zPerf: A Statistical Gray-box Approach to Performance Modeling and Extrapolation for Scientific Lossy Compression

Jinzhen Wang, Qi Chen, Tong Liu, Qing Liu*, Xubin He, *Senior Member, IEEE*

Abstract—As simulation-based scientific discovery is being further scaled up on high-performance computing systems, the disparity between compute and I/O has continued to widen. As such, domain scientists are forced to save only a small portion of their simulation data to persistent storage, and as a consequence, important physics may be discarded for data analysis. Error-bounded lossy compression has made tremendous progress recently in bridging the gap between compute and I/O, and played an increasingly important role in science productions. Nevertheless, a key hurdle to the wide adoption of lossy compression is the lack of understanding of compression performance, which is the result of the complex interaction between data, error bound, and compression algorithm. In this work, we present zPerf, a statistical gray-box performance modeling approach for scientific lossy compression. Our contributions are threefold: 1) We develop zPerf to estimate the performance of prediction-based and transform-based lossy compression techniques, based on in-depth understanding and statistical modeling for data features and core compression metrics; 2) We demonstrate the in-detailed implementation of zPerf using two case studies, where we derive the performance modeling for SZ and ZFP, two leading lossy compressors; 3) We evaluate the effectiveness of zPerf on real-world datasets across various domains. Based on the evaluation, we demonstrate the efficacy of zPerf performance model; 4) We further discuss three case studies where zPerf is applied to extrapolate the compression ratio of SZ and ZFP with alternative encoding schemes as well as ZFP with an alternative transform scheme. Through the case studies, we demonstrate the potential of zPerf for exploring the design space of lossy compression, which has hardly been studied in the literature.

Index Terms—Lossy compression, modeling, performance

1 Introduction

As simulation-based scientific discovery further scales up the model fidelity and resolution as enabled by the next-generation high-performance computing (HPC) systems, the disparity between compute and I/O has continued to widen, to the extent that only a disproportionately small amount of data from the simulation can be saved for post-processing. A consequence is that, despite the improved fidelity on the simulation side, important physics in the data can still be discarded for the subsequent data analysis. To date, there has been a multitude of efforts to reduce the I/O overhead for large-scale simulations, including in situ processing [1], [2], [3], [4], storage layer optimization (e.g., burst buffer) [5], [6], [7], [8], and data reduction [9], [10], [11], [12]. In particular, lossy compression by trading accuracy for performance has shown great promise to fundamentally solve the I/O challenge, when used in conjunction with other methods across the software/hardware stack. Nevertheless, a major roadblock to the wide adoption of lossy compression is the lack of understanding of compression performance, which are the results of complex interaction between between data, error bound, and compression algorithm. A question that domain scientists are often faced with is whether a substantial compression ratio (e.g., $> 10\times$) can be achieved for a given dataset under a realistic error bound. If not, domain scientists may forgo compressing their data so that at least they can fully preserve the data and avoid paying the precious compute time for compression. Unfortunately, in practice, such a question is mostly answered through trial-and-error, which is cumbersome and costly for large simulations.

A straightforward yet effective method to estimate the compression performance (ratio and throughput) is through sampling, which was recently explored [13]. Namely, one can generate a small sample of data and use the compression performance of the sampled data to approximate that of the full data. The key advantages of this approach are: 1) The sampled data share the similar characteristics with the full data; 2) This method treats compressors as black-box, which is easy to implement; Nevertheless, the sampling based approach does not easily allow for exploring the design space of compression without in-depth code changes in a compressor. For example, to assess *how much the performance will improve if a component in a compressor is replaced by a list of candidate solutions*, all candidate solutions must be integrated into the compressor, before one can understand the performance outcome.

This paper aims to estimate the compression perfor-

- *J. Wang, Q. Liu are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, 07102. E-mail: {jw447, qing.liu}@njit.edu*
- *Q. Chen is with the Software College of Northeastern University, China. E-mail: {20195791}@stu.neu.edu.cn*
- *T. Liu and X. He are with the Department of Computer and Information Sciences at Temple University. E-mail: {tongliu, xubin.he}@temple.edu*
- ** Corresponding author*

mance based upon the inner design of a compressor as much as possible so that we can more conveniently explore the design space of lossy compression. In particular, this paper makes the following contributions:

- We develop zPerf, a statistical gray-box approach for lossy compression performance modeling. In contrast to the sampling based approach, our work models the compression performance leveraging the statistical distribution of data features and core compression metrics. Detailed analytical models are developed for prediction-based and transform-based lossy compressors that represent different compression philosophies.
- We demonstrate two case studies, where in-detailed implementations of zPerf is derived for SZ [10], [14], [15] and ZFP [11], two state-of-the-art lossy compressors.
- We evaluate the effectiveness of zPerf across real-world datasets from various domains and verify that zPerf can achieve reasonable accuracy.
- We further use zPerf for design space explorations where we extrapolate the compression ratio of SZ and ZFP with alternative encoding schemes, as well as ZFP with an alternative transform scheme. Our results show that the performance extrapolation is accurate and the alternative encoding and transform schemes outperform the original ones in some application scenarios.

The rest of this paper is organized as follows. We first discuss the related work and motivation in Section 2 and Section 3 respectively. Then in Section 4, we present the general methodologies of zPerf as well as the implementation for two state-of-the-art lossy compression techniques, SZ and ZFP. In Section 5, we present the evaluation results of zPerf for SZ and ZFP over real-world scientific datasets along with error analysis. In Section 6, we discuss the application of zPerf for design space exploration of lossy compression, along with conclusions in Section 7.

2 RELATED WORK

In this section, we present the background and related work on lossy compression and performance modeling.

2.1 Error-bounded lossy compression

Error-bounded lossy compression techniques have been proven to be effective in reducing data volume while satisfying the accuracy based on user requirements. Motivated by the reduction potential of spline functions [16], [17], ISABELA [18] is designed to compress spatial-temporal scientific data through pre-sorting and B-spline prediction, which makes the random input data more compressible. However, the sorting operation loses locality information, and therefore an extra index needs to be stored which hurts the performance of reduction. ZFP [11] adopts a block-wise non-orthogonal transform and a customized embedded encoding scheme for compression. The non-orthogonal transform removes the data redundancy within each data block while variable-length embedded encoding compress transform coefficients one bit plane at a time. Due to the block-wise compression design, it offers random access to compressed data blocks. Di *et al.* present SZ [10], [14], [15], a curve-fitting based lossy compression algorithm. It adopts multiple curve-fitting schemes to remove the data redundancy. Then the curve-fit data points go through linear quantization and Huffman encoding, while the curve-missed data points are stored with binary representation. Among numerous lossy compression techniques, SZ and ZFP are demonstrated to be the two best error-bounded lossy compressors [13]. MGARD [12], [20] is a recently developed lossy compressor that specializes in multi-level recomposition and progressive retrieval. During compression, it decomposes original data into orthogonal coefficient levels and encodes coefficient levels using bit plane encoding. During decompression, bit planes from coefficient levels are selected based on error bounds via greedy search to achieve minimal data retrieval.

2.2 Compression performance modeling

While lossy compression can achieve higher performance than their lossless counterparts, they are often hard to choose from and configure. As a result, the performance understanding and modeling are becoming increasingly important for choosing the right compression technique and suitable configurations. Lu *et al.* [13] propose a sampling-based estimation approach for the compression ratio of SZ. The estimation is based on the similarities between the distribution of quantization levels for the full and sampled datasets. Underwood [21] proposes to search for the error bound configuration towards a target compression ratio for lossy compressors. It adopts an iterative auto-tuning approach that treats lossy compressors as black-box. While it is capable of capturing the compression ratio of lossy compressors, the black-box iterative approach may result in significant overhead in both time and memory footprint. It also lacks the ability to explore the design space of lossy compression. Tao *et al.* [22] aim to estimate rate-distortion for SZ and ZFP using sampled datasets. It incorporates the inner mechanism of SZ and ZFP into the design of online selection model. While it is capable of capturing the bit-rate and PSNR for SZ and ZFP, it does not take the compression time performance into account. Zhao *et al.* [23] present a compression-principle-based method to estimate the compression quality of SZ across different parameter settings. Such a method takes the compression design, for example, how the data is predicted and quantized, into account. Jin *et al.* [24] proposes a general-purpose analytical model to capture the ratio-quality for prediction-based lossy compression. It incorporates the modeling of quantized prediction error and the modeling of Huffman encoding efficiency to achieve the estimation of compression ratio. Another previous work by Jin *et al.* [25] aims to estimate the impact of lossy compression on post-analysis of cosmology simulation. It provides an optimization guideline to select the best-fit error-bound for each partition of the cosmology data to achieve the maximum compression efficiency. Qin *et al.* [26] propose to use deep neural networks (DNN) to capture the compressibility of a dataset for a given lossy compressor. The proposed DNN model classifies a dataset as compressible or incompressible based on a set of general features as well as compression-dependent features. The model can also capture the compression ratio based on error tolerance, however the estimation is black-box due to the nature of DNN model.

3 MOTIVATION

In the past, the compression ratio modeling has been attempted mostly using a black-box approach [13], [21], [22], [27]. Overall, the idea was to extract the salient properties of data and use the compression performance under an inexpensive setting to extrapolate that under a target setting. In particular, this can be achieved by properly sampling the dataset, measuring the compression performance of the sampled dataset, and further estimating the performance of the original dataset. Unless the sampling ratio is too small, it is anticipated that the original data and sampled data share similar characteristics, and therefore the compression performance of the sampled data can be used to approximate that of the full data. However, the prior approach suffers from the following weaknesses:

Motivation 1: The sampling based estimation does not allow us to explore the large design space of compression easily. Scientific lossy compression techniques undergo modifications frequently to improve the general performance or suit the requirements of different applications. For example, SZ recently incorporated second-order Lorenzo and regression predictors to improve the compression performance [23]. Given the large design space of compression algorithms, the compressor developers may wish to understand whether replacing a component in a compressor, e.g., the entropy encoder for quantized data, with a list of candidate solutions would lead to a substantial performance improvement.

Such design space explorations are useful to identify possible research and development opportunities for lossy compression before more labor-intensive software development is underway. However, the sampling based approach does require all candidate solutions to be fully integrated into the compressor prior to the assessment of performance outcomes, which is costly and error prone.

Motivation 2: For offline compression where data needs to be first retrieved from persistent storage, the I/O overhead of the sampling based estimation is high for extreme-scale datasets. While one could choose a low sampling ratio (e.g., $< 0.1\%$) to somewhat reduce the I/O overhead, the resulting accuracy of estimation decreases rapidly with the sampling ratio. Fig. 1 shows the estimation error of compression ratio across a range of sample ratios for SZ. It is shown that once the sampling ratio is lower than 1E-4, the estimation error exceeds 80%, and therefore a low sampling ratio is not desirable.

Motivated by the weaknesses of the sampling based approach, this work aims to develop a gray-box approach where key components within a compressor are modeled to allow for performance estimation and extrapolation.

4 GRAY-BOX COMPRESSION MODELING

In this section, we first discuss the general methodologies of zPerf gray-box performance modeling for prediction-based and transform-based lossy compression techniques. We then use SZ and ZFP as two case studies to show the detailed implementation. For the convenience of discussion, we list the notations in TABLE 1.

4.1 zPerf for prediction-based and transform-based compression

Modern error-bounded lossy compression techniques can be generally divided into two categories: prediction-based and transform-based, depending on how they remove redundancies within the dataset.

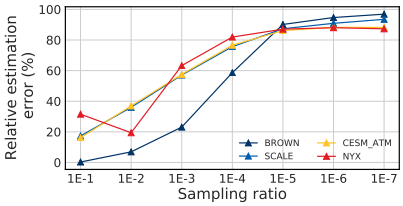


Fig. 1: Estimation error of compression ratio vs. sampling ratio for SZ. The random sampling is used in this experiment. The estimation error is averaged across relative error bounds of 1E-6, 1E-5, 1E-4, and 1E-3.

In general, prediction-based lossy compression tries to represent data points with a prediction model. Then those data points that can be predicted by the model are converted to discrete quantization codes, followed by an entropy encoding. SZ [10], [14], [15], FPZIP [30], and ISABELA [18] are three typical examples of the prediction-based lossy compression techniques. The output of prediction-based lossy compression mainly consists of the encoded quantization codes, as well as those data points cannot be predicted by predictor. The size of encoded data generally depends on the entropy encoding efficiency (i.e., bit rate) while the unpredictable data size essentially depends on the prediction efficiency (the percentage of predicted data points).

On the other hand, transform-based lossy compression performs orthogonal transforms that map the original data to de-correlated coefficients. In general, more coefficients will be close to zero if the transform is more efficient. Then the transform coefficients are further encoded while insignificant information is discarded, either by data values or by bitplanes. Typical examples of transform-based lossy compression techniques include ZFP [11] and MGARD [12]. The output of transform-based lossy compression is essentially the encoded transform coefficients, whose size depends on the encoding efficiency as well as the amount of information (e.g., number of bit planes) to be stored according to error bounds. Take ZFP for an example, the number of bit planes to encode depends on the user prescribed error bounds and the encoding efficiency is reflected by the number of bits to encode a bit-plane, whose distribution can be characterized by Laplacian distribution. The detailed discussion is included in Section 4.3. On the other hand, the number of bits to encode a bit plane in MGARD depends on the dimensionality of multi-level coefficients as well as the performance of ZSTD that compresses the bit planes after decomposition, and the number of bit planes to be encoded is determined (32 bits for single-precision floating-point data) as MGARD does not truncate bit planes during compression. Due to the limited scope, the implementation of zPerf for MGARD is not detailed in this work.

In what follows, we introduce the general procedures of zPerf for prediction-based and transform-based lossy compression, as shown in Fig. 2.

Stage 1: We feed the highly condensed data features into our model to capture the data characteristics. Generally speaking, the performance of lossy compression is the outcome of complex interactions between a compressor, error bound, and a dataset, and thus data features are important to the model. As a matter of fact, we comment that such condensed

TABLE 1: Notations.

(a) General metrics.

Symbol	Description
$\mathcal{U}$	Input dataset size (bytes).
$\mathcal{G}$	Compression ratio.
$\mathcal{F}$	Compression throughput (bytes/sec).
$\mathcal{N}$	Number of elements of input data.
$\mathcal{B}$	Number of data blocks.
$\mathcal{R}$	Value range of input data.
$\mathcal{E}$	Lossy error bound.

(b) SZ compression metrics.

Symbol	Description
ρ	Number of quantization levels for encoding.
η	Curve-fitting efficiency.
n	Number of Huffman tree nodes.
$\mathcal{J}$	Huffman tree structure size (bytes).
$\mathcal{K}$	Huffman encoding size (bytes).
$\mathcal{M}$	Curve-missed data encoding size (bytes).
$\mathbb{P}$	Curve-fitting and quantization time (secs).
$\mathbb{M}$	Curve-missed data encoding time (secs).
$\mathbb{C}$	Huffman tree construction time (secs).

(c) ZFP compression metrics.

Symbol	Description
ϵ	Maximum exponent value in each data block.
m	Number of bit planes to encode for each data block.
b	Number of encoding bits for each bit plane.
$\mathcal{P}$	Exponent values size (bytes).
$\mathcal{Q}$	Embedded encoding size (bytes).
$\mathbb{V}$	Exponent values calculation time (secs).
$\mathbb{X}$	Mantissas values conversion time (secs).
$\mathbb{T}$	Non-orthogonal transform time (secs).
$\mathbb{R}$	Transform coefficients reordering time (secs).
$\mathbb{E}$	Embedded encoding time (secs).

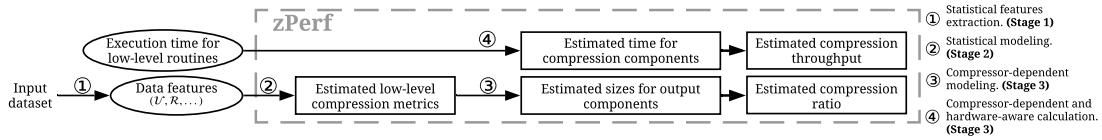
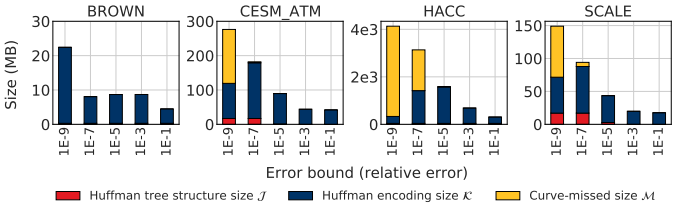


Fig. 2: The overall gray-box methodology for lossy compression modeling. The metrics circled in oval, i.e. data features and execution time for low-level routines, are input to zPerf.

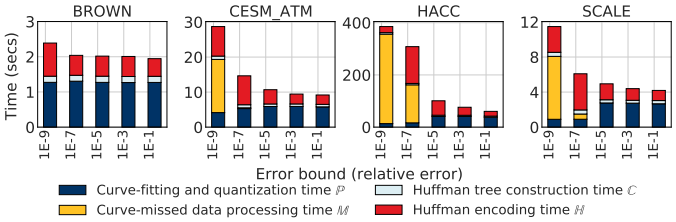
features can be made available by modern data management systems, such as ADIOS [7], which maintains an expandable set of data attributes that can be specified and computed when data is generated with the goal of simplifying the post-processing, such as query and filtering.

Stage 2: To capture the inner compression mechanism that a compressor employs, a set of low-level compression metrics need to be identified to derive the performance of lossy compression. Based upon our discussion above, the low-level metrics for prediction-based compression may include prediction efficiency and entropy encoding efficiency, while for transform-based compression, the low-level metrics may include the encoding efficiency and the number of bitplanes.

Stage 3: We then model the high-level compression metrics that are directly associated with the compression performance, including the sizes of compression output components and time of compression routines. Such modeling is enabled by the low-level compression metrics obtained in stage 2. For prediction-based lossy compression, the output components may include the output of encoded quantization codes and the representation of unpredictable data points. The compression routines may include the time to perform data prediction and entropy encoding. Similarly, for transform-based lossy compression, the output is mostly the encoded transform coefficients, while the time of compression routines mainly include the time to perform orthogonal transform and the time to encode transform coefficients.



(a) Compression output.



(b) Compression time.

Fig. 3: SZ compression performance breakdown.

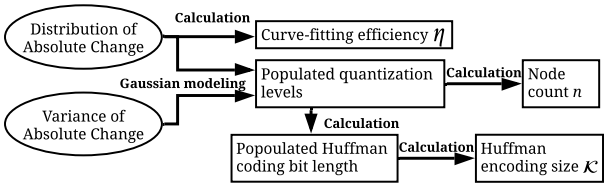


Fig. 4: Key steps in SZ modeling. The calculations for high-level metrics such as $\mathcal{J}$ and $\mathcal{M}$ are not shown.

4.2 Modeling of SZ - a case study of prediction-based lossy compression

SZ compresses input data by first predicting with Lorenzo or regression predictor, then encoding the quantization codes with Huffman encoding. As shown in Fig. 3, the size of the compressed data includes the sizes of Huffman tree structure $\mathcal{J}$, Huffman encoding $\mathcal{K}$, and curve-missed data points $\mathcal{M}$. And the compression time consists of those spent on data prediction (curve-fitting) and quantization for curve-hit data points $\mathbb{P}$, Huffman tree construction $\mathbb{C}$, Huffman encoding $\mathbb{H}$, as well as curve-missed data processing $\mathbb{M}$. These performance metrics are influenced by two low-level compression metrics: curve-fitting efficiency η and number of Huffman tree nodes n, which depends on data features.

Data features. To capture the impact of data characteristics on compression, we feed the distribution and variance of the difference between *adjacent* values into our model, as shown in Fig. 4. Overall, these features characterize the data smoothness and directly affect the outcome of SZ curve-fitting and Huffman encoding.

Low-level compression metrics. Curve-fitting efficiency η is the proportion of data points that can be represented by curve-fitting. Consider a dataset $\mathcal{D}$ with $\mathcal{N}$ elements and the 1D Lorenzo predictor for curve-fitting, the prediction for the i-th (i $\geq$ 3) data point $\mathcal{D}_i$, is simply the quantized value of the previous data point $\hat{\mathcal{D}}_{i-1}$. In particular, $\mathcal{D}_i$ is deemed to be curve-hit if the prediction error $|\mathcal{D}_i - \hat{\mathcal{D}}_{i-1}|$ falls into a prediction range γ, which can be derived from a user-prescribed error bound. In particular, γ is the product of quantization interval and the number of quantization levels ρ. For the relative error bound, the length of a quantization interval can be calculated as $\mathcal{ER}$, where $\mathcal{E}$ is the relative error bound and $\mathcal{R}$ is the data range. As such, $\gamma = \rho\mathcal{ER}$ and $\eta = \frac{1}{\mathcal{N}}\sum_{i=3}^{\mathcal{N}} I(|\mathcal{D}_i - \hat{\mathcal{D}}_{i-1}| \leq \rho\mathcal{ER})$,

where I is a unit vector. The outcome of η ultimately depends on the absolute difference between adjacent values $|\Delta\mathcal{D}_i| = |\mathcal{D}_i - \mathcal{D}_{i-1}|$, assuming $\mathcal{D}_{i-1} \approx \hat{\mathcal{D}}_{i-1}$. Specifically, we bin $\Delta\mathcal{D}$ by the prediction range associated with each error bound. For a set of k relative error bounds $\{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_k\}$ (assuming $\mathcal{E}_1 > \mathcal{E}_2 > ... > \mathcal{E}_k$), the corresponding prediction range is $\{\rho\mathcal{E}_1\mathcal{R}, \rho\mathcal{E}_2\mathcal{R}, ..., \rho\mathcal{E}_k\mathcal{R}\}$. If $|\Delta\mathcal{D}_i|$ falls into the range of $[\rho\mathcal{E}_{j+1}\mathcal{R}, \rho\mathcal{E}_j\mathcal{R})$, then $\mathcal{D}_i$ will be curve-hit at error bounds no greater than $\mathcal{E}_j$. We use $h_m^- \in \{h_1, h_2, ..., h_k\}$ to denote the number of data points where ΔD_i falls into the range of $(-\rho\mathcal{E}_m\mathcal{R}, -\rho\mathcal{E}_{m+1}\mathcal{R}]$. Correspondingly, we define $h_m^+ \in \{h_{k+1}, h_{k+2}, ..., h_{2k}\}$ to denote the number of data points where ΔD_i falls into the range of $[\rho\mathcal{E}_{m+1}\mathcal{R}, \rho\mathcal{E}_m\mathcal{R})$. Therefore, η at $\mathcal{E}_j$ can be calculated as $\eta_j = (\sum_{m=j}^{k} h_m^- + \sum_{m=k+1}^{2k+1-j} h_m^+)/\mathcal{N}$.

For 2D Lorenzo predictor, the prediction for i-th data point is expanded to use its three immediate neighbors that has been predicted, where the curve-fitting prediction error is $|\mathcal{D}_{ij} - \hat{\mathcal{D}}_{i-1,j} - \hat{\mathcal{D}}_{i,j-1} + \hat{\mathcal{D}}_{i-1,j-1}|$. Therefore, the curve-fitting efficiency can also be characterized by the distribution of the difference between adjacent values.

The number of Huffman tree nodes n is another metric that impacts the performance of SZ. For those data points that can be curve-fitted, the prediction error is quantized into at most ρ quantization levels, which is then encoded by a Huffman tree. Herein, n can be obtained during the estimation of Huffman encoding size $\mathcal{K}$ (detailed next).

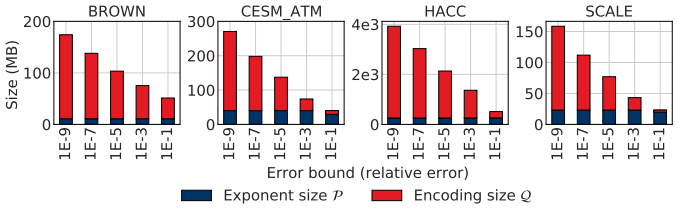
High-level compression metrics. The Huffman encoding size $\mathcal{K}$ is the sum of Huffman code length for each quantized data point, which can be calculated as $\mathcal{K} = \left\lceil \frac{1}{8} \sum_{i=1}^{\mathcal{N}} x_i \right\rceil$ where x_i is the code length (in bits) for the quantization level associated with $\mathcal{D}_i$. The code length x_i for each quantization level is determined by its frequency of occurrence – the more frequently a quantization level occurs, the shorter its code length is. As far as we know, there has been little theoretical work in the literature to model x_i. However, it was shown that the quantization level follows the Gaussian distribution [13]. Through studying the implementation of SZ, it is found that the mean of Gaussian model is located at $\frac{\rho}{2}$ and the variance is $\mathrm{Var}(\Delta\mathcal{D})/(\mathcal{E}\mathcal{R})^2$ where $\mathrm{Var}(\Delta\mathcal{D})$ denotes the variance of $\Delta\mathcal{D}$, given that a quantization level is calculated as $\Delta\mathcal{D}/(\mathcal{E}\mathcal{R})$. Hence, we can obtain n and a set of x_i, and in turn $\mathcal{K}$ by performing Huffman encoding on a small set of populated quantization levels.

The Huffman tree size $\mathcal{J}$ is the number of bytes for storing the Huffman tree. Through studying the source code of SZ, we find that four attributes are stored for each tree node: the offsets of the left and right sub-trees, the value of the node, and a flag indicating whether the current node is a leaf node. Therefore, the size of the Huffman tree can be easily calculated once n is determined.

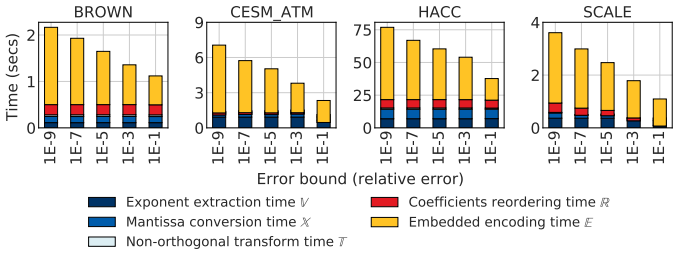
Curve-missed size $\mathcal{M}$ is the size of the curve-missed data points. These data points are first subtracted by the median of the data values and then truncated to discard the insignificant mantissa bits subject to the error bound. Based upon SZ's implementation, the number of remaining significant mantissa bits q_j at relative error bound $\mathcal{E}_j$ can be calculated as $q_j = \mathrm{Exp}(\mathcal{R}/2) - \mathrm{Exp}(\mathcal{E}_j\mathcal{R})$, where $\mathrm{Exp}(\cdot)$ denotes the exponent part of a floating-point value. In particular, $\mathrm{Exp}(\mathcal{R}/2)$ is the exponent value of the data radius and $\mathrm{Exp}(\mathcal{E}_j\mathcal{R})$ is the exponent value of a quantization level under $\mathcal{E}_j$. The rationale behind this formula is that the minimal q_j bits are required to reconstruct the normalized values of curve-missed data. Therefore, for each curve-missed data point, we need $\lfloor \frac{q_j}{8} \rfloor$ bytes to store the significant mantissa value, and additional $(q_j \bmod 8)$ bits if q_j is not multiple of 8. Given a dataset with $\mathcal{N}$ elements and curve-fitting efficiency η_j under relative error bound $\mathcal{E}_j$, we have the size of byte array $\mathcal{M}_1 = \lceil \mathcal{N}(1-\eta_j) \lfloor \frac{q_j}{8} \rfloor \rceil$ and the size of the bit array $\mathcal{M}_2 = \lceil \frac{1}{8}\mathcal{N}(1-\eta_j)(q_j \bmod 8) \rceil$. SZ further reduces the storage cost of $\mathcal{M}_1$ by performing a byte-level XOR operation between consecutive mantissa values. The number of leading zero in the result of XOR indicates the number of leading bytes between consecutive values that have the same value. As an empirical design, SZ designates 2 bits for each mantissa value to store the number of leading zero bytes. Therefore, maximally three leading zero bytes can be captured and we have the size of the leading zero byte array $\mathcal{M}_3 = \lceil \frac{1}{8} 2\mathcal{N}(1-\eta_j) \rceil = \lceil \frac{1}{4}\mathcal{N}(1-\eta_j) \rceil$. We approximate that each mantissa value has an average of 1.5 leading zero bytes for any scientific datasets. Therefore, $\mathcal{M}_1 = \lceil \mathcal{N}(1-\eta_j)(\lfloor \frac{q_j}{8} \rfloor - 1.5) \rceil$ and $\mathcal{M} = \mathcal{M}_1 + \mathcal{M}_2 + \mathcal{M}_3$.

Meanwhile, the compression time is system-dependent and the idea of modeling the compression time is to focus on analyzing the complexity of compression routines and use the measurement of low-level routines to extrapolate the compression time on a particular system. The curve-fitting and quantization time $\mathbb{P}$ is the time to perform curve-fitting and quantization on curve-hit points. In particular, SZ first maps them into ρ quantization levels and then adjusts the data prediction due to the quantization. Since SZ performs the same operation for each curve-hit data point, the time complexity of $\mathbb{P}$ is $\mathcal{O}(\mathcal{N}\eta)$. The Huffman tree construction time $\mathbb{C}$ is the time to construct the Huffman tree from quantization levels. It involves calculating the frequency of each quantization level $\mathbb{C}_1$, node insertion $\mathbb{C}_2$ and Huffman code building $\mathbb{C}_3$. The time for frequency calculation is essentially linear to the number of data points. Therefore the time complexity of $\mathbb{C}_1$ is $\mathcal{O}(\mathcal{N})$. The node insertion involves the following two steps. First, each quantization level is formed as a tree node and inserted into a tree, with each tree having one node. Then two trees with the lowest frequencies will be merged, and this step will be repeated until all trees are merged into one tree. Thus, the complexity of node insertion is $\mathcal{O}(n)$. Huffman code building involves traversing the tree and assign 0s or 1s to the tree leaves. It requires a tree traversal and the time complexity is clearly $\mathcal{O}(n)$. The Huffman encoding time $\mathbb{H}$ is the time to encode the quantized data. Essentially SZ goes over the entire dataset, looks up the Huffman coding of the quantization data, and writes the code to a buffer. The time complexity is $\mathcal{O}(\mathcal{N}\eta)$. Meanwhile, the curve-missed processing time $\mathbb{M}$ is the time to perform the aforementioned operations for curve-missed data points. Clearly, the time complexity of $\mathbb{M}$ is $\mathcal{O}(\mathcal{N}(1-\eta))$ as each curve-missed data point is processed with the same operations.

To fully model the compression time for a given system, we next obtain the timings of the hardware-dependent low-level compression routines. They include the timings of the processing of a single curve-hit point r_1, the processing of a single curve-missed point r_2, the calculation of quantization



(a) Compression output.



(b) Compression time.

Fig. 5: ZFP compression performance breakdown.

level for a data point r_3, the node insertion to Huffman tree r_4, the traversal of a tree r_5, and the retrieval and storage of Huffman coding for a data point r_6.

SZ compression performance. Given the performance metrics we discussed above and the measurements of low-level routines, we have $\mathcal{G}_{sz} = \frac{\mathcal{U}}{\mathcal{J}+\mathcal{K}+\mathcal{M}}$ and $\mathcal{F}_{sz} = \frac{\mathcal{U}}{\mathcal{N}\eta r_1+\mathcal{N}(1-\eta)r_2+\mathcal{N}r_3+n(r_4+r_5)+\mathcal{N}r_6}$ for compression ratio and compression throughput, respectively. We note that for the convenience of implementation, curve-missed data points are marked as a special quantization level 0 in SZ, and therefore r_3 and r_6 are scaled by a factor of $\mathcal{N}$, instead of $\mathcal{N}(1-\eta)$.

4.3 Modeling of ZFP - a case study of transform-based lossy compression

ZFP compresses input data by first performing block-based orthogonal transform to remove redundancies, then encoding the transform coefficients one bit plane at a time to control the output bit rate. As shown in Fig. 5a, the output size of ZFP includes the sizes of exponents values $\mathcal{P}$ and embedded encoding $\mathcal{Q}$. Meanwhile, the compression time consists of exponent extraction time $\mathbb{V}$, mantissa conversion time $\mathbb{X}$, non-orthogonal transform time $\mathbb{T}$, transform coefficients reordering time $\mathbb{R}$, and embedded encoding time $\mathbb{E}$, as demonstrated in Fig. 5b. Fundamentally, there are two low-level compression metrics that affect the high-level metrics: the number of bit planes m to encode for each block and the number of encoding bits b for each bit plane. Fig. 6 shows the key steps in ZFP modeling, which are detailed next.

Data features. The compression performance of ZFP is highly affected by the properties of input data. In particular, m is impacted by the exponents of input data for each block, while b depends on the non-orthogonal transform coefficients. Therefore, to capture the interaction between data and the compressor, we feed the distribution of block-wise mean into zPerf, based upon which the distributions of exponent and mantissa mean can be derived.

Low-level compression metrics. Let m_i denote the number of bit planes to encode for the i-th block under a user-prescribed error bound $\mathcal{E}$, which can be calculated as $m_i = \epsilon_i - \log_2 \mathcal{E} + 2(dim+1)$ [31], where ϵ_i is the maximum base-2 exponent of the block and dim is the number of

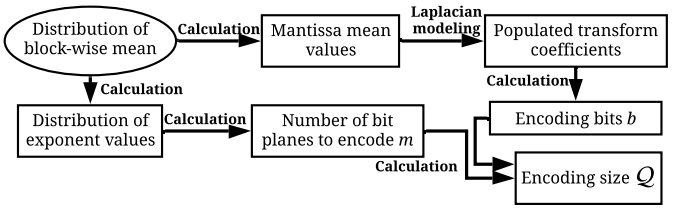


Fig. 6: Key steps in ZFP modeling.

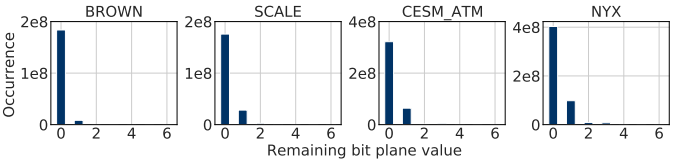


Fig. 7: The histogram of remaining values of bit planes after encoding the significant bits under the relative error of 1E-6. It is clear that the remaining values of the majority of bit planes are zero, which requires one bit to encode.

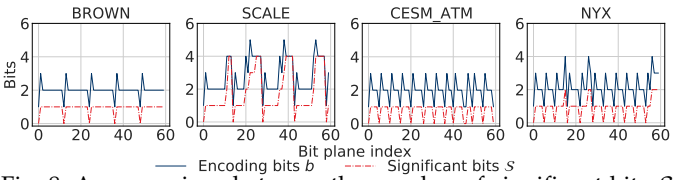


Fig. 8: A comparison between the number of significant bits $\mathcal{S}$ and encoding bits b under the relative error of 1E-6.

dimensions. We note that the additional $2(dim+1)$ bit planes are needed due to the range expansion incurred by the inverse transform during decompression. Given the distribution of block-wise mean of input data, we can populate a small set of mean values, and ϵ_i can be estimated by taking the logarithm over the populated data, based upon the fact that data values within a block are often smooth and therefore ϵ_i is close to the exponent of the mean. As such, m_i can be obtained.

The value of b is another vital metric to the ZFP compression. Let b_{ij} denote the number of bits required to encode the j-th bit plane for the i-th block. The embedded encoding of each bit plane in a block includes the following two steps: First, $\mathcal{S}_{ij}$ *significant* bits are encoded verbatim, where $\mathcal{S}_{ij}$ is the smallest number that allows $4^{dim} - \mathcal{S}_{ij}$ highest bits in all previous $j-1$ bit planes to be all zeros. Second, the remaining $4^{dim} - \mathcal{S}_{ij}$ bits are encoded using a variable-length representation. Due to the reordering of transform coefficients, we find that the remaining $4^{dim} - \mathcal{S}_{ij}$ bits of each bit plane are largely zero, as shown in Fig. 7. Note that since an extra zero bit is saved for a bit plane whose remaining bits are all zeros, b_{ij} can be estimated as $\mathcal{S}_{ij}+1$, as demonstrated in Fig. 8. Hence in this work, we use $\mathcal{S}_{ij}$ to estimate b_{ij}. Further, it is clear that $\mathcal{S}_{ij}$ depends heavily on the transform coefficients, which are the product of the non-orthogonal transform in ZFP. In order to capture the bit plane values and in turn $\mathcal{S}_{ij}$, we next discuss the distribution of transform coefficients.

It was shown in previous work [32] that the non-orthogonal transform coefficients can be modeled by Laplacian distribution with the mean of zero and the scaling factor of λ, where λ can be estimated as the block-wise mean of coefficients via the maximum likelihood estimation [33]. Consider a 1D block of input data $[x, y, z, w]$ with the corresponding mantissa $[\dot{x}, \dot{y}, \dot{z}, \dot{w}]$ and transform coefficients $[x', y', z', w']$. For a block of data that is typically smooth

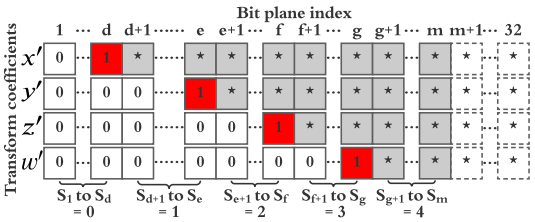


Fig. 9: A schematic of 1D embedded encoding in ZFP. The symbol $*$ indicates that the associated bit can be either 0 or 1. A dotted box indicates that the bit is discarded according to the error tolerance. The most significant 1 bit in the corresponding coefficient is highlighted in red. The value of $\mathcal{S}_j$ for a bit plane is determined by the location of the red box.

(due to the continuity in the physical quantities captured from a scientific simulation), the high-frequency components of the transform coefficients, namely y', z' and w', will be close to zero [34]. Therefore, the mean of transform coefficients can be approximated as $\frac{x'}{4}$, where x', known as the DC (zero-frequency) component, can be further calculated as the mean of input mantissa values $x' = \frac{1}{4}(\dot{x} + \dot{y} + \dot{z} + \dot{w})$ [35]. It is clear that the estimation of transform coefficients distribution ultimately comes down to the mean of mantissa of input data, which can be obtained from the populated values during the estimation of m_i. As such, the modeling of ZFP transform coefficients is complete.

Next, based upon the estimated Laplacian distribution of transform coefficients, we populate a set of coefficients and then compute the $\mathcal{S}_{ij}$ values. In Fig. 9, we provide an example of encoding a 1D floating-point block of transform coefficients. In this example, given the populated transform coefficients $[x', y', z', w']$, the indices of significant bits, marked by red boxes, can be calculated as $[d, e, f, g] = [32 - \lfloor log_2 x' \rfloor, 32 - \lfloor log_2 y' \rfloor, 32 - \lfloor log_2 z' \rfloor, 32 - \lfloor log_2 w' \rfloor]$. Then $\mathcal{S}_{ij}$ for each bit plane can be easily calculated. As such, b_{ij} can be estimated.

High-level compression metrics. Encoding size $\mathcal{Q}$ is the total size of the embedded encoding, which is the sum of b across all bit planes and all data blocks, i.e., $\mathcal{Q} = \left\lceil \frac{1}{8} \sum_{i=1}^{\mathcal{B}} \sum_{j=1}^{m_i} b_{ij} \right\rceil$ where $\mathcal{B}$ is the number of data blocks. Exponent size $\mathcal{P}$ is the byte size to store ϵ for all data blocks. According to the IEEE-754 format, ϵ takes 11 bits for double precision floating point data and therefore $\mathcal{P}$ can be calculated as $\mathcal{P} = \lceil \frac{11}{8} \mathcal{B} \rceil$. Similarly, ϵ takes 8 bits for single precision floating point data and $\mathcal{P} = \lceil \mathcal{B} \rceil$.

On the other hand, $\mathbb{V}$, $\mathbb{X}$, $\mathbb{T}$, and $\mathbb{R}$ are the total time to extract ϵ, convert input floating-point values into mantissa values, perform non-orthogonal transform, and reorder the transform coefficients, respectively for all data blocks. Clearly the complexity of $\mathbb{V}$, $\mathbb{X}$, $\mathbb{T}$, and $\mathbb{R}$ is $\mathcal{O}(\mathcal{B})$. $\mathbb{E}$ is the total time to perform embedded encoding on all transform coefficients. It is affected by both the number of bit planes across all data blocks and the number of encoding bits for each bit plane, given that the more bits a bit plane has, the longer the encoding time is. Therefore, we need to measure the encoding time for bit planes with different numbers of encoding bits. To determine each of these metrics, we directly measure the timing of the following low-level compression routines: the calculation of the common exponent on a single block r_1, the calculation of mantissa values on a single block r_2, the non-orthogonal transform on a single block of mantissa values r_3, the reordering of a single block of transform coefficients r_4, and the embedded encoding of a bit plane r_5. For r_5, we calculate the average time to store a bit plane with encoding bits of 1, 2, 3 and 4, respectively.

ZFP performance. Given the compression metrics and the measurements of the low-level compression routines, we have $\mathcal{G}_{zfp} = \frac{\mathcal{U}}{\mathcal{P}+\mathcal{Q}}$ for the compression ratio, and $\mathcal{F}_{zfp} = \frac{\mathcal{U}}{\mathcal{B}(r_1+r_2+r_3+r_4)+\mathbb{E}}$ for the compression throughput.

TABLE 2: Dataset description.

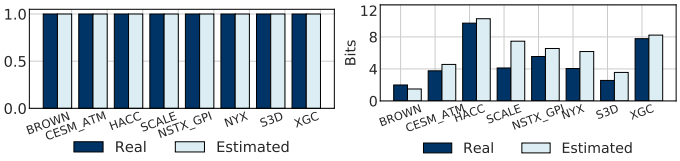
Dataset	Size	Description
1D experiments		
BROWN	257 MB	Synthetic, generated to specified regularity
CESM_ATM	643 MB	Climate simulation
HACC	4 GB	Cosmology: particle simulation
SCALE	539 MB	Climate simulation
2D experiments		
NSTX_GPI	361 MB	NSTX Gas Puff Image (GPI) data
NYX	513 MB	Cosmology: Adaptive mesh hydrodynamics + N-body cosmological simulation
S3D	11 GB	Combustion simulation
XGC	324 MB	Fusion simulation

5 Evaluation

In this section, we evaluate zPerf across eight scientific datasets (described in TABLE 2) from the Scientific Dataset Reduction Benchmark [36]. Specifically, we show the results of four datasets with 1D compression experiments and four with 2D compression experiments. We conduct the compression experiments using SZ (Version 2.1.7) and ZFP (Version 0.5.5) on two leading HPC systems, Cori [37] at National Energy Research Scientific Computing Center, and Summit [38] at Oak Ridge National Laboratory, to test the compression throughput estimation accuracy of our model. We test the modeling performance of zPerf under the relative error bounds [1E-6, 1E-5, 1E-4, 1E-3] as moderate error bounds generally play a more important role in production as compared to extreme ones. Specifically, since the ZFP APIs do not support the relative error bound, we set the absolute error bound of ZFP to the product of the data range and relative error bound, so that different error magnitudes can be covered in our experiments. In what follows, we show the modeling results of the low-level and high-level compression metrics. We then compare zPerf against the sampling approach in terms of the estimation accuracy.

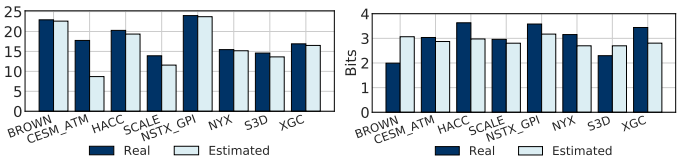
5.1 Low-level compression metrics

For SZ, the estimated curve-fitting efficiency η and the average Huffman coding bit length are shown in Fig. 10. For η, it can be estimated accurately mainly because adjacent data points are observed to be smooth, i.e., $\hat{\mathcal{D}}_{i-1} \approx \mathcal{D}_{i-1}$. Therefore, the histogram of $\Delta\mathcal{D}$ versus error tolerance can well capture the number of curve-fit points. On the other hand, the average Huffman coding bit length, which directly impacts the outcome of $\mathcal{K}$, is obtained through performing Huffman encoding on the quantization levels, which are populated based on the Gaussian distribution. It is noted that the average Huffman coding bit length is accurate for most cases, due to the similarity between the distributions of original and populated quantization levels. In particular, we observe that datasets like *NYX* and *SCALE* demonstrates lower data smoothness, such that the number of quantization levels needed for Huffman encoding is hard to capture.



(a) Curve-fitting efficiency η. (b) Huffman coding bit length.

Fig. 10: SZ low-level metrics estimation. Results are averaged across relative error bounds.



(a) Num of bitplanes for a block. (b) Num of bits for a bitplane.

Fig. 11: ZFP low-level metrics estimation. Results are averaged across relative error bounds.

TABLE 3: Execution time of low-level routines. All measurements were conducted using the *SCALE* dataset under the relative error bound of 1E-6. Each execution time is averaged across 10 runs.

(a) SZ on Cori

Routine	Avg. time (ns)
r_1	32.63
r_2	384.91
r_3	4.46
r_4	4.91
r_5	5.76
r_6	4.58

(b) SZ on Summit

Routine	Avg. time (ns)
r_1	62.63
r_2	452.91
r_3	7.46
r_4	5.71
r_5	7.36
r_6	5.32

(c) ZFP on Cori

Routine	Avg. time (ns)
r_1	11.8
r_2	8.19
r_3	2.39
r_4	17.19
r_5	2.72

(d) ZFP on Summit

Routine	Avg. time (ns)
r_1	20.8
r_2	16.19
r_3	4.39
r_4	26.19
r_5	3.72

Therefore, the estimated Huffman coding bit lengths based upon the populated quantization levels deviate from original values.

For ZFP, we demonstrate the modeling results of the number of bit planes m to encode for each data block and the number of encoding bits b for each bit plane in Fig. 11. The value of m can be well captured for most datasets due to the fact that ϵ is observed to be close to the exponent of the input data mean. For b, the estimation error is caused by the inadequacy of our model to capture the difference between the DC component (x') and high-frequency components (y', z' and w') of transform coefficients. Therefore, when we populate transform coefficients based on the estimated Laplacian distribution, the difference between x' and y' is underestimated (assuming $x' \gg y' > z' > w'$), causing the number of bit planes between the d-th and e-th bit plane to be underestimated, as shown in Fig. 9. On the other hand, our model does not handle the cases of $b > 4$. Such cases are counted towards $b = 4$ automatically, which leads to the overestimation for the number of bit planes with $b = 4$. However, the bit planes with $b > 4$ make up a small portion of the total number of bit planes. Therefore, the error is observed to be insignificant.

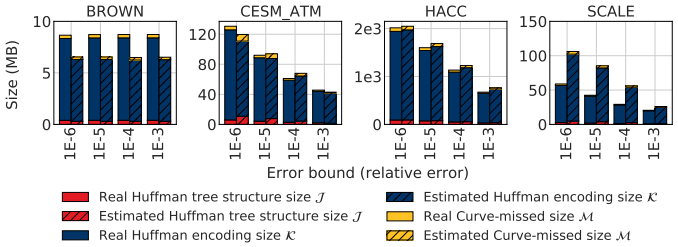
5.2 High-level compression metrics

We further evaluate the estimation of high-level metrics for SZ and ZFP. Specifically, for SZ, we show the estimated results of $\mathcal{J}$, $\mathcal{K}$ and $\mathcal{M}$ under in Fig. 12. Note that the overall height of each bar indicates the final compressed size. The deviation of n as a result of the discrepancy between the original and the quantization levels produced by the Gaussian distribution is further propagated to $\mathcal{J}$ due to the linear relationship between n and $\mathcal{J}$. Overall, the accuracy of $\mathcal{J}$ can be well maintained for the compressed size, while the average estimation error of $\mathcal{K}$ is less than 45%, except for *NYX* and *SCALE*. We note that the estimation error of $\mathcal{K}$ mainly comes from the estimation error of Huffman coding bit length, which is discussed in Section 5.1.

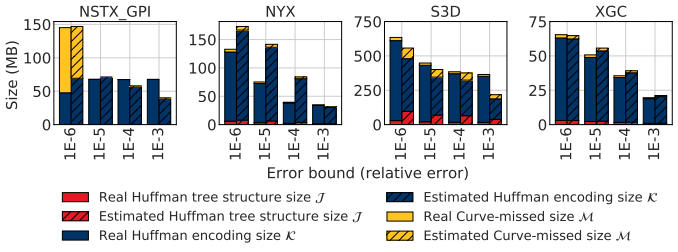
As aforementioned, to estimate the system-dependent compression time, we need to measure the timings of the low-level compression routines r_i. TABLE 3a and TABLE 3b show the execution time of low-level routines of SZ on Cori and Summit respectively, which were measured using the *SCALE* dataset under the relative error bound of 1E-6. Each measured execution time is averaged across 10 runs. In particular, r_1 is measured by the total curve-fitting and quantization time divided by the number of curve-hit data points, r_2 is measured by the curve-missed data processing time divided by the number of curve-missed data points, and r_3 is measured by the time to retrieve quantization levels divided by the total number of data points. The estimation for compression time components for SZ is shown in Fig. 13, where the compression time for Cori is shown in Fig. 13a, Fig. 13b and the compression time for Summit is shown in Fig. 13c, Fig. 13d. Since $\mathbb{P}$ is linear to η, its estimation is fairly accurate. The Huffman tree construction time $\mathbb{C}$ can be calculated as $\mathcal{N}r_3 + n(r_4 + r_5)$. Overall, since $\mathbb{C}$ is dominated by $\mathcal{N}r_3$ ($\mathcal{N} \gg n$) and r_3 is deterministic, its accuracy is good and the estimation error mostly comes from n. For the Huffman encoding time $\mathbb{H}$, it can be accurately estimated since the amount of time to encode one quantization level is fairly constant for a particular system.

For ZFP, we demonstrate the estimation results of $\mathcal{P}$ and $\mathcal{Q}$ in Fig. 14a and Fig. 14b respectively for 1D and 2D compression. In particular, $\mathcal{P}$ can be well estimated since it is linearly related to the number of blocks $\mathcal{B}$. For $\mathcal{Q}$, it is the sum of b of all bit planes across all data blocks. The estimation error of $\mathcal{Q}$ essentially comes from b, which has been discussed in Section 5.1. TABLE 3c and TABLE 3d list the measured execution time of low-level routines in ZFP on Cori and Summit respectively. Specifically for r_5, we manually set the bit planes values so that the corresponding encoding bits for each bit plane is 1, 2, 3 and 4. For bit planes with different number of encoding bits, we calculate the average encoding time over 1,000,000 blocks with 64 bit planes, and the average time to store a bit plane is 2.72 ns on Cori and 3.72 ns on Summit. We show the estimation results for $\mathbb{V}$, $\mathbb{X}$, $\mathbb{T}$, $\mathbb{R}$, and $\mathbb{E}$ in Fig. 15. As we mentioned earlier, $\mathbb{V}$, $\mathbb{X}$, $\mathbb{T}$ and $\mathbb{R}$ are linear to $\mathcal{B}$. The estimation error for these timings mainly comes from the approximation of using low-level routines time measured on *SCALE*, while the estimation error of $\mathbb{E}$ mainly comes from the estimation of encoding bits for each bit plane.

Overall, zPerf is effective in capturing the trend of com-



(a) Compression output components (1D compression).



(b) Compression output components (2D compression).

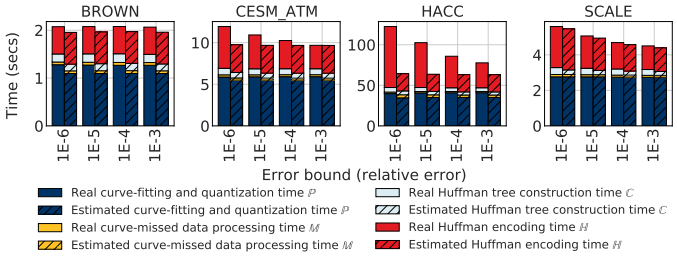
Fig. 12: SZ compression size estimation.

pression performance, despite the statistical approximation (e.g., the Gaussian modeling for SZ and Laplace modeling for ZFP) in estimating the compression metrics. Generally, zPerf achieves better estimation results for ZFP than for SZ. The reason is that the low-level compression metrics of ZFP, m and b, can be well modeled. On the one hand, m is directly calculable based on $\mathcal{E}$ and ϵ. On the other hand, b does not demonstrate drastic changes across all datasets used in our work. Based on our observation, the values of b typically range from 1 to 6.

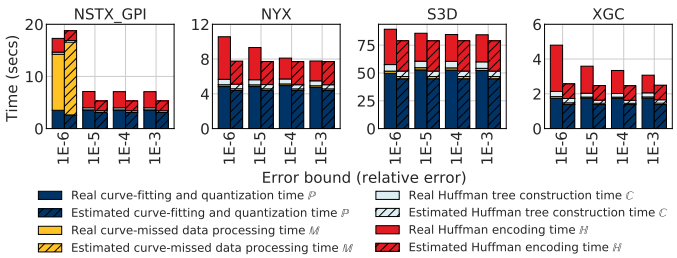
5.3 zPerf vs. sampling based approach

We next compare zPerf with the sampling approach [13] regarding the estimation accuracy. We assess the estimation accuracy using the *mean relative error* (MRE), which is defined as the average ratio between the absolute estimation error and the original compression performance. Overall, the estimation accuracy of the sampling approach is impacted by the sampling ratio, while the size of populated metrics affects the performance of zPerf as well. As a result, the sampling ratio and the population ratio (defined as the ratio between the size of the populated values and the size of the original data) are key parameters in our evaluation. On the other hand, as the prior work [13] pointed out that for the estimation of compression ratios, the sampling approach offers a biased estimation for compressors without bounded locality, such as SZ, and an unbiased estimation for compressors with bounded locality, such as ZFP. Therefore, we anticipate that the sampling approach works well for ZFP, but not SZ.

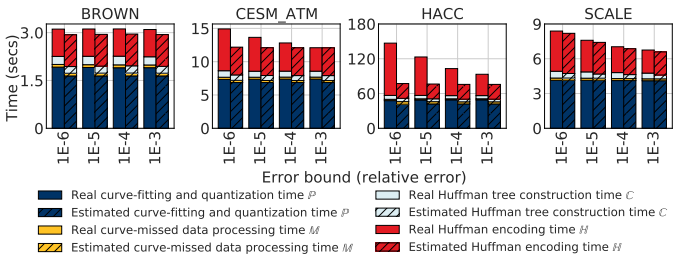
In this section, we vary both the population and sampling ratios from 1E-1 to 1E-7 and compare the MRE of zPerf and sampling approach. In Fig. 16 and Fig. 17, we measure the MRE of estimation for SZ and ZFP, respectively. In each figure, we display the MRE for estimating compression ratio (1D and 2D scenarios) as well as compression throughput (on Cori and Summit). Due to the limited space, we only display compression throughput estimation for the 1D scenario. It is observed that the accuracy of the sampling approach generally degrades (MRE increasing) when the sampling ratio decreases. Note that due to the



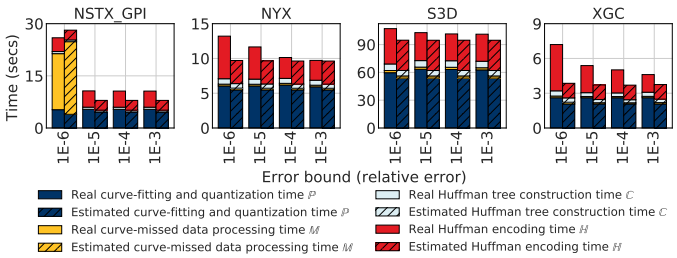
(a) Compression time components on Cori (1D Compression).



(b) Compression time components on Cori (2D Compression).



(c) Compression time components on Summit (1D Compression).

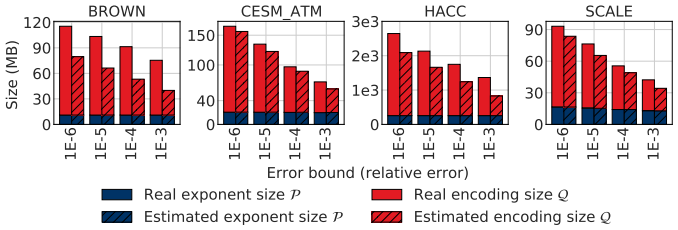


(d) Compression time components on Summit (2D Compression).

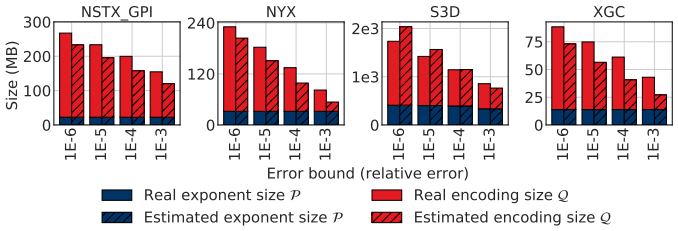
Fig. 13: SZ compression time estimation.

bounded locality of ZFP, the sampling approach yields a low error for compression ratio (Fig. 17a and Fig. 17b). For SZ, zPerf generally outperforms the sampling approach in the estimation of compression ratio. It is because the estimation of Huffman tree structure deteriorates rapidly when the sampling ratio decreases, while it can be better maintained by zPerf through the Laplacian modeling. On the other hand, the MRE of the compression throughput using zPerf is observed to be insensitive to the population ratio. This suggests that if the compression throughput is a metric of interest (e.g., for online compression), zPerf provides a good estimation even with a small set of populated values. We find that this is because the two largest components of compression time, $\mathbb{P}$ and $\mathbb{H}$, have the complexity of $\mathcal{O}(\mathcal{N})$, which are not directly affected by the population ratio.

For ZFP, the MRE of the compression throughput using the sampling approach increases with the decreasing of the sampling ratio. While the MRE of the sampling approach is generally lower than the MRE of zPerf, we find that they achieve similar performance at low sampling ratios.



(a) Compression output components (1D compression).



(b) Compression output components (2D compression).

Fig. 14: ZFP compression size estimation.

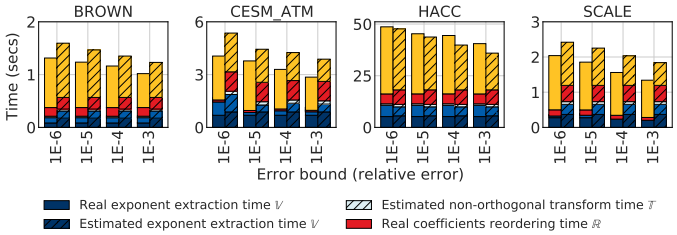
Generally speaking, both models are anticipated to work well at high sampling and population ratios (e.g., 1E-1 and 1E-2). However, the MRE of the sampling approach deteriorates rapidly as the sampling ratio becomes small– a key disadvantage when estimating the performance of extreme-scale datasets that require a small sampling ratio.

We further compare the running time overhead of both zPerf and sampling approach, as shown in Fig. 18. It can be shown that for SZ (Fig. 18a), zPerf yields a lower overhead than sampling approach after sampling ratio drops below 1E-4, which further demonstrates the advantage of zPerf when estimating compression performance at low sampling ratios. For ZFP, zPerf requires longer running time as compared to sampling approach. However, it is still beneficial given that zPerf can provide better estimation at low population ratios.

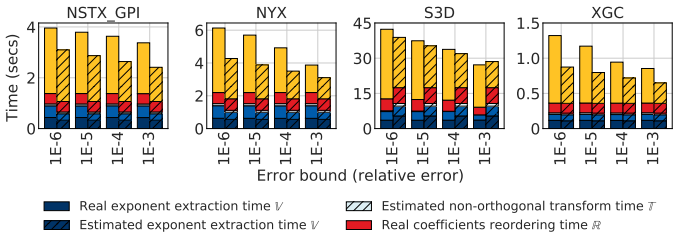
5.4 zPerf vs. state-of-the-art

We next quantitatively compare zPerf with state-of-the-art in compression performance modeling.

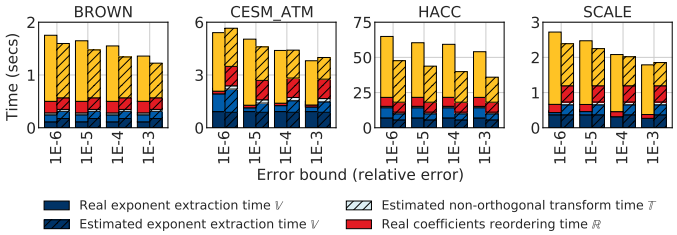
Tao *et al.* [22] employ a rate-distortion estimation method for bit-rate estimation. As compared to zPerf, this work neither explored the design space of lossy compression nor attempted the modeling the compression throughput. Rather, it uses a sampling-based approach for performance estimation by compressing the sampled data directly. As such, it is anticipated that it can outperform zPerf, albeit unable to predict the performance for a potentially new design for a compressor. For the compression ratio modeling, it measured the performance under high sampling ratios (no lower than 1%) and did not attempt lower sampling ratios (e.g., 0.1% and 0.01%) that are important for the modeling of compression for large data in a cost-effective way. By allowing for low sampling ratios, zPerf incurs substantially less memory footprint, e.g., 100X less memory at a sampling ratio of 0.01% as compared to that of 1%, with an insignificant degradation of modeling accuracy (e.g., by 10% to 15%). For *CESM_ATM* at a sampling ratio of 1%, this previous work achieves an average estimation error of 7.5% for SZ and 5.7% for ZFP, while zPerf achieves 19.1% for SZ and 20.68% for ZFP (again, with the added capability of exploring new algorithms in a compressors).



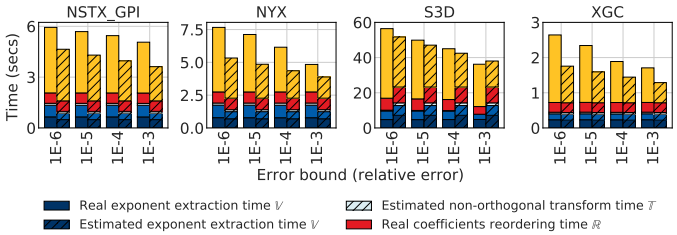
(a) Compression time components on Cori (1D Compression).



(b) Compression time components on Cori (2D Compression).



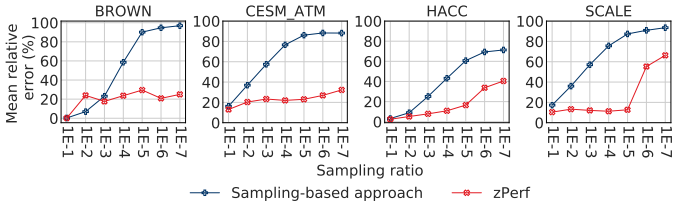
(c) Compression time components on Summit (1D Compression).



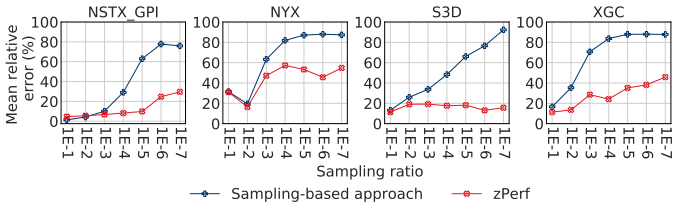
(d) Compression time components on Summit (2D Compression).

Fig. 15: ZFP compression time estimation.

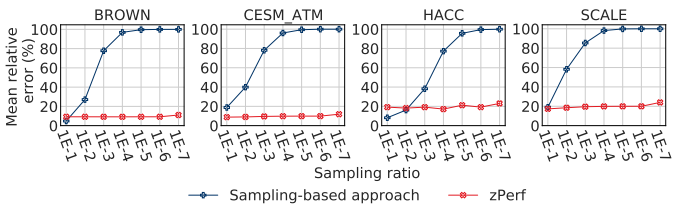
Zhao *et al.* [23] achieved an average estimation error of 5% in most cases when the sampling ratio is 8%. In contrast, zPerf achieves an average error of 10.54% under a sampling ratio of 10% for SZ. However, they did not present quantitative results across datasets and the work focused only on SZ. Jin *et al.* [24] adopted a modularized approach for the compression ratio and quality estimation for prediction-based lossy compression. While the modular estimation is similar to zPerf, it only focuses on prediction-based lossy compression and does not study other types of techniques. As reported in the paper, their approach achieves an average estimation error of 9.66%, 8.08%, 3.83%, and 6.46% for *CESM_ATM*, *Nyx*, *HACC*, and *Brown*, respectively, under a sampling ratio of 1%. However, the error configuration was not disclosed and we could not further compare it to zPerf. Meanwhile, the average estimation error of zPerf under a sampling ratio of 1% for corresponding datasets are



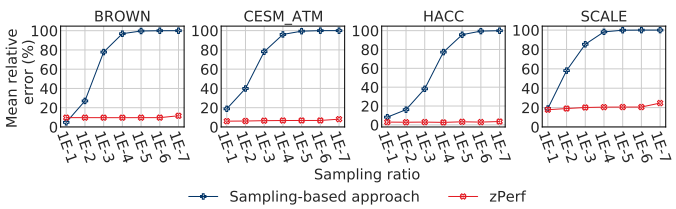
(a) Compression ratio estimation (1D Compression).



(b) Compression ratio estimation (2D Compression).



(c) Compression throughput estimation on Cori (1D compression).



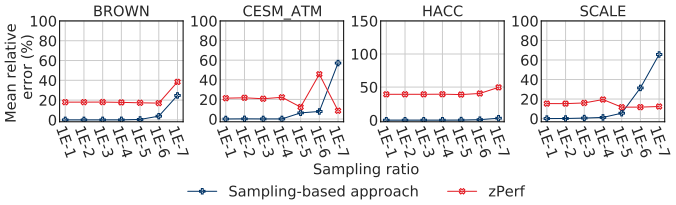
(d) Compression throughput estimation on Summit (1D compression).

Fig. 16: zPerf estimation error compared to the sampling approach for SZ under sampling ratios from 1E-1 to 1E-7. The estimation error is averaged across error bounds.

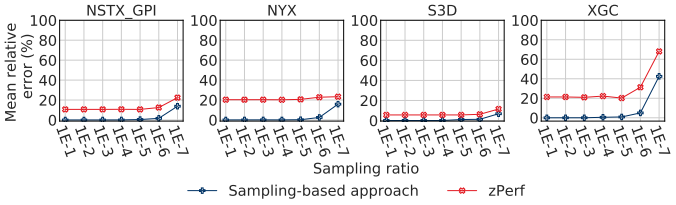
19.1%, 18.2%, 18.8%, and 9.8%, respectively. Jin *et al.* [25] also achieved the modeling of compression ratio for *Nyx*. Their approach is based on the empirical analysis that the bit-rate to error bound ratio for a compression method on a dataset is similar across error bounds. The model only targets the case where the compression ratio is larger than 16, given the goal is to improve the quality of visualization after compression. There is no quantitative evaluation presented for the modeling accuracy.

6 Performance Extrapolation

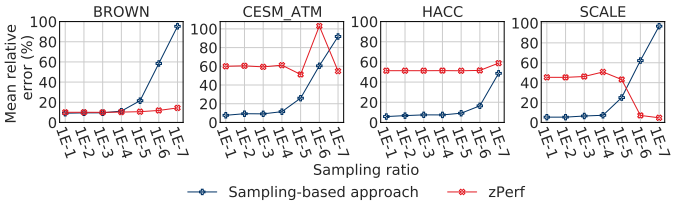
Identifying new opportunities for lossy compression has become increasingly difficult, given the large algorithmic and software-hardware co-design space to explore. For developers of lossy compression, a question often arises: *would a new component in the compressor improve the overall performance for some application scenarios?* To answer this question, developers must implement a new version of the compressor first, followed by labor-intensive testing and maintenance. In this section, we demonstrate the application of using zPerf to explore the design space of lossy compression. We discuss three case studies where we estimate the impact of alternative entropy encoding schemes on SZ and ZFP, as well as alternative transform scheme on ZFP. Specifically, for entropy encoding study, we replace the Huffman encoding



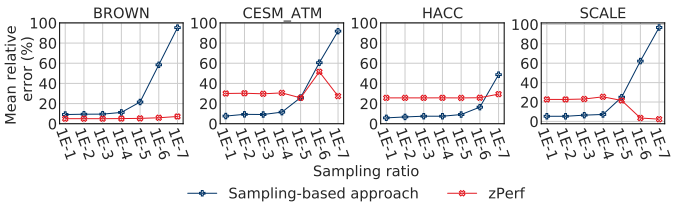
(a) Compression ratio estimation (1D Compression).



(b) Compression ratio estimation (2D Compression).

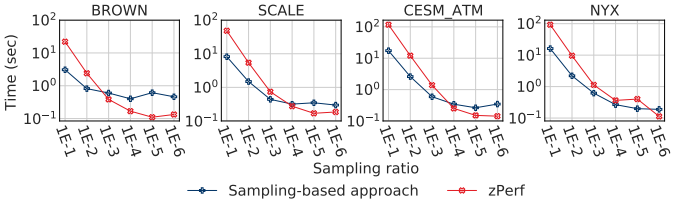


(c) Compression throughput estimation on Cori (1D compression).

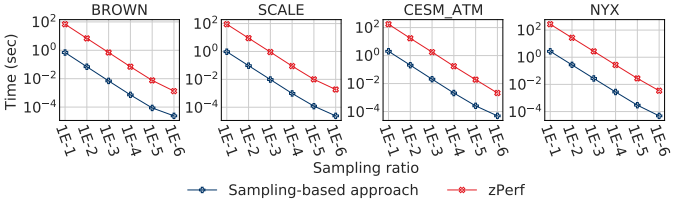


(d) Compression throughput estimation on Summit (1D compression).

Fig. 17: zPerf estimation error compared to the sampling approach for ZFP under sampling ratios from 1E-1 to 1E-7. The estimation error is averaged across error bounds.



(a) SZ compression performance estimation overhead.



(b) ZFP compression performance estimation overhead.

Fig. 18: Running time overhead of zPerf compared to the sampling approach under sampling ratios from 1E-1 to 1E-6.

in SZ with the ZFP lossless compression and the embedded encoding in ZFP with Huffman encoding. For transform scheme study, we replace the customized non-orthogonal transform in ZFP with discrete cosine transform (DCT).

Case study 1: exploring ZFP lossless encoding for SZ. SZ currently employs Huffman encoding to compress the quantization levels based upon the observation that the

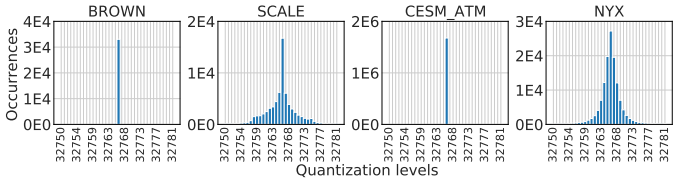


Fig. 19: Distribution of SZ quantization levels.

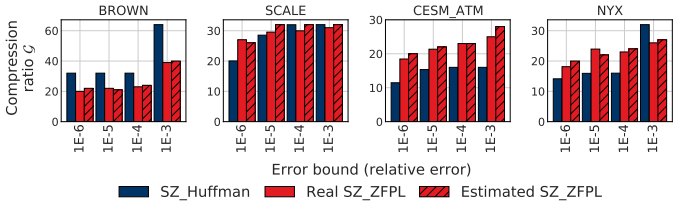


Fig. 20: Compression ratio of *SZ_Huffman*, *SZ_ZFPL*.

distribution of quantization levels is Gaussian [13]. As such, they can be efficiently compressed with Huffman encoding. In this case study, we consider the possibility of encoding the SZ quantization levels with the ZFP lossless mode, which adopts a modified decorrelating transform to map the input floating point data to transformation coefficients. Then during the variable-length encoding, bit-planes are no longer truncated to achieve lossless encoding. The intuition of adopting the ZFP lossless encoding for compressing quantization levels is that the variable-length encoding scheme by ZFP leverages the similarity among the transformation coefficients. That is, the smoother the input data is, the more efficient the decorrelating transform will be; thus, a lower bit-rate can be achieved in encoding. Given that the SZ quantization levels are generally highly similar, as shown in Fig. 19, it is reasonable to use ZFP lossless mode as the backend for compression.

As discussed in Section 4.3, the output of ZFP compression consists the exponent value size $\mathcal{P}$ and encoding size $\mathcal{Q}$. While $\mathcal{P}$ mainly depends on the number of data blocks, $\mathcal{Q}$ can be calculated by the number of data blocks $\mathcal{B}$, as well as the bits to encode each bit-plane b_{ij}: $\mathcal{Q} = \left\lceil \frac{1}{8} \sum_{i=1}^{\mathcal{B}} \sum_{j=1}^{m_i} b_{ij} \right\rceil$. Specifically in the lossless compression mode, all the bit planes are encoded. Therefore, the output of ZFP lossless compression depends solely on the number of bits to encode a bit-plane. As previously discussed, the modeling of b_{ij} comes down to capturing the significant bits in transform coefficients, which depends on the input data. Therefore, following the approach we developed in Section 4.3, we model the transform coefficients with Laplacian distribution and calculate the number of significant bits for each block. In Fig. 20, we show the measured and estimated compression ratio of SZ using the ZFP lossless encoding, denoted as *SZ_ZFPL*, versus *SZ_Huffman*. It is worth noting that *SZ_ZFPL* achieves much lower compression ratios than *SZ_Huffman* on *Brown*. The reason is that while the ZFP lossless mode can encode the transform coefficients of quantization levels efficiently, the *Brown* dataset is of double-precision, thus ZFP lossless mode needs to store many additional bit-planes. For other datasets, it is shown that *SZ_ZFPL* typically outperforms *SZ_Huffman* when the error bound is tight. The reason is that, when the error bound is tight, more quantization levels are used to encode the curve-fitting error, resulting a Huffman tree with more branches and longer codes. When

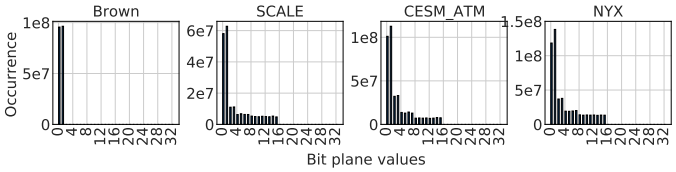


Fig. 21: Distribution of ZFP transform coefficient bit plane.

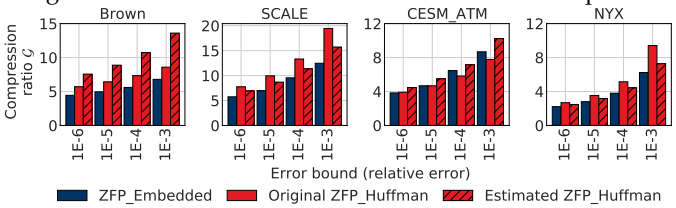


Fig. 22: Compression ratio of *ZFP_Embedded*, *ZFP_Huffman*.

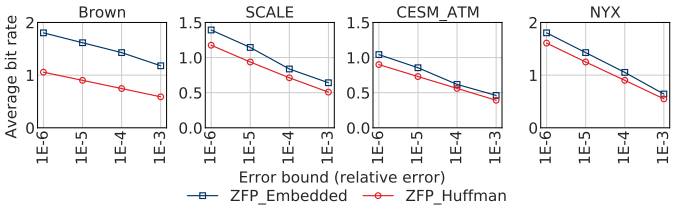


Fig. 23: Bit-rate achieved by *ZFP_Embedded* and *ZFP_Huffman*.

error bound is loose, less quantization levels are used and thus shorter codes. However, *SZ_ZFPL* still needs to encode all bit-planes of the transformation coefficients, resulting lower compression ratios.

Case study 2: exploring Huffman encoding for ZFP. ZFP currently employs a customized embedded encoding to compress the transform coefficient bit planes within each block. The design of block-wise compression is primarily to support random access to the compressed data. However, block-wise encoding does not exploit the potential similarity between bit planes across blocks. In this work, we consider exploring such similarity using Huffman encoding. The rationale behind using Huffman encoding is that bit plane values usually consist of a small set of distinct values, as shown in Fig. 21, which is due to the fact that each bit plane consists of a limited number of bits. For example, the value of a 1D bit plane with 4 bits can only range from 0 to 31, while the value of a 2D bit plane with 16 bits can range from 0 to 65,535. Therefore, the bit plane can be suitable for Huffman encoding.

As discussed in Section 4.2, the estimation of Huffman encoding output, i.e., $\mathcal{J}$ and $\mathcal{K}$, essentially comes down to the distribution of Huffman coding bit length. As such, we can acquire such a distribution by performing Huffman encoding on a small set of transform coefficients populated based on the Laplacian model. Therefore, the compression ratio of Huffman encoding based ZFP, denoted as *ZFP_Huffman*, can be calculated as $\mathcal{G}_{ZFP\_Huffman} = \frac{\mathcal{U}}{\mathcal{P}+\mathcal{K}+\mathcal{J}}$. In Fig. 22, we demonstrate the measured and estimated compression ratios of *ZFP_Huffman*, compared with the compression ratio of the original embedded encoding based ZFP, denoted as *ZFP_Embedded*. It is shown that our model can accurately capture the compression ratio of *ZFP_Huffman* to reflect the performance difference between two encoding schemes. Generally, *ZFP_Huffman* achieves higher compression ratios than *ZFP_Embedded*. Such a performance outcome demonstrates the efficiency of compressing non-orthogonal transform coefficients bit plane

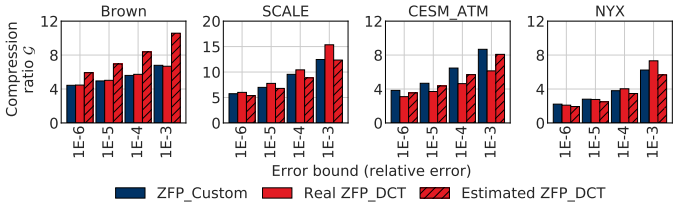


Fig. 24: Compression ratio of *ZFP_Custom* and *ZFP_DCT*.

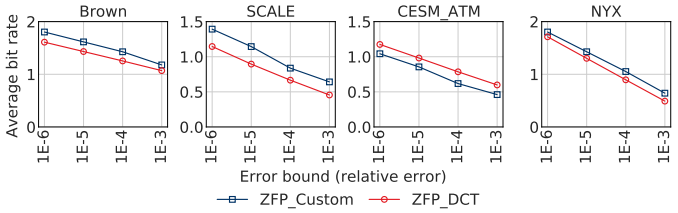


Fig. 25: Bit-rate achieved by *ZFP_Custom* and *ZFP_DCT*.

values using Huffman encoding. Fig. 23 demonstrates the average bit rate achieved by *ZFP_Embedded* and estimated *ZFP_Huffman*. It is shown that both *ZFP_Embedded* and *ZFP_Huffman* demonstrate relatively steady bit rates that changes linearly over relative error bounds.

Case study 3: exploring discrete cosine transform for ZFP. ZFP originally adopts a customized non-orthogonal transform to decorrelate the values of each block. The advantage of the customized approach is the computational efficiency achieved by lift implementation and bit operations. However the decorrelation efficiency might not be optimal as it also depends on the input data. In this work, we consider using DCT to replace the customized non-orthogonal transform scheme. In Fig. 24, we demonstrate the measured and estimated compression ratio of DCT-based ZFP, denoted as *ZFP_DCT*, compared with the compression ratio of original ZFP, denoted as *ZFP_Custom*. We also show the average bit rate achieved by DCT-based ZFP and original ZFP in Fig. 25. It is shown that DCT-based ZFP outperforms the original ZFP on *Brown*, *SCALE*, and *NYX*. Such performance demonstrates both the efficiency of correlating scientific data using DCT and the motivation of exploring transform schemes in ZFP compression.

Observation: *Overall, the results illustrate the effectiveness and potential benefit of zPerf in exploring the design space. In particular, the alternative SZ_RLE achieves higher compression ratios at loose error bounds, while ZFP_Huffman consistently outperforms the original encoding in ZFP. As such, the compressor developers can understand the performance benefits before labor-intensive development are underway, and make more informed decisions for future opportunities.*

7 CONCLUSIONS

In this paper, we present zPerf, a gray-box approach for lossy compression performance modeling and estimation. Based on the understanding of the inner compression mechanism, we discuss the modeling and estimation of compression ratio and throughput for two state-of-the-art lossy compressors, SZ and ZFP. We thoroughly evaluate the accuracy of zPerf on eight scientific datasets and compare the performance of zPerf against the sampling-based approach. The evaluation results demonstrate the effectiveness of zPerf. We also illustrate the benefit of zPerf for design space exploration of lossy compression. In the future, we would like to extend our work by: 1) Investigating the error propagation from low-level compression metrics, like the average bit-length, and the number of bit planes to encode, to high-level compression performance; 2) Quantifying the data smoothness to improve the modeling of low-level metrics that are highly data dependent; 3) Comparing zPerf with more advanced sampling techniques, like block-wise and histogram-based sampling; 4) Incorporating more compression techniques, like SZ3 and MGARD, for which we could not discuss in full details due to the space limit.

REFERENCES

[1] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci *et al.*, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–9.

[2] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "PreDatA–preparatory data analytics on peta-scale machines," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[3] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi, "Enabling in-situ execution of coupled scientific workflow on multi-core platform," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 1352–1363.

[4] M. Gamell, I. Rodero, M. Parashar, J. C. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. G. Landge, A. Gyulassy, P. McCormick *et al.*, "Exploring power behaviors and trade-offs of in-situ data analytics," in *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*. IEEE, 2013, pp. 1–12.

[5] W.-k. Liao, A. Ching, K. Coloma, A. Nisar, A. Choudhary, J. Chen, R. Sankaran, and S. Klasky, "Using MPI file caching to improve parallel write performance for large-scale scientific applications," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2007, pp. 1–11.

[6] W.-k. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols," in *SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. IEEE, 2008, pp. 1–12.

[7] 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.

[8] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu, "An ephemeral burst-buffer file system for scientific applications," in *SC'16*, 2016.

[9] 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.

[10] 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.

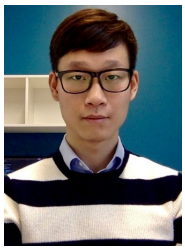
[11] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[12] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the multivariate case," *SIAM Journal on Scientific Computing*, vol. 41, no. 2, pp. A1278–A1303, 2019.

[13] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu, and Z. Qiao, "Understanding and modeling lossy compression schemes on hpc scientific data," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS 18)*, 2018, pp. 1–10.

[14] 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," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 438–447.

[15] 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 *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 1129–1139.
[16] S. Wold, "Spline functions in data analysis," *Technometrics*, vol. 16, no. 1, pp. 1–11, 1974.
[17] X. He and P. Shi, "Monotone b-spline smoothing," *Journal of the American statistical Association*, vol. 93, no. 442, pp. 643–650, 1998.
[18] 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.
[19] W. Austin, G. Ballard, and T. G. Kolda, "Parallel tensor compression for large-scale scientific data," in *2016 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2016, pp. 912–922.
[20] X. Liang, Q. Gong, J. Chen, B. Whitney, L. Wan, Q. Liu, D. Pugmire, R. Archibald, N. Podhorszki, and S. Klasky, "Error-controlled, progressive, and adaptable retrieval of scientific data with multilevel decomposition," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–13.
[21] R. Underwood, S. Di, J. C. Calhoun, and F. Cappello, "Fraz: a generic high-fidelity fixed-ratio lossy compression framework for scientific floating-point data," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 567–577.
[22] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing lossy compression rate-distortion from automatic online selection between SZ and ZFP," *IEEE Transactions on Parallel and Distributed Systems*, 2019.
[23] K. Zhao, S. Di, X. Liang, S. Li, D. Tao, Z. Chen, and F. Cappello, "Significantly improving lossy compression for hpc datasets with second-order prediction and parameter optimization." Association for Computing Machinery, Inc, 6 2020, pp. 89–100.
[24] S. Jin, S. Di, S. Byna, D. Tao, and F. Cappello, "Improving prediction-based lossy compression dramatically via ratio-quality modeling," *arXiv preprint arXiv:2111.09815*, 2021.
[25] S. Jin, J. Pulido, P. Grosset, J. Tian, D. Tao, and J. Ahrens, "Adaptive configuration of in situ lossy compression for cosmology simulations via fine-grained rate-quality modeling," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, 2021, pp. 45–56.
[26] Z. Qin, J. Wang, Q. Liu, J. Chen, D. Pugmire, N. Podhorszki, and S. Klasky, "Estimating lossy compressibility of scientific data using deep neural networks," *IEEE Letters of the Computer Society*, vol. 3, no. 1, pp. 5–8, 2020.
[27] J. Wang, T. Liu, Q. Liu, X. He, H. Luo, and W. He, "Compression ratio modeling and estimation across error bounds for lossy compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1621–1635, 2020.
[28] L. Noordsij, S. van der Vlugt, M. A. Bamakhrama, Z. Al-Ars, and P. Lindstrom, "Parallelization of variable rate decompression through metadata," in *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2020, pp. 245–252.
[29] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–24.
[30] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, Sept 2006.
[31] P. Lindstrom. (2021) zfp 0.5.5 documentation. [Online]. Available: https://zfp.readthedocs.io/en/release0.5.5/modes.html#fixed-accuracy-mode/
[32] E. Y. Lam and J. W. Goodman, "A mathematical analysis of the dct coefficient distributions for images," *IEEE transactions on image processing*, vol. 9, no. 10, pp. 1661–1666, 2000.
[33] T. Eltoft, T. Kim, and T.-W. Lee, "On the multivariate laplace distribution," *IEEE Signal Processing Letters*, vol. 13, no. 5, pp. 300–303, 2006.
[34] J. Diffenderfer, A. L. Fox, J. A. Hittinger, G. Sanders, and P. G. Lindstrom, "Error analysis of ZFP compression for floating-point data," *SIAM Journal on Scientific Computing*, vol. 41, pp. A1867–A1898, 2019.
[35] K. Jack, "Chapter 5 - digital video processing," in *Digital Video and DSP*, ser. Instant Access, K. Jack, Ed. Burlington: Newnes, 2008, pp. 125–150. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780750689755000054
[36] C. Franck, A. Mark, B. Julie, B. Martin, C. Jong Youl, C. Emil Mihai, D. Sheng, G. Hanqi, L. Peter, and T. Ozan. (2021) Scientific data reduction benchmarks. [Online]. Available: https://sdrbench.github.io/
[37] (2021) Cori. [Online]. Available: https://www.nersc.gov/systems/cori/
[38] (2021) Summit - oak ridge leadership computing facility. [Online]. Available: https://www.olcf.ornl.gov/summit/

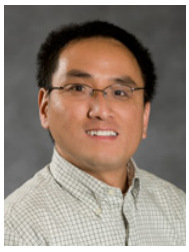


Jinzhen Wang is currently a PhD student in the Department of Electrical and Computer Engineering at NJIT. He received his B.S. from Shandong University, China, in 2015 and his M.S. in Electrical Engineering from NJIT in 2017. His research interests include high performance computing and scientific data management.

Qi Chen is currently an undergradute student in the College of Software at Northeastern University in China. His research interests include high performance computing and software engineering.

Tong Liu received the B.S. degrees in computer science from Huazhong University of Science and Technology, China, in 2015. He obtained his PhD degree at Temple University in 2021. His research interests include computer systems, data storage, cloud computing, high performance computing, and data reliability.

Qing Liu is an Assistant Professor in the Department of Electrical and Computer Engineering at NJIT and Joint Faculty with Oak Ridge National Laboratory. Prior to that, he was a staff scientist at Computer Science and Mathematics Division, Oak Ridge National Laboratory for 7 years. He received his Ph.D. in Computer Engineering from the University of New Mexico in 2008, M.S. and B.S., from Nanjing University of Posts and Telecom, China, in 2004 and 2001, respectively.



Xubin He received the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1995 and 1997, respectively, and the PhD degree in electrical engineering from University of Rhode Island, Kingston, RI, in 2002. He is currently a professor in the Department of Computer and Information Sciences, Temple University, Philadelphia, PA. His research interests include computer architecture, data storage systems, virtualization, and high availability computing. Dr. He received the Ralph E. Powe Junior Faculty Enhancement Award in 2004 and the Sigma Xi Research Award (TTU Chapter) in 2005 and 2010. He is a senior member of the IEEE, a member of the IEEE Computer Society and USENIX.