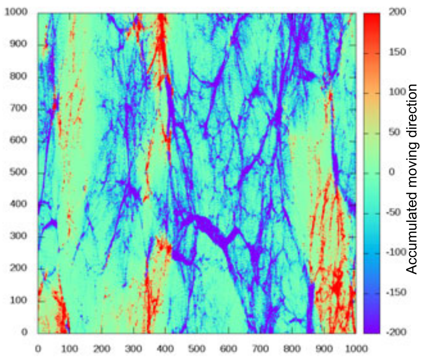
Fig. 13. Visualization of accumulated kinetic energy (HACC).



(a) Original raw data



(b) Decompressed data with ABS_ERR=6.2



(c) Decompressed data with REL_ERR=0.1

Fig. 14. Particle moving direction along Z-axis (HACC).

Fig. 13 plots the slice images of the accumulated kinetic energy in the space. Specifically, we split the space into 1,000 slices each having 1000×1000 blocks and then aggregate the kinetic energy of the particles in the middle 10 slices, plotting the slice images based on three datasets (original raw data and two decompressed datasets with different types of error bounds) respectively. We also zoom in the bottom-left square by splitting it further into 1000×1000 small blocks to observe the possible difference. We compare the three slice images at a full resolution, while they look exactly the same with each other. This means that the point-wise relative error bound = 0.1 can already lead to a satisfactory effect from the perspective of the overall visualization.

We also analyze the impact of the decompressed data on the aggregated moving directions of the particles in the simulation. We just analyze the moving direction along the Z-axis here, because the studies based on Y-axis and X-axis lead to the similar results. The aggregated Z-axis moving direction of the particles in a block is defined as as the sum of the moving direction of the particles along Z-axis in the

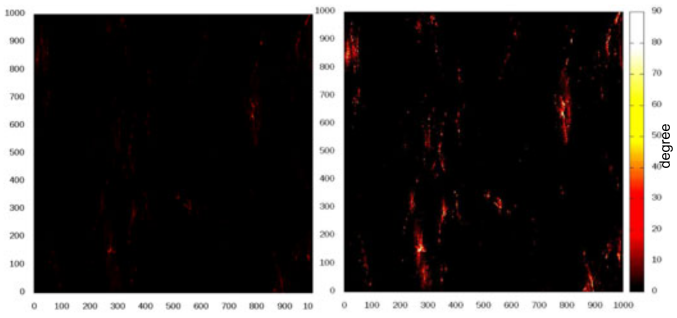

(a) REL_ERR=0.1 (b) ABS_ERR=6.2

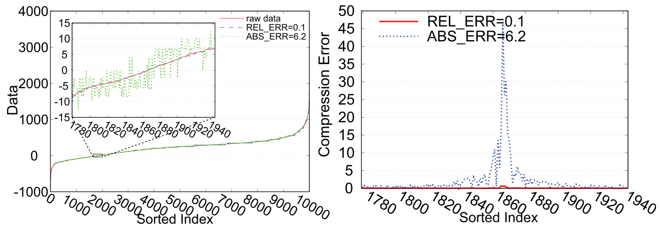
Fig. 15. Accumulated speed angles between original particle and decompressed particle (HACC).

same block, as shown blow:

$$Amd_{(Z\text{-}axis)} = \sum_{particle_i \in Block_I} \frac{vz_i}{\sqrt{vx_i^2 + vy_i^2 + vz_i^2}}, \tag{7}$$

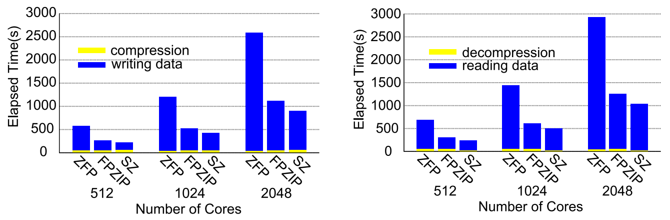
where vx_i, vy_i and vz_i are the velocity values along the three dimensions respectively. Fig. 14 presents the aggregated moving direction of the 10 middle slices for the original data and decompressed data respectively. The regions with positive values and negative values indicate the overall particles in those regions are moving closer to or farther away from the observer respectively. We can see that the two types of decompressed data are both very satisfactory to users from the perspective of the overall visualization effect. Specifically, it is extremely hard to observe a tiny difference among the three images even though we zoom in them to a very high resolution.

However, the compression with point-wise relative error bound can retain the details much more effectively than the compression with absolute error bound, especially for the low-velocity particles, as illustrated by Fig. 15. This figure presents the accumulated speed's angles between original particles and their corresponding decompressed particles, based on the particles located in the middle 10 slices and with velocity potential lower than 100. In this figure, the brighter color a region exhibits, the larger the distortion (i.e., accumulated speed's angles) between the original particles and decompressed particles in that region. We can clearly observe that the decompressed data with absolute error bound leads to much larger discrepancies of the speed angles from the original particles than the decompressed data with relative error bound does. The reason is that the former suffers from much larger compression errors for the close-to-zero velocity values as confirmed in Fig. 16.



(a) decompression data vs. raw data (b) relative errors

Fig. 16. Analysis of compression errors based on HACC velocity).



(a) Dumping data to PFS (b) Loading data from PFS

Fig. 17. Parallel performance evaluation using HACC (velocity fields)).

Fig. 16 presents the absolute errors and relative errors for the two types of compressions respectively. We select the first 10,000 particles to demonstrate the compression errors and other particles exhibit the similar results. We also sorted the compression errors for the purpose of the simpler observation. Based on Fig. 16a, it is observed that the overall absolute errors under the two compression types are both negligible compared to their data values for the high-velocity particles with fast speeds. However, the zoomed-in figure shows that compression with relative error bound has little absolute errors while the compression with absolute error bound suffers from too large errors to tolerate. In other words, the decompressed velocities of the low-speed particles are invalid under the compression with absolute error bound. This can be explained clearly using Fig. 16b, which shows the relative errors of the low-speed particles under the two types of compressions. Specifically, the maximum relative error could reach up to 45 times as large as the data value under the absolute-error-bound based compression, while it is only about 50 percent in the worse case for our designed relative-error-based compression.

We finally perform a parallel experiment with up to 2,048 cores from a supercomputer [28] to assess the I/O performance gain under our proposed compressor, as shown in Fig. 17. Specifically, our experiment is weak-scaling: i.e., we lunched different numbers of ranks in parallel and let each rank write/load simulation data in the same size such that the total data size increases with the execution scales. We adopt the point-wise relative error bound of 0.1 for SZ and FPZIP, because such a setting already leads to a good visual quality as we discussed above. For ZFP, we adopt the precision of 20. As a comparison, the total I/O times of writing the original data to the PFS is about 5,600 seconds, and the I/O time of reading data is about 5,800 seconds, when running the parallel experiment with 2,048 cores. As shown in Fig. 17a, when using 2,048 cores, the total time of dumping simulation data (i.e., compression time + writing compressed data) is only about 895 seconds when equipped with our compressor, which means a throughput improvement of 5.25X compared with the original data writing time and an improvement of 24 percent than the second-best solution using FPZIP. The data loading throughput can be improved by 560+ percent compared with reading the original dataset and by 21 percent than the solution employing the second best compressor FPZIP, when using 2,048 cores.

7 Conclusion and Future Work

Point-wise relative error bound is significant to respect the multi-resolution demand duringlossy compression. In this paper, we present two novel strategies that can significantly improve the compression ratio based on the relative error bound. The strategy A (called block-based strategy) splits the whole data set into multiple small blocks and then computes the real absolute error bound for each block, which is particularly effective on the compression of relatively smooth data with multiple dimensions. Strategy B (called multi-threshold-based strategy) splits the entire data value range into multiple groups with exponential sizes. This method is particularly effective on the data that exhibits spiky value ranges (e.g., the 1D data arrays in N-body simulation). We evaluate our proposed compression methods using two well-known datasets from the climate simulation and N-body simulation community respectively. The key insights are summarized as follows:

- Our relative error bound based compression strategies exhibit the best compression ratios than other state-of-the-art compressors on both of the two datasets. Specifically, the compression ratio of SZ is higher than the second-best compressor (FPZIP) by 17.2–618 percent on CESM-ATM climate simulation data and by 31–210 percent on HACC particular simulation data.
- Based on both CESM-ATM data and HACC simulation data, we demonstrate that the relative error bound based compression can effectively retain more details than the absolute error bound based compression does, without any degradation of overall data visualization.
- As for compression/decompression rate, FPZIP runs the fastest on the compression based on relative error bound, while SZ runs the fastest on the decompression from among all the compressors.
- Parallel experiments with up to 2,048 cores show that SZ improves the I/O performance by 520-560 percent than writing/reading the original HACC data, and by 21-24 percent than the second best compressor FPZIP.

In the future, we plan to customize more effective strategies to further improve the compression ratio for the non-smooth data sets such as N-body simulation data.

Acknowledgments

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative. The material was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357, and supported by the US National Science Foundation under Grant No. 1619253.

References

[1] SZ 1.4.13.5. [Online]. Available: https://github.com/disheng222/SZ/archive/v1.4.13.5.tar.gz, Jun. 2018.
[2] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, and K. Heitmann, "HACC: Extreme scaling and performance across diverse architectures," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2013, pp. 1–10.
[3] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *Proc. IEEE 30th Int. Parallel Distrib. Process. Symp.*, 2016, pp. 730–739.
[4] 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.*, 2017, pp. 1129–1139.
[5] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2674–2683, Dec. 2014.
[6] Gzip compression. [Online]. Available: http://www.gzip.org
[7] BlosC compressor. [Online]. Available: http://blosc.org
[8] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.
[9] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Proc. Data Compression Conf.*, 2006, pp. 133–142.
[10] B. E. Usevitch, "JPEG2000 compatible lossless coding of floating-point data," *J. Image Video Process.*, vol. 2007, no. 1, p. 22, 2007.
[11] M. Burtscher and P. Ratanaworabhan, "High throughput compression of double-precision floating-point data," in *Proc. Data Compression Conf.*, 2007, pp. 293–302.
[12] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka, "Exploration of lossy compression for application-level checkpoint/restart," in *Proc. IEEE 29th Int. Parallel Distrib. Process. Symp.*, 2015, pp. 914–922.
[13] Z. Chen, S. W. Son, W. Hendrix, A. Agrawal, W. Liao, and A. Choudhary, "NUMARCK: Machine learning algorithm for resiliency and checkpointing," in *Proc. IEEE/ACM Supercomput.*, 2014, pp. 733–744.
[14] S. Di and F. Cappello, "Optimization of error-bounded lossy compression for hard-to-compress HPC data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 129–143, Jan. 1 2018.
[15] A. Omeltchenko, T. J. Campbell, R. K. Kalia, X. Liu, A. Nakano, and P. Vashishta, "Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm," *J. Comput. Physics Commun.*, vol. 131, no. 1, pp. 78–85, 2000.
[16] D. Tao, S. Di, Z. Chen, and F. Cappello, "In-depth exploration of single-snapshot lossy compression techniques for N-Body simulations," in *Proc. IEEE Int. Conf. Big Data*, 2017, pp. 486–493.
[17] D. Lee, A. Sim, J. Choi, and K. Wu, "Novel data reduction based on statistical similarity," in *Proc. 28th Int. Conf. Sci. Statistical Database Manage.*, 2016, pp. 21:1–21:12.
[18] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data," in *Proc. 17th Eur. Conf. Parallel Process.*, 2011, pp. 366–379.
[19] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 1245–1250, Sep./Oct. 2006.
[20] Community Earth Simulation Model (CESM). [Online]. Available: https://www2.cesm.ucar.edu/
[21] Gzip. [Online]. Available: https://zlib.net
[22] S. Paul and B. Bandyopadhyay, "A novel approach for image compression based on multi-level image thresholding using Shannon entropy and differential evolution," in *Proc. IEEE Students' Technol. Symp.*, 2014, pp. 56–61.
[23] H. Rekha and P. Samundiswary, "Image compression using multilevel thresholding based absolute moment block truncation coding for WSN," in *Proc. Int. Conf. Wireless Commun. Signal Process. Netw.*, 2016, pp. 396–400.
[24] S. G. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Trans. Image Process.*, vol. 9, no. 9, pp. 1532–1546, Sep. 2000.
[25] M. Cheriet, J. N. Said, and C. Y. Suen, "A recursive thresholding technique for image segmentation," *IEEE Trans. Image Process.*, vol. 7, no. 6, pp. 918–921, Jun. 1998.
[26] Hurrican ISABEL simulation data. [Online]. Available: http://vis.computer.org/vis2004contest/data.html, 2004.
[27] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, and S. A. Mickelson, "A methodology for evaluating the impact of data compression on climate simulation data," in *Proc. 23rd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2014, pp. 203–214.
[28] Bebop cluster. [Online]. Available: https://www.lcrc.anl.gov/systems/resources/bebop
[29] B. Welch, "POSIX IO extensions for HPC," in *Proc. 4th USENIX Conf. File Storage Technol.*, 2005, pp. 1–24.

[30] R. Thakur, W. Gropp, and E. Lusk, "On implementing MPI-IO portably and with high performance," in *Proc. 6th Workshop I/O Parallel Distrib. Syst.*, 1999, pp. 23–32.

[31] A. Turner, "Parallel I/O performance." [Online]. Available: https://www.archer.ac.uk/training/virtual/2017–02-08-Parallel-IO/2017_02_ParallelIO_ARCHERWebinar.pdf, 2017.

Sheng Di received the master's degree from the Huazhong University of Science and Technology, in 2007 and the PhD degree from the University of Hong Kong, in 2011. He is currently an assistant computer scientist with Argonne National Laboratory. His research interest includes resilience on high-performance computing (such as silent data corruption, optimization checkpoint model, and in-situ data compression) and broad research topics on cloud computing (including optimization of resource allocation, cloud network topology, and prediction of cloud workload/hostload). He is working on multiple HPC projects, such as detection of silent data corruption, characterization of failures and faults for HPC systems, and optimization of multilevel checkpoint models. He is a senior member of the IEEE.

Dingwen Tao received the bachelor's degree in mathematics from the University of Science and Technology of China, in 2013 and the PhD degree in computer science from the University of California, Riverside, in 2018. He is currently an assistant professor of computer science with the University of Alabama. Prior to this, he worked as an R&D intern with Brookhaven National Laboratory, Argonne National Laboratory, and Pacific Northwest National Laboratory. His research interests include high-performance computing, parallel and distributed systems, big data analytics, resilience and fault tolerance, scientific data compression algorithms and software, numerical algorithms and software, etc. He has published more than 15 peer-reviewed papers in top HPC and parallel and distributed conferences and journals, such as IEEE BigData, Cluster, IPDPS, the *IEEE Transactions on Parallel and Distributed Systems*, the *ACM High-Performance Parallel and Distributed Computing*, PPoPP, ACM/IEEE SC, and the *International Journal of High Performance Computing Applications*. He is a member of the IEEE.

Xin Liang received the bachelor's degree from Peking University, in 2014. He is working toward the PhD degree at the University of California, Riverside. His research interest includes parallel and distributed systems, fault tolerance, high-performance computing, large-scale machine learning, big data analysis, and quantum computing.

Franck Cappello is a program manager and senior computer scientist with ANL. Before moving to ANL, he held a joint position with Inria and the University of Illinois at Urbana Champaign, where he initiated and co-directed from 2009 the Inria Illinois-ANL Joint Laboratory on Petascale Computing. Until 2008, he led a team with Inria, where he initiated the XtremWeb (Desktop Grid) and MPICH-V (fault-tolerant MPI) projects. From 2003 to 2008, he initiated and directed the Grid5000 project, a nationwide computer science platform for research in large-scale distributed systems. He has authored papers in the domains of fault tolerance, high-performance computing, Grids and contributed to more than 70 program committees. He is an editorial board member of the *IEEE Transactions on Parallel and Distributed Systems*, the *International Journal on Grid Computing*, the *Journal of Grid and Utility Computing*, and the *Journal of Cluster Computing*. He is/was program co-chair for IEEE CCGRID 2017, Award chair for ACM/IEEE SC15, Program co-chair for ACM HPDC2014, Test of time award chair for IEEE/ACM SC13, Tutorial co-chair of IEEE/ACM SC12, Technical papers co-chair at IEEE/ACM SC11, Program chair of HiPC2011, pPogram co-chair of IEEE CCGRID 2009, Program Area co-chair IEEE/ACM SC09, general chair of IEEE HPDC 2006. He is a fellow of the IEEE and a member of the ACM.