Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

TOWARD SMART AND EFFICIENT SCIENTIFIC DATA MANAGEMENT

by
Jinzhen Wang

Scientific research generates vast amounts of data, and the scale of data has significantly increased with advancements in scientific applications. To manage this data effectively, lossy data compression techniques are necessary to reduce storage and transmission costs. Nevertheless, the use of lossy compression introduces uncertainties related to its performance. This dissertation aims to answer key questions surrounding lossy data compression, such as how the performance changes, how much reduction can be achieved, and how to optimize these techniques for modern scientific data management workflows.

One of the major challenges in adopting lossy compression techniques is the trade-off between data accuracy and compression performance, particularly the compression ratio. This trade-off is not well understood, leading to a trial-and-error approach in selecting appropriate setups. To address this, the dissertation analyzes and estimates the compression performance of two modern lossy compressors, SZ and ZFP, on HPC datasets at various error bounds. By predicting compression ratios based on intrinsic metrics collected under a given base error bound, the effectiveness of the estimation scheme is confirmed through evaluations using real HPC datasets.

Furthermore, as scientific simulations scale up on HPC systems, the disparity between computation and input/output (I/O) becomes a significant challenge. To overcome this, error-bounded lossy compression has emerged as a solution to bridge the gap between computation and I/O. Nonetheless, the lack of understanding of compression performance hinders the wider adoption of lossy compression. The dissertation aims to address this challenge by examining the complex interaction

between data, error bounds, and compression algorithms, providing insights into compression performance and its implications for scientific production.

Lastly, the dissertation addresses the performance limitations of progressive data retrieval frameworks for post-hoc data analytics on full-resolution scientific simulation data. Existing frameworks suffer from over-pessimistic error control theory, leading to fetching more data than necessary for recomposition, resulting in additional I/O overhead. To enhance the performance of progressive retrieval, deep neural networks are leveraged to optimize the error control mechanism, reducing unnecessary data fetching and improving overall efficiency.

By tackling these challenges and providing insights, this dissertation contributes to the advancement of scientific data management, lossy data compression techniques, and HPC progressive data retrieval frameworks. The findings and methodologies presented pave the way for more efficient and effective management of large-scale scientific data, facilitating enhanced scientific research and discovery.

In future research, this dissertation highlights the importance of investigating the impact of lossy data compression on downstream analysis. On the one hand, more data reduction can be achieved under scenarios like image visualization where the error tolerance is very high, leading to less I/O and communication overhead. On the other hand, post-hoc calculations based on physical properties after compression may lead to misinterpretation, as the statistical information of such properties might be compromised during compression. Therefore, a comprehensive understanding of the impact of lossy data compression on each specific scenario is vital to ensure accurate analysis and interpretation of results.

TOWARD SMART AND EFFICIENT SCIENTIFIC DATA MANAGEMENT

by
Jinzhen Wang

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

Helen and John C. Hartmann Department of
Electrical and Computer Engineering

August 2023

APPROVAL PAGE

TOWARD SMART AND EFFICIENT SCIENTIFIC DATA MANAGEMENT

Jinzhen Wang

Dr. Qing Liu, Dissertation Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Nirwan Ansari, Committee Member — Date
Distinguished Professor of Electrical and Computer Engineering, NJIT

Dr. Mengchu Zhou, Committee Member — Date
Distinguished Professor of Electrical and Computer Engineering, NJIT

Dr. Roberto Rojas-Cessa, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Xubin He, Committee Member — Date
Professor of Computer and Information Sciences,
Temple University, Philadelphia, PA

BIOGRAPHICAL SKETCH

Author: Jinzhen Wang

Degree: Doctor of Philosophy

Date: August 2023

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2023
- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2017
- Bachelor of Science in Internet of Things Engineering,
  Shandong University, Jinan, Shandong, China, 2015

Major: Electrical Engineering

Presentations and Publications:

Jinzhen Wang, Qi Chen, Tong Liu, Qing Liu, Xubin He, "Zperf: A Statistical Gray-Box Approach to Performance Modeling and Extrapolation for Scientific Lossy Compression," *in IEEE Transactions on Computers*, early access, 2023.

Jinzhen Wang, Xin Liang, Ben Whitney, Jieyang Chen, Qian Gong, Xubin He, Lipeng Wan, Scott Klasky, Norbert Podhorszki, Qing Liu, "Improving Progressive Retrieval for HPC Scientific Data Using Deep Neural Network," *In IEEE 38th International Conference on Data Engineering (ICDE)*, accepted for publication, 2023.

Tong Liu, Jinzhen Wang, Qing Liu, Shakeel Alibhai, Tao Lu, Xubin He, "High-ratio Lossy Compression: Exploring the Autoencoder to Compress Scientific Data," *IEEE Transactions on Big Data*, vol. 9, no. 1, pp. 22-36, 2023.

Jinzhen Wang, Pascal Grosset, Terece L Turton, James Ahrens, "Analyzing the Impact of Lossy Data Reduction on Volume Rendering of Cosmology Data," *2022 IEEE/ACM 8th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD)*, pp. 11-20, 2022.

Nan Wang, Tong Liu, Jinzhen Wang, Qing Liu, Shakeel Alibhai, Xubin He, "Locality-based Transfer Learning on Compression Autoencoder for Efficient Scientific Data Lossy Compression," *Journal of Network and Computer Applications*, vol. 205, pp. 103452, 2022.

Xinying Wang, Lipeng Wan, Jieyang Chen, Qian Gong, Ben Whitney, Jinzhen Wang, Ana Gainaru, Qing Liu, Norbert Podhorszki, Dongfang Zhao, Feng Yan, Scott Klasky, "Unbalanced Parallel I/O: An Often-Neglected Side Effect of Lossy Scientific Data Compression," *2021 7th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-7)*, pp. 26-32, 2021.

Tong Liu, Shakeel Alibhai, Jinzhen Wang, Qing Liu, Xubin He, "Reducing the Training Overhead of the HPC Compression Autoencoder via Dataset Proportioning," *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1-7, 2021.

Zhenlu Qin, Jinzhen Wang, Qing Liu, Jieyang Chen, Dave Pugmire, Norbert Podhorszki, Scott 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.

Jinzhen Wang, Tong Liu, Qing Liu, Xubin He, Huizhang Luo, Weiming 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, 2019.

Tong Liu, Shakeel Alibhai, Jinzhen Wang, Qing Liu, Xubin He, Chentao Wu, "Exploring Transfer Learning to Reduce Training Overhead of HPC Data in Machine Learning," *2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1-7, 2019.

Huizhang Luo, Dan Huang, Qing Liu, Zhenbo Qiao, Hong Jiang, Jing Bi, Haitao Yuan, Mengchu Zhou, Jinzhen Wang, Zhenlu Qin, "Identifying Latent Reduced Models to Precondition Lossy Compression," *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 293-302, 2019.

Zhenbo Qiao, Tao Lu, Huizhang Luo, Qing Liu, Scott Klasky, Norbert Podhorszki, Jinzhen Wang, "SIRIUS: Enabling Progressive Data Exploration for Extreme-Scale Scientific Data," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 900-913, 2018.

Huizhang Luo, Qing Liu, Zhenbo Qiao, Jinzhen Wang, Mengxiao Wang, Hong Jiang, "DuoModel: Leveraging Reduced Model for Data Reduction and Re-Computation on HPC Storage," *IEEE Letters of the Computer Society*, vol. 1, no. 1, pp. 5-8, 2018.

To my family, for their unwavering love and support, and for believing in me even when I didn't believe in myself. Thank you for being my rock and safe harbor and for constantly pushing me to strive for excellence. This accomplishment is as much yours as it is mine.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my advisor Dr. Qing Liu for his guidance, support, and encouragement throughout my research project. His insightful feedback and constructive criticism helped me refine my ideas and develop a deeper understanding of my research topic.

I would also like to thank the members of my dissertation committee, Professor Xubin He, Professor Nirwan Ansari, Professor Mengchu Zhou, and Professor Roberto Rojas-Cessa, for their thoughtful feedback and constructive criticism, which have helped me to improve the quality and rigor of my research. Their insights and suggestions have challenged me to think more deeply and critically about my topic, and I am grateful for their generosity with their time and expertise.

In addition, I would like to express my gratitude to the Department of Electrical and Computer Engineering for providing the resource and support necessary to complete this dissertation.

I would like to extend my thanks to my colleagues, Zhenbo Qiao, Zhenlu Qin, Weiming He, and Qirui Tian, for their support, encouragement, and valuable feedback throughout my dissertation research. Their insightful comments and suggestions were instrumental in improving the quality of my research.

I am deeply grateful to my Mom, Yu Jin, and my Dad, Weihua Wang, for their unwavering love, support, and encouragement throughout my academic journey. Their love and support gave me the strength to overcome the challenges and obstacles that I encountered during my research.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

High-performance computing (HPC) is moving rapidly to the era of exascale, as empowered by the recent advances in system architecture and hardware and software ecosystems. The continuous scaling of application performance has enabled science to be done at an unparalleled microscopic level and new scientific breakthroughs that were not possible in the past. Nevertheless, with the increasing model resolution and fidelity, post-hoc data analytics has become increasingly cumbersome due to the high cost of moving data from persistent storage to memory and performing computation. Similarly, the aggregation and transportation of large volumes of data across multiple sites incur significant overheads for scientific applications running on grids due to the limited bandwidth of network [53]. To address this challenge, various methods have been developed to allow for more efficient data exploration for scientific simulations, and they generally tackle the issue from the following three dimensions: improving storage and I/O systems [27, 31, 51, 64, 67, 68, 70], in situ processing [7, 11, 14, 33], and data reduction [15,17,21,41,47,50]. None of these methods are deemed to be the silver bullet to solving the long-standing problem. Rather, they are designed to function at a single level in the system stack and complement each other to achieve the best application outcome.

In particular, there has been a multitude of efforts re-designing storage and I/O system for HPC, most notably the insertion of a burst buffer layer [51] into the HPC storage stack, and the use of emerging storage devices, such as NVRAM, die-stacked memory. Meanwhile, in-situ processing offers an alternative paradigm that allows data analysis to be done in memory while the simulation is running without being forced to move data to persistent storage for post-processing. In contrast, data reduction has shown great promise to fundamentally solve the I/O challenge when

used in conjunction with other methods across the software/hardware stack. For large datasets generated from high-performance computing (HPC) simulations, data reduction techniques aim to lower data volume and velocity so that the overhead of I/O and data analysis is more tractable. In general, data reduction takes advantage of the inherent redundancy in data, and state-of-the-art floating-point compressors can be either lossless [15, 26, 50, 56] or lossy [10, 18, 19, 21, 41, 47, 55], depending on whether there is information loss during compression. On one hand, while lossless compression incurs no information loss, the resulting reduction performance is often mild and far from being sufficient when dealing with extreme-scale datasets, *e.g.*, those of petabytes produced by simulations running at scale. On the other hand, by trading accuracy for performance, Lossy compression offers a much higher reduction performance, *e.g.*, a 400X of compression ratio as reported in prior work [21], to mitigate the bottleneck of I/O.

In reality, domain scientists commonly accept and leverage information loss to lower the application complexity. A classic example is the particle-in-cell (PIC) numerical solver, in which the number of macro-particles injected into the system is orders of magnitude less than that of physical particles, thus greatly reducing the computational complexity [60]. Therefore, due to its high reduction performance, lossy compression is often the preferred path forward for large-scale data management. When using lossy compressors, such as SZ [21] and ZFP [47], domain scientists are required to specify an error bound (or precision) in order to control the loss of accuracy of their data. Nonetheless, domain scientists often face a question: 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 it with full precision and avoid paying precious computing time for compression. Unfortunately, such a question is mostly answered through trial and error, which is cumbersome and costly for large simulations.

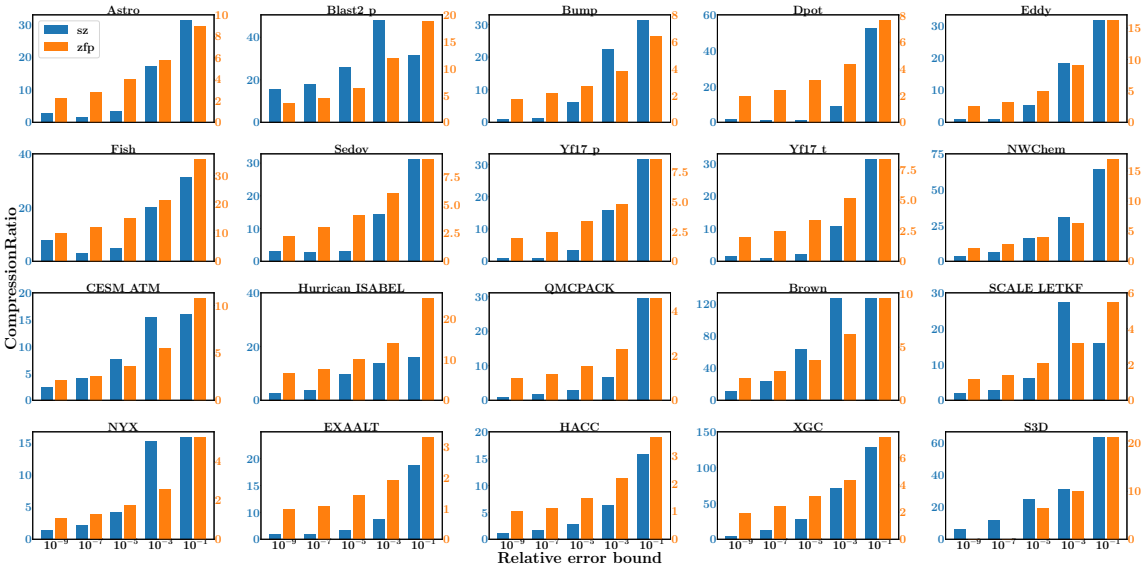


Figure 1.1 Compression ratio vs. error bound (SZ and ZFP).

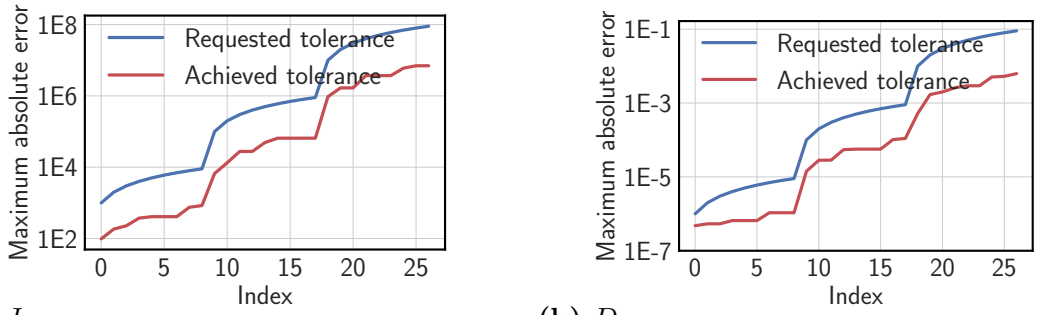
Intuitively speaking, a positive correlation exists between error bound and compression ratio. That is, the looser the error bound is, the higher the compression ratio can be achieved. Prior work [54] has shown that a looser error bound in SZ can result in a higher hit ratio of curve-fitting, improving the compression ratio. Yet, this may not be true across all error bounds. Figure 1.1 shows such relationships in SZ and ZFP on twenty scientific datasets, respectively. For ZFP, while it is clear that the compression ratio increases monotonically as the error bound loosens, the trend cannot be characterized by a simple linear or exponential relationship. Meanwhile, for SZ, the monotonically increasing trend is only applicable to some datasets, *e.g.*, *Bump*, *Sedov*, *NWChem*, *QMPACK*, *HACC*, *XGC* and *S3D*. For others, the compression ratio decreases slightly initially and then quickly ramps up. With such counter-intuitive trends of compression ratios, it is hard for domain scientists to select an appropriate error bound that satisfies their reduction goals while minimizing the loss of information. They may have to exhaust many error bounds before identifying a satisfactory one, and this process can be both time- and resource-consuming for large datasets since compression is highly computation-intensive in nature.

To that end, Chapter 2 aims to gain a deep understanding of the inner mechanisms of lossy compressors and develop modeling techniques to guide the selection of appropriate error bounds. The goal is to predict the order of magnitude of compression ratios so that the estimation can be useful in science production in order to satisfy the storage constraints.

Chapter 3 further extends the sampling-based compression ratio modeling into a statistical gray-box approach for lossy compression performance modeling, named as zPerf. In contrast to the sampling-based approach, zPerf models the compression performance by leveraging the statistical distribution of data features and core compression metrics. It provides superiority of less estimation overhead and the ability to explore the large design space of compression techniques.

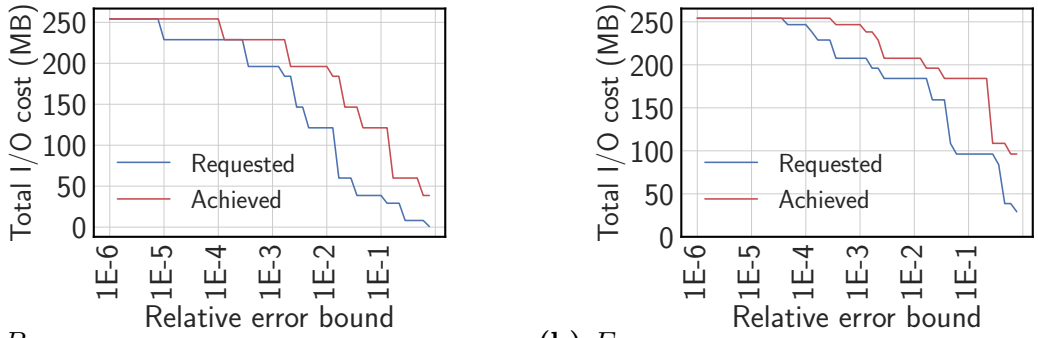
Despite the recent success of lossy compression in HPC applications, it suffers from the following weaknesses: 1) Nearly all lossy compressors require data to be compressed at a given error bound. Once the data is compressed, the error tolerance cannot be adjusted after the fact. This causes problems for datasets that are shared within a large community of users where the accuracy needs can be diverse. 2) Lossy compressors were mostly designed to reduce the storage footprint, and after decompression, the same degrees of freedom of data will be fed into data analysis. As such, most lossy compressors do not reduce the cost of computation.

Driven by the aforementioned weaknesses, an error-controlled data decomposition and progressive recomposition were recently designed [8]. The key ideas are that, during storage, data are transformed into different levels of precision coefficients using a combination of a multi-grid-like decomposition and bit-plane encoding. Then upon retrieval, a progressive retrieval framework fetches part of the decomposed data and recomposes an approximation to the original data with reduced accuracy. A major advantage is that the degrees of freedom are reduced so that the cost of both I/O and computation can be controlled in a fine-grained manner. Nevertheless, as pointed out



(a) J_x **(b)** D_u

Figure 1.2 Requested vs. achieved error tolerance.



(a) B_x **(b)** E_x

Figure 1.3 Requested vs. achieved I/O.

by another work [45], the progressive retrieval framework employs an over-pessimistic error control, as shown in Figure 1.2, resulting in fetching more data than needed, as shown in Figure 1.3. Therefore, additional overhead exists during the data retrieval, hurting the I/O performance.

Intuitively speaking, the retrieved data size depends on both the user-prescribed error bounds as well as the data characteristics. For example, more data should be retrieved if users require a higher data accuracy. On the other hand, if the original data demonstrate stronger smoothness, less data is needed for reconstruction. Therefore, the amount of data that needs to be retrieved is of the complex interplay between data characteristics and the error bounds and is hard to capture quantitatively. In Chapter 4, a DNN-based progressive retrieval framework is proposed to leverage the Deep Neural Network (DNN) to capture such complex interactions so that a minimum amount of data can be retrieved to reduce the I/O overhead.

As for future research work, Chapter 5 highlights the importance of investigating the impact of lossy data compression on downstream analysis. Lossy compression techniques, which aim to reduce scientific data sizes, can have significant implications for the accuracy, reliability, and interpretability of compressed data. It is vital for scientific users to understand how much data compression be done without compromising the data integrity and downstream analysis. In this chapter, an ongoing effort to investigate the impact of lossy compression techniques on scientific visualization is discussed.

In the following table, we list all the datasets used in this dissertation. Specifically, the name, dimension, data type, volume size, and brief descriptions of the datasets are included.

Table 1.1 Dataset Description

Dataset	**Dimension**	**Type**	**Size**	**Description**
Dpot	$[1, 20694]$	double	166 KB	Electric potential deviation in a plasma physics simulation
Astro	$[1, 65536]$	double	524 KB	Velocity magnitude in a supernova simulation
Fish	$[1, 65536]$	double	524 KB	Velocity magnitude in a CFD calculation
Sedov	$[1, 78144]$	double	625 KB	Pressure of strong shocks in a hydrodynamical simulation
Blast2_p	$[1, 578880]$	double	5 MB	Pressure of strong shocks in a gas-dynamical simulation
Eddy	$[1, 282616]$	double	2 MB	Velocity in a 2D solution to Navier-Stokes equations
Yf17_p	$[1, 97104]$	double	777 KB	Pressure in a computational fluid dynamics calculation
Yf17_t	$[1, 97104]$	double	777 KB	Temperature in a computational fluid dynamics calculation
Bump	$[1, 55692]$	double	446 KB	Flow density of an axisymmetric bump
CEMS_ATM	$[26, 1800, 3600]$	single	674 MB	Climate simulation
EXAALT	$[1, 2869440]$	single	12 MB	Molecular dynamics simulation
Hurricane_ISABEL	$[100, 500, 500]$	single	100 MB	Climate simulation of hurricane
HACC	$[1, 280953867]$	single	1 GB	Cosmology: particle simulation
NYX	$[1, 512, 512]$	single	537 MB	Cosmology: Adaptive mesh hydrodynamics + N-body cosmological simulation
NWChem	$[1, 102953248]$	double	824 MB	Two-electron repulsion integrals computed over Gaussian-type orbital basis sets
QMCPACK	$[115, 69, 69, 88]$	single	631 MB	Many-body ab initio Quantum Monte Carlo
S3D	$[500, 500, 500]$	double	11 GB	Combustion simulation
XGC	$[20694, 512]$	double	339 MB	Fusion Simulation
Brown	$[1, 8388609]$	double	268 MB	Synthetic, generated to specified regularity
SCALE_LETKF	$[98, 1200, 1200]$	single	565 MB	Climate simulation
NSTX_GPI	$[369357, 64, 80]$	double	361 MB	NSTX Gas Puff Image (GPI) data

CHAPTER 2

COMPRESSION RATIO MODELING AND ESTIMATION FOR LOSSY COMPRESSION ACROSS ERROR BOUNDS

2.1 Motivation

A previous work [54] first proposed a sampling-based methodology to predict SZ and ZFP compression ratios. The key idea is to use sampled data to extrapolate the compression ratios of the full data, leveraging the statistical similarity between the two datasets. Despite the fact that the estimation scheme achieves high accuracy, the outcome of the estimation is sensitive to the sampling ratio. With a higher sampling ratio (thus smaller sampled data), more information is lost, and thus the estimation will deviate from the true compression ratio. Conversely, the scheme can achieve fair estimations with a lower sampling ratio, such as 1% and 0.1%. Nevertheless, extreme-scale datasets at the level of petabytes will still be very costly to handle after being down-sampled to terabytes. Further, to estimate the compression ratios at multiple error bounds, one must compress the down-sampled data at each target error bounds, which would be too expensive. At last, estimating the compression ratio of the full dataset from the sampled dataset is not always feasible since this approach strictly requires the bounded locality [30].

In light of the issues above, we take a new direction to achieve compression ratio estimation across error bounds. Our approach is motivated by the correlation between compression metrics and error bounds. We show the compression metrics of SZ and ZFP on the *Dpot* dataset followed by the Pearson correlation with the logarithm of the error bound in Tables 2.1 and 2.2, respectively. A detailed explanation of these compression metrics can be found in Subsections 2.2.3 and 2.2.4.

For SZ, most compression metrics, *HitRatio*, *OutlierSize*, *Mean of quantization factor* and *Variance of quantization factor* are highly correlated to the error bound.

Table 2.1 SZ Compression Metrics Across Error Bounds

Compression Metric	EB_1	EB_2	EB_3	EB_4	Correlation w. log of error bound
Error-bound	1E-9	1E-7	1E-5	1E-3	1.00
NodeCount	20,551	40,421	13,383	1,467	-0.66
HitRatio	0.78	0.98	1.0	1.0	0.82
QuantizationFactor	20,551	40,421	13,383	1,467	-0.66
Mean of quantization factor	798,747	1,024,573	1,048,574	1,048,576	-0.87
Variance of quantization Factor	207,742,027,628	30,042,598,502	37,446,718	3,770	0.87
TreeSize	184,960	363,790	120,448	13,204	-0.66
EncodeSize	49,900	70,639	50,503	27,354	-0.64
OutlierSize	74,205	4,623	0	0	-0.80

Table 2.2 ZFP Compression Metrics Across Error Bounds

Compression Metric	EB_1	EB_2	EB_3	EB_4	Correlation w. log of error bound
Error-bound	1E-9	1E-7	1E-5	1E-3	1.00
BitsPerBitplane	3.50	3.48	3.44	3.37	-0.98
MaxPrec	33.84	26.84	19.84	13.84	-1.0
MaxExp	2.84	2.84	2.84	2.84	N/A
BlockSize	118.53	93.34	68.15	46.56	-0.99

Yet, *NodeCount*, *TreeSize*, *EncodeSize*, *OutlierSize* and *QuantizationFactor* are less correlated to the error bound. This is because *NodeCount* is observed to increase first and decrease later with the error bound, thus being less correlated with the error bound, and it is a dominating factor that in turn affects *TreeSize*, *EncodeSize*, *OutlierSize*, and *QuantizationFactor*.

Similarly, for ZFP, the compression metrics with Pearson correlation coefficients to the logarithms of error bounds are shown in Table 2.2. It is shown that all parameters are highly correlated to the error bound, except *MaxExp* which is constant across the error bound. We note that the Pearson correlation coefficient is undefined for a random variable with zero variance.

This observation motivates us to leverage the correlation and capture the trend of compression metrics in order to estimate the compression ratio. We next discuss the methodology of capturing the trend of compression metrics and compression ratio estimation.

2.2 Compression Ratio Estimation Across Error Bounds

In this section, we first discuss the general methodology of estimating compression ratios across error bounds. We then develop the prediction models for SZ and ZFP.

For both compressors, our approach is to first predict the internal compression metrics across error bounds and then estimate compression ratios.

2.2.1 Basics

For the convenience of discussion, we list the notations used in the paper in Table 2.3. The datasets used for evaluation are briefly described in Table 1.1. The approach here is that by compressing data once at EB_{base} and additionally collecting a small set of compression metrics, one can capture the characteristics of data and behavior of a compressor, as well as the interplay between them. Based on this, we can further assess the compression ratio at EB_{new}. The compression ratio, denoted as *CompressionRatio*, is defined as the ratio of the original data size, *InputSize*, to the compressed data size, *OutputSize*, as shown in Equation (2.1). Clearly, for a given dataset, the problem of modeling *CompressionRatio* comes down to the modeling of *OutputSize*.

$$CompressionRatio = \frac{InputSize}{OutputSize} \tag{2.1}$$

As mentioned in Chapter 1, the error bound EB_i controls the tolerance of information loss during data compression. In general, there are two types of error bounds, *absolute* and *relative* error bounds, that are widely used in HPC data compression. Assume a data point has a value denoted as V, the absolute error bound is an upper bound of the difference between the original value and the decompressed value, so the decompressed value is in the range of $[V - EB_i, V + EB_i]$. In contrast, the relative error bound allows for an error that is relative to V and has an error tolerance range of $[V \cdot (1 - EB_i)$, $V \cdot (1 + EB_i)]$. Unless otherwise specified, we adopt the relative error bound in our work, since it results in commensurate information loss for both high and low values.

Table 2.3 A List of Symbols for Sampling-based Performance Modeling

Symbols	Description
General	
InputSize	Size of uncompressed dataset
OutputSize	Size of compressed dataset
CompressionRatio	The ratio of *InputSize* to *OutputSize*
EB_{base}, EB_{new}	The base error bound and target error bound to predict *CompressionRatio*
SZ compression metrics	
NodeCount	Number of Huffman tree nodes
HitRatio	Curve-fitting hit ratio
QuantizationFactors	Number of quantization factors used in Huffman encoding
TreeSize	Size of Huffman tree structure (in bytes)
EncodeSize	Size of Huffman encoding for all nodes (in bytes)
OutlierSize	Size of the binary representation of curve-missed points (in bytes)
qf_ebase_min, qf_ebase_max	The smallest and largest quantization factor at EB_{base}
qf_enew_min, qf_enew_max	The smallest and largest quantization factor at EB_{new}
ZFP compression metrics	
BitsPerBitplane	Number of bits used in encoding each bit plane
MaxPrec	Maximum number of bit planes to encode in order to meet the accuracy demand
MaxExp	The common (largest) exponential of each block
BlockSize	Size of each block data (in bits)

2.2.2 Methodology

To predict the compression ratio at error bound EB_{new}, we first perform a standard compression at error bound EB_{base}. During this process, we collect a set of compression metrics (detailed in Table 2.3) that have shown to be correlated with error bounds (Section 2.1). For example, we experimentally observed in SZ that the distribution of quantization factors, a key intermediate compression product that is more compressible than the original data, are highly similar at different error bounds and can be approximated by the Gaussian distribution. This critical observation enables us to extrapolate the quantization factors from EB_{base} to EB_{new} (Subsection 2.2.3). Once the quantization factors are obtained at EB_{new}, we can further construct the new Huffman tree and extrapolate the compression ratio. Also, for ZFP, due to its block-wise operation (Subsection 2.2.4), characterizing the number of bits used per each block via two parameters, the number of bit planes to encode and the average bits used to encode each bit plane, will allow us to predict the average block size at other error bounds.

The general methodology is described as follows:

- **Step 1**: We run a standard compression at EB_{base} and collect a set of internal compression metrics, which are compressor dependent.

- **Step 2**: We analyze the compression metrics at EB_{base} and build models to predict them at EB_{new}. The intuition behind this is that by capturing the compression metrics, we can indirectly understand the data characteristics as well as how a compressor reacts to the data. These can be further exploited to extrapolate the compression performance.

- **Step 3**: We use the estimated compression metrics to further predict the compression ratio at EB_{new}.

Next, we discuss the estimation of SZ and ZFP, respectively.

2.2.3 Prediction of SZ compression ratio

In this chapter, we focus our discussion on SZ 2.0. Since some of the datasets have already been linearized, we compress all datasets as one-dimensional across the board for consistency. Therefore, the proposed regression-based prediction model is not used in our work since it targets multi-dimensional compression. For each data point, SZ checks whether it can be predicted by its previous points using either linear or quadratic curve-fitting, subject to a user-specified error bound. If so, this data point is deemed to be curve-fitted and is further discretized using a quantization factor followed by Huffman encoding. The intuition is that if there is local smoothness in data, the likelihood that data points are distributed closely around the predicted value is high. Therefore, after quantization, data points could potentially be mapped to an identical quantization factor and thus can be further compressed using Huffman encoding. The number of quantization factors, denoted as *QuantizationFactor*, is the number of discrete levels that SZ maps a curve-fitted data value into. This parameter can be either prescribed by the user or calculated by the compressor based on the data range and error bound. If the data point cannot be predicted by its previous

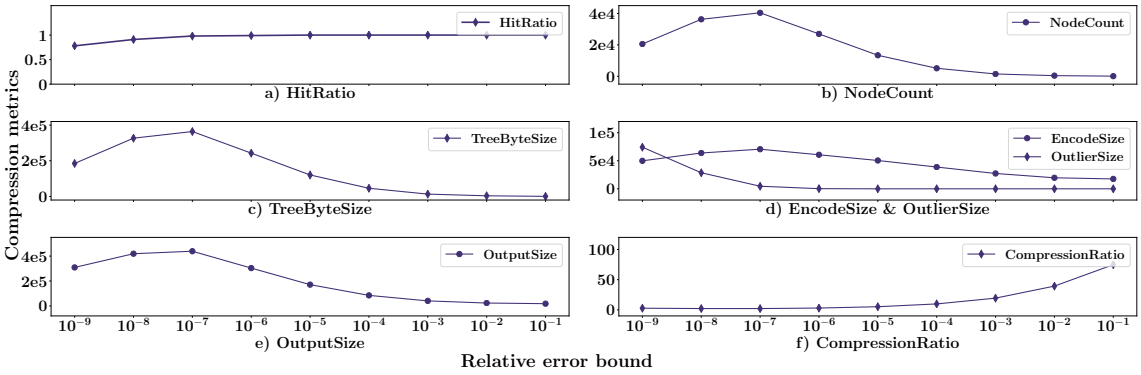


Figure 2.1 SZ compression metrics vs. relative error bound.

points, it is deemed to be curve-missed and is encoded using binary representation analysis. Namely, it utilizes *IEEE format 754* to represent curve-missed data points where data values are normalized and truncated based on the error bound, and then optimized by a leading-zero-based compression method [21].

As such, the size of SZ-compressed data consists of Huffman tree size, Huffman encoding size for curve-fitted data points, and binary representation size for curve-missed data points, as shown in Equation (2.2). We next first analyze the impact of each individual component on *OutputSize*.

$$OutputSize = TreeSize + EncodeSize + OutlierSize \tag{2.2}$$

SZ Compression and Its Internal Metrics We observe that most of the compression power in SZ comes from the curve-fitting and Huffman encoding. For example, for *Astro*, as shown in Figure 2.1, when the relative error bound is higher than 10^{-8}, *HitRatio* is consistently above 90%. Thus, the majority of data points are hit by curve-fitting. Therefore, we focus on curve-hitting and Huffman encoding in what follows.

HitRatio. A key compression metric that measures the effectiveness of curve-fitting is *HitRatio*, which is the percentage of data points that can be curve-fitted

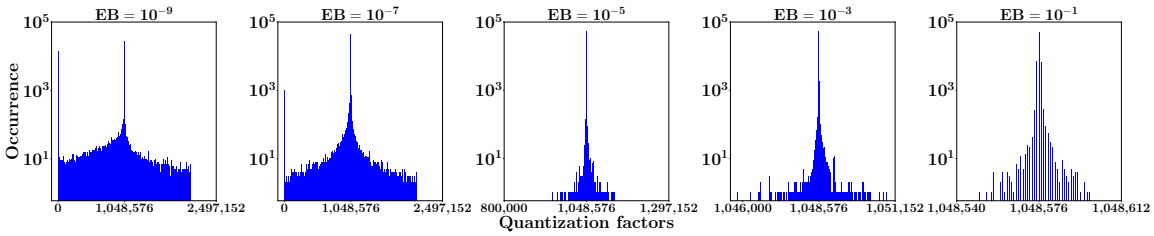


Figure 2.2 Quantization factor distribution (*Astro*).

and encoded using the Huffman tree under a given error bound. In Figure 2.1(a), we observe that for *Astro*, *HitRatio* increases monotonically from 70% to approximately 100% when the error bound loosens from 10^{-9} to 10^{-1}. Intuitively, the associated Huffman tree quantities *TreeSize* and *EncodeSize* should also increase since more data points need to be encoded as a result of increasing *HitRatio*. Nevertheless, the results in Figure 2.1(c) and (d) show that *TreeSize* and *EncodeSize* increase at first and then drop after the error bound reaches 10^{-7}. Thus, despite being an important metric, *HitRatio* may not be the sole factor in determining the outcome of compression.

NodeCount. Here *NodeCount* is the number of Huffman tree nodes used to encode the quantization factors. We observe that the resulting *TreeSize* and *EncodeSize* follow a similar trend as *NodeCount* across error bounds, indicating that *NodeCount* is another key factor that affects compression. Namely, it increases at first and then drops when the error bound reaches 10^{-7}, which is also the point where *HitRatio* reaches 100% (Figure 2.1(b)). Since *NodeCount* is equivalent to the unique number of quantization factors used in Huffman encoding, we aim to study the distribution of quantization factors across different error bounds and understand the trend of *NodeCount*. It can be seen from Figure 2.2 that quantization factors exhibit similar shapes across error bounds from 10^{-9} to 10^{-7}, but the shape narrows drastically thereafter. Namely, the range of quantization factors decreases from $[0, 2497152]$ to $[1048540, 1048612]$, and the number of points represented by each factor, indicated by the bar height, increases. We observe that fewer quantization factors are used after *HitRatio* approximates to 100%. The reason is that no more

data points can be curve-fitted, and further loosening the error bound will result in more data points being covered by a single quantization factor. The decreasing of unique quantization factors further leads to the decreasing of *NodeCount*.

Therefore, we believe *HitRatio* and *NodeCount* are the two main metrics affecting the compression of SZ. To extrapolate the compression ratio, we need to first model *HitRatio* and *NodeCount*, respectively.

SZ Modeling and Estimation We aim to predict SZ compression metrics from the base error bound, EB_{base}, to another error bound, EB_{new}, and further predict $CompressionRatio_{new}$ - the compression ratio at EB_{new}. To this end, we first discuss the modeling of *HitRatio* and *NodeCount*.

HitRatio. For SZ, *HitRatio* can be fairly well predicted since whether a data point is a hit or a miss only involves a simple comparison between the prediction error and the radius of hit, denoted as *HitRadius*, which is calculated as $HitRadius = EB_{new} \cdot V$, where V is the first value of each data segment. The prediction error is the difference between the predicted value, *e.g.*, using linear or quadratic curve fitting, and the real value. When compressing data at EB_{base}, if the prediction error at EB_{new} is no greater than *HitRadius*, the data point is considered a hit; otherwise, it is considered a miss. Thus, scanning all data points when compressing data at EB_{base}, *HitRatio* at EB_{new} can be additionally obtained with essentially no extra overhead.

The results of *HitRatio* estimation are shown in Figure 2.3, where the estimated *HitRatio* (in red) is compared against the real values (in blue) for error bounds from 10^{-9} to 10^{-1}. Overall the estimation of *HitRatio* is accurate, and the trend of *HitRatio* against the error bound is well modeled. Nevertheless, the estimation deviates from the real value for *Dpot* at error bounds of 10^{-7} and 10^{-5}, and *Eddy* at 10^{-7}. We comment that the deviation is caused by the simplification in modeling *HitRatio*. Namely, during compression, SZ calculates *HitRadius* on the basis of segments that

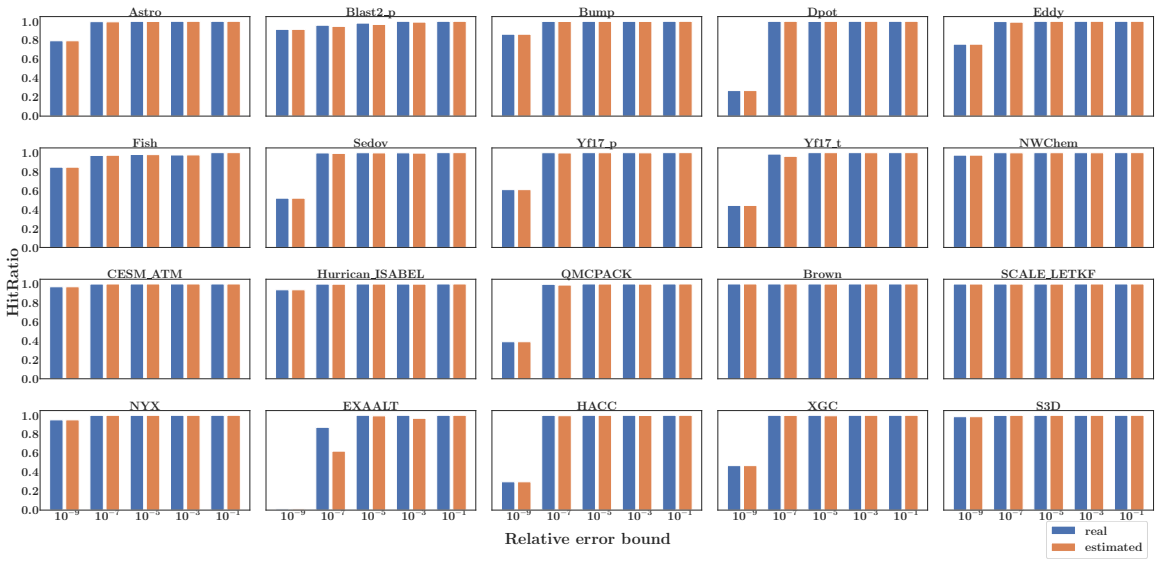


Figure 2.3 *HitRatio* estimation on evaluation datasets.

consists of 32 data points. However, during our estimation, we use the same *HitRadius* for all data points for simplification, which would otherwise require storing a vector of *HitRadius* and can be expensive for large datasets. This simplification can cause inaccuracy for *HitRatio* prediction. Also, the prior work on SZ [21] suggests that one should use the preceding compressed values instead of the original values to predict future values so that the predicted values are bounded by the error bounds. Since the difference between compressed values and original values is limited by the error bound, which is typically under 10^{-1}, we use the preceding original values for prediction to simplify the design.

NodeCount. *NodeCount* can be estimated utilizing the distribution of quantization factors. We further notice that, among all datasets we evaluated, the distributions of quantization factors across different error bounds are highly similar, and they follow the Gaussian distribution. In general, to characterize a Gaussian distribution, one only needs to determine the mean and variance. A caveat is that, despite the fact that the quantization factor distributions of EB_{base} and EB_{new} are observed to have identical means, they exhibit different variances, as shown in Figure 2.2. Therefore, when extrapolating *NodeCount* from EB_{base} to EB_{new}, the variance

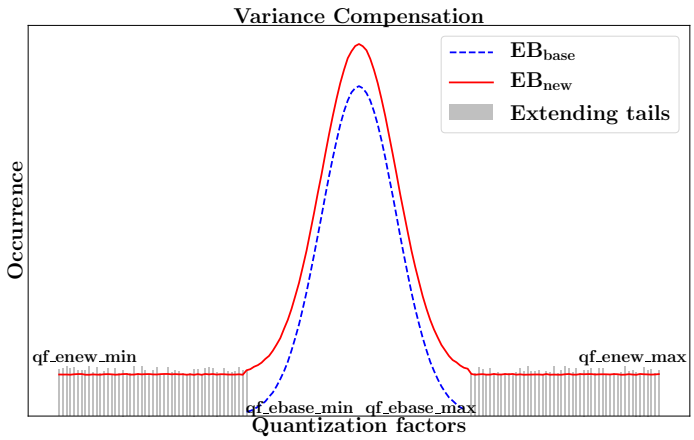


Figure 2.4 Variance compensation for quantization factor distribution.

needs to be compensated. We notice that when the error bound is enlarged from EB_{base} to EB_{new}, assuming $EB_{new} > EB_{base}$, those data points missed under EB_{base} but hit under EB_{new} essentially extend the tails of the Gaussian distribution under EB_{base}, thus increasing the variance. Hence, compensating the variance at EB_{new} comes down to obtaining the distribution of those newly hit points on both tails. Since the tails of the Gaussian distribution are relatively smooth, we simplify the problem by using a uniform distribution to model the added tails.

We illustrate the variance compensation scheme in Figure 2.4. The distribution of quantization factors at EB_{new} (red curve) has a wider range than the distribution at EB_{base} (blue curve). We model the added tails of newly hit data points (gray bars) using the uniform distribution. To this end, we need to identify the range of added tails, i.e., [*qf_enew_min, qf_ebase_min*] and [*qf_ebase_max, qf_enew_max*]. We note that *qf_ebase_min* and *qf_ebase_max* are the minimum and maximum values of quantization factors at EB_{base}. For *qf_enew_min* and *qf_enew_max*, we use the minimum and maximum of newly hit data points as approximations. We note that the newly hit data points at EB_{new} can be easily obtained by simply comparing *HitRadius* with the curve-fitting prediction error.

Next, we apply the uniform distribution to randomly generate two sets of values in the range of [*qf_enew_min, qf_ebase_min*] and [*qf_ebase_max, qf_enew_max*],

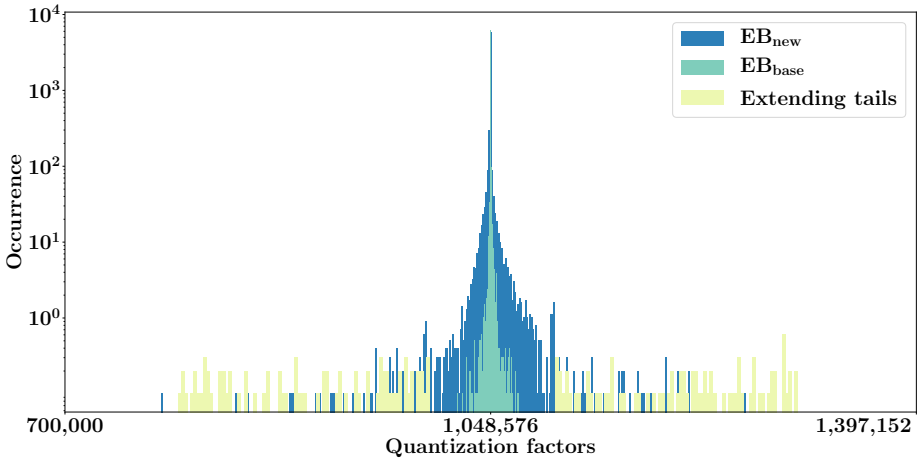


Figure 2.5 Quantization factor variance compensation.

respectively, to compensate for the difference in quantization factor distribution. The two sets of quantization factors generated from the uniform distribution are combined with the original distribution at EB_{base} to approximate the distribution at EB_{new}. We show the compensation results of *astro* in Figure 2.5. We can see that the distributions under EB_{base} (in green bars) and EB_{new} (in blue bars) have the same mean value of 1,048,576. The variances of distribution under EB_{new} and EB_{base} are 37,446,718, and 375,682, respectively. And the compensated variance of distribution at EB_{base} with the extended tails is 35,443,720, which is very close to the true variance at EB_{new}. Once the quantization factor distribution at EB_{new} is obtained, we can estimate *NodeCount* by counting the number of unique quantization factors in the estimated distribution.

CompressionRatio. Based upon the *HitRatio* and *NodeCount* estimations, we next describe the complete *CompressionRatio* estimation. The extrapolation of *CompressionRatio* from EB_{base} to EB_{new} involves the following steps:

- **Step 1**: Run the standard SZ compression for a given dataset at EB_{base}. Calculate *HitRadius* at EB_{new}. Record the SZ compression metrics in Table 2.3. In addition, record the prediction error for each value.

- **Step 2**: Extrapolate *HitRatio* to EB_{new} based on estimated *HitRadius* and the recorded prediction error.

- **Step 3**: Calculate *qf_enew_min* and *qf_enew_max*. Construct the quantization factor distribution and estimate the *NodeCount* at EB_{new}.

- **Step 4**: Estimate *TreeSize*, *EncodeSize*, and *OutlierSize*, based on the estimated *NodeCount* and *HitRatio*. The estimated output size, $OutputSize_{new}$, is the sum of the estimated $TreeSize_{new}$, $EncodeSize_{new}$ and $OutlierSize_{new}$.

In particular, for Step 4, the methodology used here, similar to our previous work [54], is based upon the following observations: 1) the Huffman tree size is proportional to the tree node count; 2) the encoding size is related to the depth of Huffman tree; and 3) the size of outliers is proportional to the number of outliers, with the size of each outlier being similar. The values of $TreeSize_{new}$, $EncodeSize_{new}$, and $OutlierSize_{new}$ at EB_{new} can be estimated as follows.

$$TreeSize_{new} = TreeSize_{base} \cdot \frac{NodeCount_{new}}{NodeCount_{base}}$$

$$EncodeSize_{new} = EncodeSize_{base} \cdot \frac{log_2 NodeCount_{new}}{log_2 NodeCount_{base}}$$

$$OutlierSize_{new} = OutlierSize_{base} \cdot \frac{OutlierCount_{new}}{OutlierCount_{base}}$$

Therefore,

$$OutputSize_{new} = TreeSize_{new} + EncodeSize_{new} + OutlierSize_{new}$$

$$CompressionRatio_{new} = \frac{InputSize}{OutputSize_{new}}$$

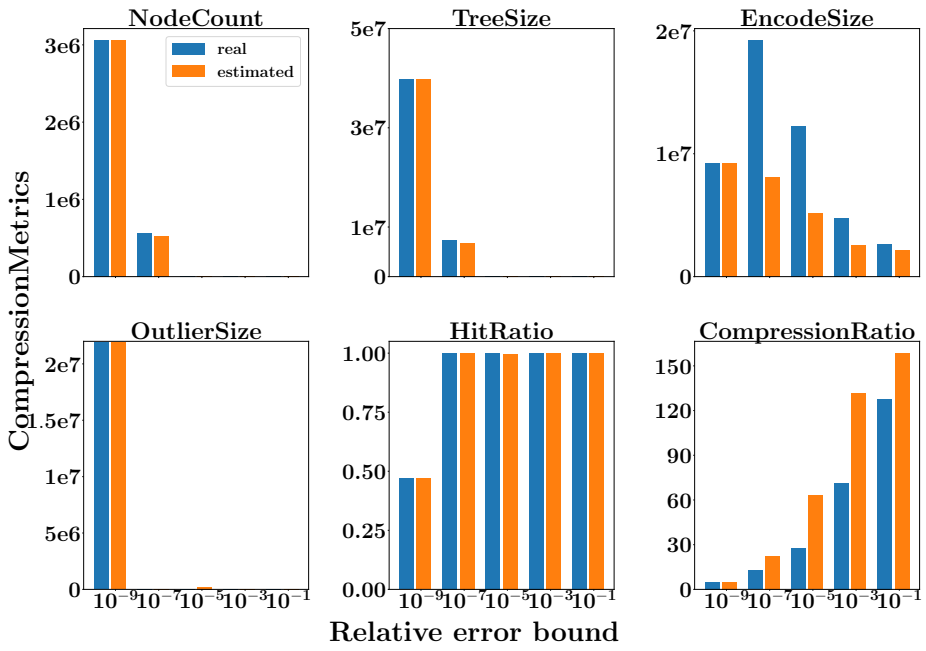


Figure 2.6 SZ compression metric estimation.

We take the *Astro* dataset as an example to illustrate the estimation of each component, as shown in Figure 2.6. We can see that the two key factors, *HitRatio* and *NodeCount*, are well predicted. The non-linearity observed in *NodeCount* is also captured, which leads to the modeling of *TreeSize*, *EncodeSize*, and *OutlierSize*. A more comprehensive evaluation of SZ *CompressionRatio* is presented in Section 2.3.

2.2.4 Prediction of ZFP compression ratio

ZFP Compression and Its Internal Metrics ZFP compresses data based on blocks. For each block, ZFP first transforms it into a set of floating-point mantissas along with a common exponent. The common exponent is computed from the largest absolute value in the block, resulting in mantissas in the range of $[-1, 1]$. Next, the floating-point mantissas are converted to fixed-point values and then taken into a reversible orthogonal transformation. The transformation, similar to the discrete cosine transform (DCT) used in JPEG image encoding, decorrelates spatially correlated values, resulting in near-zero transform coefficients that are typically more

compressible. The transform incurs equal importance for each transform coefficient, and each bit of coefficients within the same bit plane has the same impact on accuracy [47] [49]. Thus, transform coefficients can be compressed using embedded encoding [61] where a bit plane of coefficients is encoded at a time. Note that the number of bit planes to be encoded can be calculated from the user-specified precision.

ZFP can work in different modes depending on the user's requirements. In this work, we choose the fixed-accuracy mode, in which the absolute error bound is set by the parameter *accuracy*. The data points in each block are compressed up to a common minimum number of bit planes to meet the target error tolerance.

MaxPrec and BitsPerBitplane. Given that ZFP compresses data by blocks, *OutputSize* is simply the sum of the size of all compressed blocks (see in Equation (2.3)). We denote the compressed size of block i as $BlockSize_i$ (in bits), where $0 \leq i < n$ and n is the total number of blocks.

$$OutputSize = \frac{1}{8}\sum_{i=0}^{n-1} BlockSize_i \tag{2.3}$$

In the extreme case where a block has all zero values, *BlockSize* is one and only a single bit of zero is written. If a block has non-zero values, *BlockSize* is the total number of bits used to encode the values in the block. To control the accuracy of the compressed dataset, ZFP operates on a bit plane of coefficients at a time using the embedded encoding (as shown in Figure 2.7). When compressing the coefficients in a block by bit planes, *BlockSize* depends on *MaxPrec*, the number of bit planes to encode, and *BitsPerBitplane*, the number of bits consumed to encode each bit plane (see Equation (2.4)).

$$BlockSize_i = \sum_{j=0}^{MaxPrec-1} BitsPerBitplane_{ij} \tag{2.4}$$

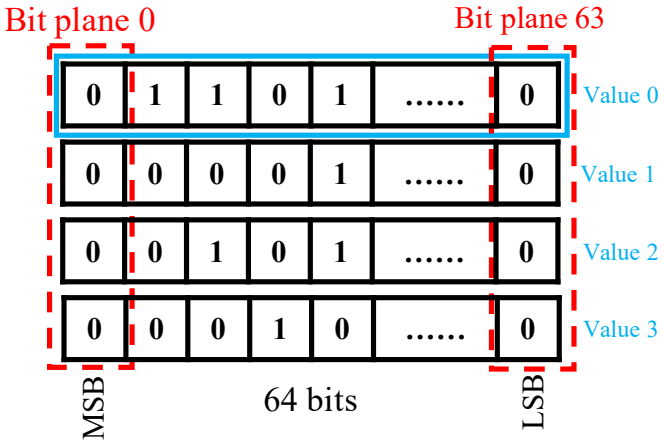


Figure 2.7 Bit planes in a ZFP block.

where $BitsPerBitplane_{ij}$ is the j-th bit plane to encode for block i. In order to estimate *CompressionRatio* of ZFP, we must model and estimate these two parameters.

Note that ZFP in the fixed accuracy mode only supports the absolute error bound. In order to select a spectrum of error bounds that covers both loose and tight bounds, we set the absolute error bound to the product of the root mean square (RMS) and the prescribed relative error bound. The definition of root mean square is defined as follows:

$$RMS = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}} \tag{2.5}$$

where $x_1, x_2, ...x_n$ is a set of values.

ZFP Modeling and Estimation In this section, we aim to model and estimate two key compression metrics, *MaxPrec* and *BitsPerBitplane*, and further predict *CompressionRatio*. The central idea is to take advantage of the correlation between compression metrics and error bounds to make predictions.

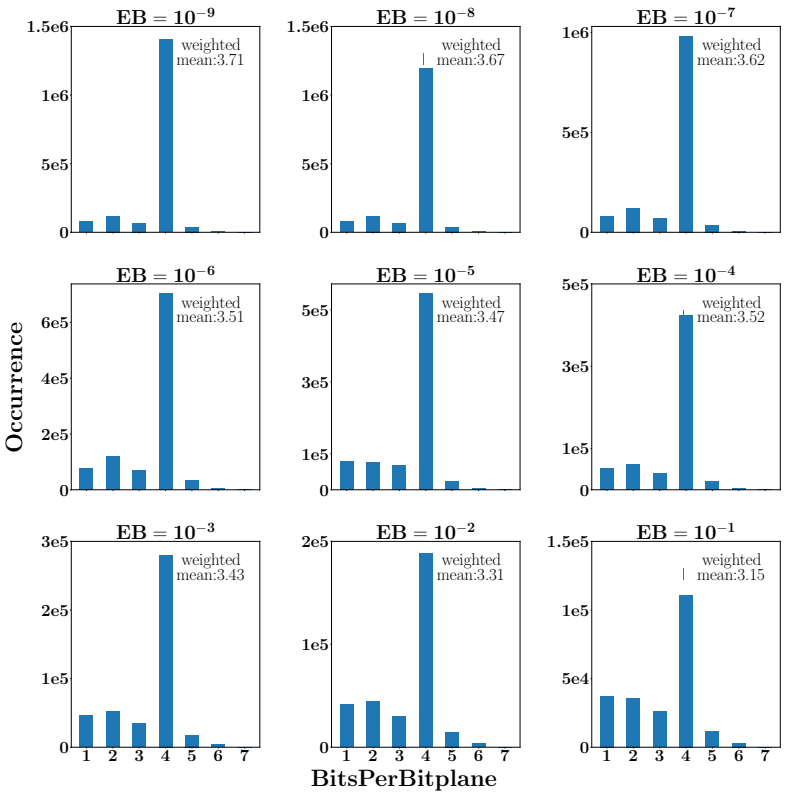


Figure 2.8 Histogram of bits used in each bit plane (*Eddy*).

As shown in Table 2.2, *MaxExp*, the maximum exponent value in a block, is constant across error bounds. *MaxPrec* decreases as the error bound loosens. It indicates that each block requires fewer bit planes to be encoded to satisfy the error tolerance. For *BitsPerBitplane*, although it decreases monotonically as well, we observe that the average number of bits to encode each bit plane decreases slowly from 3.50 to 3.37 when the error bound loosens from 10^{-9} to 10^{-3}. The reason is that this quantity highly depends on the actual bits of each bit plane but is not impacted by the error bound. Therefore, *BitsPerBitplane* is deemed to be less sensitive to the error bound, thus highly predictable at EB_{new}.

MaxPrec. We studied the ZFP implementation* and notice that *MaxPrec* is empirically calculated by Equation (2.6). In the equation, *MaxExp* is the largest

*This is of ZFP 0.5.3

exponent value in a block, and $log_2Accuracy$ is the smallest bit plane number that should be encoded to [48].

$$MaxPrec = MaxExp - log_2Accuracy + 2 \cdot (1 + d) \tag{2.6}$$

BitsPerBitplane. For a bit plane that has non-zero values, an embedded encoding scheme is applied to encode the bits of each of 4^d numbers into two parts: the first m bits are encoded verbatim where the value of m depends on the previous bit plane; then n bits are used to encode the remaining $4^d - m$ bits using run-length encoding. The total number of bits $(m + n)$ used to encode is in general data dependent, and it is non-trivial to calculate the exact number of bits used to encode each bit plane. Nevertheless, we observe that the distribution of *BitsPerBitplane* is highly similar for each dataset across error bounds. It is shown in Figure 2.8 that for *Eddy*, each bit plane uses around 3.5 bits on average, and this stands true for all error bounds. Therefore, we use a weighted average of *BitsPerBitplane* at EB_{base} to approximate that at EB_{new}. In *eddy* for example, the total number of bit planes to encode at error bound 10^{-9} is 1715895, among which, there are $[78281, 120245, 68547, 1404963, 36905, 6882, 72]$ bit planes using $[1, 2, 3, 4, 5, 6, 7]$ bits to encode, respectively. Thus, the weighted average of *BitsPerBitplane* is 3.71.

The weighted averages of *BitsPerBitplane* for *Eddy* across error bounds are shown in Figure 2.8. It is observed that *BitsPerBitplane* decreases slowly from 3.71 to 3.15 when the error bound loosens from 10^{-9} to 10^{-1}, validating the intuition that fewer bits are needed to encode each bit plane at a looser error bound. It also suggests that *BitsPerBitplane* is insensitive to the error bound. This conclusion stands for all twenty datasets and the complete results of *BitsPerBitplane* distributions are shown in Figure 2.12 (Subsection 2.3.2).

CompressionRatio. Now that we have modeled *MaxPrec* and *BitsPerBitplane*, $BlockSize_i$ can be approximated by the product of *MaxPrec* and the weighted average of $BitsPerBitplane_{ij}$, denoted as $\overline{BitsPerBitplane_i}$.

$$\begin{aligned} BlockSize_i &= \sum_{j=0}^{MaxPrec-1} BitsPerBitplane_{ij} \\ &\approx MaxPrec \cdot \overline{BitsPerBitplane_i} \end{aligned} \tag{2.7}$$

We take the *Eddy* dataset as an example to show the estimation result of *BlockSize*. As shown in Figure 2.9, the distributions of real (in red) and estimated (in blue) *BlockSize* are highly similar across error bounds. We note that the estimation error is a result of using the weighted average of *BitsPerBitplane* at EB_{base} for prediction.

With *BlockSize* modeled, *OutputSize* can be estimated using Equation (2.3). The process of compression ratio estimation can be broken down into the following steps:

- **Step 1**: For a given dataset, we run the ZFP compression in the fixed-accuracy mode at EB_{base}, to obtain *MaxExp* of each block and *BitsPerBitplane* of each bit plane.
- **Step 2**: We calculate the weighted average of *BitsPerBitplane* at EB_{base} and calculate the *MaxPrec* for error bound EB_{new}.
- **Step 3**: We estimate *BlockSize* at EB_{new} using Equation (2.7) and calculate the compressed data size *OutputSize* and *CompressionRatio*.

The results of compression ratio prediction for all datasets can be found in Subsection 2.3.2.

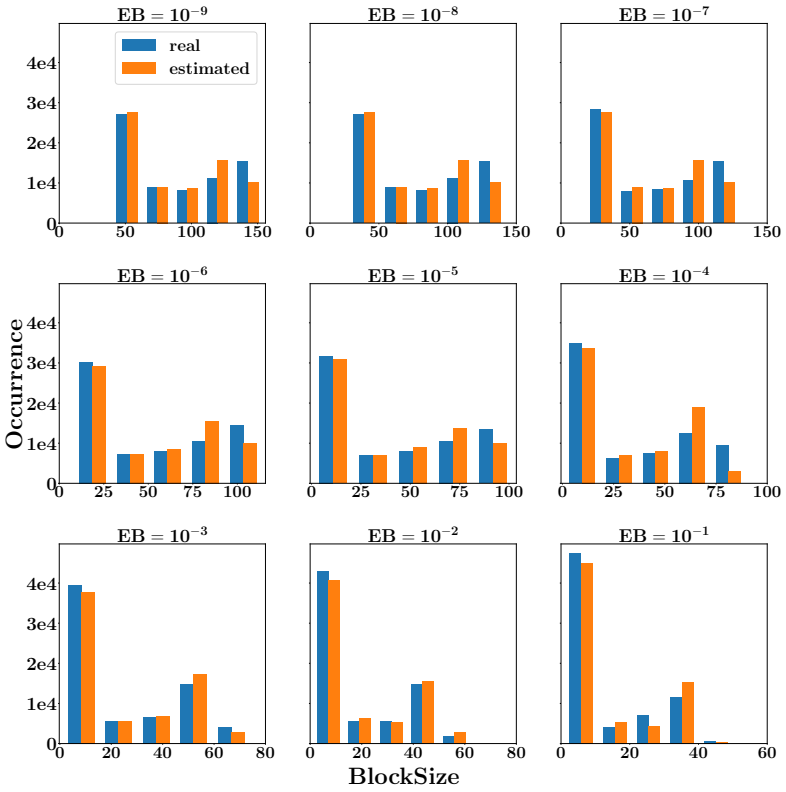


Figure 2.9 Distribution of real and estimated *BlockSize* across error bounds.

2.3 Evaluation

In this section, we present evaluations of our models for SZ and ZFP, respectively, on twenty real scientific datasets. Among them, eleven datasets are adopted from a suite of *Scientific Data Reduction Benchmarks* [25] which contains data from real-world scientific simulations, including climate simulation [2], hurricane simulation [1], cosmological simulation [6], molecular dynamic simulation [5], N-body cosmological simulation [9], example molecular 2-electron integral values [24], weather simulation [46], many-body ab initio Quantum Monte Carlo [39], combustion simulation [40], and fusion simulation [16]. We compare the estimated compression ratios with the real ones across error bounds. Note that EB_{base} is set to 10^{-9}. We also quantitatively analyze the estimation error of compression ratios for both compressors.

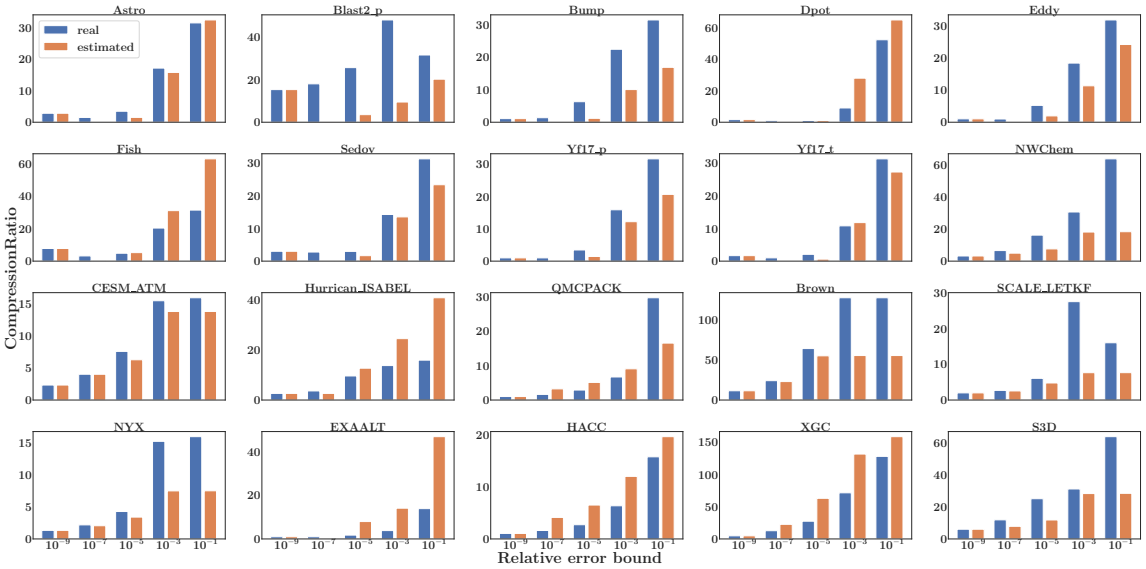


Figure 2.10 SZ compression estimation.

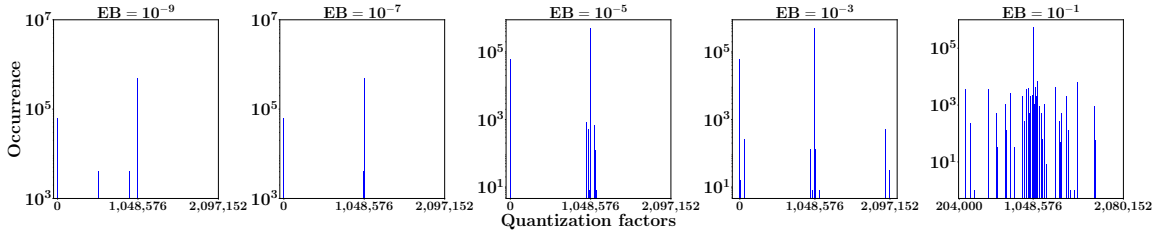


Figure 2.11 Quantization factor distribution.

2.3.1 SZ compression estimation

We apply the proposed estimation scheme to extrapolate the compression ratios of SZ across error bounds from 10^{-9} to other error bounds, as shown in Figure 2.10. We further show the prediction error for each dataset in Figure 2.14. For most datasets, the proposed scheme can capture the trend of compression ratio well and make reasonable estimations under most error bounds. Nevertheless, for *Blast2_p*, the scheme shows a substantial departure from the real compression ratios. This is due to the unique data distribution of *Blast2_p*, in which data points mostly center around two values. As a result, the quantization factor deviates from the exact Gaussian by a large margin (Figure 2.11), and this, in turn, affects the accuracy of *NodeCount* estimation.

For *Eddy*, *Sedov* and *Yf17_p*, although the relative estimation errors shown in Figure 2.14 are high, the absolute estimation errors are not. For example, for *Eddy*, the estimation errors at error bound 10^{-5} and 10^{-3} are 0.45 and 1.74, respectively. None of the absolute estimation errors exceeds an order of magnitude difference - the goal of compression estimation is to capture the trend of compression ratio, as opposed to predicting the precise value.

Furthermore, it is observed that for loose error bounds, *e.g.*, 10^{-2} or 10^{-1}, the estimation is less accurate (except for *Astro*). The reduced accuracy is caused by the following: 1) our approximation of using uniform distribution to model newly hit points, and 2) the amount of newly hit points becomes very small as the error bound increases, thus making it hard to statistically capture their characteristics. We comment that loose error bounds are often not preferred in scientific productions due to the significant information loss, and they are listed here only for comparisons.

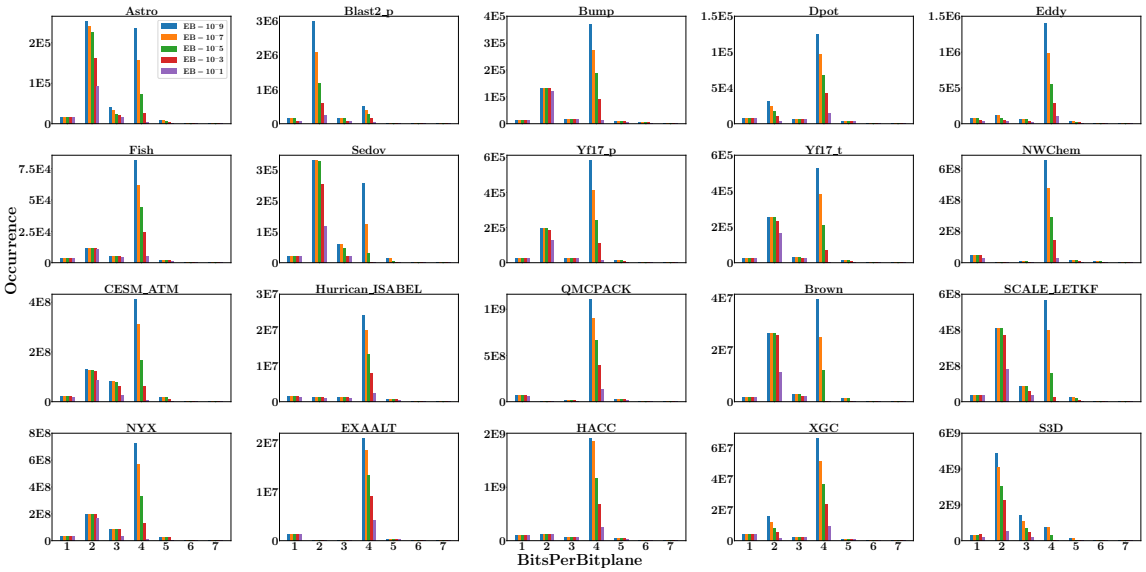


Figure 2.12 Distribution of *BitsPerBitplane* over error bound.

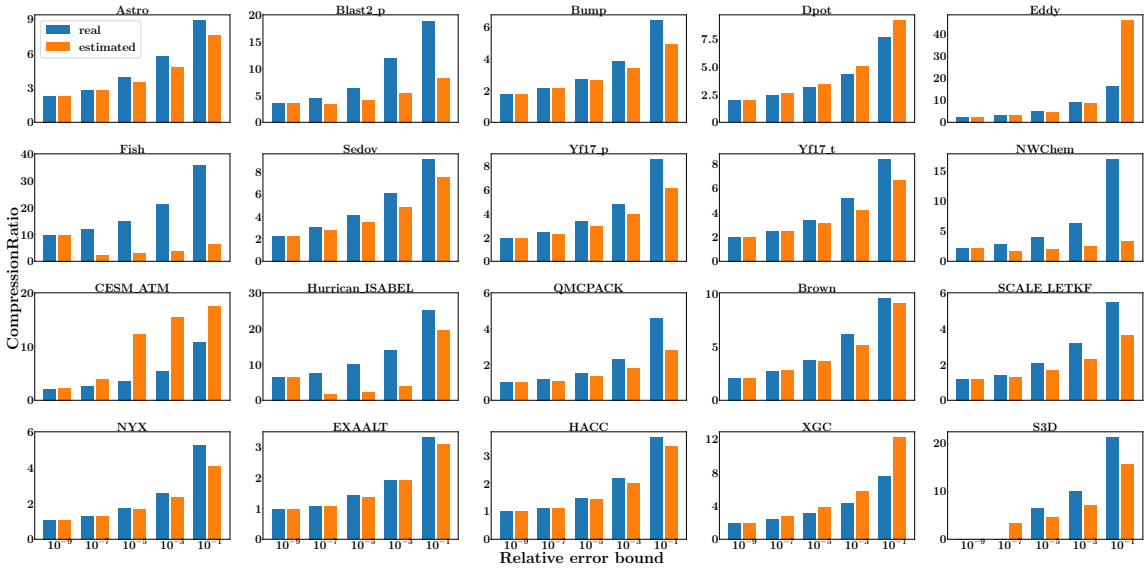


Figure 2.13 ZFP compression estimation.

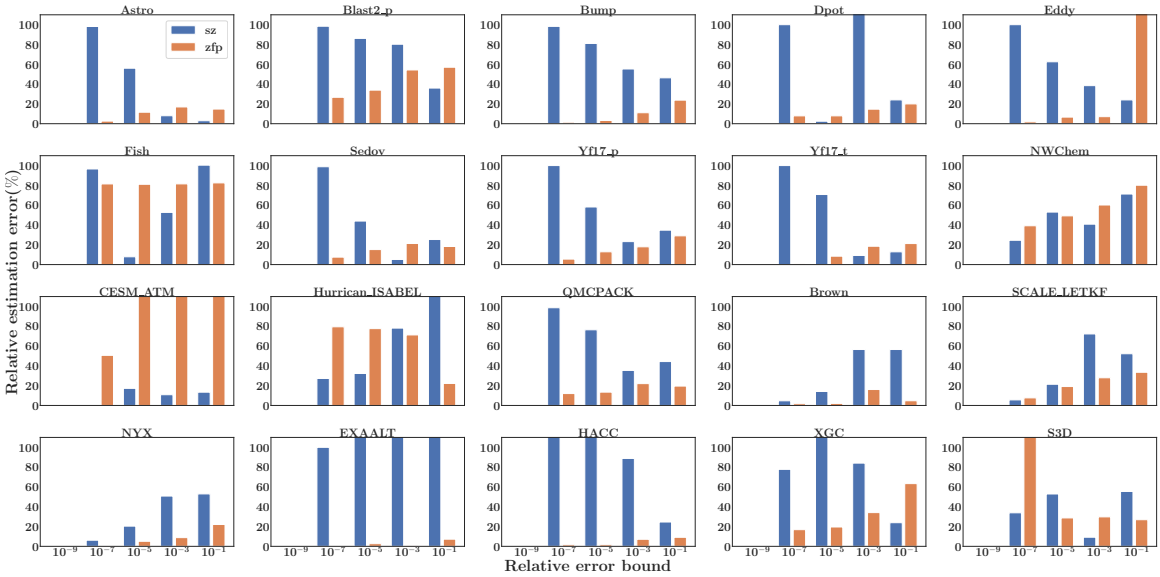


Figure 2.14 Compression estimation error (SZ and ZFP).

2.3.2 ZFP compression estimation

We first evaluate the estimation of *BitsPerBitplane*. In Figure 2.12, we plot the distributions of *BitsPerBitplane* under error bounds from 10^{-9} to 10^{-1}. It can be seen that the distribution of *BitsPerBitplane* maintains a similar shape across error bounds for all datasets.

The results of ZFP compression estimation are shown in Figure 2.13. Overall, we can capture the trend of compression ratio well across error bounds. For *Yf17_p*, *Yf17_t* and *Sedov*, our proposed model overestimates the output size which results in lower compression ratios. This is because *BitsPerBitplane* under $EB_{base}(10^{-9})$ is always larger than others (see in Figure 2.12) due to tighter error tolerance. The relative estimation error of *CompressionRatio* for ZFP is shown in Figure 2.14. As compared to SZ (Figure 2.14), the estimation error of *CompressionRatio* for ZFP is significantly lower than SZ. For SZ, the quantization factor approximation involves significant simplifications of Gaussian tails, while ZFP does not have this problem. In addition, the estimation error generally increases as the error bound deviates from the EB_{base} (i.e., 10^{-9} in our runs). The largest relative estimation error observed is below 35% (for *Fish* and *Yf17_t* data at error bound 10^{-1}) while we observe that the absolute estimation error is 4.75 and 4.23, respectively.

2.4 Conclusions

Motivated by the insufficient understanding of lossy compressors, this chapter thoroughly studies the mechanisms of two lossy compressors, SZ and ZFP. In particular, we examine how the error bound influences the compression ratio and identify the key factors that affect the outcome of compression. This work develops modeling techniques to predict compression ratios based on the estimation of key compression metrics across a set of error bounds. For SZ, we focus on the modeling and estimation of *HitRatio* and *NodeCount*; whereas for ZFP we capture the trend of *MaxPrec* and *BitsPerBitplane*. We evaluate the modeling and estimation schemes on real HPC datasets. The results show that our estimation scheme achieves good accuracy on the compression ratios of SZ and ZFP for all datasets across error bounds. Our work is beneficial to domain scientists for choosing error bounds when compressing large datasets on HPC systems.

CHAPTER 3

ZPERF: A STATISTICAL GRAY-BOX APPROACH TO PERFORMANCE MODELING FOR LOSSY COMPRESSION

3.1 Motivation

In the past, compression ratio modeling has been attempted mostly using a black-box approach [54,63,65,66]. 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. Yet, 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 [69].

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. Regardless, the sampling-based approach does require all

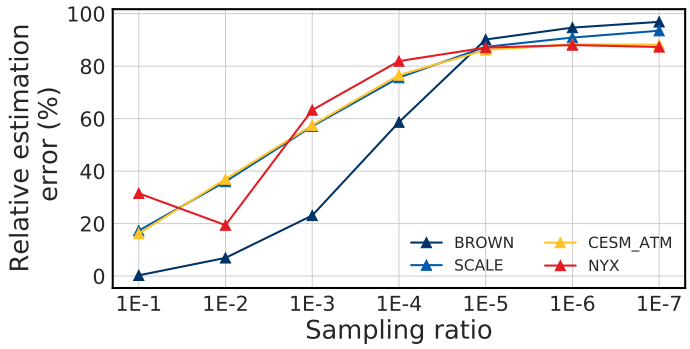


Figure 3.1 Estimation error of compression ratio vs. sampling ratio for SZ.

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. Figure. 3.1 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 chapter aims to develop a gray-box approach where key components within a compressor are modeled to allow for performance estimation and extrapolation.

3.2 Gray-box Compression Modeling

In this section, we first discuss the general methodologies of 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 as follows, we list the notations in Table 3.1.

Table 3.1 A List of Symbols for Gray-box Performance Modeling

Symbols	Description
General	
U	Input dataset size (bytes).
G	Compression ratio.
F	Compression throughput (bytes/sec).
N	Number of elements of input data.
B	Number of data blocks.
R	Value range of input data.
E	Lossy error bound.
SZ compression metrics	
ρ	Number of quantization levels for encoding.
η	Curve-fitting efficiency.
n	Number of Huffman tree nodes.
J	Huffman tree structure size (bytes).
K	Huffman encoding size (bytes).
M	Curve-missed data encoding size (bytes).
P	Curve-fitting and quantization time (secs).
M	Curve-missed data encoding time (secs).
C	Huffman tree construction time (secs).
H	Huffman encoding time (secs).
ZFP compression metrics	
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.
P	Exponent values size (bytes).
Q	Embedded encoding size (bytes).
V	Exponent values calculation time (secs).
X	Mantissas values conversion time (secs).
T	Non-orthogonal transform time (secs).
R	Transform coefficients reordering time (secs).
E	Embedded encoding time (secs).

3.2.1 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.

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 [21, 44, 62], FPZIP [50], and ISABELA [41] are three typical examples of 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 the 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 bit planes. Typical examples of transform-based lossy compression techniques include ZFP [47] and MGARD [8]. 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 Subsection 3.2.3. 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.

We now introduce the general procedures of zPerf for prediction-based and transform-based lossy compression, as shown in Figure 3.2.

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,

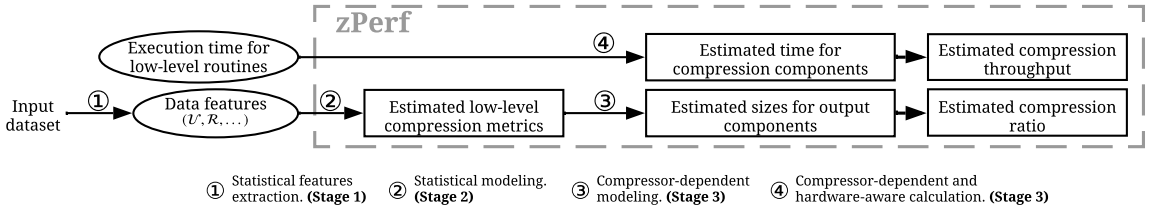
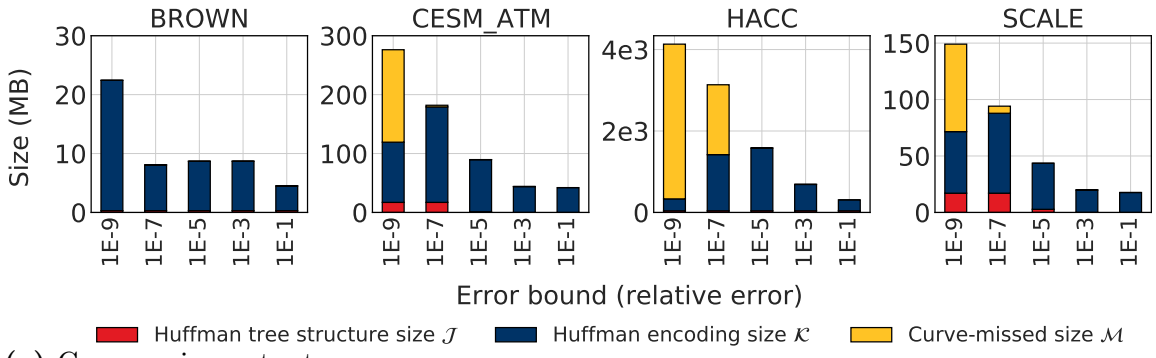


Figure 3.2 Overall gray-box methodology for lossy compression modeling.

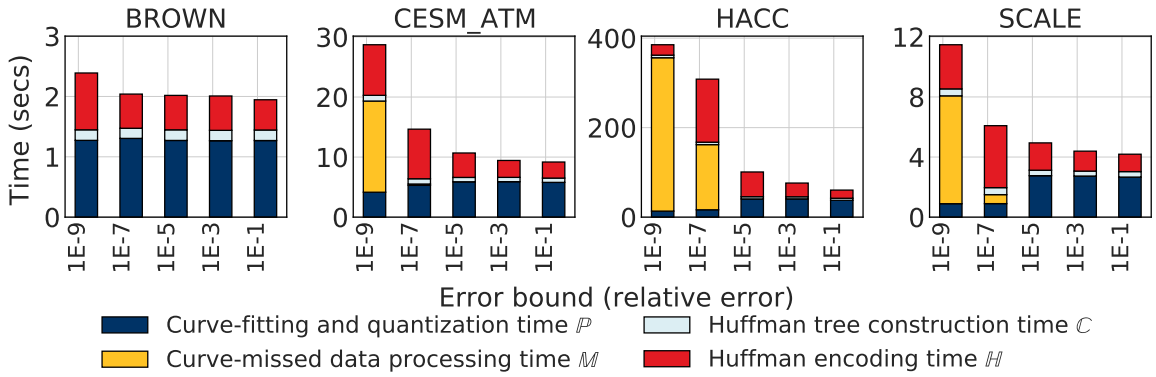
we comment that such condensed features can be made available by modern data management systems, such as ADIOS [52] which maintains an expandable set of data attributes that can be specified and computed when data is generated with the goal of simplifying the post-processing, such as query and filtering.

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 bit planes.

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 includes the time to perform orthogonal transform and the time to encode transform coefficients.



(a) Compression output.



(b) Compression time.

Figure 3.3 SZ compression performance breakdown.

3.2.2 SZ modeling: a case study of prediction-based compression

SZ compresses input data by first predicting with Lorenzo or regression predictor, then encoding the quantization codes with Huffman encoding. As shown in Figure 3.3, 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 depend 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

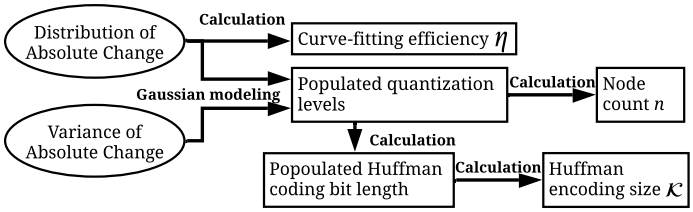


Figure 3.4 Key steps in SZ modeling.

our model, as shown in Figure 3.4. 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 the 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 the 2D Lorenzo predictor, the prediction for i-th data point is expanded to use its three immediate neighbors that have 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 [54]. Through studying the implementation of SZ, it is found that the mean of the Gaussian model is located at $\frac{\rho}{2}$ and the variance is $\text{Var}(\Delta\mathcal{D})/(\mathcal{E}\mathcal{R})^2$ where $\text{Var}(\Delta\mathcal{D})$ 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.

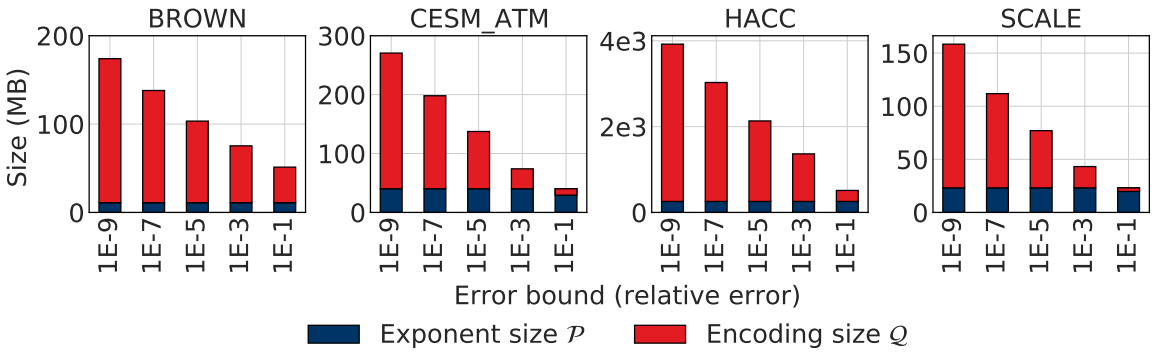
Curve-missed size $\mathcal{M}$ is the size of the curve-missed data points. These data points are first subtracted by the median of the data values and then truncated to discard the insignificant mantissa bits subject to the error bound. Based upon SZ's implementation, the number of remaining significant mantissa bits q_j at relative error bound $\mathcal{E}_j$ can be calculated as $q_j = \text{Exp}(\mathcal{R}/2) - \text{Exp}(\mathcal{E}_j\mathcal{R})$, where $\text{Exp}(\cdot)$ denotes the exponent part of a floating-point value. In particular, $\text{Exp}(\mathcal{R}/2)$ is the exponent value of the data radius and $\text{Exp}(\mathcal{E}_j\mathcal{R})$ 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 zeros 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 dataset. 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. 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 maps them into ρ

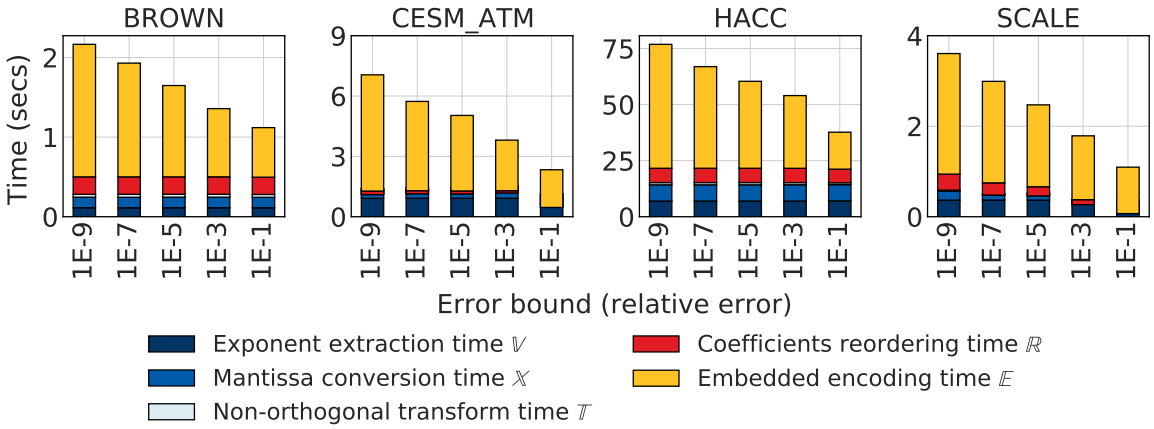
quantization levels and then adjusts the data prediction due to the quantization. Since SZ performs the same operation for each curve-hit data point, the time complexity of $\mathbb{P}$ is $\mathcal{O}(\mathcal{N}\eta)$. The Huffman tree construction time $\mathbb{C}$ is the time to construct the Huffman 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, which 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 assigning 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 operations for curve-missed data points. 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 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 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,



(a) Compression output.



(b) Compression time.

Figure 3.5 ZFP compression performance breakdown.

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 $\mathcal{N}(1-\eta)$.

3.2.3 ZFP modeling: a case study of transform-based compression

ZFP compresses input data by first performing a block-based orthogonal transform to remove redundancies, then encoding the transform coefficients a bit plane at a time to control the output bit rate. As shown in Figure 3.5a, 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 Figure 3.5b. Fundamentally, two

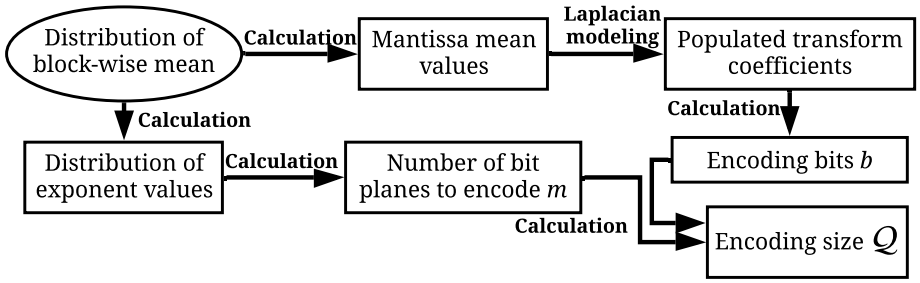


Figure 3.6 Key steps in ZFP modeling.

low-level compression metrics 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. Figure 3.6 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 $m_i = \epsilon_i - \log_2 \mathcal{E} + 2(dim + 1)$ [48], where ϵ_i is the maximum base-2 exponent of the block and dim is the number of 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 the 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

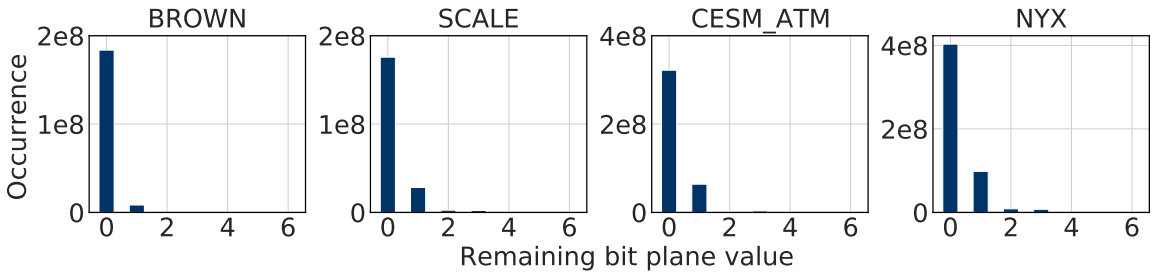


Figure 3.7 Histogram of remaining values of bit planes after encoding the significant bits under the relative error of 1E-6.

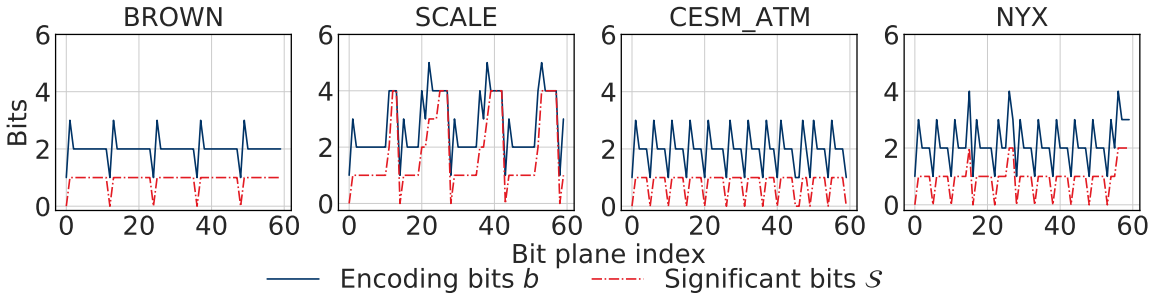


Figure 3.8 Comparison between the number of significant bits $\mathcal{S}$ and encoding bits b under the relative error of 1E-6.

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 Figure 3.7. 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 Figure 3.8. 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. To capture the bit plane values and further $\mathcal{S}_{ij}$, we next discuss the distribution of transform coefficients.

It was shown in previous work [42] 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

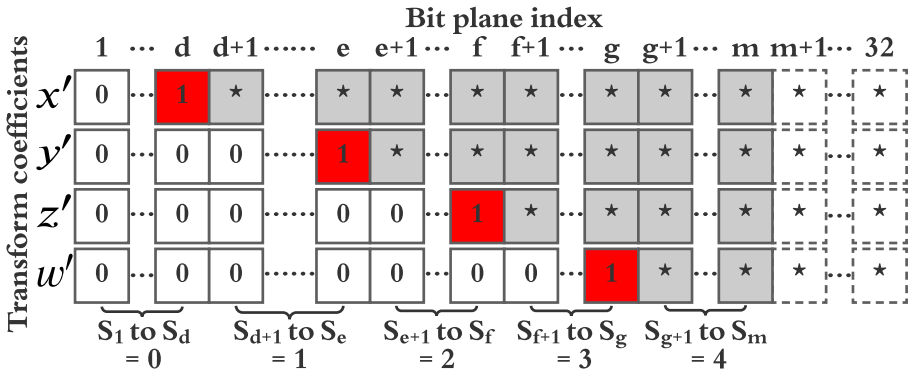


Figure 3.9 Schematic of 1D embedded encoding in ZFP.

maximum likelihood estimation [23]. 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 (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 [22]. 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 $x' = \frac{1}{4}(\dot{x} + \dot{y} + \dot{z} + \dot{w})$. It is clear that the estimation of transform coefficients distribution ultimately comes down to the mean of the 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 Figure 3.9, 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., $\mathcal{Q} =$

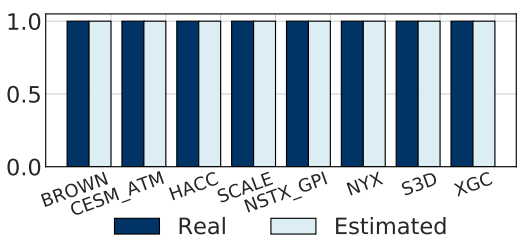
$\left\lceil \frac{1}{8} \sum_{i=1}^{\mathcal{B}} \sum_{j=1}^{m_i} b_{ij} \right\rceil$ where $\mathcal{B}$ is the number of data blocks. Exponent size $\mathcal{P}$ is the byte size to store ϵ for all data blocks. According to the IEEE-754 format, ϵ takes 11 bits for double precision floating point data, and therefore $\mathcal{P}$ can be calculated as $\mathcal{P} = \left\lceil \frac{11}{8}\mathcal{B} \right\rceil$. 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 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.

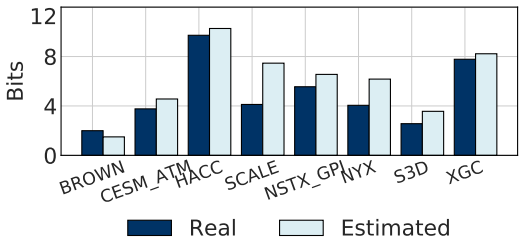
ZFP performance. Given the compression metrics and the measurements of the low-level compression routines, we have $\mathcal{G}_{zfp} = \frac{\mathcal{U}}{\mathcal{P}+\mathcal{Q}}$ for the compression ratio, and $\mathcal{F}_{zfp} = \frac{\mathcal{U}}{\mathcal{B}(r_1+r_2+r_3+r_4)+\mathbb{E}}$ for the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

3.3 Evaluation

In this section, we evaluate zPerf across eight scientific datasets (described in TABLE 1.1) from the Scientific Dataset Reduction Benchmark [25]. Specifically, we show the results of four datasets with 1D compression experiments and four



(a) Curve-fitting efficiency η.



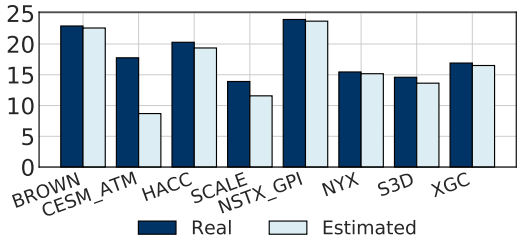
(b) Huffman coding bit length.

Figure 3.10 SZ low-level metrics estimation.

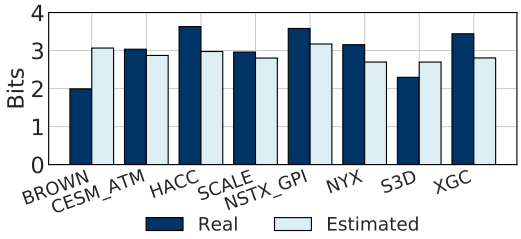
with 2D compression experiments. We conduct the compression experiments using SZ (Version 2.1.7) and ZFP (Version 0.5.5) on two leading HPC systems, Cori [3] at National Energy Research Scientific Computing Center, and Summit [4] at Oak Ridge National Laboratory, to test the compression throughput estimation 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 the 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. For the rest of the section, we show the modeling results of the low-level and high-level compression metrics in Subsections 3.3.1 and 3.3.2, respectively. We then compare zPerf against the sampling approach in terms of estimation accuracy in Subsection 3.3.3.

3.3.1 Low-level compression metrics

For SZ, the estimated curve-fitting efficiency η and the average Huffman coding bit length are shown in Figure 3.10. For η, it can be estimated accurately mainly because adjacent data points are observed to be smooth, i.e., $\hat{\mathcal{D}}_{i-1} \approx \mathcal{D}_{i-1}$. Thus, 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



(a) Num of bitplanes for a block.

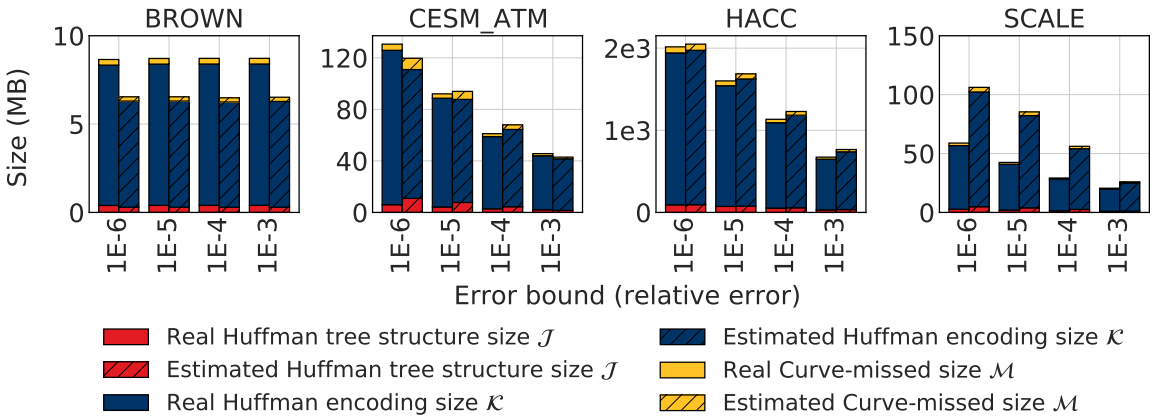


(b) Num of bits for a bitplane.

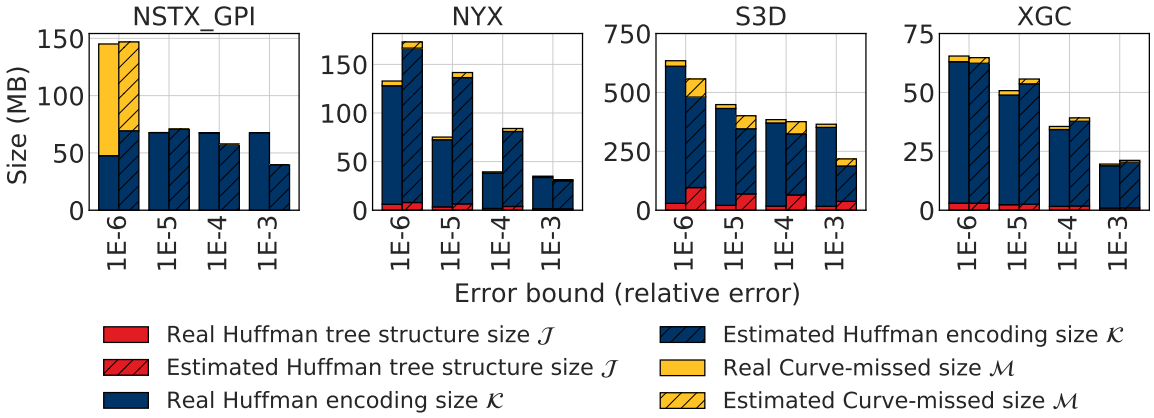
Figure 3.11 ZFP low-level metrics estimation.

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* demonstrate lower data smoothness, such that the number of quantization levels needed for Huffman encoding is hard to capture. Consequently, the estimated Huffman coding bit lengths based on the populated quantization levels deviate from the 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 Figure 3.11. 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 Figure 3.9. 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 of 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. Hence, the error is observed to be insignificant.



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

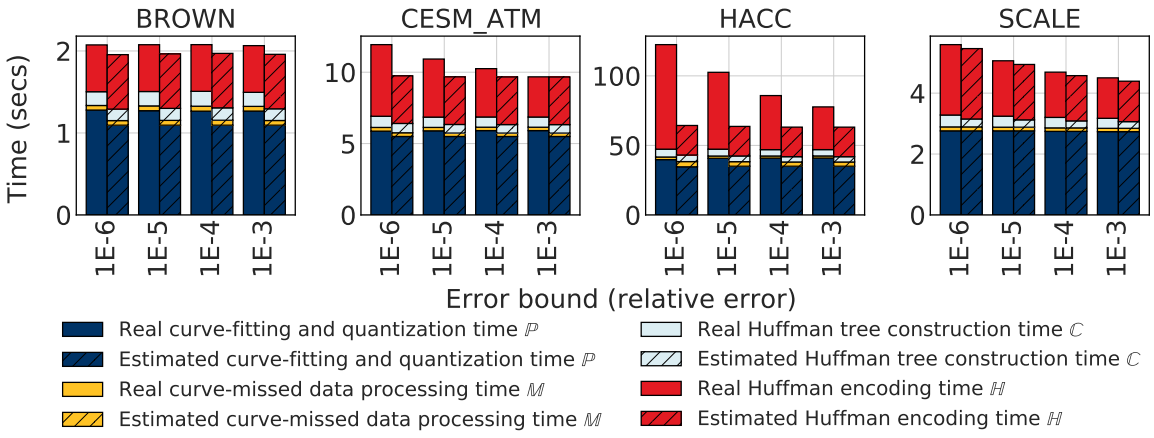


(b) Compression output components (2D compression).

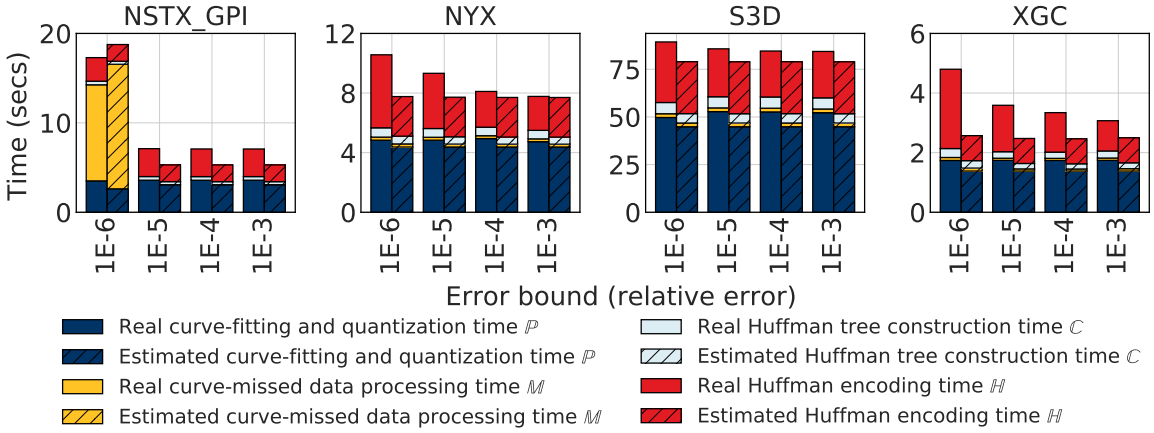
Figure 3.12 SZ compression size estimation.

3.3.2 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 Figure 3.12. 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 Subsection 3.3.1.



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



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

Figure 3.13 SZ compression time estimation on Cori.

As mentioned in Section 3.2, to estimate the system-dependent compression time, we need to measure the timings of the low-level compression routines r_i. Table 3.2 shows 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 ten 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.

Table 3.2 Execution Time of Low-level Routines

(a) SZ on Cori

Routine	Exec. Time
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

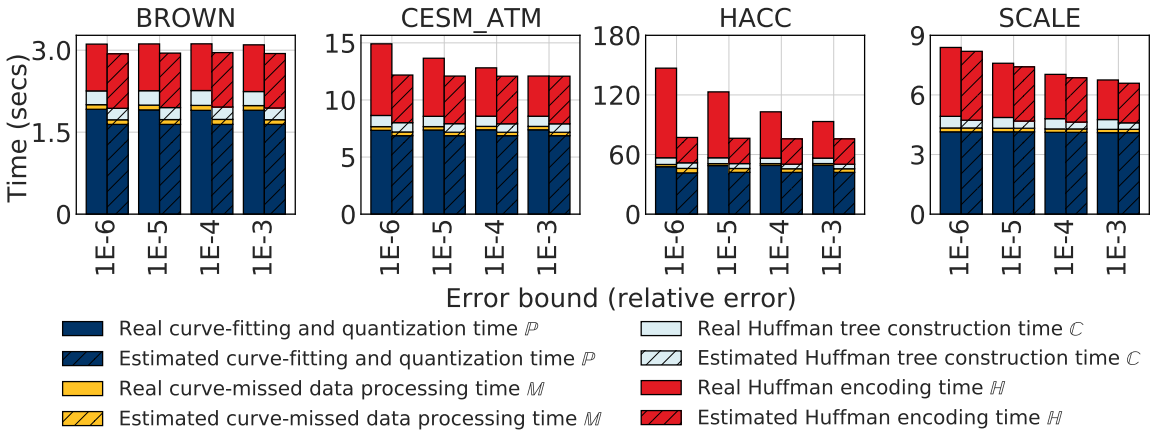
Routine	Exec. Time
r_1	62.63
r_2	452.91
r_3	7.46
r_4	5.71
r_5	7.36
r_6	5.32

(c) ZFP on Cori

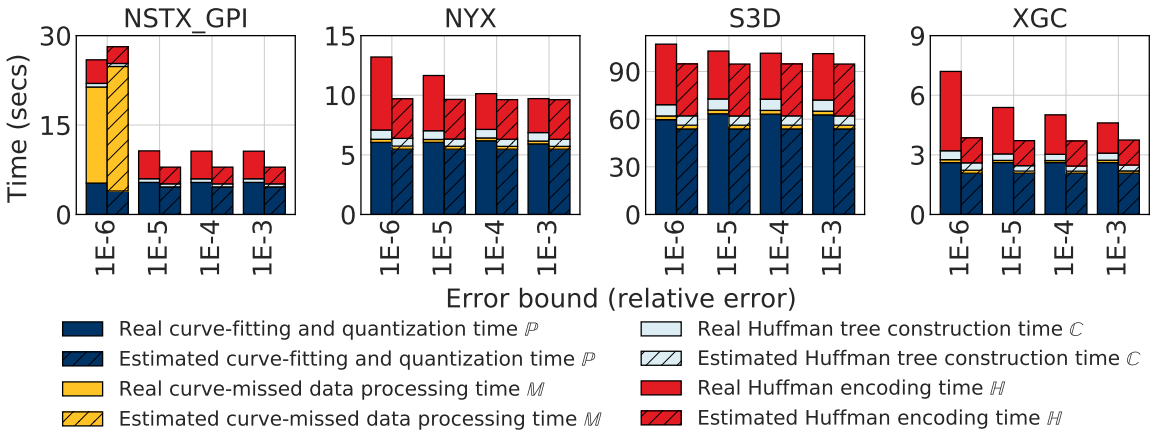
Routine	Exec. Time
r_1	11.8
r_2	8.19
r_3	2.39
r_4	17.19
r_5	2.72

(d) ZFP on Summit

Routine	Exec. Time
r_1	20.8
r_2	16.19
r_3	4.39
r_4	26.19
r_5	3.72



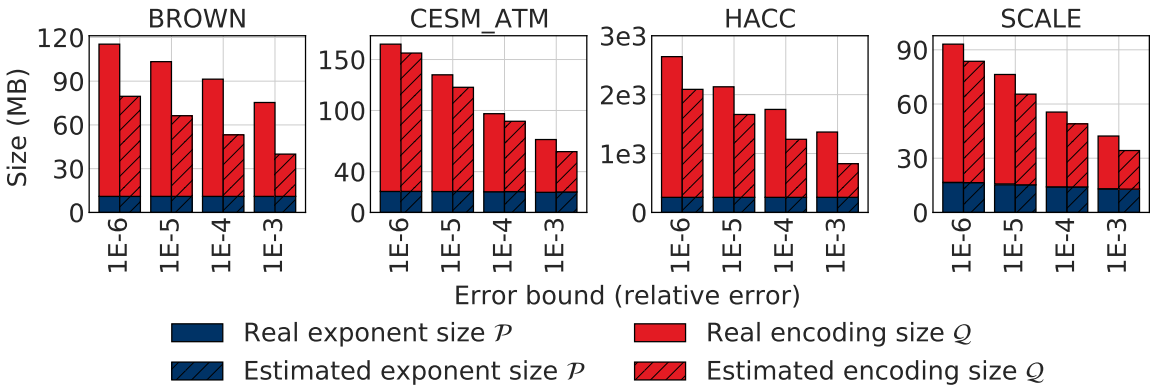
(a) Compression time components on Summit (1D Compression).



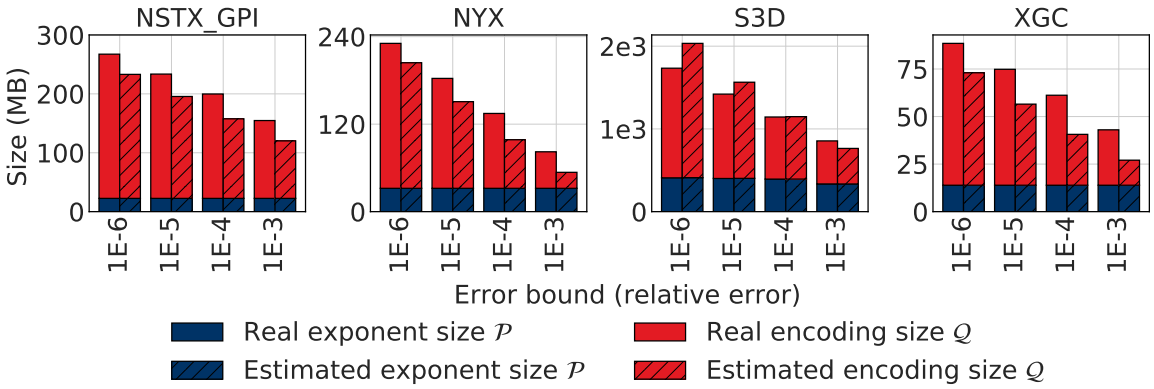
(b) Compression time components on Summit (2D Compression).

Figure 3.14 SZ compression time estimation on Summit.

The estimation for compression time components for SZ is shown in Figures 3.13 and 3.14, where the compression time for Cori is shown in Figure 3.13, and the compression time for Summit is shown in Figure 3.14. Since $\mathbb{P}$ is linear to η, its estimation is fairly accurate. The Huffman tree construction time $\mathbb{C}$ can be calculated as $\mathcal{N}r_3+n(r_4+r_5)$.



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

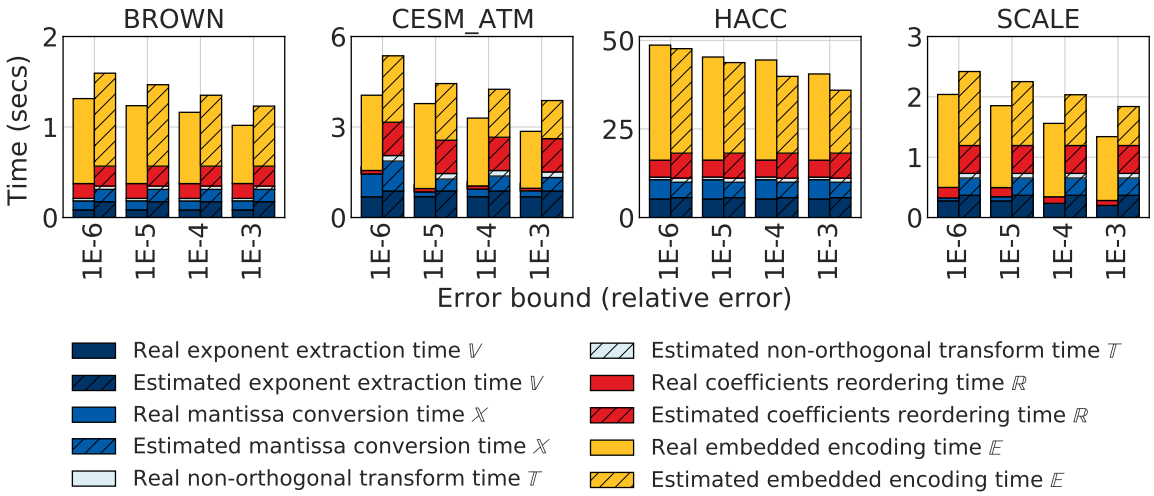


(b) Compression output components (2D compression).

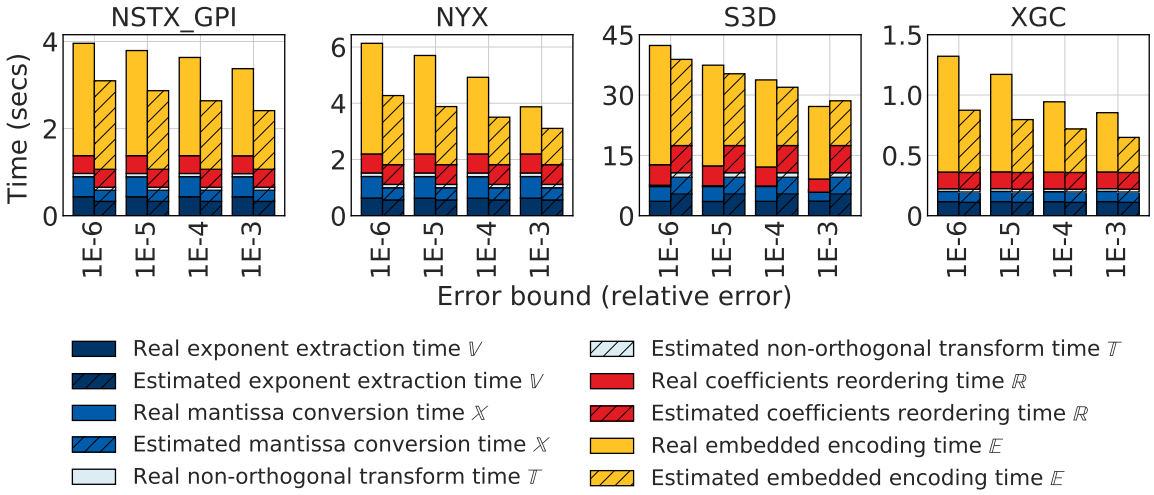
Figure 3.15 ZFP compression size estimation.

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. The Huffman encoding time $\mathbb{H}$ 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 Figure 3.15 for 1D and 2D compression. In particular, $\mathcal{P}$ can be well estimated since it is linearly related to the number of blocks $\mathcal{B}$. $\mathcal{Q}$ 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 Subsection 3.3.1. Table 3.2 lists the measured execution time of low-level routines in ZFP on Cori and Summit, respectively. Specifically for r_5, we manually set the bit plane values so that the corresponding encoding bits for each bit plane are 1, 2, 3 and



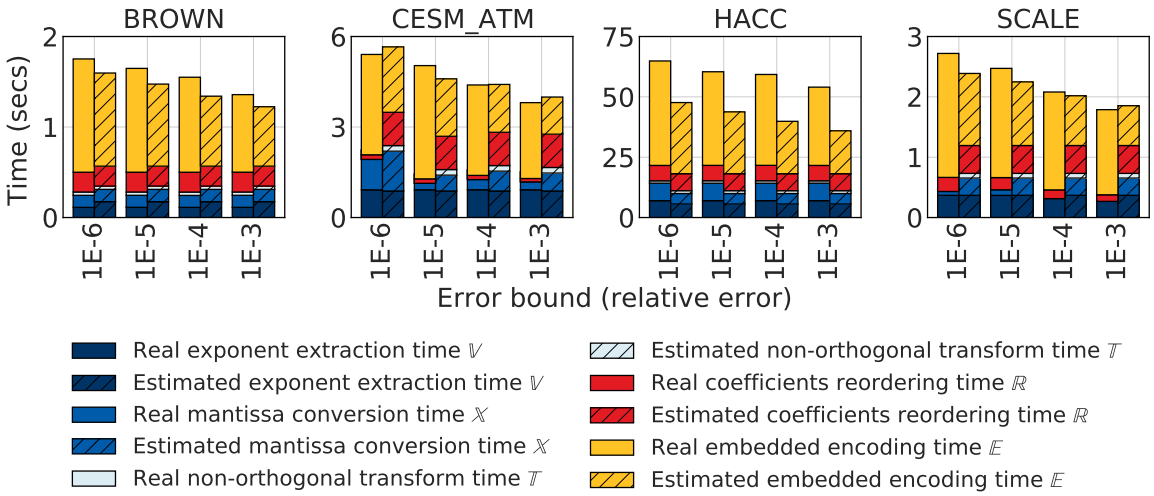
(a) Compression time components on Cori (1D Compression).



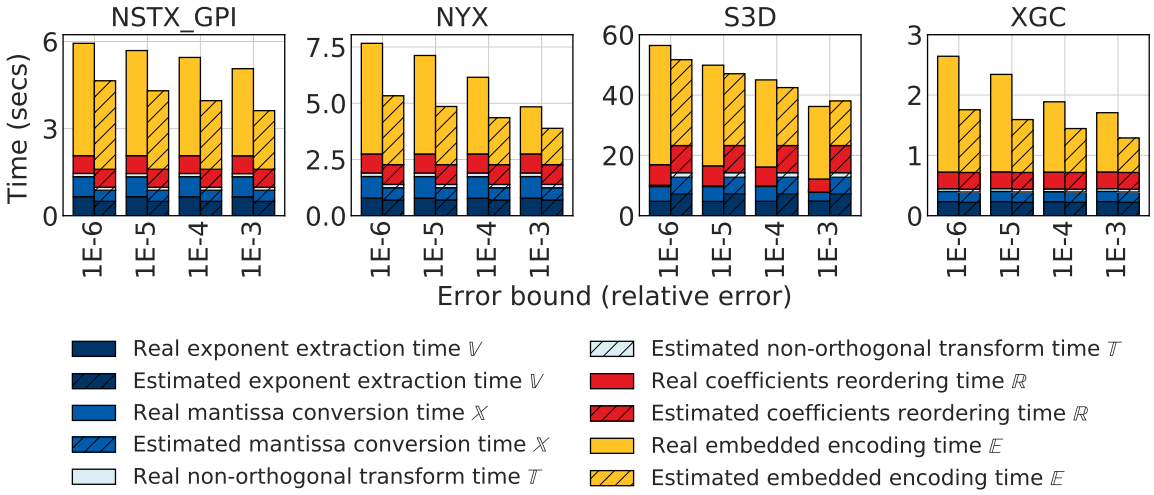
(b) Compression time components on Cori (2D Compression).

Figure 3.16 ZFP compression time estimation on Cori.

4. For bit planes with different numbers 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 Figures 3.16 and 3.17. As we mentioned in Subsection 3.2.3, $\mathbb{V}$, $\mathbb{X}$, $\mathbb{T}$ 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



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



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

Figure 3.17 ZFP compression time estimation on Summit.

the estimation error of $\mathbb{E}$ mainly comes from the estimation of encoding bits for each bit plane.

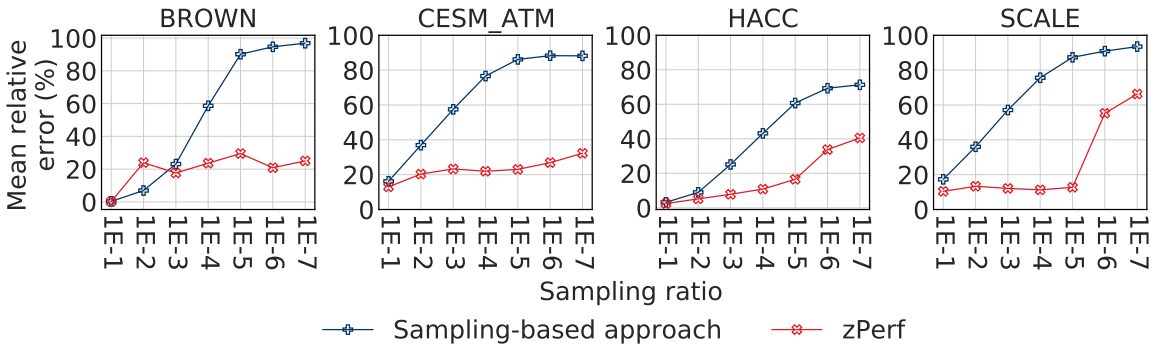
Overall, zPerf effectively captures the trend of compression performance, despite the statistical approximation (*e.g.*, the Gaussian modeling for SZ and Laplace modeling for ZFP) in estimating the compression metrics. Generally, zPerf achieves better estimation results for ZFP than for SZ. The reason 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.

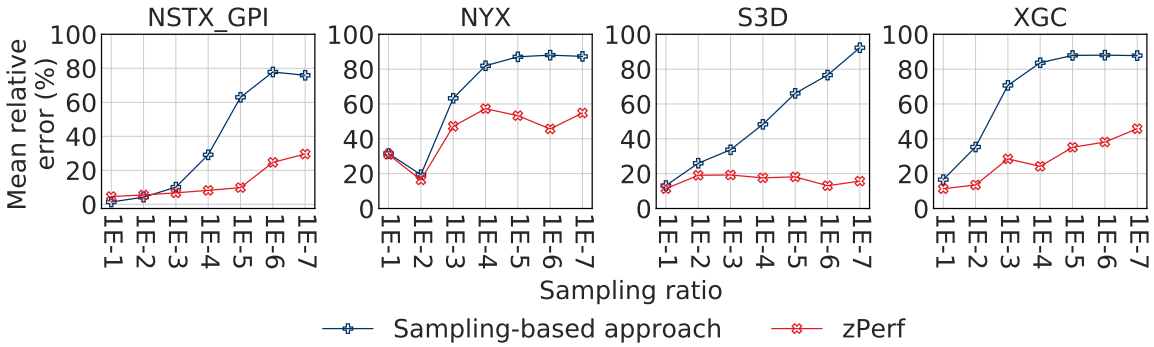
3.3.3 zPerf vs. sampling-based approach

We next compare zPerf with the sampling approach [54] 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 [54] pointed out that for the estimation of compression ratios, the sampling approach offers a biased estimation for compressors without bounded localities, such as SZ, and an unbiased estimation for compressors with bounded localities, such as ZFP. Therefore, we anticipate that the sampling approach works well for ZFP, but not SZ.

In this section, we vary the population and sampling ratios from 1E-1 to 1E-7 and compare the MRE of zPerf and the sampling approach. In Figures 3.18 and 3.19, we measure the MRE of compression ratio and compression throughput estimation for SZ, respectively. In Figures 3.20 and 3.21, we show the MRE of compression ratio and compression throughput estimation for ZFP. 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



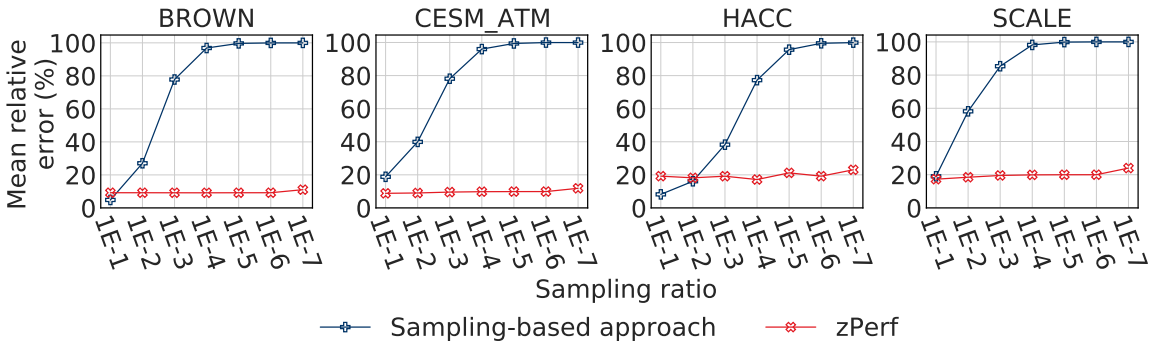
(a) Compression ratio estimation (1D Compression).



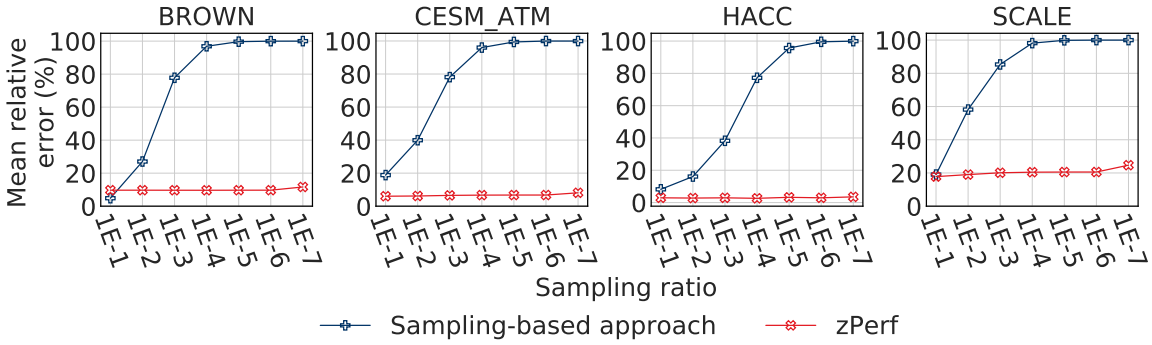
(b) Compression ratio estimation (2D Compression).

Figure 3.18 zPerf estimation error compared to the sampling approach for SZ compression ratio under sampling ratios from 1E-1 to 1E-7.

the sampling ratio decreases. Note that due to the bounded locality of ZFP, the sampling approach yields a low error for compression ratio (Figure 3.20). For SZ, zPerf generally outperforms the sampling approach in estimating 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.



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

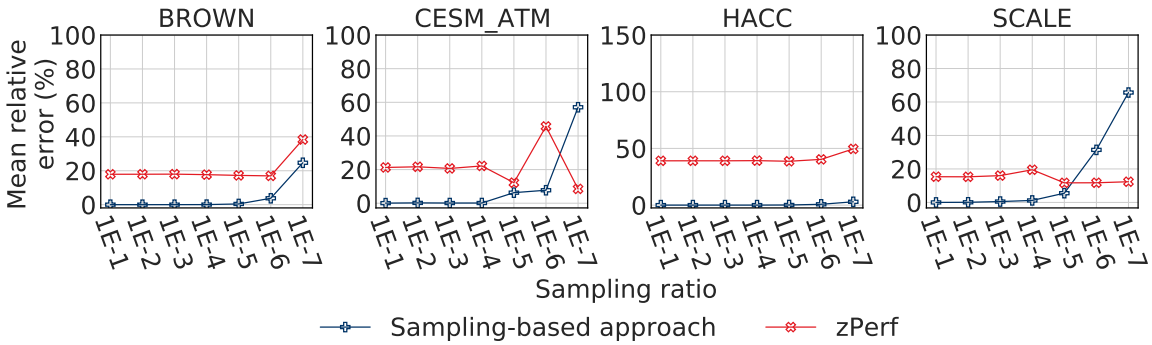


(b) Compression throughput estimation on Summit (1D compression).

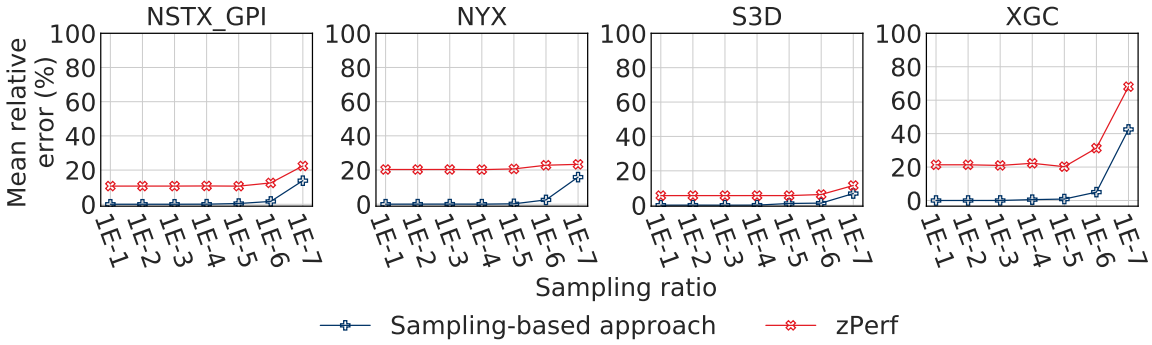
Figure 3.19 zPerf estimation error compared to the sampling approach for SZ compression throughput under sampling ratios from 1E-1 to 1E-7.

For ZFP, the MRE of the compression throughput using the sampling approach increases with decreasing 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. Nonetheless, 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 the sampling approach, as shown in Figure 3.22. It can be shown that for SZ (Figure 3.22a), zPerf yields a lower overhead than the sampling approach after the 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 a longer



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



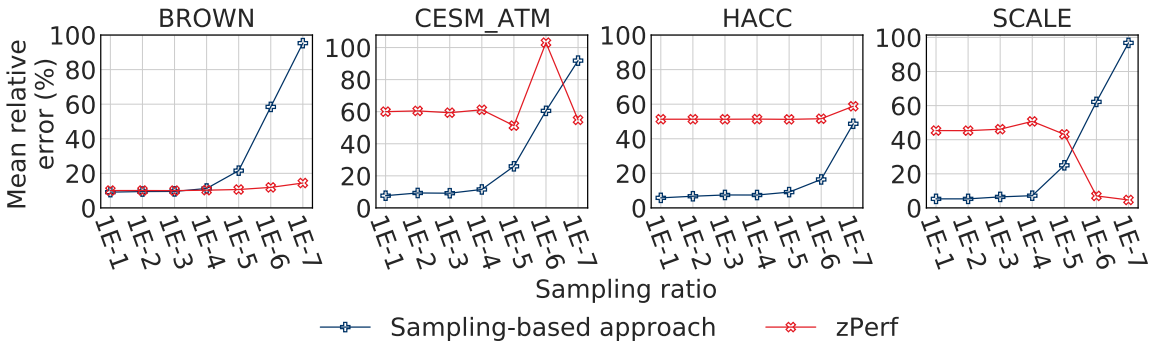
(b) Compression ratio estimation (2D Compression).

Figure 3.20 zPerf estimation error compared to the sampling approach for ZFP compression ratio under sampling ratios from 1E-1 to 1E-7.

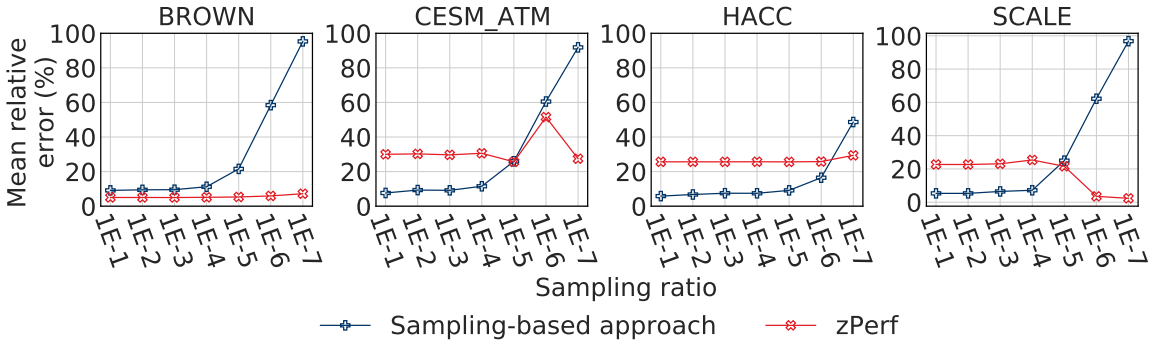
running time as compared to the sampling approach. Nevertheless, it is still beneficial given that zPerf can provide better estimation at low population ratios.

3.3.4 zPerf vs. state-of-the-art

Tao *et al.* [63] employ a rate-distortion estimation method for bit-rate estimation. Compared to zPerf, this work neither explored the design space of lossy compression nor attempted to model 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 of 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



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

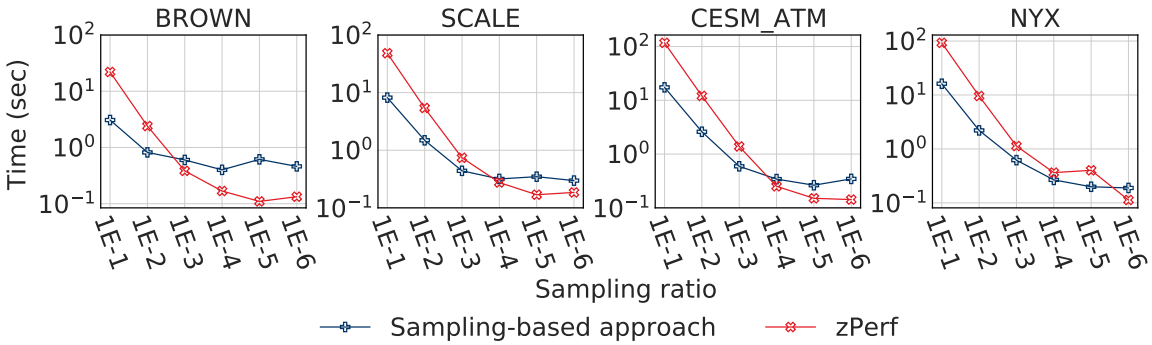


(b) Compression throughput estimation on Summit (1D compression).

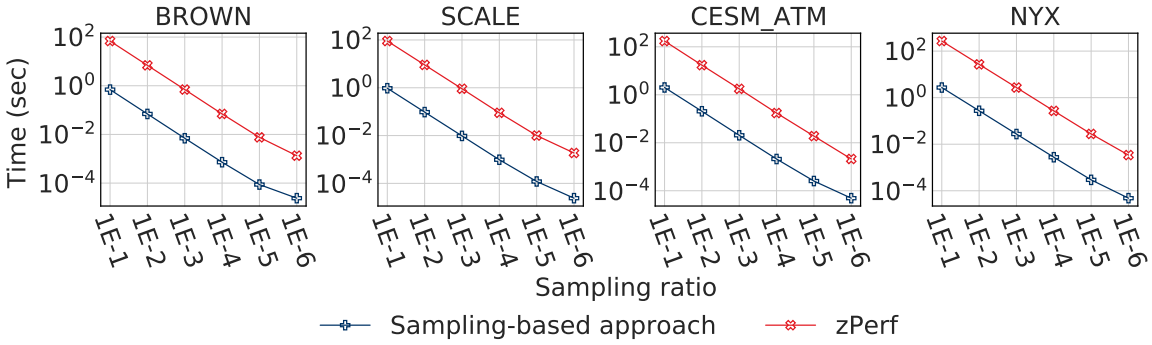
Figure 3.21 zPerf estimation error compared to the sampling approach for ZFP compression throughput under sampling ratios from 1E-1 to 1E-7.

important for the modeling of compression for large data cost-effectively. 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 (with the added capability of exploring new algorithms in a compressor).

Zhao *et al.* [69] 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. Yet, they did not present quantitative results across datasets and the work focused only on SZ. Jin *et al.* [35] adopted a modularized approach for the compression ratio and quality estimation for prediction-based lossy



(a) SZ compression performance estimation overhead.



(b) ZFP compression performance estimation overhead.

Figure 3.22 Running time overhead of zPerf compared to the sampling approach under sampling ratios from 1E-1 to 1E-6.

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 is 19.1%, 18.2%, 18.8%, and 9.8%, respectively. Jin *et al.* [37] 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 visualization quality after compression. There is no quantitative evaluation presented for the modeling accuracy.

3.4 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 schemes on ZFP. Specifically, for the entropy encoding study, we replace the Huffman encoding in SZ with the ZFP lossless compression and the embedded encoding in ZFP with Huffman encoding. For the transform scheme study, we replace the customized non-orthogonal transform in ZFP with a 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 distribution of quantization levels is Gaussian [54]. 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 coeffi-

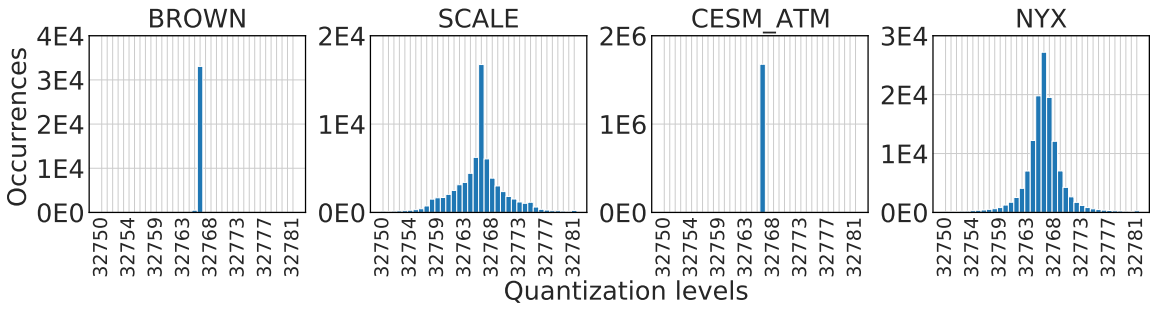


Figure 3.23 Distribution of SZ quantization levels.

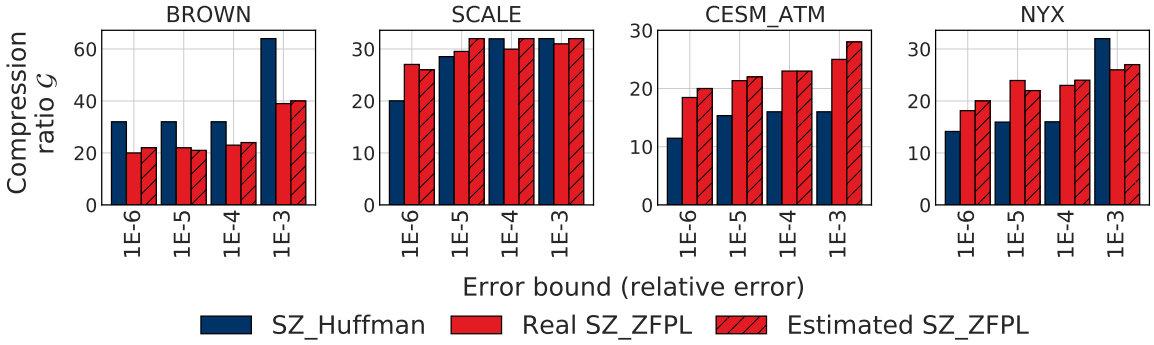


Figure 3.24 Compression ratio of *SZ_Huffman*, *SZ_ZFPL*.

cients. 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 Figure 3.23, it is reasonable to use ZFP lossless mode as the backend for compression.

As discussed in Subsection 3.2.3, 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. Thus, 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. Accordingly, following the approach we developed in Subsection 3.2.3, we model the transform coefficients with Laplacian distribution and calculate the number of significant bits for each block. In Figure 3.24, we show the measured and estimated compression ratio of SZ using the

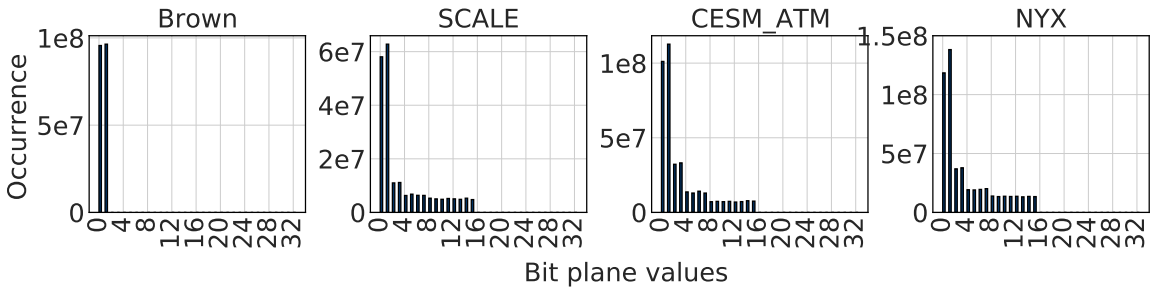


Figure 3.25 Distribution of ZFP transform coefficient bit plane.

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 in a Huffman tree with more branches and longer codes. When the error bound is loose, fewer quantization levels are used and thus shorter codes. However, *SZ_ZFPL* still needs to encode all bit-planes of the transformation coefficients, resulting in 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. Yet, 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 Figure 3.25, 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

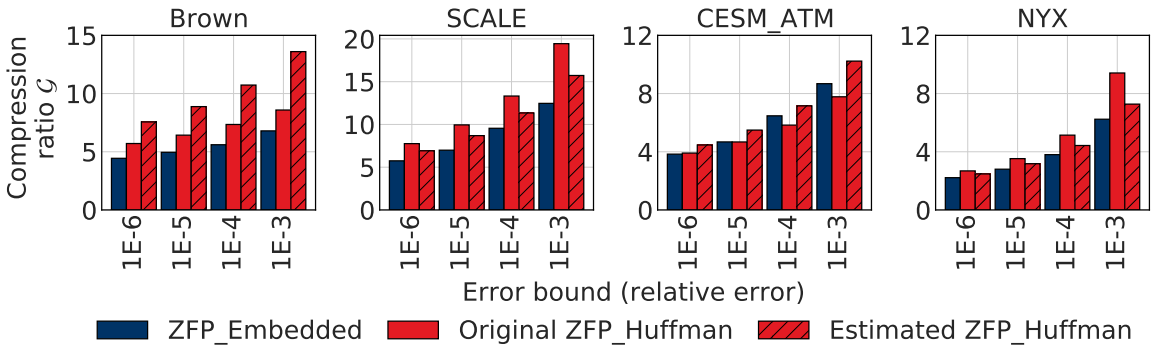


Figure 3.26 Compression ratio of *ZFP_Embedded*, *ZFP_Huffman*.

31, while the value of a 2D bit plane with 16 bits can range from 0 to 65,535. Hence, the bit plane can be suitable for Huffman encoding.

As discussed in Subsection 3.2.2, 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. Consequently, 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 Figure 3.26, 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*. 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 values using Huffman encoding. Figure 3.27 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 change 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.

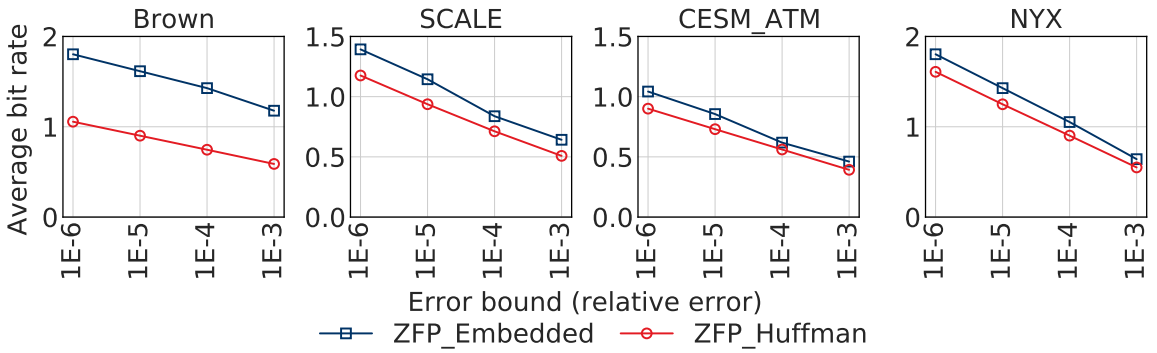


Figure 3.27 Bit-rate achieved by *ZFP_Embedded* and *ZFP_Huffman*.

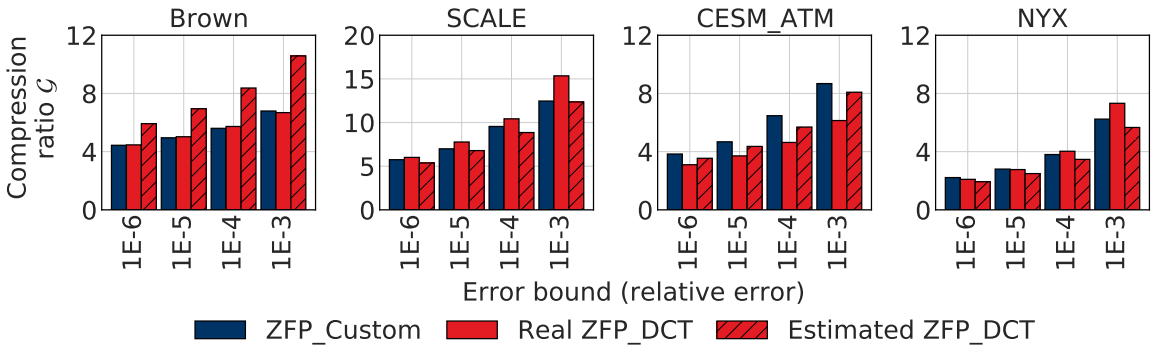


Figure 3.28 Compression ratio of *ZFP_Custom* and *ZFP_DCT*.

The advantage of the customized approach is the computational efficiency achieved by lift implementation and bit operations. Nevertheless, 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 Figure 3.28, 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 Figure 3.29. 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*

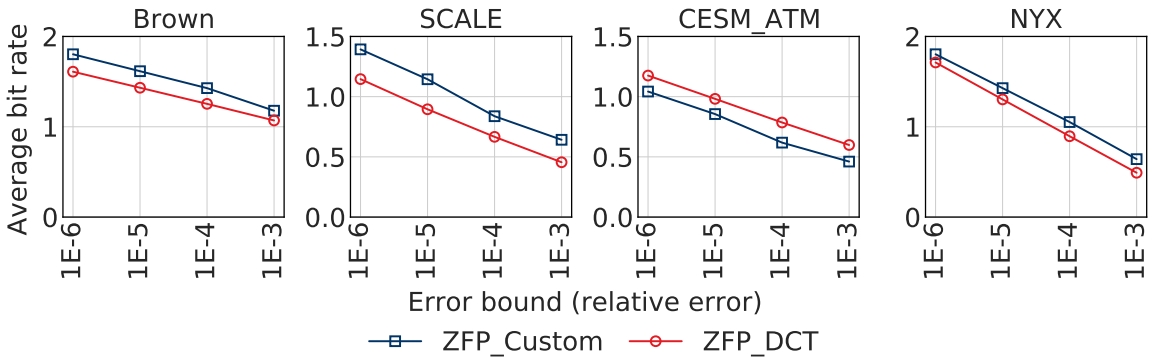


Figure 3.29 Bit-rate achieved by *ZFP_Custom* and *ZFP_DCT*.

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.

3.5 Conclusions

In this chapter, 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.

CHAPTER 4

IMPROVING PROGRESSIVE RETRIEVAL FOR HPC SCIENTIFIC DATA USING DEEP NEURAL NETWORK

4.1 Background and Motivation

4.1.1 Progressive data retrieval

Progressive data retrieval aims to address the problem of high computation cost at full data resolution, as well as data transmission when I/O bandwidth is constrained. The idea is to transmit a small part of the data each time, based on the user-prescribed accuracy requirements, so that the transmission can be finished efficiently. The general approach is first to decompose data into a multi-resolution hierarchy level of coefficients, such that, the highest coefficient level with the lowest resolution contains the most information. The design of such a hierarchy (*e.g.*, the number of levels and the resolution of each level) follows the storage hierarchy of HPC systems. For example, the highest level data, which is supposed to be frequently accessed, can be placed on the fastest storage tier (*e.g.*, NVMe); while the lowest level data should be placed on the slowest storage tier (HDD or tapes), given it will be least likely to be accessed.

During data transmission, one or more coefficient levels are transmitted based on the system I/O bandwidth and user requirement for data accuracy. Then the transmitted data levels can be recomposed to generate an approximation to the original data with reduced accuracy. Due to the complex I/O bandwidth status and various user requirements, it is vital for the retrieval framework to support fine-grained progressiveness. As such, bit-plane-based encoding schemes are commonly used in representing the coefficient level data, in order to support truncation on the bit stream. For example, Hoang *et al.* proposed to use binary encoding to represent the tree-based hierarchical data structure [32]. Another example is MGARD [8, 45],

a recently developed transform-based lossy compressor that adopts nega-binary encoding for representation, which will be detailed as follows.

4.1.2 MultiGrid Adaptive Reduction of Data (MGARD)

MGARD combines the features of lossy data compression with multi-level decomposition and progressive retrieval. During compression, a data decomposer first transforms original multi-dimensional data to multi-level coefficients using orthogonal L2 projection and interpolation. Then an interleaver linearizes the coefficient levels to 1D for precision encoding. The linearized coefficients are then encoded in the bit-plane fashion and compressed losslessly with ZSTD. In order to support error-bounded progressive retrieval, an error matrix is further collected to represent the error incurred on coefficient levels by partially quantizing bit-planes. The encoded coefficient levels, error matrix as well as other metadata are written to files and placed across the storage hierarchy.

During decompression, MGARD first estimates the number of bit-planes to be retrieved from each coefficient level with a maximum error estimator. Such an error estimator is able to predict the maximum data reconstruction error based on the number of bit-planes as well as the error matrix collected during compression. Thus MGARD recursively finds the minimum number of bit-planes to retrieve that satisfies the user-requested error control. Then a size interpreter calculates the retrieval size as well as the precision segments to fetch from the storage hierarchy. Then the bit-planes retrieved from various precision segments are recomposed to form the decompressed data.

4.1.3 Over-aggressive error control

The key capability of a progressive data retrieval framework is to identify the amount of data needed given users' prescribed error bounds. For example, MGARD [45] adopts

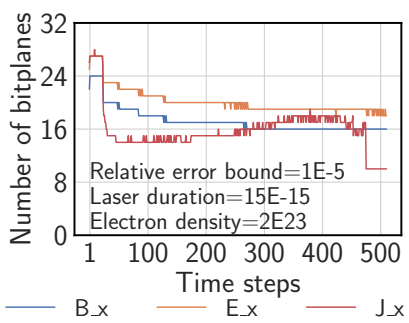
a theory-based error control scheme, that first computes the significance of each bit-plane that contributes towards the reconstructed data accuracy. Then the number of bit-planes that should be retrieved under a user-prescribed error bound can be deduced by progressively comparing the achieved accuracy and requested error bound for each bit-plane.

Nevertheless, the error control theory developed by the early works [8] estimates the maximum absolute error bound using absolute row sum which neglects the cancellation between positive and negative errors. Therefore, there is a significant gap between the user-requested error tolerance and the actual error achieved, which results in the number of bit-planes retrieved larger than what is required and a higher I/O performance overhead. As shown in Figure 1.2, the achieved error tolerance is constantly lower than requested by orders of magnitude. Correspondingly, the achieved I/O cost is significantly higher than requested, as shown in Figure 1.3.

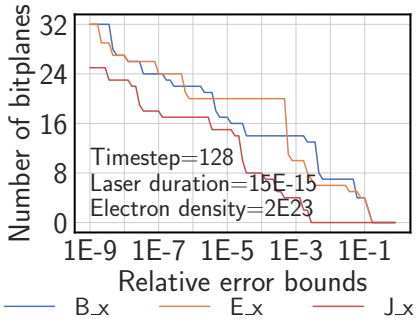
Motivation 1: *The theory-based error control is overly pessimistic and incurs high overheads for data retrieval. As such, a more precise error control method is needed to improve the I/O performance of data retrieval.*

4.1.4 High dimensionality of bit-plane retrieval

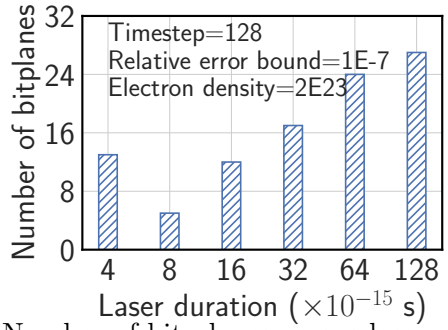
Given the various scenarios, the number of bit-planes to be retrieved is a multi-variant function that depends on simulation timesteps, error tolerances, as well as data characteristics, which are directly affected by simulation input parameters. In Figure 4.1, we demonstrate MGARD's number of bit-planes versus such variables. As demonstrated in Figure 4.1a, the number of bit-planes across timesteps demonstrates non-linear behaviors. In Figure 4.1b, we show that the number of bit-planes reduces while the tolerance loosens. In Figure 4.1c and Figure 4.1d, we show the number of bit-planes also manifests complicated behaviors with the change of *electron density* and *laser peak amplitude*, which are two initial conditions to the WarpX



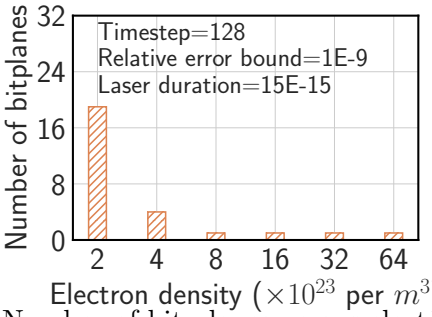
(a) Number of bit-planes across timesteps.



(b) Number of bit-planes versus error bounds.



(c) Number of bit-planes versus laser duration.



(d) Number of bit-planes versus electron density.

Figure 4.1 MGARD number of bit-planes versus timesteps, relative error bounds, and simulation-dependent parameters.

simulation experiment. Therefore, estimating MGARD's number of bit-planes is a non-linear high-dimensional problem that is difficult to solve with traditional modeling techniques, and we seek to tackle it with Deep Neural Network (DNN) as a data-driven approach.

Motivation 2: *The number of bit-planes to be retrieved to satisfy a given error bound is highly sophisticated and can be affected by many factors. Therefore it is beneficial to capture using the DNN-based approaches.*

4.2 DNN-based Progressive Retrieval

In this section, we first formulate the research problem of improving progressive retrieval for HPC scientific data towards minimum I/O overhead and then discuss the details of design and implementation. For the convenience of discussion, we adopt MGARD as an example of a bit-plane-based progressive data retrieval framework

Table 4.1 A List of Symbols for DNN-based Progressive Retrieval

Symbols	Description
D	The retrieved data size (bytes).
L	The number of coefficient levels.
e	User-requested maximum absolute error.
err	Maximum absolute error of reconstructed data.
Err	Maximum absolute error of coefficient levels.
B	The total number of bit-planes on each coefficient level.
b_l	The number of bit-planes to retrieve on coefficient level l.
$\mathcal{L}(x, y)$	The loss function used to control the prediction error.
$\mathbb{S}$	The sizes of bit-planes across coefficient levels (bytes).
$\mathcal{F}$	The mapping function from input error bound to b.
F	A set of data features used in DNN training.

to discuss the details of our design. We first list the commonly used notations in Table 4.1.

4.2.1 Problem formulation

The HPC progressive I/O overhead, which is also the data retrieval size, depends on both the number of coefficient levels as well as the number of bit-planes of retrieval on each level. Denote $l(0 \leq l < L)$ as the current coefficient level where L is the total number of coefficient levels, $b_l(0 \leq b_l \leq B)$ as the number of bit-planes to retrieve for level l where B is the total number of bit-planes on each coefficient level (32 for single-precision floating-point data), $k(0 \leq k \leq b)$ as the current bit-plane index, and $\mathbb{S}$ as the sizes of bit-planes on each level. We have the data retrieval size D as the accumulation of bit-plane sizes across coefficient levels, as shown in Equation (4.1).

$$D = \sum_{l=0}^{L-1} \sum_{k=0}^{b_l} \mathbb{S}_{lk} \tag{4.1}$$

In particular, the total number of coefficient levels L is jointly determined by both data resolution and user input during data decomposition. On the other hand, the sizes of bit-planes across coefficient levels $\mathbb{S}$ also depend on the resolutions of coefficient levels, which are also determined during data decomposition. Therefore, both L and $\mathbb{S}$ can be considered as constant during data retrieval, and the overall size

of progressive retrieval D can be directly calculated by the number of bit-planes of retrieval b_l. Ideally, the purpose of the progressive retrieval framework is to compute the number of bit-planes of retrieval b_l based on the user-requested maximum absolute error e, as shown in Equation (4.2), to the extent that the achieved maximum error err after recomposition is less than but very closed to e, so that users can understand the information loss incurred by progressive retrieval. Yet, as we mentioned in Section 4.1, the current framework achieves a significantly lower maximum absolute error than the one user requested ($err \ll e$), such that the err became almost agnostic to users while resulting in higher b_l and larger D.

$$b_l = \arg\min_e \mathcal{F}(e), b_l \in [b_0, b_1, ...b_{L-1}] \tag{4.2}$$

In this work, our objective is to improve the performance of the progressive retrieval framework by optimizing the error control mechanism of the original MGARD. In particular, we aim to find a better mapping function that users can utilize to guide the choice of error bounds and data retrieval. While the relation between e and b_l is unclear, err and b_l are corresponding to each other by nature. Therefore, we aim to directly learn the mapping between the achieved maximum error and the number of bit-planes, leading to our first approach.

Approach A: For each field of data from scientific applications, we first perform compression experiments using the original MGARD under an extensive set of input error bounds e, record the number of bit-planes of retrieval b_l as well as the achieved error err. Then, we can train a DNN model to predict b_l with err, which can be used to take the place of the error interpreter in the original MGARD. This predictive model, named D-MGARD, takes the achieved maximum error err, along with a set of data features, as input and directly predicts the number of bit-planes of retrieval b_l

for each coefficient level. It bypasses the original error control mechanism in MGARD completely and predicts the number of bit-planes based on the inherent correlation from the data retriever.

The reasons why we choose DNN over traditional ML methods for D-MGARD are mainly two-fold: 1) Given the large problem space and complexity of our work (large numbers of simulation configurations, timesteps, and compression error bounds), the number of bit-planes exhibits extremely complicated and non-linear behaviors where DNN is naturally anticipated to perform well; 2) With the possible extension of problem space, where we seek to predict more data attributes with the model, the feature selection and engineering is challenging for ML methods whereas DNN can benefit from more advanced model architecture to extract features automatically.

As D-GMARD achieves the prediction of the number of bit-planes in a purely data-driven way, such a model provides limited interpretability as it treats the mapping as a black box. In the following, we propose another approach that is more closely coupled with the philosophy of MGARD.

Approach B: Upon recomposition, MGARD estimates the reconstruction error based on the partially retrieved coefficient levels. Given the resolution and mesh structure of the original dataset, the coefficient level error Err is converted to data reconstruction error err. As we discussed previously, the pessimistic error estimation from Err to err is the key reason for the aggressive error control, which leads to the extra I/O overhead. Therefore we propose to improve the error estimation with DNN. We propose to design a DNN model, named E-MGARD to predict the data reconstruction error err given the coefficient level error Err. Such a model replaces the original MGARD to achieve more precise error control.

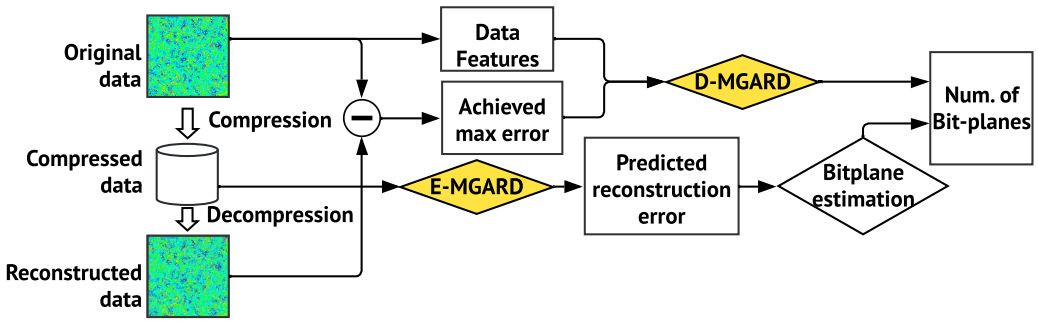
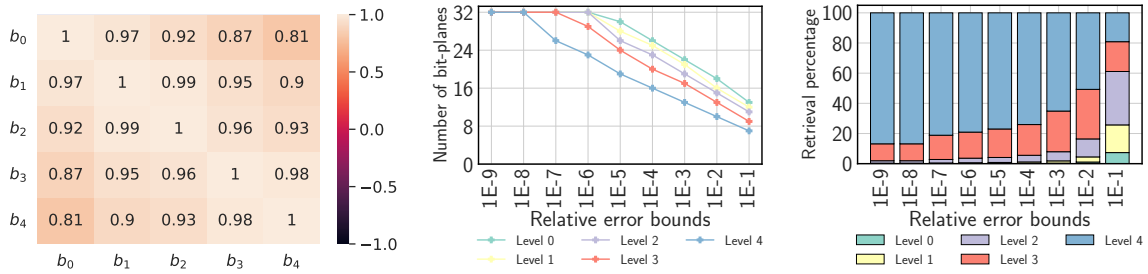


Figure 4.2 Overview of DNN-based progressive data retrieval framework.

4.2.2 Design overview

We present the overall design of the framework in Figure 4.2. We showed D-MGARD and E-MGARD in the framework we previously introduced. In the following, we discuss the design details of D-MGARD and E-MGARD toward the numbers of bit plane prediction, as well as the selection between the two approaches.

D-MGARD: number of bit-planes estimation The idea is to learn a prediction model that directly maps between the number of bit-planes and the achieved maximum error of reconstructed data. Such that when users require the reconstructed data to a specific maximum error, our model can predict the number of bit-planes that need to be fetched. This approach does not take into consideration how MGARD handles the input data but rather treats all processing and transformation operations of MGARD as a black box. The D-MGARD contains the following steps: 1) run the compression experiments under a set of absolute errors; 2) collect the achieved maximum errors as well as the numbers of bit-planes fetched from coefficient levels under achieved absolute errors; 3) train a multi-target prediction model with the achieved maximum error as input and numbers of bit-planes as the target. In order to account for the impact of data characteristics on the performance of MGARD, the prediction model also takes a set of statistical data features as input.



(a) Correlation of the numbers of bit planes.

(b) Number of bit planes from coefficient levels.

(c) Breakdown of retrieval size from coefficient levels.

Figure 4.3 MGARD retrieval performance across relative error bounds.

Compared to the traditional theory-based predictor, the proposed D-MGARD directly learns the mapping between the number of bit-planes to the achieved maximum absolute error. In order to achieve this, the error bounds that we feed into the model as input are *achieved* error (red curves in Figure 1.2) we collected before training rather than the one users requested.

D-MGARD essentially performs *regression* tasks by taking data features and user-requested error bounds as input and producing the number of bit-planes for each coefficient level as output. Generally speaking, we consider the scenarios where the data is decomposed to more than one level. Therefore, such a regression task is also called *multi-output regression*.

The first and most intuitive approach is to train a multi-level perceptron (MLP) model with input and output layer dimensions matching the dimension of training variables and targets, respectively. However, as pointed out by literature [20], such an MLP model usually suffers from low training accuracy, as the correlation among target variables is not accounted for.

In Figure 4.3a, we demonstrate the correlation matrix of the numbers of bit-planes across five levels. As shown in the figure, the numbers of bit-planes are strongly correlated with each other. Such correlation is caused by: 1) the values of bit-plane numbers, which all are integers in the range of [0, 32]; 2) the greedy-based bit-plane retriever fetches bit-planes by accuracy efficiency ranking. Therefore, the number of

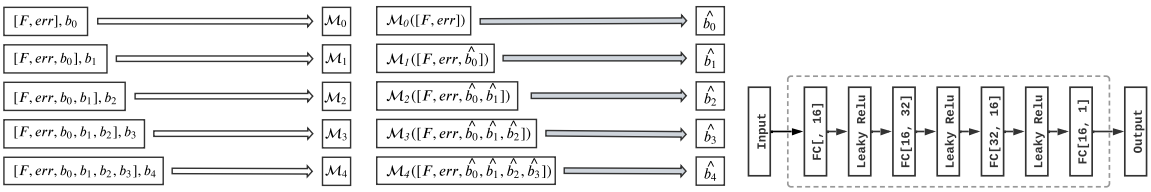
bit-planes to be retrieved from a certain level depends not only on the user-requested error bound but also on the number of bit-planes of all other levels.

The accuracy efficiency for a bit-plane used by MGARD's greedy-based retriever is the ratio between the error reduction by retrieving the bit-plane and its resolution. Based on our investigation, we observe that for bit-planes with the same bit location, those on a higher level generally demonstrate higher accuracy efficiency than those on a lower level, as the difference in the resolution across coefficient levels. Therefore, bit-planes on higher levels are usually retrieved first. We show such behavior in Figure 4.3b, where we display the number of bit-planes retrieved from each of the five coefficient levels across relative error bounds during data retrieval. As shown in the figure, level 0 always contributes the most bit-planes while level 4 always contributes the least.

We further demonstrate the breakdown of bit-plane sizes across five coefficient levels in Figure 4.3c. It is interesting to show that, despite the lowest number of bit-planes level 4 contributing to the retrieved data, it holds the most significant proportion of the retrieved data size under most error bounds. The only exception is the relative error bound of 1E-1 where the data accuracy requirement is very low so there is almost no need to read from level 3 and level 4.

Such breakdown suggests that the numbers of bit-planes across the coefficient levels are not of the same importance, rather those on a lower level play more important roles in determining the retrieved data size. Therefore, in order to minimize the I/O overhead, it is more important to capture the numbers of bit-planes for lower levels than for higher levels.

In order to leverage the correlation among the numbers of bit-planes and consider the weighted importance of the numbers of bit-planes across coefficient levels, we design D-MGARD as a chained multi-output regression model (CMOR) as shown in Figure 4.4. Let $L = 5$ in this case as an example. Essentially, the idea



(a) Offline training stage. **(b)** Online prediction stage. **(c)** MLP network on each level.

Figure 4.4 Chained multi-output regression (CMOR) model for a five-level hierarchy.

is to train a separate MLP for each level, denoted as $\mathcal{M}_l (0 \leq l < 5)$, to capture the number of bit-planes. In order to leverage the correlation among the numbers of bit-planes, the training variables for each level include not only the general variables like data features $\mathbb{F}$ and achieved maximum absolute error err, but also the numbers of bit-planes from previous levels $[b_0, ...b_{l-1}]$. For example, as shown in Figure 4.4a, $\mathcal{M}_1$ is trained with b_0 as an additional feature, $\mathcal{M}_2$ with b_0 and b_1, ... and so on. During the online prediction, as shown in Figure 4.4b, the number of bit-planes for each level is predicted with each pre-trained model following a sequential order, from level 0 to level 4.

Such model design is closely coupled with the greedy-based bit-plane retriever as well as the characteristics of the number of bit-planes we discussed above. Specifically, the most important number of bit-planes (on level 4) is trained with the most features. On the other hand, all MLP models can be trained in parallel, without causing additional training overhead.

The network architecture of the MLP model on each level is shown in Figure 4.4c. We configure the network with six fully-connected hidden layers and the activation function of leaky ReLu in between. We note that the dimension of the input layer is different across levels, accounting for the additional training features on each level, while the dimension of the output layer is always one.

During the model training, we seek to minimize the loss function, which is the average prediction error of bit-plane numbers for each level, as shown in Equation (4.3).

$$\mathcal{L}(b_i, \hat{b_i}) = \frac{1}{L} \sum_{i=0}^{L-1} \ell(b_i, \hat{b_i}) \tag{4.3}$$

Common choices of loss function $\ell(x, y)$ generally include the mean squared error (MSE) and the mean absolute error (MAE). Yet, based on our investigation, neither is a good choice in our case. On the one hand, controlling prediction error with MAE often leads to long tails in the distribution of prediction error, indicating the existence of large outliers. The reason is that MAE does not penalize the large prediction errors enough. On the other hand, controlling with MSE can capture those outliers fine but often leads to large average prediction errors. The reason is that MSE, by taking square on prediction errors, is not sensitive to small prediction errors. Based on empirical experiments on various loss functions, we observe superior training accuracy can be obtained in our case by controlling the prediction error with the Huber loss function [34], which is defined in Equation (4.4).

$$\ell(x, y) = \begin{cases} \frac{1}{2}(x-y)^2, & \text{if} |x-y| < \delta \\ \delta(|x-y| - \frac{1}{2}\delta), & \text{otherwise} \end{cases} \tag{4.4}$$

The Huber loss is a combination of MSE and MAE that is less sensitive to outliers than the MSE. It is quadratic for small prediction errors less than a certain threshold δ and linear for large prediction errors beyond δ. In this work, we observe that setting $\delta = 1$ achieves the best training accuracy. Such that the loss function used in our work can be written as follows.

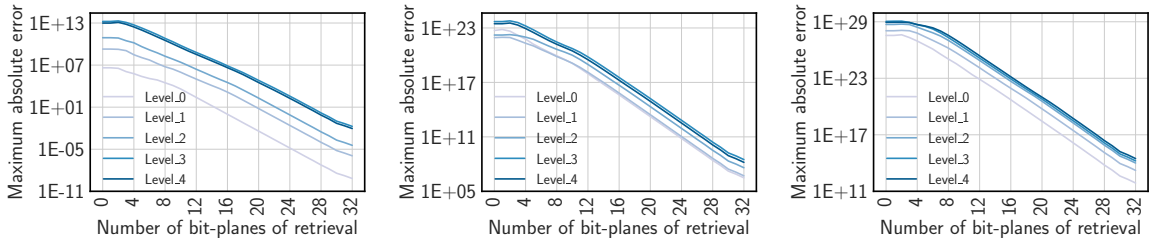
$$\ell(b_i, \hat{b_i}) = \begin{cases} \frac{1}{2}(b_i - \hat{b_i})^2, & \text{if} |b_i - \hat{b_i}| < 1 \\ (|b_i - \hat{b_i}| - \frac{1}{2}), & \text{otherwise} \end{cases} \tag{4.5}$$

E-MGARD: error control optimization According to the error control of MGARD recomposition, the maximum error between original data and recomposed data is bounded by the following Equation (4.6), where $Err[l][b_l]$ denotes the absolute error of l-th ($0 \leq l < L$) coefficient level when retrieving the first $b_l (0 \leq b_l \leq B)$ bit planes and C is a constant that maps the absolute error of coefficient levels to the absolute error of the reconstructed data. It has been noted [8] that C depends on the data characteristics, specifically the mesh structure of the input dataset. Therefore, it has to be manually derived based on the mesh structure of each scientific application. Nevertheless, the derived mapping constant still suffers from sub-optimal performance. As shown in Figure 1.2, the mapping constant yields over-pessimistic error control and thus results in extra I/O overhead.

$$err \leq C \sum_{l=0}^{L-1} Err[l][b_l] \tag{4.6}$$

One issue with the current error control approach is that the same mapping constant is applied to all coefficient levels, implying that the absolute error incurred from progressive retrieval on each coefficient level has the same impact on the absolute error of the reconstructed dataset. We demonstrate that this is not the case in Figure 4.5.

In Figure 4.5, we demonstrate the absolute error of each coefficient level incurred by progressively retrieving an increasing number of bit-planes. It can be shown in the figure that the magnitude of absolute error across coefficient levels shows significant



(a) B_x **(b)** E_x **(c)** J_x

Figure 4.5 Absolute error of progressive retrieval from coefficient levels.

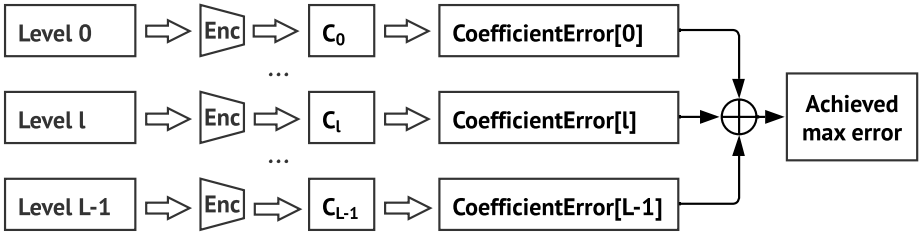


Figure 4.6 Design of E-MGARD error prediction model.

differences. Therefore, when the same mapping constant C is applied to different coefficient levels, the absolute error of the reconstructed dataset is biased towards the absolute error of lower coefficient levels, reducing the granularity of error control and over-pessimistic error estimation.

In order to tackle this issue, we propose to learn a mapping constant for each coefficient level. We present the design of our approach, named E-MGARD, in Figure 4.6. Specifically, the *Enc* block in the figure is an encoder network with hidden layer dimensions of 2048, 512, 128 and 8. The activation function is ReLu. As shown in the figure, for each coefficient level from a decomposed dataset, an encoder network transforms it into a set of latent features which is further used to predict the mapping constant for the current level. Then the achieved maximum error of reconstructed data can be calculated by Equation (4.7), where $C_l \in \{C_0, C_1, ...C_{L-1}\}$ denotes the learned mapping constant for each coefficient level.

$$err \leq \sum_{l=0}^{L-1} C_l Err[l][b_l] \tag{4.7}$$

4.2.3 Selection between D-MGARD and E-MGARD

Between the two approaches we proposed in this section, the advantage of D-MGARD is the easiness of learning the prediction model, namely users do not need to have a deep understanding of MGARD, or other progressive retrieval methods. Nevertheless, D-MGARD suffers from mainly two disadvantages: 1) the difficulty to summarize a set of data features to account for the impact of data characteristics on the MGARD performance, which will limit the modeling capability and result in estimation error; 2) the lack of understanding how MGARD processes and transforms the input data. Especially, the lack of understanding of MGARD prevents a more precise prediction model from being developed.

On the other hand, E-MGARD focuses more on improving the error control of the original MGARD. It is more precise in the modeling scope compared to D-MGARD. Especially for the bit-plane estimation using coefficient level error (as shown in Figure 4.2), where D-MGARD takes it as a part of the black box and estimates with a chained multi-output regression model, E-MGARD leverages the greedy-based estimation method from original MGARD for calculation, which is more accurate by nature. However, the design of E-MGARD is tightly coupled with the recomposition scheme of MGARD, making it hard for E-MGARD to extend to other progressive retrieval methods.

4.3 Performance Evaluation

In this section, we present the evaluation results for our DNN-based progressive retrieval framework. We first discuss the experiment setup as follows.

4.3.1 Experimental setup

1) Hardware platform: We perform the progressive decomposition and recomposition experiments on the Summit supercomputer at Oak Ridge National Laboratory [4].

All experiments are performed on a single CPU node with two 16-core 3.0 GHz AMD EPYC processors and 256GB of main memory. We train and evaluate the DNN-based progressive retrieval framework on the Google Colab environment with a single GPU. The Colab environment is driven by three Intel Xeon processors with 26GB memory, as well as an NVIDIA Tesla P100 GPU with 16GB HBM.

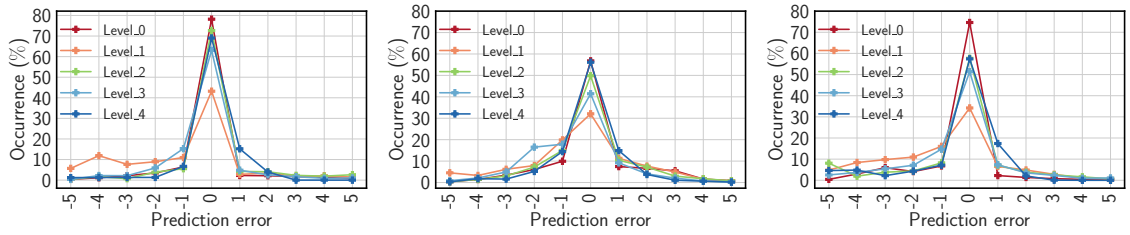
2) Scientific datasets: We use Gray-Scott application [58] and WarpX application [57] as evaluation datasets in this work. The datasets are obtained by running simulations on Summit. The detailed information of the datasets is shown in Table 4.2. All datasets are double-precision floating-point values.

Table 4.2 Scientific Applications

Application	**Fields of use**	**Dimensions**	**Timesteps**
Gray-Scott	D_u, D_v	512^3	512
WarpX	B_x, E_x, J_x	512^3	512

3) MGARD compression experiment: In order to cover various application scenarios where users can have arbitrary error-bound requirements, we compress the simulation datasets using MGARD under a large range of relative error bounds, including 81 error-bound values. Specifically, the relative error bounds used in this work are [1E-9, 2E-9, ... 8E-1, 9E-1]. We then collect the number of bit-planes on each coefficient level as well as the achieved maximum error associated with each input error bound to assemble the training records for D-MGARD. We also store the decomposed coefficient level data as the training set for E-MGARD. In order to compress the multiple timesteps of application data, we assume the data range of each field at each timestep is calculated during the simulation and available during compression.

4) Training configurations: For each field of Gray-Scott and WarpX application, we train our models on the first 256 timesteps of the dataset and test on the remaining ones. The D-MGARD is trained with a learning rate of 0.00005, batch size of 256, and E-MGARD is trained with a learning rate of 0.00001, batch size of 64. Both models



(a) *J_x* **(b)** *B_x* **(c)** *E_x*

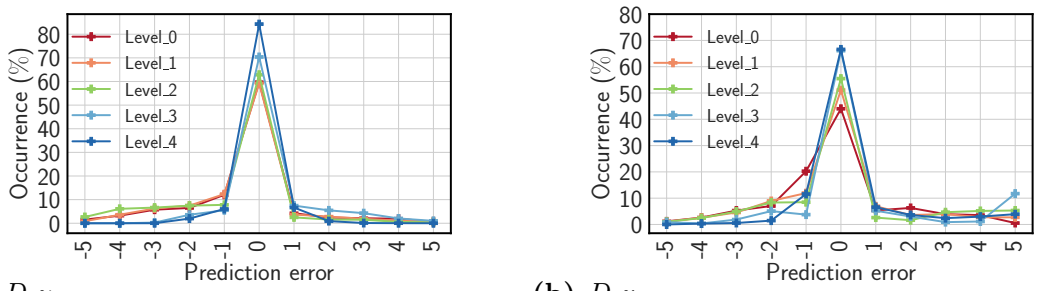
Figure 4.7 Prediction error distribution of D-MGARD on WarpX application.

have been trained for 300 epochs, and the cost of training for each model is 0.7 hours and 1.2 hours, respectively. We would like to point out that, in our work, our split of training data via timesteps is to mimic the way scientific applications dump data during simulation. Throughout the simulation process, applications will periodically dump variable data which are evolving with time. The purpose of training on the first half of timesteps and testing on the later half is to evaluate whether the trained model can be reused when the data show changes over time. Under the scenario of "train once, infer many times", we expect our model can be reused on new datasets generated from the same applications and the training time can be amortized.

For the rest of this section, we evaluate the performance of our proposed framework. Specifically, we first evaluate the prediction accuracy of D-MGARD across simulation timesteps as well as across data resolutions. Then for E-MGARD, we evaluate the achieved maximum error of reconstructed data against input error and achieved a maximum error of original MGARD. We then evaluate both approaches on the total retrieval size against the original MGARD.

4.3.2 Prediction accuracy across simulation timesteps

Considering the nature of scientific applications that evolves with the simulation timesteps, the most important question we need to address in order for our model to be adopted for production is whether our model can be trained on early timesteps and applied to future ones. Therefore, for a certain application dataset, we train on the



(a) *D_u* **(b)** *D_v*

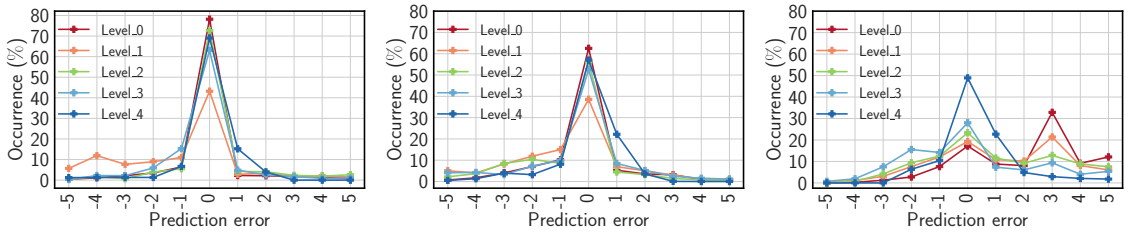
Figure 4.8 Prediction error distribution of D-MGARD on Gray-Scott application.

first half of timesteps and test the model performance on the second half. In Figure 4.7, we present the distribution of D-MGARD prediction error on the WarpX laser-driven electron acceleration application. We trained the model on the first 256 timesteps of J_x field and evaluated the prediction accuracy on J_x, B_x as well as E_x. As shown in Figure 4.7a, the absolute prediction error generally ranges between -5 and 5, where positive error indicates over-estimation and negative error indicates under-estimation. For J_x, more than 60% predictions are made without error for levels 1 - 4, with an additional 20% prediction resulting in prediction error by one bit-plane at most. Similar performance holds for B_x and E_x as well, as shown in Figures 4.7b and 4.7c that the majority of predictions are made correctly. Furthermore, the prediction error decreases from level 0 to level 4 (more predictions are without error), indicating that the numbers of bit-planes of retrieval on lower coefficient levels are better captured.

In Figure 4.8, we show the prediction error distribution of D-MGARD on the Gray-Scott application. Similarly, D-MGARD performs well on lower coefficient levels to capture the number of bit-planes of retrieval that more than 60% of predictions are made without error.

4.3.3 Prediction accuracy across simulation resolutions

In this section, we aim to answer the following question: can the D-MGARD model be applied across simulation resolutions? As running compression experiments and collecting training data on full-resolution data can be resource-demanding, it can



(a) Resolution of 64^3. **(b)** Resolution of 128^3. **(c)** Resolution of 256^3.

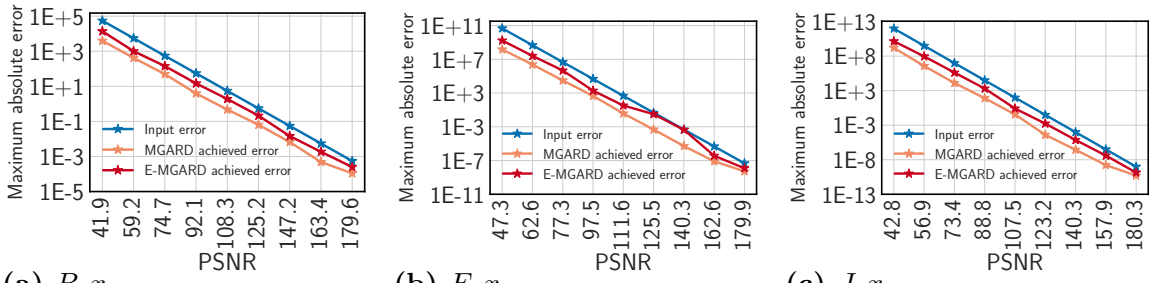
Figure 4.9 Prediction error distribution of D-MGARD across data resolutions.

be beneficial if users can train D-MGARD on low-resolution data and apply it to high-resolution data of the same application. Hereby, we train the model on the J_x field of the WarpX application with the resolution of 64^3 and test on the resolutions of 128^3 and 256^3. We demonstrate the distribution of prediction error in Figure. 4.9. As shown in the figure, the D-MGARD model performs well when being trained on the resolution of 64^3 and tested on the resolutions of 64^3 and 128^3. When being tested on the resolution of 256^3, the prediction accuracy drops significantly. The main reason for such behavior is that more local features manifest in the higher-resolution data, which causes the change of data characteristics and the deviation of MGARD performance, which is hard for D-MGARD to capture.

Despite that the D-MGARD model does not work well when the resolution of testing data deviates too much from the resolution of training data, we note that the 80% of predictions on level 4 still achieve error by at most one bit-plane.

4.3.4 Achieved maximum error against original MGARD

In this section, we demonstrate the achieved maximum error by E-MGARD. As we discussed previously, E-MGARD improves the error control by predicting the maximum absolute error of reconstruction data with improved mapping from coefficient level error to reconstruction data error. In Figure 4.10, we show the achieved maximum error by E-MGARD across PSNR, compared to the maximum error achieved by original MGARD as well as the user-requested absolute error bound.

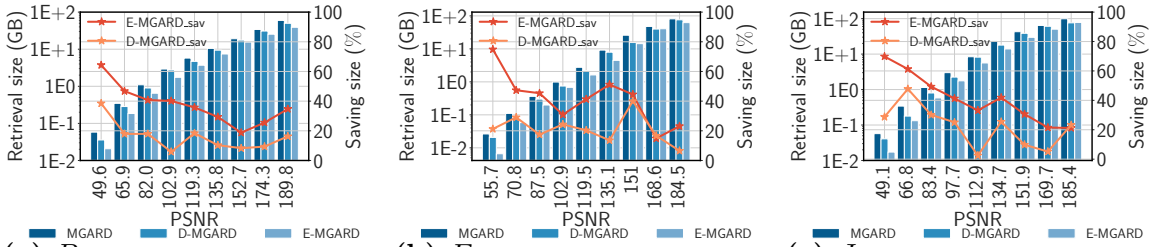


(a) B_x **(b)** E_x **(c)** J_x

Figure 4.10 E-MGARD achieved maximum absolute error as compared with original MGARD as well as input error bound.

It can be shown that the maximum absolute error values achieved by E-MGARD lie closer to the user-requested error, thus providing better error control.

4.3.5 Retrieval size against original MGARD



(a) B_x **(b)** E_x **(c)** J_x

Figure 4.11 Total retrieval size of D-MGARD and E-MGARD compared with original MGARD across 512 timesteps.

In Figure 4.11, we demonstrate the total retrieval size incurred by D-MGARD and E-MGARD as compared to the original MGARD. The retrieval sizes are accumulated across a total of 512 timesteps. It is clear that E-MGARD achieves the least retrieval size and thus the lowest I/O overhead.

We define the percentage of saved retrieval size in the following equation:

$$Sav = |D_{mgard} - D_{new}|/D_{mgard} \tag{4.8}$$

where D_{new} represents the retrieval size incurred by either D-MGARD or E-MGARD. As shown in the figure (red and orange curves), D-MGARD reduces the retrieval size between 5% and 40%, whereas E-MGARD achieves the saving between 20% and 80%. It can be observed that E-MGARD typically achieves the highest saving percentage at small PSNR values, indicating that it holds stronger advantages over D-MGARD and original MGARD when I/O is very limited and users are asking for a low resolution of data being transmitted.

We would also like to point out that, for most evaluation cases, E-MGARD would achieve a decompression error lower than the input error, although there is no theoretical bound. The reasons are mainly two-fold: 1) The coefficient level reconstruction error is incurred by quantifying the coefficient on each level. This error is further used to estimate the compression error, which guides the selection of bit planes. In reality, the coefficient level reconstruction error with different signs can be canceled with each other. Yet, there is a lack of theoretical analysis of the signs, and MGARD treats it as there is no error cancellation. Thus, the coefficient level reconstruction error is over-estimated. 2) The predicted compression error will be lower than the input error bound, which is the outcome of MGARD's greedy search.

As E-MGARD only tunes the values of mapping constant C, the second issue can be improved with more granularity. Therefore, the difference between the predicted compression error and coefficient level reconstruction error will be minimized. Nevertheless, the first issue is beyond the scope of our work as our model depends on the collected quantization error to predict compression error as well. Therefore, even with the optimized mapping constants, the numbers of bit-planes to be fetched are still generally more than required, and the compression error of E-MGARD would still be lower than the input.

It is still possible that the E-MGARD achieves a larger compression error than the input, meaning that the error bound is not respected. Cases like these will happen

under three conditions: 1) The error bound is extremely small (1E-9, 2E-9), so the mapping constants are highly impactful. 2) The mapping constant is significantly underestimated, leading to the underestimation of predicted compression error and the number of bit-plane. 3) The coefficient level reconstruction error is not over-estimated, meaning that there is no error cancellation across the coefficient levels, which is extremely unlikely for a real dataset.

In this section, we have demonstrated the prediction accuracy of D-MGARD on the number of bit-planes of retrieval across simulation timesteps, data resolution as well as data fields. The prediction error is very low for most cases as the majority of prediction error is by one bit-plane. Nevertheless, we still notice large prediction errors occurring despite low probability. We think the prediction error is caused by feature selection and engineering. Given the lack of understanding between the number of bit-planes and the achieved maximum error, it remains a challenge to choose a set of intuitive features for prediction. Such a challenge motivates us further to explore more advanced DNN models in the future. On the other hand, we have demonstrated the effectiveness of E-MGARD in optimizing the error control, which results in a more precise maximum error of reconstructed data. While E-MGARD is proven to be beneficial, Figure 4.11 shows that the most percentage of saving occurs at low PSNR scenarios, indicating imbalanced performance across input error bounds.

Furthermore, Our proposed work is designed and implemented in conjunction with MGARD by replacing some of the error control functionality within the original MGARD. For example, D-MGARD replaces the error estimator as well as the bit-plane retriever by directly predicting the number of bit-planes of retrieval for each coefficient level based on the user-requested maximum error. Then, the size interpreter in MGARD can calculate the retrieval size as well as the precise segment to fetch the data. E-MGARD only bypasses the error estimator by producing a more precise estimation of the achieved maximum data reconstruction error. Accordingly, MGARD

can calculate the number of bit-planes of retrieval using the original greedy-based iterative method. Both models will be deployed to work inside the progressive retrieval framework, provided they are previously trained on each application dataset. At this point, both models should be trained offline beforehand. The trained model can be used for inference with the specification of the model parameters through the high-level MGARD decompress API. Currently, the design and deployment of D-MGARD and E-MGARD are separated, and it depends on users to choose between two approaches as we discussed in Subsection 4.2.3. We would also like further to investigate the combination of two models in the future.

4.4 Conclusion

This chapter proposes a DNN-based progressive retrieval framework to reduce the I/O by minimizing the data fetched. To this end, we design two prediction models to estimate the number of bit-planes of retrieval (D-MGARD) and improve the error control (E-MGARD) under user-requested error bounds. We evaluate our proposed DNN-based progressive retrieval framework on two scientific application datasets. We demonstrate that our framework holds significant advantages over the traditional approach. By evaluating against the original theory-based progressive data retrieval framework, our solution can access significantly fewer data (between 5% and 40% with D-MGARD, between 20% and 80% with E-MGARD). Our approach brings opportunities for improving scientific applications running on computing clusters, including HPC systems and grid computing environments.

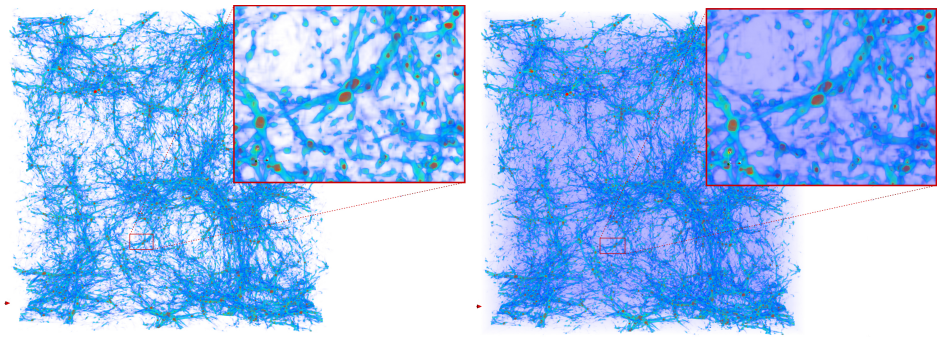
CHAPTER 5

FUTURE WORK

As we previously introduced, with the increase in computing power of supercomputers, scientists can now run more accurate simulations that generate more data than ever before. However, that increase in computing power has not been matched by an increase in storage, meaning that more data is now generated than can be stored. One of the fields generating the greatest amount of simulation data is cosmology. Cosmology simulations, such as Nyx [9] and HACC [29], which study the origin and evolution of the universe, can generate several terabytes to petabytes of data per run. Therefore, scientists are trying to incorporate lossy data reduction techniques into the data management workflow to reduce the storage needed. In this chapter, we take the Nyx cosmological simulation as an example and introduce the necessity of investigating the impact of lossy data compression on downstream scientific analytic tasks.

One of the most common ways to examine data in cosmology is visualization [38], and the most commonly used algorithm for visualizing 3D cosmology data is volume rendering [43]. As shown in Figure 5.1, volume rendering allows us to investigate the structure of cosmology datasets by examining the cosmic web's halos (dense regions) and filaments.

While the quality of data required for analysis, for example, to ensure that the power spectrum only deviates by at most 1%, is quite high, the quality needed for visualization is usually lower, hence allowing for higher compression ratios [36]. Error-bound lossy compression algorithms preserve all the values in a dataset but will add/subtract a bounded error for each of the dataset's values to enable data reduction. While the volume rendering algorithm performs a number of additive operations over samples in the dataset, which can increase the errors, the human visual system is



(a) Original **(b)** Compressed

Figure 5.1 Volume rendering of dark matter density field of a Nyx dataset.

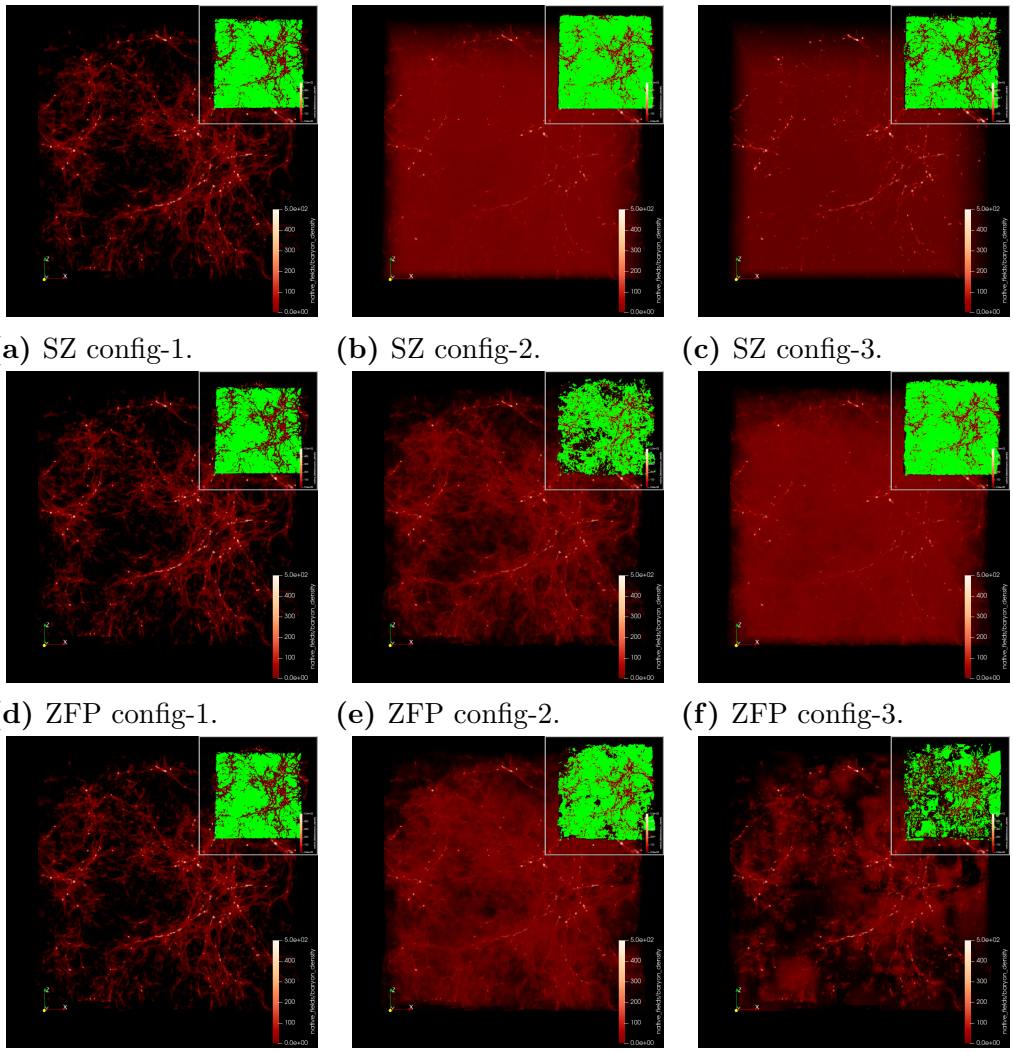
quite efficient at filtering out noise [59], meaning that the images generated are still usable at error levels that would be too high for data analysis. The key question is how much we can reduce the data before noticeable artifacts are introduced. The image in Figure 5.1b has been generated from a dataset compressed using MGARD [8], an error-bound lossy compressor, using an absolute error of 2 (dark matter density range is [0, 5452]). While the structures are still visible, blue-gray background noise has been added to the data.

In this chapter, we introduce an ongoing effort to investigate how applying different kinds and amounts of data reduction affects images created using the volume rendering algorithm on the Nyx dataset. We first visually inspect the image visualization results and assess the impact of lossy data compression on decompressed data. Then, to quantify the perceivable difference in the visualization, we compute several Image Quality Assessment (IQA) metrics and attempt to determine which are more accurate at flagging errors in the visualization.

5.1 Visual inspection of visualization quality

Due to the difference in design philosophy, each error-bound lossy compressor uses a different method to achieve data reduction. Consequently, each one produces different kinds of errors and amounts of compression. By varying the error-bound inputs, we

obtain a set of decompressed data of *Baryon_density* field from the Nyx dataset, with fixed compression ratios. We then demonstrate the volume rendering visualization results in Figure 5.2.



(a) SZ config-1. **(b)** SZ config-2. **(c)** SZ config-3.

(d) ZFP config-1. **(e)** ZFP config-2. **(f)** ZFP config-3.

(g) MGARD config-1. **(h)** MGARD config-2. **(i)** MGARD config-3.

Figure 5.2 Volume rendering for *Baryon_density*.

Figure 5.2 shows a visualization of *Baryon_density* compressed with SZ, ZFP, and MGARD, with three error-bound configurations. An inset image located in the upper right corner of each visualization shows the rendered image difference, which retains the pixel value when identical to the reference image value but toggles to green otherwise.

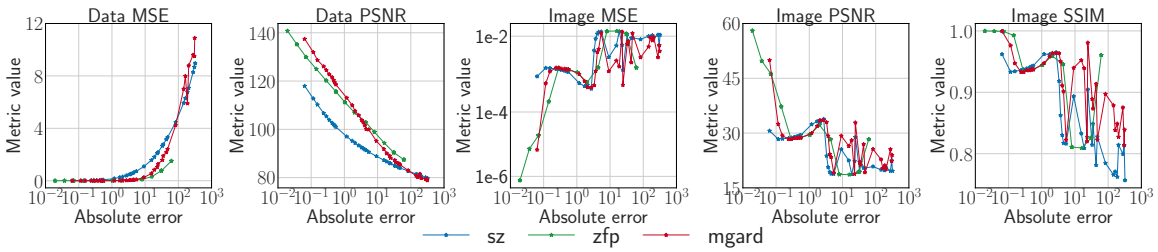


Figure 5.3 Data-based versus image-based quality metrics of *Baryon_density*.

As shown in Figure 5.2, the image difference becomes more and more significant as the error bound increases, and more importantly is different depending on the compressor that was used. For example, SZ adopts linear prediction for each data point using eight surrounding neighbors, while ZFP performs a non-orthogonal transform on each data block before encoding. Such smoothing operations can change the values of background noise, which are usually very close to zero, to positive values, which then emerge in the visualization. Given the different image difference patterns from different compressors, the choosing of compression techniques for the image quality remains an unsolved problem, which requires more in-detailed investigation, including quantitative evaluation of more scientific application data, which we discuss as follows.

5.2 Quantitative evaluation of visualization quality

In Figure 5.3, we show the data-based error metrics (data MSE and data PSNR) and image metrics (image MSE, image PSNR, and image SSIM) for SZ, ZFP, and MGARD for *Baryon_density*. The data-based error metrics change monotonically across absolute errors due to the error control mechanism employed by each compressor. However, the image-based quality metrics demonstrate rather drastic and complex behaviors, as they react to the artifacts in the images differently. As shown in the figure, the image qualities drop around the absolute error of 10^{-1} and bounce back around 10^{0}, then drop again markedly.

SSIM has been previously shown to be a reliable estimator of how errors in images will be perceived by humans [12, 13, 28]. When comparing data-based MSE and PSNR to image-based MSE, PSNR as well as SSIM, we can see that the patterns in the Data MSE and Data PSNR have a very poor (if any) correlation with SSIM and thus cannot be used to determine if reconstructed data would be suitable for image analysis. On the other hand, the Image MSE and Image PSNR do a better job of capturing the fluctuations detected by SSIM. Nevertheless, Image MSE and Image PSNR, and SSIM are still not considered ideal quality metrics as they do not react differently to the compression artifacts introduced by different compression techniques. To date, there are no ideal image quality metrics that are well suited to assess the visualization quality after scientific data compression. More evaluation on domain-specific image quality assessment metrics is necessary for effective quantitative evaluation of scientific visualization.

REFERENCES

[1] A simulation of a hurricane from the national center for atmospheric research. http://vis.computer.org/vis2004contest/data.html, 2004. Accessed: 2023-05-23.

[2] Community earth simulation model (cesm). http://www.cesm.ucar.edu/, 2018. Accessed: 2023-05-23.

[3] Cori. https://www.nersc.gov/systems/cori/, 2021. Accessed: 2021-03-27.

[4] Summit - oak ridge leadership computing facility. https://www.olcf.ornl.gov/summit/, 2021. Accessed: 2021-03-27.

[5] Exaalt: Molecular dynamics at exascale for materials science. https://www.exascaleproject.org/project/exaalt-molecular-dynamics-at-the-exascale-materials-science/, 2023. Accessed: 2023-05-23.

[6] Hardware/hybrid accelerated cosmology code (hacc). https://press3.mcs.anl.gov/cpac/projects/hacc/, 2023. Accessed: 2023-05-23.

[7] James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H Rogers, and Mark Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 424–434. IEEE, 2014.

[8] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. Multilevel techniques for compression and reduction of scientific data—the multivariate case. *SIAM Journal on Scientific Computing*, 41(2):A1278–A1303, 2019.

[9] Ann S Almgren, John B Bell, Mike J Lijewski, Zarija Lukić, and Ethan Van Andel. Nyx: A massively parallel amr code for computational cosmology. *The Astrophysical Journal*, 765(1):39, 2013.

[10] Woody Austin, Grey Ballard, and Tamara G Kolda. Parallel tensor compression for large-scale scientific data. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 912–922. IEEE, 2016.

[11] Utkarsh Ayachit, Andrew Bauer, Earl PN Duque, Greg Eisenhauer, Nicola Ferrier, Junmin Gu, Kenneth E Jansen, Burlen Loring, Zarija Lukić, Suresh Menon, et al. Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 79. IEEE Press, 2016.

[12] Allison H Baker, Dorit M Hammerling, Sheri A Mickelson, Haiying Xu, Martin B Stolpe, Phillipe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, et al. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, 9(12):4381–4403, 2016.

[13] Allison H Baker, Dorit M Hammerling, and Terece L Turton. Evaluating image quality measures to assess the impact of lossy data compression applied to climate simulation data. In *Computer Graphics Forum*, volume 38, pages 517–528. Wiley Online Library, 2019.

[14] Janine C Bennett, Hasan Abbasi, Peer-Timo Bremer, Ray Grout, Attila Gyulassy, Tong Jin, Scott Klasky, Hemanth Kolla, Manish Parashar, Valerio Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, page 49, 2012.

[15] Martin Burtscher and Paruj Ratanaworabhan. Fpc: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58(1):18–31, 2009.

[16] Choongseock Chang. Multiphysics magnetic fusion reactor simulator, from hot core to cold wall. `https://www.olcf.ornl.gov/caar/xgc/`, 2023.

[17] Zhengzhang Chen, Seung Woo Son, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. Numarck: Machine learning algorithm for resiliency and checkpointing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 733–744, 2014.

[18] Daniel Cohen-Or, David Levin, and Offir Remez. Progressive compression of arbitrary triangular meshes. In *Proceedings Visualization '99 (Cat. No.99CB37067)*, pages 1–7, 1999.

[19] Michael Deering. Geometry compression. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 13–20, New York, NY, USA, 1995. Association for Computing Machinery.

[20] Kenan Cem Demirel, Ahmet Sahin, and Erinç Albey. Ensemble learning based on regressor chains: A case on quality prediction. In *DATA*, pages 267–274, 2019.

[21] Sheng Di and Franck Cappello. Fast error-bounded lossy HPC data compression with SZ. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 730–739, 2016.

[22] James Diffenderfer, Alyson L. Fox, Jeffrey A. Hittinger, Geoffrey Sanders, and Peter G. Lindstrom. Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing*, 41:A1867–A1898, 2019.

[23] Torbjørn Eltoft, Taesu Kim, and Te-Won Lee. On the multivariate laplace distribution. *IEEE Signal Processing Letters*, 13(5):300–303, 2006.

[24] JT Fermann and EF Valeev. Libint: Machine-generated library for efficient evaluation of molecular integrals over gaussians, 2003. *Freely available at http://libint. valeyev. net/or one of the authors*, 2013.

[25] Cappello Franck, Ainsworth Mark, Bessac Julie, Burtscher Martin, Choi Jong Youl, Constantinescu Emil Mihai, Di Sheng, Guo Hanqi, Lindstrom Peter, and Tugluk Ozan. Scientific data reduction benchmarks. `https://sdrbench.github.io/`. Assessed: 2023-05-23.

[26] Jean-loup Gailly. Gzip: The data compression program. `https://www.gnu.org/software/gzip/manual/gzip.pdf`, 2016. Accessed: 2016-10-27.

[27] Ana Gainaru, Guillaume Aupy, Anne Benoit, Franck Cappello, Yves Robert, and Marc Snir. Scheduling the I/O of HPC applications under congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS 15)*, pages 1013–1022. IEEE, 2015.

[28] Verislav T. Georgiev, Anna Karahaliou, Spyros Skiadopoulos, Nikolaos Arikidis, Alexandra Kazantzi, George Panayiotakis, and Lena Costaridou. Quantitative visually lossless compression ratio determination of jpeg2000 in digitized mammograms. *Journal of Digital Imaging*, 26(3):427–439, 2013.

[29] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, and Katrin Heitmann. Hacc: Extreme scaling and performance across diverse architectures. In *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2013.

[30] Danny Harnik, Ronen Kat, Dmitry Sotnikov, Avishay Traeger, and Oded Margalit. To zip or not to zip: Effective resource usage for real-time compression. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 229–241, San Jose, CA, February 2013. USENIX Association.

[31] Stephen Herbein, Dong H Ahn, Don Lipari, Thomas RW Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. Scalable I/O-aware job scheduling for burst buffer enabled HPC clusters. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 69–80. ACM, 2016.

[32] Duong Hoang, Harsh Bhatia, Peter Lindstrom, and Valerio Pascucci. High-quality and low-memory-footprint progressive decoding of large-scale particle data. In *2021 IEEE 11th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 32–42, New Orleans, LA, USA, October 2021. IEEE.

[33] Dan Huang, Qing Liu, Scott Klasky, Jun Wang, Jong Youl Choi, Jeremy Logan, and Norbert Podhorszki. Harnessing data movement in virtual clusters for in-situ execution. *IEEE Transactions on Parallel and Distributed Systems*, 30(3):615–629, 2018.

[34] Peter J. Huber. *Robust Estimation of a Location Parameter*, pages 492–518. Springer New York, New York, NY, 1992.

[35] Sian Jin, Sheng Di, Jiannan Tian, Suren Byna, Dingwen Tao, and Franck Cappello. Improving prediction-based lossy compression dramatically via ratio-quality modeling. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2494–2507, 2022.

[36] Sian Jin, Pascal Grosset, Christopher Biwer, Jesus Pulido, Jiannan Tian, Dingwen Tao, and James Ahrens. Understanding gpu-based lossy compression for extreme-scale cosmological simulations. *IEEE International Parallel and Distributed Processing Systems*, 05 2020.

[37] Sian Jin, Jesus Pulido, Pascal Grosset, Jiannan Tian, Dingwen Tao, and James 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*, pages 45–56, 2021.

[38] Brian R. Kent. Editorial: Techniques and methods for astrophysical data visualization. *Publications of the Astronomical Society of the Pacific*, 129(975):058001, apr 2017.

[39] Jeongnim Kim, Andrew D Baczewski, Todd D Beaudet, Anouar Benali, M Chandler Bennett, Mark A Berrill, Nick S Blunt, Edgar Josué Landinez Borda, Michele Casula, David M Ceperley, et al. Qmcpack: An open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter*, 30(19):195901, 2018.

[40] Hemanth Kolla and Jacqueline H Chen. Turbulent combustion simulations with high-performance computing. *Modeling and Simulation of Turbulent Combustion*, pages 73–97, 2018.

[41] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F. Samatova. Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par 2011 Parallel Processing*, pages 366–379, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[42] Edmund Y Lam and Joseph W Goodman. A mathematical analysis of the dct coefficient distributions for images. *IEEE Transactions on Image Processing*, 9(10):1661–1666, 2000.

[43] Fangfei Lan, Michael Young, Lauren Anderson, Anders Ynnerman, Alexander Bock, Michelle A. Borkin, Angus G. Forbes, Juna A. Kollmeier, and Bei Wang. Visualization in astrophysics: Developing new methods, discovering our universe, and educating the earth. *Computer Graphics Forum*, 40(3):635–663, 2021.

[44] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447. IEEE, 2018.

[45] Xin Liang, Qian Gong, Jieyang Chen, Ben Whitney, Lipeng Wan, Qing Liu, David Pugmire, Rick Archibald, Norbert Podhorszki, and Scott 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*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[46] Guo-Yuan Lien, Takemasa Miyoshi, Seiya Nishizawa, Ryuji Yoshida, Hisashi Yashiro, Sachiho A Adachi, Tsuyoshi Yamaura, and Hirofumi Tomita. The near-real-time scale-letkf system: A case of the september 2015 kanto-tohoku heavy rainfall. *SOLA*, 13:1–6, 2017.

[47] Peter Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[48] Peter Lindstrom. Zfp 0.5.5 documentation. `https://zfp.readthedocs.io/en/release0.5.5/modes.html#fixed-accuracy-mode/`, 2021. Accessed: 2021-03-27.

[49] Peter Lindstrom, Po Chen, and En-Jui Lee. Reducing disk storage of full-3d seismic waveform tomography (f3dt) through lossy online compression. *Computers and Geosciences*, 93:45–54, 2016.

[50] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[51] Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. On the role of burst buffers in leadership-class storage systems. In *012 ieee 28th symposium on mass storage systems and technologies (MSST 12)*, pages 1–11. IEEE, 2012.

[52] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, et al. Hello adios: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience*, 26(7):1453–1473, 2014.

[53] Rongxing Lu, Xiaohui Liang, Xu Li, Xiaodong Lin, and Xuemin Shen. Eppa: An efficient and privacy-preserving aggregation scheme for secure smart grid communications. *IEEE Transactions on Parallel and Distributed Systems*, 23(9):1621–1631, 2012.

[54] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Matthew Wolf, Tong Liu, and Zhenbo Qiao. Understanding and modeling lossy compression schemes on HPC scientific data. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS 18)*, pages 1–10, 2018.

[55] Tao Lu, Eric Suchyta, Dave Pugmire, Jong Choi, Scott Klasky, Qing Liu, Norbert Podhorszki, Mark Ainsworth, and Matthew Wolf. Canopus: A paradigm shift towards elastic extreme-scale data analytics on HPC storage. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 58–69, 2017.

[56] Dirk Meister, Jurgen Kaiser, Andre Brinkmann, Toni Cortes, Michael Kuhn, and Julian Kunkel. A study on data deduplication in HPC storage systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, 2012.

[57] Andrew Myers, A Almgren, Ligia Diana Amorim, J Bell, Luca Fedeli, Lixin Ge, Kevin Gott, David P Grote, M Hogan, Axel Huebl, et al. Porting warpx to gpu-accelerated platforms. *Parallel Computing*, 108:102833, 2021.

[58] John E Pearson. Complex patterns in a simple system. *Science*, 261(5118):189–192, 1993.

[59] Michael Pratte, Sam Ling, Jascha Swisher, and Frank Tong. How attention extracts objects from noise. *Journal of Neurophysiology*, 110, 06 2013.

[60] Alexander Pukhov. Particle-in-cell codes for plasma-based particle acceleration. *arXiv preprint arXiv:1510.01071*, 2015.

[61] Jerome M Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[62] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck 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)*, pages 1129–1139. IEEE, 2017.

[63] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Optimizing lossy compression rate-distortion from automatic online selection between sz and zfp. *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[64] Sagar Thapaliya, Purushotham Bangalore, Jat Lofstead, Kathryn Mohror, and Adam Moody. Managing I/O interference in a shared burst buffer system. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 416–425. IEEE, 2016.

[65] Robert Underwood, Sheng Di, Jon C Calhoun, and Franck 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)*, pages 567–577. IEEE, 2020.

[66] Jinzhen Wang, Tong Liu, Qing Liu, Xubin He, Huizhang Luo, and Weiming He. Compression ratio modeling and estimation across error bounds for lossy compression. *IEEE Transactions on Parallel and Distributed Systems*, 31(7):1621–1635, 2020.

[67] Teng Wang, Sarp Oral, Michael Pritchard, Bin Wang, and Weikuan Yu. Trio: Burst buffer based I/O orchestration. In *2015 IEEE International Conference on Cluster Computing*, pages 194–203. IEEE, 2015.

[68] Teng Wang, Sarp Oral, Yandong Wang, Brad Settlemyer, Scott Atchley, and Weikuan Yu. Burstmem: A high-performance burst buffer system for scientific applications. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 71–79. IEEE, 2014.

[69] Kai Zhao, Sheng Di, Xin Liang, Sihuan Li, Dingwen Tao, Zizhong Chen, and Franck Cappello. Significantly improving lossy compression for HPC datasets with second-order prediction and parameter optimization. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, pages 89–100, New York, NY, USA, 2020. Association for Computing Machinery.

[70] Qing Zheng, Kai Ren, Garth Gibson, Bradley W Settlemyer, and Gary Grider. Deltafs: Exascale file systems scale better without dedicated servers. In *Proceedings of the 10th Parallel Data Storage Workshop (PDSW)*, pages 1–6, 2015.