UNIVERSITY OF CALIFORNIA
RIVERSIDE

Lossy Compression for Exascale Scientific Applications

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Kai Zhao

June 2022

The Dissertation of Kai Zhao is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I am extremely grateful to my advisor, Dr. Zizhong Chen, for his encouragement, guidance, and support through every stage of my Ph.D. Without him, I would not have been here to complete my degree and to continue my academic career. Dr. Chen led me to the world of research and allowed me to explore the directions I am interested in freely. Dr. Chen has deep understanding in many fields and he always gave valuable advice on my research topics. I learned the way to be a good researcher from him. Apart from academia work, Dr. Chen cares about my work-and-life balance. I truly appreciate his efforts in solving my two-body issue with my wife.

It is my honor to be mentored by Dr. Franck Cappello and Dr. Sheng Di during my internship at Argonne National Laboratory. The invaluable spirits of academic rigor and critical thinking of Dr. Cappello have guided my continual pursuit of excellence in research. Dr. Di is always passionate and self-motivated, and I am deeply inspired by his research enthusiasm. Dr. Di led me into the topic of lossy compression and provided tremendous support for my research.

I would also like to thank Dr. Rajiv Gupta and Dr. Zhijia Zhao for being my dissertation committee members. Not only did they provide many insightful comments on my thesis, but also they provided great support for my career path.

I want to say thank you to my collaborators, Dr. Robert Underwood, Dr. Maxim Dmitriev, Dr. Thierry-Laurent Tonellot, Dr. Xiaodong Yu, Dr. Danny Perez, Yuanjian Liu, and Sian Jin. I am very lucky to be friends with many brilliant researchers, including Dr. Dingwen Tao, Dr. Hongbo Li, Dr. Jieyang Chen, Dr. Xin Liang, Dr. Sihuan Li, Dr.

Chengshuo Xu, Xiaolin Jiang, Dr. Kaiming Ouyang, Yuanlai Liu, Yujia Zhai, Quan Fan, Elisabeth Giem, Jinyang Liu, Jiajun Huang, Ziyang Jia, Jiannan Tian, Yafan huang, and Dr. Yuede Ji. I appreciate the help from all of them, and I would like to especially thank Dr. Tao, Dr. Liang, Dr. Li, and Dr. Ji for their enthusiastic help for my research and career.

Funding Acknowledgment I appreciate the funding supports from National Science Foundation (under Grants No. 1305624, No. 1513201, and No. 1619253), from the Exascale Computing Project 17-SC-20-SC (a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration), and from the U.S. Department of Energy Office of Science (under contract DE-AC02-06CH11357).

Publication Acknowledgment I acknowledge that part of the thesis has been published or released on line previously.

- Chapter 2 [114] was published in the proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, Stockholm, Sweden, June 23 - 26, 2020.

- Chapter 3 [112] was published in proceedings of the 37th IEEE International Conference on Data Engineering, Chania, Crete, Greece, Apr 19 - 22, 2021.

- Chapter 4 [115] was published in the proceedings of the 38th IEEE International Conference on Data Engineering, Virtual Event, May 9 - 12, 2022.

To my wife, Qian Wang, and my parents for all the support.

ABSTRACT OF THE DISSERTATION

Lossy Compression for Exascale Scientific Applications

by

Kai Zhao

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2022
Dr. Zizhong Chen, Chairperson

Today's scientific simulations are producing vast volumes of data that cannot be stored and transferred efficiently because of limited memory capacity, storage capacity, and network bandwidth. The situation is getting worse over time because of the ever-increasing gap between relatively slow data transfer speed and fast-growing computation power in modern supercomputers. Error-bounded lossy compression is becoming one of the most critical techniques for resolving the big scientific data issue, in that it can significantly reduce the scientific data volume while guaranteeing that the reconstructed data is valid for users because of its compression-error-bounding feature.

This thesis proposes three new lossy compressors for scientific applications across different domains. The first compressor exploits effective strategies by using 2nd-order regression and 2nd-order Lorenzo predictors to improve the prediction accuracy of SZ2 which is one of the best lossy compressors. It also contains an efficient approach to select the best-fit parameter setting, by conducting a comprehensive priori compression quality analysis and exploiting an efficient online controlling mechanism.

The second compressor uses a dynamic spline interpolation approach with a series of optimization strategies to further improve the compression quality. On the one hand, cubic spline interpolation is included to represent high order data variation, which obtains much higher prediction accuracy over linear regression for datasets with a high-order variation. On the other hand, we derive the constant coefficients in our interpolation approach such that the coefficient storage overhead can be completely eliminated. We further propose a dynamic optimization strategy to select the best predictor between the interpolation approach and the multilevel Lorenzo predictor to improve the overall compression quality.

The third compressor specifically targets molecular dynamics (MD) simulation data. MD simulations can produce a large volume of data because they could involve trillions of atoms for hundreds of millions of snapshots. Traditional lossy compressors are not optimized for MD applications because of MD data's trajectory type and irregular shape. We propose the MDZ compressor which contains three methods to fully leverage the data characteristics in both spatial and temporal domains. An adaptive solution is provided to automatically select the best-fit method during runtime.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the ever-increasing scale of today's scientific simulations, vast amounts of scientific data are produced at every simulation run. Climate simulation [47], for example, can generate hundreds of terabytes of data in tens of seconds. A cosmology simulation, such as Hardware/Hybrid Accelerated Cosmology (HACC) [37] can produce dozens of petabytes of data when it performs an N-body simulation with up to several trillion particles. Such a vast amount of scientific data needs to be stored for post hoc analysis, creating a huge challenge to the scientific data management systems [5, 20, 34, 49, 55]. Many scientists also need to share the large amounts of data across different sites (i.e., endpoints) through a data-sharing web service (such as the Globus toolkit [31]) on the Internet. Thus, the ability to significantly compress extremely large scientific datasets with controlled data distortion is critical to today's science work.

Compression techniques for scientific data have been studied for years. Lossless compressors are not suitable for scientific data in that the scientific data are composed mainly of floating-point values that involve disordered ending mantissa bits in their binary representations, such that few repeated patterns could be found in the data streams. Error-bounded lossy compression has been considered a promising solution because not only can it significantly reduce the data size (by $10\times$ or even $100\times$) but it can also strictly control the data distortion based on user-specified error bounds [113]. In fact, error-bounded lossy compressors have been broadly verified as helpful in saving storage space and improving I/O performance for many production-level applications across different domains, such as cosmology [46, 74], molecular dynamics [28], climate [40, 84], and quantum computing [50].

The SZ compression model has been recognized by independent assessments [70, 91, 101] as the best-in-class error-bounded lossy compressor for scientific datasets, especially because it has gone through many careful optimizations [25, 45, 58, 60, 91, 92, 98–100]. Our research objective is to significantly improve the compression quality of the SZ compression model for most of the datasets across from different domains. Such a research goal is challenging. On the one hand, SZ has been developed for many years, so its design and implementation have been tuned to a fairly optimized level, making further improvement to the compression quality difficult. On the other, many parameter settings (such as block size, dimension order, and regression order) are involved in the prediction-based compression model, making it nontrivial to select the best-fit combination to get the optimal compression quality, especially because of fairly diverse data characteristics in the datasets.

In this thesis, we leverage novel prediction methods and parameter optimization techniques to improve the compression quality for the SZ model. Our contributions can be summarized as follows.

- We propose novel prediction methods including 2nd-order Lorenzo, 2nd-order regression, and spline interpolation. On the one hand, cubic spline interpolation is included in our novel approach to represent high order data variation, which obtains much higher prediction accuracy over linear-regression for datasets with nonlinear data variation characteristics. On the other hand, we derive the constant coefficients in our interpolation approach such that the coefficient storage overhead can be completely eliminated. We further propose a dynamic optimization strategy to select the best predictor from between the novel spline interpolation approach and the multilevel Lorenzo predictor to improve the overall compression quality.

- We design an efficient approach selecting the best-fit parameter settings during the compression. Specifically, we perform a comprehensive priori compression quality analysis to filter out the inferior settings based on error bounds and data characteristics, and we then exploit an efficient online controlling mechanism to determine the best-fit setting at runtime.

- We carefully characterize a number of different MD simulation datasets and exploit some of the key patterns identified in the MD data to significantly improve compression ratios. We design an adaptive error-bounded lossy compressor for MD datasets which fully leverages the specific characteristics in both spatial and temporal dimensions. We integrate our solution into the MD package LAMMPS. Evaluation shows our solution has negligible time overhead in real-world MD simulations under different scales and settings.

Chapter 2

Enhance SZ2 With Second-Order Prediction and Parameter Optimization

2.1 Introduction

Error-bounded lossy compression can be categorized into two models: prediction-based or transform-based. In general, the former performs data prediction for each value in the dataset and then converts the floating-point values to integer quantization codes, followed by an entropy encoding [39] and dictionary coding [118]. SZ [25,60,91], FPZIP [66], and ISABELA [52] are three typical examples adopting the prediction-based model. The transform-based compression model performs orthogonal data transforms to convert the original dataset to another data domain and then removes insignificant values [83] or adopts embedded coding [64] to shrink the size. Typical examples are ZFP [64] and wavelet-based compression [83]. Much prior work [10, 25, 65] has demonstrated that SZ2 and ZFP are the two top error-bounded lossy compressors in most cases; however, none of them can always exhibit the best compression quality on all datasets.

In this chapter, we successfully leverage adaptive parameter optimization techniques with a series of optimization strategies on data prediction, which can significantly improve the compression quality for SZ2 with the same level of data distortion. Our contributions can be summarized as follows.

- We develop optimization strategies utilizing 2nd-order Lorenzo and 2nd-order regression prediction to improve the prediction accuracy significantly for SZ, such that the overall compression quality can be improved prominently in many cases.

- We design an efficient approach selecting the best-fit parameter settings during the compression. Specifically, we perform a comprehensive priori compression quality analysis to filter out the inferior settings based on error bounds and data characteristics, and we then exploit an efficient online controlling mechanism to determine the best-fit setting at runtime.

- We evaluate the compression quality and performance by running our new compression solution on a supercomputer with 4,096 cores, as compared with other state-of-the-art error-bounded lossy compressors.

The rest of the chapter is organized as follows. In Section 2.2, we discuss related work. In Section 2.3, we formulate the research problem. In Section 2.4, we provide an overview of our design and implementation. Section 2.5 and Section 2.6 describe our major solution (2nd-order prediction and parameter optimization) in detail. In Section 2.7, we present the evaluation results from using multiple real-world simulation datasets on a supercomputer.

2.2 Related Work

To mitigate the storage burden and I/O bottleneck presented by huge volumes of data, researchers have developed many data compressors. Lossless compressors [7,15,24,39, 116,118] can guarantee that reconstructed data suffer from no data distortion; however, they cannot significantly reduce the scientific data size because of the random ending mantissa bits in the floating-point values. Their compression ratios are usually around 2 [65, 81, 88], far from the desired level for large-scale scientific simulations running on modern HPC systems [17, 30].

In contrast, error-bounded lossy compressors have been effective in significantly reducing the science data volume for extreme-scale simulations while being able to strictly control the data distortion based on user requirements on pointwise compression errors. Two state-of-the-art models exist for error-bounded lossy compression: prediction-based [19,25,32,52,59,60,66,91] and transform-based [21,64,83,107]. SZ [25,60,91], ISABELA [52], FPZIP [66], and NUMARCK [19] are typical prediction-based compressors. Prior work [60] shows that SZ leads the compression quality among all the prediction-based compressors. SZ includes four key steps: data prediction, linear-scaling quantization, customized variable-length encoding, and dictionary encoding such as gzip [24] or zstd [118]. Vapor [21] and

ZFP [64] are typical transform-based compressors. They use different data transformation methods (wavelet transform and a customized (non)orthogonal transform, respectively) and different encoding algorithms. Recent research [64, 91] indicates that ZFP is one of the best error-controlled lossy compressors for scientific simulation datasets. ZFP compresses the dataset block by block (blocksize: 4×4 for 2D data and 4×4×4 for 3D data). Each block involves four steps: exponent alignment, fixed-point alignment, (non)orthogonal block transform to decorrelate the values, and embedded encoding of the ordered coefficients one "bit plane" at a time.

No existing error-bounded lossy compressor can always exhibit the best compression quality (or rate distortion) over all other compressors in most cases. Prior experiments [93], for example, show that neither SZ nor ZFP consistently provides the best compression results on the 13 fields of the Hurricane ISABEL dataset or on the 100+ fields of the CESM-ATM climate simulation dataset.

To address this issue, some researchers studied how to improve the compression quality by combining the two compression models intuitively. Lu et al. [70] concluded that SZ and ZFP were the two best error-bounded lossy compressors. The authors also proposed a solution to estimate the compression ratios for SZ and ZFP, respectively. In [93], they explored an online approach that can select the better strategy between SZ and ZFP in terms of peak signal-to-noise ratio (PSNR). This solution, however, [60] is subject to the existing compression quality and performance of SZ and ZFP. Moreover, the two related works both used the outdated version of SZ (SZ1.4), which exhibits much worse compression quality than does the latest SZ version (SZ2.0) [60]. Liang et al. [58] proposed a compression method that treats ZFP's data transform as one predictor (called a transform-based predictor) in the SZ compression model and selects the better one between SZ's built-in predictor and the transform-based predictor, which can prominently improve the compression quality beyond SZ and ZFP. Compared with all these works, we develop an efficient approach that can further improve the compression quality of SZ. Specifically, experiments with multiple real-world HPC simulation datasets show that our approach can improve the compression quality by 10%∼46% over the second-best approach in most cases.

2.3 Problem Formulation

Our objective is to significantly improve the compression quality for SZ. Similar to related work [58, 60, 91], we focus on structured datasets (i.e., 1D, 2D, or 3D structured mesh), because unstructured datasets (unable to be represented by a regular mesh grid) either need particular compression strategies [4] or are treated as 1D datasets for simplicity.

The error-bounded lossy compression problem can be formulated as follows: Given a structured mesh dataset (denoted by $D = \{d_1, d_2, \cdots, d_N\}$) with N floating-point data values, how can the data be compressed to obtain a high compression quality, while the reconstructed data (denoted D') still strictly respect user-specified pointwise error bounds? In the error-bounded lossy compression community, three ways have been formulated to assess compression quality in general.

1. Checking the compression ratio (which is defined as the ratio of the original raw data size to the compressed data size) based on the same error bound for different compressors.

2. Using rate distortion, a common indicator in the visualization community. Rate distortion involves two metrics: bit rate and data distortion. Bit-rate distortion is the average number of bits used to represent one data point after compression. The smaller the bit rate, the higher the compression ratio. Distortion is usually evaluated by using the peak signal-to-noise ratio, which is defined in Formula (2.1). In general, the higher the PSNR, the better the compression result.

$$PSNR = 20 \cdot \log_{10}(\max(d_i) - \min(d_i)) - 10\log_{10}(MSE(D, D')) \tag{2.1}$$

   where MSE stands for mean-squared error between D and D'. Rate distortion is arguably the most important indicator because some domain scientists care about the overall statistical errors, especially for visualization purposes.

3. Checking the visual quality of the reconstructed data compared with the original raw data, by aligning the compression ratios to the same level for different compressors. This method is also widely used by existing error-bounded lossy compression developers [21,25,52,64,66,83,91] and HPC application users [37,50,74,84].

We use all three assessment metrics for comparing our solution with other state-of-the-art lossy compressors such as SZ [25,91] and ZFP [64]. We will also evaluate the I/O performance of these compressors on a supercomputer.

2.4 Design Overview

We adopt the SZ compression model because it exhibits the best compression quality (rate distortion) from among the different compressors in literature and as confirmed by our experiments. SZ adopts four stages in the compression: data prediction, linear-scale quantization, Huffman encoding, and dictionary encoding (such as Zstd [118]). Here we focus mainly on how to improve the data prediction accuracy with as little overhead as possible and how to determine the best parameter settings for the overall compression. Our work involves only the prediction and quantization steps because the other steps involve lossless compression that already has optimized settings.

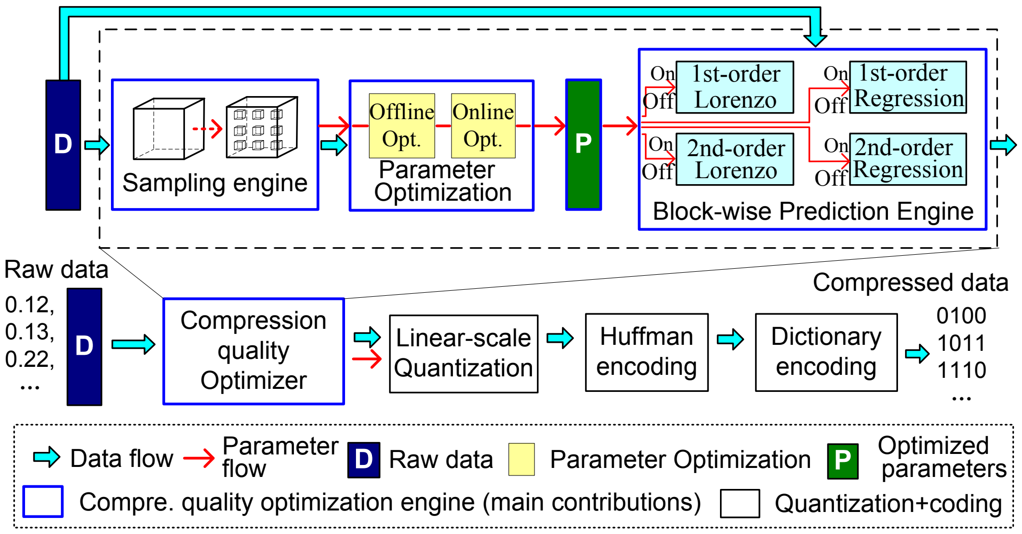


Figure 2.1: Design overview of our solution

We present the design overview of our method in Figure 2.1, in which we highlight the main contributions by purple rectangles. Specifically, we develop a compression quality optimizer that includes three key engines working systematically: sampling engine, parameter optimization engine, and blockwise prediction engine.

- *Sampling Engine.* The sampling engine is designed for significantly reducing the overall overhead of our compression quality optimization solution. At the compression runtime, our approach selects a small portion of the whole dataset by a uniform sampling method, and the subsequent steps (i.e., parameter optimization and blockwise selection) are performed on top of the sampled dataset.

- *Parameter Optimization Engine.* The parameter optimization engine addresses two critical issues: (1) how to estimate the overall compression ratio as accurately as possible based on the sampled dataset and (2) how to select the best-fit parameters as efficiently as possible. As for the first issue, simply assembling a new dataset with the uniformly sampled data blocks and performing lossy compression on top of it would cause a large deviation of the estimation (demonstrated later). Accordingly, we develop an effective method that can estimate the compression ratios accurately for various parameter settings. As for the second issue, we design a two-stage (offline and online) optimization strategy that can find the best-fit parameter setting with a fairly low time complexity at runtime. Details are given in Section 2.6.

- *Blockwise Prediction Engine.* Blockwise prediction is the most important step in our design. In addition to the traditional prediction method [25, 60, 91] (either 1st-order Lorenzo or 1st-order regression), we introduce two new prediction methods, 2nd-order Lorenzo and 2nd-order regression, which can improve the overall compression quality significantly. Based on the optimized parameter settings selected by the parameter optimization engine, the blockwise prediction engine checks the compression quality for each data block and selects the best choice from among the four prediction methods for each block. The 2nd-order prediction methods is detailed in Section 2.5, and how to select the best-fit prediction method is described in Section 2.6.

2.5 Second-Order Data Prediction

In addition to the original 1st-order prediction methods, we propose to use 2nd-order Lorenzo and 2nd-order regression prediction, which can significantly improve the compression quality.

2.5.1 Second-Order Lorenzo Prediction

Second-order Lorenzo prediction was proposed by other researchers conceptually in the literature. For instance, it was called *two-layer prediction* in [91]. However, no compressors are using this idea in practice because of its limitations (detailed later). For instance, the authors in [91] reported that they did not achieve higher prediction accuracy in their experiments with 2nd-order Lorenzo prediction. In our work, we combine 2nd-order Lorenzo prediction with other prediction methods to make it work effectively. In what follows, we review the 1st-order and 2nd-order Lorenzo predictor and then discuss the pros and cons of the two predictors and in what situations 2nd-order Lorenzo is better than 1st-order Lorenzo prediction.

We illustrate the 1st-order Lorenzo and 2nd-order Lorenzo prediction in Figure 2.2 (using a 2D dataset as an example). As shown in the figure, the 1st-order prediction involves 3 data points per data prediction while the 2nd-order prediction requires 7 nearby data points for predicting each value along the scanning order.

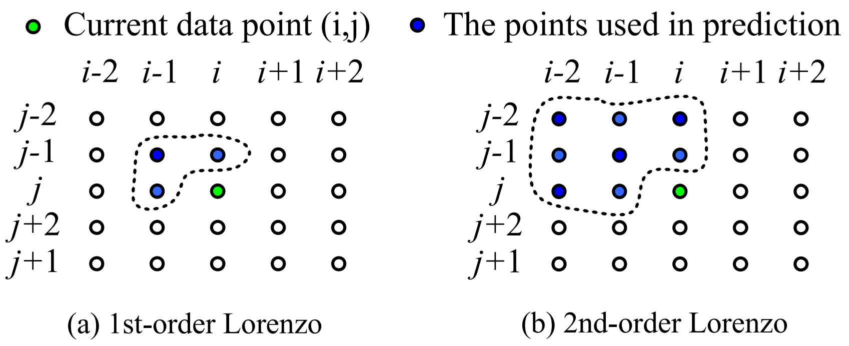


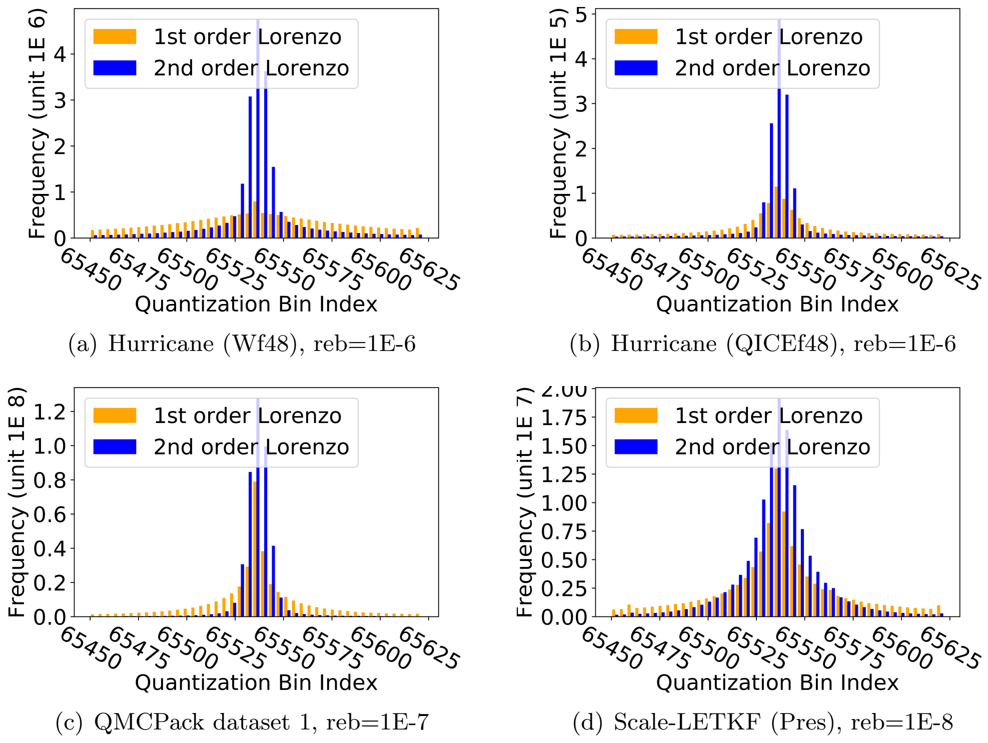
Figure 2.2: 1st-order Lorenzo predictor vs. 2nd-order predictor

In general, the more data points used, the higher the prediction accuracy will be. For example, the average prediction accuracy on the QMCPack dataset [50] is about 0.00197 and 0.00062 when using 1st-order and 2nd-order Lorenzo predictor, respectively. On the other hand, we note that SZ needs to use the decompressed data with biased values to do the prediction instead of the original data, in order to fully respect the preset error bound during the decompression. In this sense, the more data points involved, the more the compression errors impact the prediction accuracy, causing a lower prediction accuracy. Tao et al. [91] demonstrated that 2nd-order Lorenzo prediction does not work as well as the 1st-order Lorenzo, so they adopted only the 1st-order Lorenzo in the released SZ compressor.

We note, however, that 2nd-order Lorenzo prediction may significantly improve the compression ratio, especially when the error bound is required to be relatively low. Figure 2.3 demonstrates the frequency distribution of quantization bins generated by the 1st-order and 2nd-order Lorenzo predictors with the same compression error bound for four example datasets. In principle, the sharper the distribution is, the higher the compression ratio will be. We observe that when the relative error bound[1] is set to the order of 1E-6∼1E-8, the

[1]Relative error bound here refers to value-range-based error bound, which is defined as the ratio of absolute error bound to the data value range.

2nd-order Lorenzo predictor turns out to be better than the 1st-order Lorenzo. The key reason is discussed as follows. SZ has to perform the data prediction using decompressed data each with certain errors, which may impact the prediction accuracy in turn. If the error bound is small enough, the impact of decompressed data to the prediction accuracy will be very small. This result is also verified by our evaluation of the percentage breakdown of different predictors used in compression (discussed in Section 2.7).



(a) Hurricane (Wf48), reb=1E-6

(b) Hurricane (QICEf48), reb=1E-6

(c) QMCPack dataset 1, reb=1E-7

(d) Scale-LETKF (Pres), reb=1E-8

Figure 2.3: Frequency distribution of quantization bins between 1st- and 2nd-order Lorenzo prediction

2.5.2 Second-Order Regression-Based Prediction

In this subsection, we describe how we design the 2nd-order regression predictor in terms of the 2nd-order polynomial multivariate regression. The basic idea is constructing a 2nd-order regression hyperplane based on the coordinates and the values of all data in a specific data block and minimizing the mean squared error by derivation. In what follows, we first discuss the generic formula and then extend it to fit the blockwise design in compression.

The generic formula can be derived based on a m-dimensional dataset ($n_1 \times n_2 \times \cdots \times n_m$). The independent variable vector of the m-dimensional dataset is denoted as $\boldsymbol{x} = (x_1, x_2, .., x_m)$. Its corresponding dependent variable vector is $f_{\boldsymbol{x}}$. We use $f^{2r}(\boldsymbol{x})$ to denote the prediction value of $\boldsymbol{x}$ by 2nd-order regression. $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, ..\beta_{m_2})$ represents the coefficient vector in which β_0 is the intercept coefficient. We denote the total number of coefficients by $m_2 = m \times (m+1)/2 + m + 1$.

$$\begin{gathered} f^{2r}(\boldsymbol{x}) = t(\boldsymbol{x})^T \boldsymbol{\beta}, \\ where t(\boldsymbol{x}) = (1, x_1, x_2, \cdots, x_m, \\ x_1^2, x_1x_2, x_1x_3, \cdots, x_1x_m, \\ x_2^2, x_2x_3, \cdots, x_2x_m, \\ \cdots, x_mx_m), \end{gathered} \tag{2.2}$$

The 2nd-order regression predictor uses $f^{2r}(\boldsymbol{x})$ to estimate the dependent variable $f_{\boldsymbol{x}}$. The objective is to minimize the mean squared error between $f^{2r}(\boldsymbol{x})$ and $f_{\boldsymbol{x}}$, as shown in Equation (2.3).

$$f_{obj} = \arg\min_{\beta} \sum_{\forall \boldsymbol{x}} (t(\boldsymbol{x}^T)\boldsymbol{\beta} - f(\boldsymbol{x}))^2 \tag{2.3}$$

This objective function is hard to solve with a closed-form solution because of the unknown dimension m. Thus, we resolve it for each specific dimension separately. For simplicity, we describe our solution using a 3D dataset case, which can be extended to other dimensions easily without loss of generality.

For a 3D dataset, (x_1, x_2, x_3) in Formula (2.2) can be replaced by coordinates (i, j, k), where $0 \le i < n_1, 0 \le j < n_2, 0 \le k < n_3$. Then, the objective function can be simplified to

$$\begin{gathered} f_{obj} = \arg\min_{\beta} \textstyle\sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \sum_{k=0}^{n_3-1} (\beta_0 + \beta_1 i + \beta_2 j + \beta_3 k \\ + \beta_4 i^2 + \beta_5 ij + \beta_6 ik + \beta_7 j^2 + \beta_8 jk + \beta_9 k^2 - f_{ijk})^2. \end{gathered} \tag{2.4}$$

It can be solved by setting all partial derivatives to 0.

$$\textstyle\sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \sum_{k=0}^{n_3-1} \boldsymbol{A} \boldsymbol{\beta}^T = \boldsymbol{V}^T \tag{2.5}$$

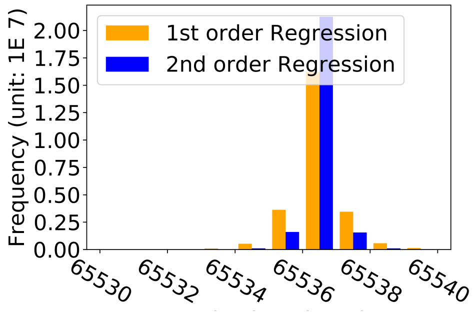
$$A = \begin{bmatrix} 1 & i & j & k & i^2 & ij & ik & j^2 & jk & k^2 \\ i & i^2 & ij & ik & i^3 & i^2j & i^2k & ij^2 & ijk & ik^2 \\ j & ij & j^2 & jk & i^2j & ij^2 & ijk & j^3 & j^2k & jk^2 \\ k & ik & jk & k^2 & i^2k & ijk & ik^2 & j^2k & jk^2 & k^3 \\ i^2 & i^3 & i^2j & i^2k & i^4 & i^3j & i^3k & i^2j^2 & i^2jk & i^2k^2 \\ ij & i^2j & ij^2 & ijk & i^3j & i^2j^2 & i^2jk & ij^3 & ij^2k & ijk^2 \\ ik & i^2k & ijk & ik^2 & i^3k & i^2jk & i^2k^2 & ij^2k & ijk^2 & ik^3 \\ j^2 & ij^2 & j^3 & j^2k & i^2j^2 & ij^3 & ij^2k & j^4 & j^3k & j^2k^2 \\ jk & ijk & j^2k & jk^2 & i^2jk & ij^2k & ijk^2 & j^3k & j^2k^2 & jk^3 \\ k^2 & ik^2 & jk^2 & k^3 & i^2k^2 & ijk^2 & ik^3 & j^2k^2 & jk^3 & k^4 \end{bmatrix}$$

$$\boldsymbol{V} = (V_1, V_2, .., V_9), V_t = \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \sum_{k=0}^{n_3-1} g_{ijk}(t) * f_{ijk}, 0 \le t \le 9$$

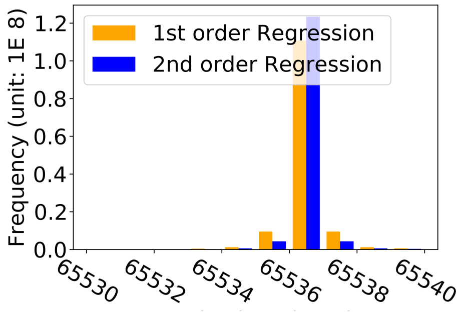
$g_{i,j,k}(t)$ returns tth element from list $[1, i, j, k, i^2, ij, ik, j^2, jk, k^2]$

Denote $f^A_{n_1,n_2,n_3} = \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \sum_{k=0}^{n_3-1} A$. The solution $\boldsymbol{\beta}$ equals $(f^A_{i,j,k})^{-1} \boldsymbol{V}^T$. Since $f^A_{n_1,n_2,n_3}$ is fixed under given (n_1, n_2, n_3), its inverse matrix can be calculated beforehand. During the process of 2nd-order regression, we just need to compute $\boldsymbol{V}$ followed by a matrix-vector multiplication to get the solution $\boldsymbol{\beta}$ with the optimized coefficients. Based on the optimized coefficients, the prediction value for each data point (i, j, k) can be written as $f^{2r}(i, j, k) = \beta_0 + \beta_1 i + \beta_2 j + \beta_3 k + \beta_4 i^2 + \beta_5 ij + \beta_6 ik + \beta_7 j^2 + \beta_8 jk + \beta_9 k^2$.

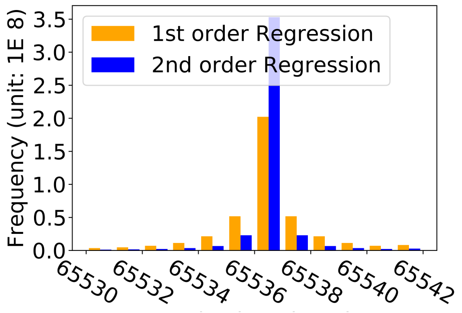
Figure 2.4 demonstrates the frequency distribution of quantization bins generated after the 1st-order regression predictor versus the 2nd-order regression predictor with the same compression error bound for four example datasets. The 2nd-order regression exhibits



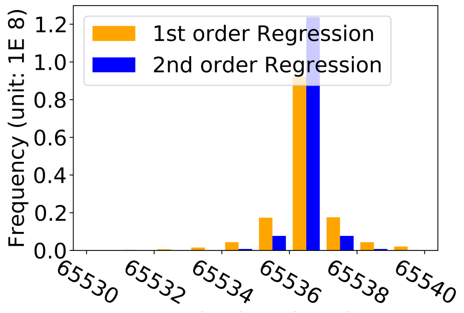
(a) Hurricane(TCf48), reb=1E-3



(b) Nyx(velocityz), reb=1E-3



(c) QMCPack dataset 1, reb=1E-4



(d) Scale-LETKF(W), reb=1E-3

Figure 2.4: Frequency distribution of quantization bins between 1st- and 2nd-order regression prediction

sharper distribution than does the 1st-order regression, which means that the 2nd-order regression will likely obtain higher compression ratios in these cases.

Next we compress the 10 coefficients ($\beta_0 \sim \beta_1 0$) used to construct the hyperplane. Specifically, we compress them using the SZ compressor, because this can lead to outstanding compression ratios that other compressors such as FPZIP [66], ZFP [64], or bit-truncation methods [33] cannot achieve.

2.6 Parameter Optimization

In this section, we describe another important contribution, which can further improve the compression ratios prominently.

The key idea is to optimize the parameter settings involved in the whole compression. This is motivated by our observation that different parameter settings (such as block size, number of quantization bins) may affect the compression quality.

Based on our new compression design supporting 2nd-order prediction, we summarize a total of 12 critical parameters for the whole compression. Five of them are from SZ (version 2.0), as shown in Table 2.1, and the other 7 parameters are based on the 2nd-order prediction we designed, as shown in Table 2.2.

From among the 7 parameters related to the 2nd-order prediction, four of them are of Boolean values used to control the four prediction methods (1st-order/2nd-order + Lorenzo/regression). For example, if *use_lorenzo* is set to false, the Lorenzo predictor will be excluded in the whole process of blockwise prediction. The other three parameters are used to control the compression of the coefficients for the 2nd-order regression.

Table 2.1: SZ parameters

Type	**Name**	**Explanation**
Input	data	Original data
Input	dim	The dimension of original data
Input	reb	Value range based relativity error bound
Param	block_size	Block size used by predictors
Param	pred_dim	The dimension used by Lorenzo and regression predictors, pred_dim $\leq$ dim
Param	quan_bins	Number of bins used in quantization algorithm
Param	reg_coef_intercept	Error bound for compressing the intercept coefficient of regression predictor
Param	reg_coef_linear	Error bound for compressing the linear coefficients of regression predictor

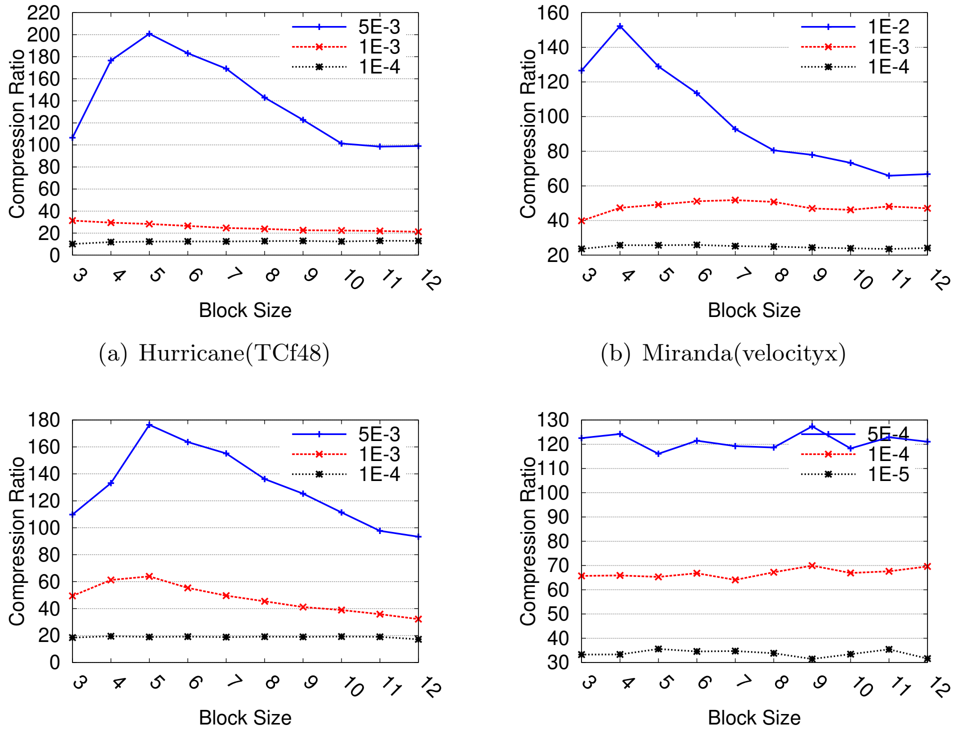
The 12 parameters are determined by our in-depth analysis of their impact on the compression ratios based on experiments using 5 real-world simulation datasets each

Table 2.2: Extended parameters in our solution

Name	**Explanation**
enable_lorenzo	Enable Lorenzo predictor or not
enable_2ndlorenzo	Enable 2nd-order Lorenzo predictor or not
enable_regression	Enable regression predictor or not
enable_2ndregression	Enable 2nd-order regression predictor or not
2ndreg_coef_intercept	Error bound for compressing the intercept coefficient of 2nd-order regression predictor
2ndreg_coef_linear	Error bound for compressing the linear coefficients of 2nd-order regression predictor
2ndreg_coef_poly	Error bound for compressing the polynomial coefficients of 2nd-order regression predictor

with multiple time steps, involving about 100 fields and thousands of data files in total. Different settings of these parameters may lead to largely different compression ratios. We demonstrate three examples in Figure 2.5, Figure 2.6, and Figure 2.7. For instance, based on Figure 2.5(a), SZ's compression ratio is 180:1 and 100:1 on Hurricane(TCf48) with the error bound 5E-3, when its block size is set to 5 and 11, respectively. In Figure 2.6, none of *pred_dim* = 2 or *pred_dim* = 3 can always exhibit the best compression ratio when the error bound is between 1E-3 and 1E-4. In Figure 2.7, 8192 is the best setting for quan_bin to compress the Hurricane dataset with a 1E-5 error bound. However, in order to compress the same dataset with 1E-7 error bound, the best setting for *quan_bin* is 1024.

In the following text, we describe the detailed optimization strategies, including optimization of estimating compression quality by sampled datasets, offline parameter optimization, and online parameter optimization.

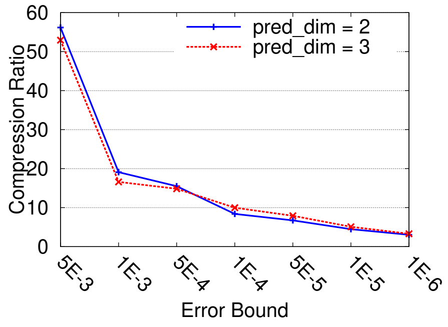


(a) Hurricane(TCf48)
(b) Miranda(velocityx)
(c) QMCPack dataset 1
(d) Scale-LETKF(QI)

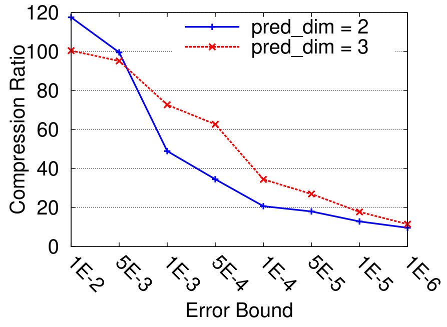
Figure 2.5: Change of compression ratios with block sizes

2.6.1 Optimizing Compression Quality Estimation Over Sampled Dataset

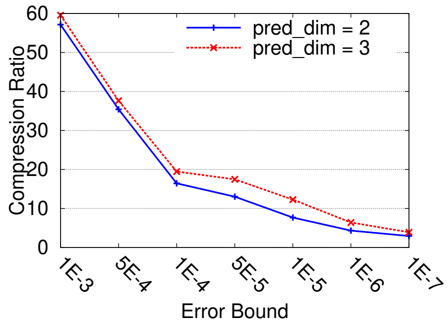
Accurately estimating the compression quality based on the sampled dataset is critical to selecting the best-fit parameter settings and predictors at runtime. To this end, we design an approach that takes into account how the data will be predicted and quantized for each block, in that the existing compression quality estimation methods are not suitable for our case. Lu et al. [70], for example, proposed a sampling-based estimation method based on the distribution of quantization bins, which can estimate the compression ratios of SZ to a certain extent. Since this method can support only Huffman encoding but not dictionary encoding (zstd), it cannot be applied to our estimation. Moreover, since



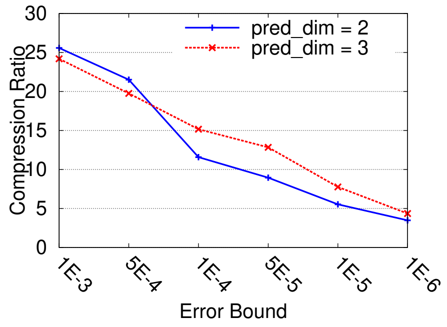
(a) Hurricane(Uf48)



(b) Miranda(density)



(c) QMCPack dataset 1



(d) Scale-LETKF(V)

Figure 2.6: Change of compression ratios with various prediction dimensions

it is designed based on SZ 1.4, which has no regression predictor, it cannot estimate the compression ratios accurately for SZ2.

Another idea is adopting a black-box compression quality estimation method by ignoring the detailed compression principles. In order to control the overhead, it needs to estimate the real compression ratio for the overall dataset based on the sampled datasets. That is, one can estimate the compression ratios by simply assembling a new dataset using the sampled data blocks and compressing the assembled dataset by a particular compressor such as SZ 2.0. Such a black-box estimation method, however, may easily cause biased estimation of compression ratios because it totally ignores the compression principle.

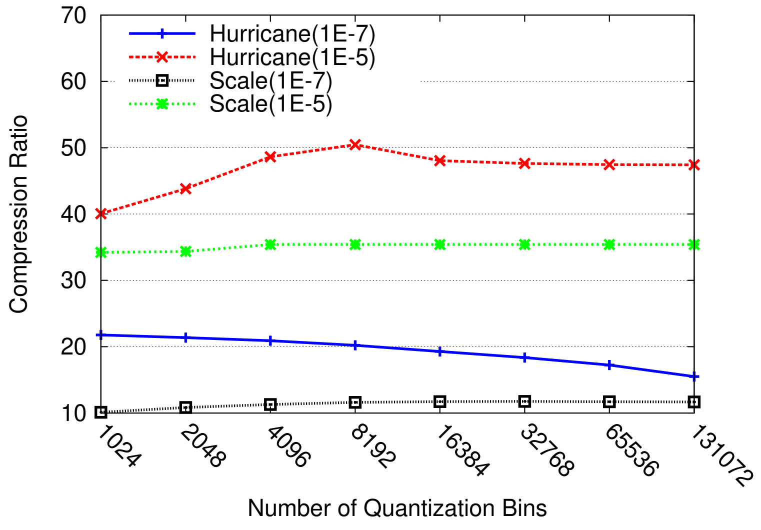
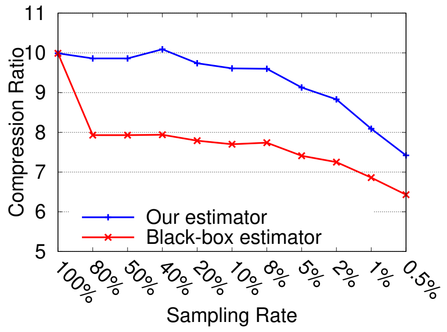


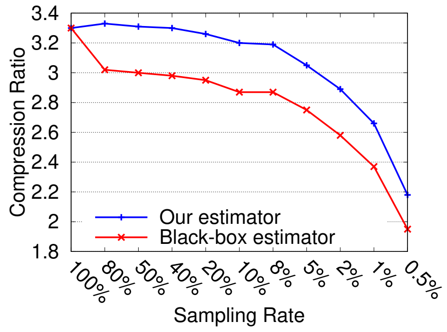
Figure 2.7: Change of compression ratios with numbers of quantization bins (Hurricane (QCLOUDf48) and Scale (QI))

Unlike the simple black-box estimation method, we take into account how the data will be used in the compression steps. Specifically, we ensure that the sampled block size is consistent with the block size to be used in the compression steps. Our estimation method also leverages the data points from other adjacent blocks to estimate the prediction accuracy in each compression block.

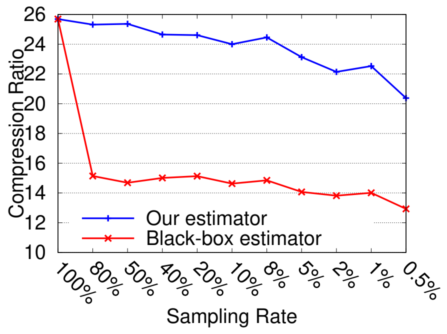
Figure 2.8 presents the significant improvement of our compression-principle-based estimation method over the black-box estimation method. We can clearly see that even under a small sampling rate 8%, the compression ratios can be estimated accurately, with only about 5% estimation errors in most cases. Accordingly, we set the sampling rate to 8% in our experiments.



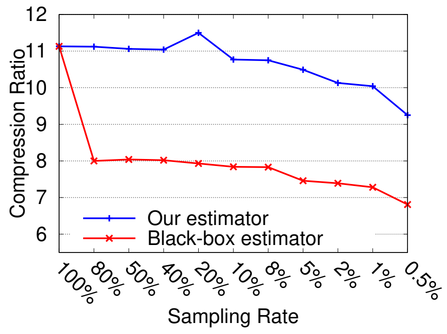
(a) Hurricane(Uf48), reb=1E-4



(b) Hurricane(Uf48), reb=1E-6



(c) Miranda(velocityy), reb=1E-4



(d) Miranda(velocityy), reb=1E-5

Figure 2.8: Comparison of estimation accuracy (sampling rate refers to the fraction of sampled data to the full data; sampling rate = 100% refers to the full dataset)

2.6.2 Offline Parameter Optimization

Finding the best parameter combination for a given dataset is a multivariable optimization problem. The objective is to find the maximum compression ratio using the same compression function and original data. Gradient-based algorithms such as gradient descent are difficult to apply for this problem since it is unclear whether the compression function itself is a differentiable function; also, the derivative is hard to obtain even if this function is differentiable. Derivative-free methods such as coordinate descent and (meta)heuristic methods such as simulated annealing, genetic algorithm, and ant colony optimization could be used to find the approximate global optimization in a large search space if the derivative is

Table 2.3: Range of parameters

Name	**Value Range**	**Values to be Tested**	**Outstanding Candidates**
enable_lorenzo	[True,False]	True, False	True
enable_2ndlorenzo	[True,False]	True, False	True, False
enable_regression	[True,False]	True, False	True
enable_2ndregression	[True,False]	True, False	True, False
pred_dim	[1,2,3]	1,2,3	2,3
block_size	$\mathbb{N}$	3,4,5,6,7,8,9,10,11,12,15,20,25,30	4,5,6,7,8
quan_bins	$\mathbb{N}$	s[1], 4096, 8192, 16384, 32768, 65536, 131072	s[1], 16384
reg_coef_intercept	$\mathbb{R}^+$	0.01, **0.1**, 0.5, **1**, 2, 5, 10, 20, 50, 100	1
reg_coef_linear	$\mathbb{R}^+$	0.01, 0.1, 0.5, **1**, 2, 5, **b**[2], 10, 20, 50, 100	b[2]
2ndreg_coef_intercept	$\mathbb{R}^+$	0.01, **0.1**, 0.5, **1**, 2, 5, 10, 20, 50, 100	0.1
2ndreg_coef_linear	$\mathbb{R}^+$	0.01, **0.1**, **0.5**, 1, 2, 5, 10, 20, 50, 100	0.5
2ndreg_coef_poly	$\mathbb{R}^+$	0.01, 0.1, 0.5, **1**, **2**, 5, 10, 20, 50, 100	2

[1] s means use the estimation value provided by SZ
[2] b means use block_size as reg_coef_linear
Bold values in column 3 are used in the first step of manual parameter search.

unknown or nonexistent. However, those (meta)heuristic algorithms are all time-consuming (generally requiring 20+ iterations to converge a near-global optimum solution [104]). By contrast, we need to control the number of iterations to under 20 such that the analysis overhead can be limited within 100% when the sampling rate is set to 8%. To this end, we propose an offline + online parameter optimization method.

The offline algorithm manually searches for the best parameters by testing as many parameter combinations as possible and analyzes the data generated by this process to get the best candidates. We manually tested more than 30K combinations of parameters and analyzed all the results to get the best candidate parameters. For parameters with discrete numbers (such as *pred_dim*), we evaluate all the possible values. For the parameters with continuous numbers (such as block_size or the error bounds of compressing regression coefficients), we evaluate 10+ values for each parameter; those values are actually outstanding settings based on our numerous experiments with many datasets. That is, the values outside this range are unlikely to achieve a good compression quality, based on our experience.

The pseudocode of the manual parameter search algorithm is demonstrated in Algorithm 1. In the first step (lines 2–6), the goal is to optimize the parameters with priority on the 1st-/2nd-order Lorenzo predictor. Two value sets are used for regression-related parameters, decided by our prior experience. In the second step (lines 7–9), the goal is to optimize the parameters of the 1st-order regression predictor: $reg\_coef\_intercept$, and reg_coef_linear. The third step (lines 10–12) is to optimize the parameters of the 2nd-order regression predictor: $2ndreg\_coef\_intercept$, $2ndreg\_coef\_linear$, $2ndreg\_coef\_poly$. The final step (lines 13–15) is to optimize the *quan_bin* since it is independent of predictors.

Algorithm 1 Manual parameter search

```
Input: raw data D, relative error bound reb
Output: list of parameter settings and its compression ratio

 1: compressMode ← no_sampling
 2: for (enable_lorenzo, enable_2ndlorenzo, enable_2ndregression, pred_dim, block_size) in (values from
    Table 2.3) do
 3:     for (reg_coef_intercept, reg_coef_linear, 2ndreg_coef_intercept, 2ndreg_coef_linear, 2ndreg_coef_poly) in
        (bold values from Table 2.3) do
 4:         Do compression, Record parameter settings and compression ratio
 5:     end for
 6: end for
 7: for (enable_regression, block_size, reg_coef_intercept, reg_coef_linear) in (values from Table 2.3) do
 8:     Do compression, Record parameter settings and compression ratio
 9: end for
10: for (enable_2ndlorenzo, block_size, 2ndreg_coef_intercept, 2ndreg_coef_linear, 2ndreg_coef_poly) in (val-
    ues from Table 2.3) do
11:     Do compression, Record parameter settings and compression ratio
12: end for
13: for quan_bin in (Values from Table 2.3) do
14:     Do compression, Record parameter settings and compression ratio
15: end for
```

We analyze the data generated by a manual parameter search to get the outstanding candidates. The manual search was conducted offline; that is, it does not involve runtime overhead for compression. The manual search results are maintained separately based on data fields. For each field, we first identify the best compression ratio (denoted as *best_ratio*) and then collect good parameter settings whose compression ratios are larger than 95% $\times$ *best_ratio*. Having gleaned relatively good parameter combinations for each field, we can choose any one parameter combination selected and use it to do compression, which can achieve at least a 95% top compression ratio. Then we collect the outstanding candidates for each individual parameter statistically based on a prior probability. Specifically, if some parameter value appears frequently (larger than 85%), we put it in the outstanding-candidate set. For instance, if the parameter *block_size*=5 appears in the good candidate parameter combinations for 86 fields from among 100 fields, we choose it as one of the outstanding candidates. The final results are shown in the last column of Table 2.3. By this selection method, we considerably reduce the number of parameter values to be focused on during online parameter optimization.

2.6.3 Online Parameter Optimization

Our solution searches the best parameters based on the outstanding candidates generated by the offline optimization. This auto parameter search process is an online process, which means it will be executed every time when we run the compressor. The subsets generated by sampling are used to find the best parameters. After that, the original datasets will be compressed by our compressor using the best parameters.

Parameters with multiple outstanding candidates in Table 2.3 will be evaluated to find the best setting. There are 5 parameters that need to be evaluated and we clarify them to 3 groups: *pred_dim* and *enable_2ndlorenzo* as group 1, *block_size* and *enable_2ndregression* as group 2, *quan_bins* as group 3. The evaluation is performed group by group since parameters between groups have little correlation in terms of the compression process. To find the best settings, Group 1, 2, and 3 require 4, 10, and 2 iterations respectively, according to the number of outstanding candidates of each parameter in Table 2.3. Thus, there are 16 iterations in total in our auto parameter search to choose the best setting regarding the first 5 parameters. The remaining 7 parameters have only one outstanding candidate each; thus, they do not need to be optimized during this step. Using a heuristic algorithm such as simulated annealing or a derivative-free algorithm such as coordinate descent is unnecessary for the auto parameter search because there are only 16 iterations in total which is already efficient.

Although the online auto parameter search performs on top of the outstanding candidates generated by an offline parameter optimization, this solution is also efficient on new datasets, as we verify in Section 2.7.4.

2.7 Performance Evaluation

In this section, we present the evaluation results based on the datasets produced by five real-world scientific simulations from different domains.

2.7.1 Experimental Settings

Table 2.4: Applications used in our experiments

Name	Domain	# Fields	Size Per Snapshot
Hurricane [40]	Weather	13	1.3 GB (= 13× 96MB)
Miranda [73]	Hydrodynamics	7	1 GB (= 7 × 144MB))
QMCPack [50]	Atom/Molecules	4	∼17 GB (=0.6 + 3.4 +13) GB
Scale-LETKF [84]	Weather	12	6.4 GB (=12×539MB)
NYX [74]	Cosmology	6	3.1 GB (=6×512MB)

Table 2.4 describes the five applications, which all require compression techniques to store big science data [37,50,74,84]. In particular, QMCPack here involves three datasets that are stored in three scales—288×115×69×69 (1 field), 816×115×69×69 (2 fields), and 6192×115×69×69 (1 field)—corresponding to 0.6 GB, 3.4 GB, and 13 GB, respectively. We call them QMCPack dataset 1, QMCPack dataset 2, and QMCPack dataset 3, respectively. Since our experiments involve parallel processes each with several gigabytes, the de facto total data size is up to 10+ terabytes for one application in our experiments, when the execution scale is 4,096 cores.

We conducted our experiments on the Bebop supercomputer [12] at Argonne National Laboratory using up to 4,096 cores. Specifically, the experiments involve 64∼128 nodes, and each node is equipped with 128 GB memory and two Intel Xeon E5-2695 v4 processors (each with 16 cores). Its storage system adopts a General Parallel File System (GPFS) equipped with 2 I/O nodes, and the I/O system is a typical high-end supercomputer facility. We perform data writing/reading by a file-per-process method with POSIX I/O [109] in parallel.[1]

We compare our solution with three state-of-the-art lossy compression methods: SZ2.1.8 [60], ZFP0.5.5 [64], and a hybrid model [58], which have been confirmed as the best in class [58, 70]. The hybrid model merges the SZ2.0 and ZFP0.3.1 to get the best compression quality, while suffering from 200% time overhead [58].

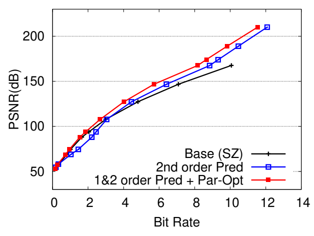
In what follows, we first present the compression quality results based on second-order prediction and parameter optimization and then present the overall compression qual-

[1]Another researcher [103] verified that POSIX I/O has comparable performance with parallel I/O, such as MPI-IO [95] when reading/writing thousands of files simultaneously on GPFS. We also further verified that the read/write performance difference of POSIX IO and MPI-IO is within ±10% on this supercomputer, when the execution scales between 2k cores and 8k cores.

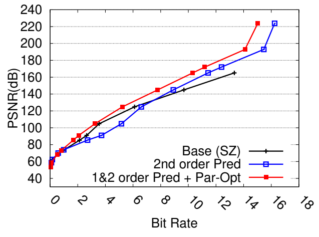
ity in terms of the indicators defined in our problem formulation (Section 2.3). We also evaluate the I/O performance gain by running a series of parallel experiments on a supercomputer with up to 4,096 cores, and compare the results with those of other existing state-of-the-art compressors.

2.7.2 Assessment of Second-Order Prediction

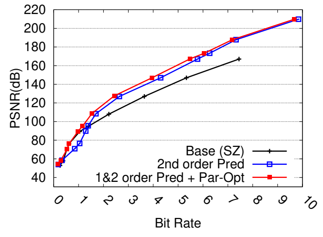
In Figure 2.9, we present the rate distortion improvement obtained with the second-order prediction (shown as blue curves in the figure) over the original design in SZ 2.0 (called Base(SZ) and shown as black curves in the figure) that uses the 1st-order prediction. As mentioned in Section 2.3, the higher the PSNR, the better the compression quality; and the lower the bit rate, the higher the compression ratio. We can clearly see that using 2nd-order prediction (see Section 2.5) can significantly improve the compression quality over the original SZ with 1st-order prediction in many cases, especially with relatively high bit rates or relatively high precision. For instance, the compression ratios can be improved by about 50% when the PSNR is greater than 120 dB for the Miranda and QMCPack simulation. The main reason is the high-order nature of the datasets. However, we can also see that at some bit rates, the original SZ with 1st-order prediction outperforms the one with 2nd-order prediction. As shown in Figure 2.9, for instance, 1st-order prediction is much better than 2nd-order prediction when the bit rate is in the range of [1.5,5] for the Scale-LETKF(Pres) field. This result provides motivation for adopting both 1st- and 2nd-order predictions in the compression.



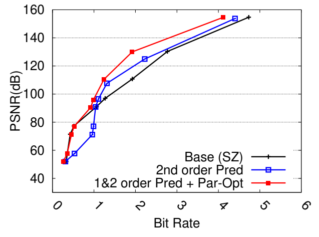
(a) Hurricane



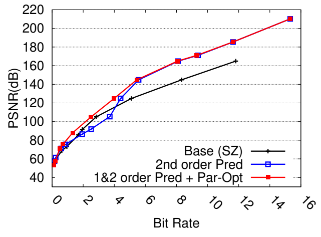
(b) Hurricane(Pf48)



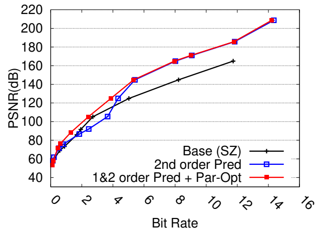
(c) Miranda



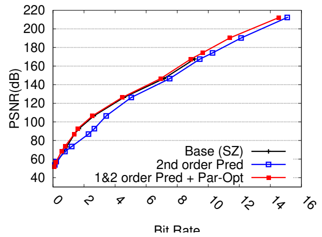
(d) Miranda(density)



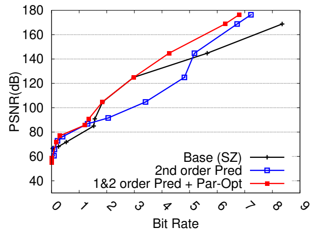
(e) QMCPack - dataset 1



(f) QMCPack - dataset 2



(g) Scale-LETKF



(h) Scale-LETKF(Pres)

Figure 2.9: Breakdown compression quality analysis

2.7.3 Assessment of Parameter Optimization

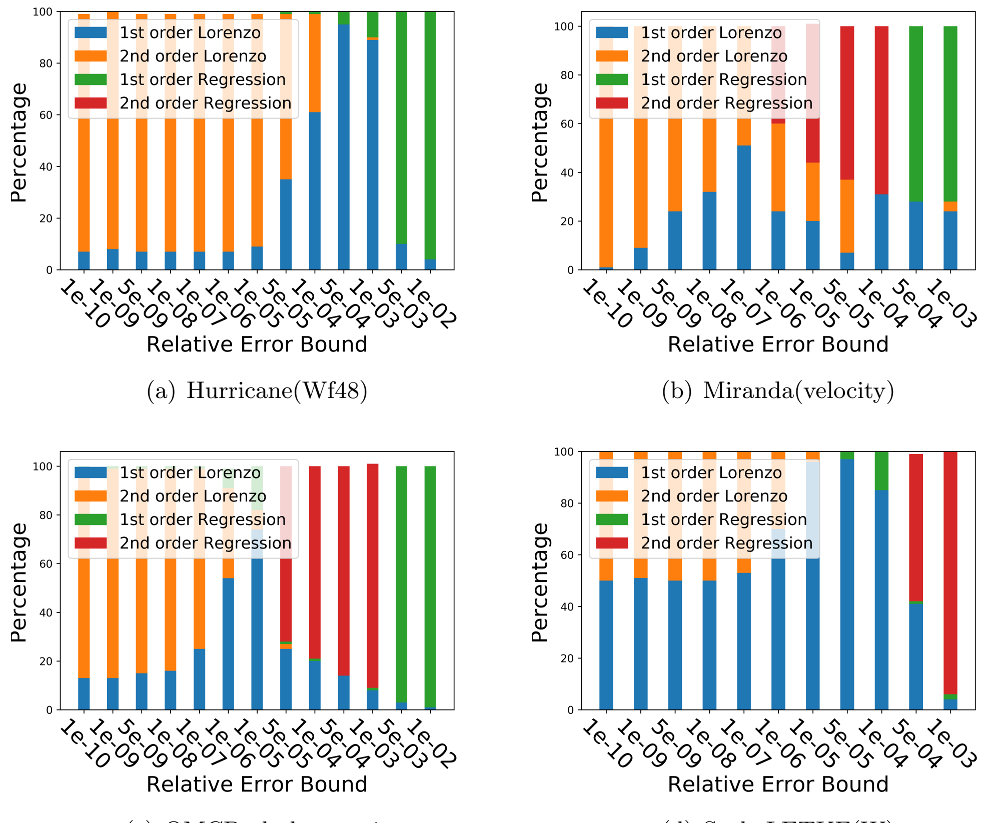
In Figure 2.9, we also demonstrate the further compression quality improvement (see the red curves versus the blue curves) when using parameter optimization strategies on top of 2nd-order prediction. In absolute terms, the rate distortion can be improved by 4%∼50% in most cases, depending on the bit rates. Such a significant improvement is attributed to our design of integrating both 1st-order prediction and 2nd-order prediction (four predictors in total) and an efficient online parameter optimization strategy selecting the best-fit parameter setting at runtime in fine granularity (as per block) (for details, see

Section 2.6.1 and Section 2.6). The variation in the rate distortion improvement shows that the default parameter settings of the original SZ is nearly optimal for some datasets while it is far from the optimal level for some other datasets. This confirms the significance of our parameter optimization in order to achieve the optimal results for all datasets.

To demonstrate the effect of our parameter optimization engine, in Figure 2.10 we illustrate the percentage breakdown of the four different prediction methods used in the compression of different applications or fields. We clearly observe an interesting distribution pattern of the four prediction methods in terms of different error bounds. Specifically, when the error bound is relatively large (such as 1E-2), the regression-based predictor would take a major role, since the Lorenzo predictor may suffer from huge prediction errors in this situation because of the impact of decompressed data (keep in mind that Lorenzo prediction has to be performed by using decompressed data during the compression stage). When the error bound is relatively small, the Lorenzo prediction would outperform the regression-based prediction. In particular, when the error bound is extremely small, our optimization engine selects the 2nd-order Lorenzo predictor in most blocks. This action is consistent with our analysis in Section 2.5.1: many of the application datasets actually exhibit high-order smoothness, such that the 2nd-order Lorenzo predictor is more accurate for data prediction, especially with small compression error bounds.

2.7.4 Overall Compression Quality

In Figure 2.11 we present the overall compression quality (rate distortion) based on five real-world scientific simulation datasets, and we demonstrate the result of one exam-



(a) Hurricane(Wf48)

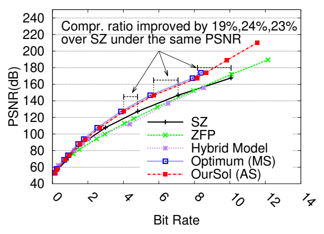
(b) Miranda(velocity)

(c) QMCPack dataset 1

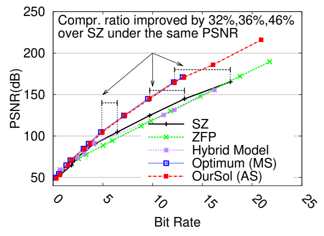
(d) Scale-LETKF(W)

Figure 2.10: Percentage breakdown of four predictors used in the blockwise compression

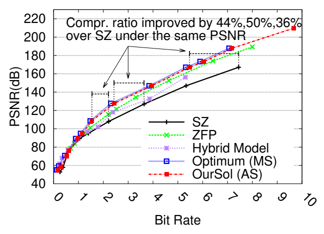
ple field for Hufficane ISABEL, Miranda, and Scale-LETKF, respectively. The blue curve (called optimum) refers to the ideal level obtained by our offline parameter searching (MS) for optimal parameters. As highlighted in the figures, our compression solution can improve the compression ratios over SZ (see red curve versus black curve) by 20+% for Hurricane, by ∼40+% for Miranda, and by ∼30+% for QMCPack, respectively, with the same PSNR. Our solution also exhibits the best compression quality from among all existing compressors on the three applications. Specifically, with the same PSNR, its overall compression ratio is higher than that of the second best compressor generally by 20∼25% and by 5∼10% and 20∼30% for the three applications, respectively. For some specific fields, the improvement can be up to 46%, as shown in Figure 2.11(b). As for the simulation Scale-LETKF and NYX, our solution still leads to the best compression quality from among all the compressors, although it has no prominent improvement over the second-best compressor, probably because the default parameter setting of the original SZ is also (or nearly) the best choice in those cases.



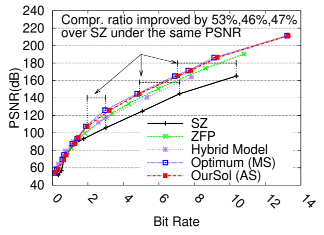
(a) Hurricane (overall)



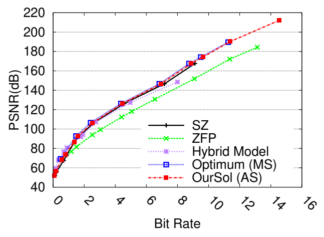
(b) Hurricane (Wf48)



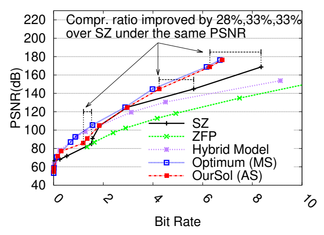
(c) Miranda (overall)



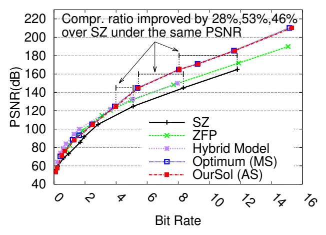
(d) Miranda (velocityx)



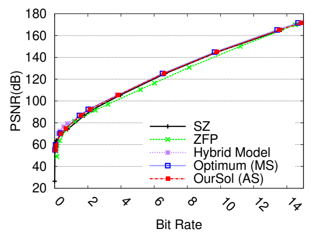
(e) Scale-LETKF (overall)



(f) Scale-LETKF (Pres)



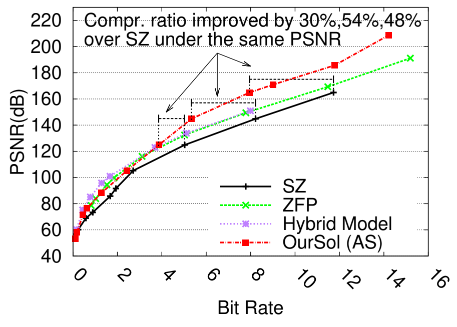
(g) QMCPack dataset 1 (overall)



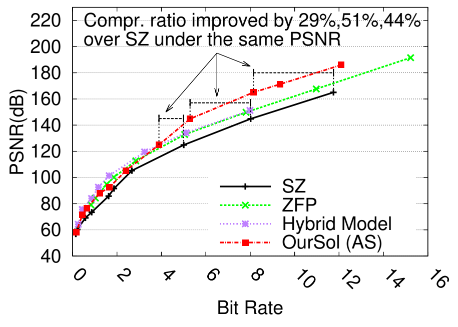
(h) NYX (overall)

Figure 2.11: Overall evaluation

Figure 2.12 presents the compression quality of the QMCPack dataset 2 and dataset 3 compared with the QMCPack dataset 1 shown in Figure 2.11(g). Note that our offline parameter optimization was performed not based on these two QMCPack datasets, which are largely different from the QMCPack dataset 1 in scale. Based on the figure, we clearly see that for both datasets our solution can still get much better compression quality than the others can. This means that our optimization method can also be applied effectively on new simulation datasets that were not included in our offline optimization analysis.



(a) QMCPack - dataset 2



(b) QMCPack - dataset 3

Figure 2.12: Evaluation on multiple QMCPack dataset

We also evaluate the autocorrelation metric of the compression errors (as shown in Table 2.5), in order to check the randomness of the compression errors. The users generally expect to see close-to-zero autocorrelation results, because this introduces less bias to their post-analysis. Table 2.5 shows that our solution achieves comparable autocorrelation values of compression errors compared with SZ, indicating the same randomness of compression errors.

Table 2.5: Lag one autocorrelation of compression error

Dataset	Error Bound (reb)	Autocorrelation (lag=1)		
		SZ	ZFP	Our Solution
Hurricane (Uf48)	1E-3	0.040711	0.151458	0.053633
	1E-5	0.001358	0.115680	0.001687
Miranda (velocityz)	1E-3	0.211425	0.343711	0.216588
	1E-5	0.071940	0.266735	0.059465
QMCPack (dataset 1)	1E-3	0.211425	0.374731	0.241557
	1E-5	0.022431	0.217974	0.028725

2.7.5 I/O Performance Evaluation

In this subsection, we present the parallel I/O evaluation results based on two scientific simulations (Hurricane and Miranda) on the Bebop supercomputer [12]. The value-range-based relative error bounds are set to 1E-6 and 1E-5 respectively. We first show that the lossy compression with these two error bounds leads to fairly high precision of the reconstructed data compared with the original raw data. We then show the parallel I/O performance when using different compressors.

The reconstructed data under lossy compression with these error bounds are of fairly high precision. On the one hand, some domain scientists [9] recommend keeping the structural similarity index measure (SSIM) [108] no less than 0.99995, based on their postanalysis using existing lossy compressors. The reconstructed data in our experiments here can get an overall SSIM up to 0.99999+, so the data are supposed to be acceptable to users w.r.t. SSIM. On the other hand, to confirm that the error bounds in our evaluation lead to high precision of the reconstructed data, we demonstrate the visual quality of the reconstructed data for the two applications in Figure 2.13 and Figure 2.14. We zoom in on a small region to 625× for each image.

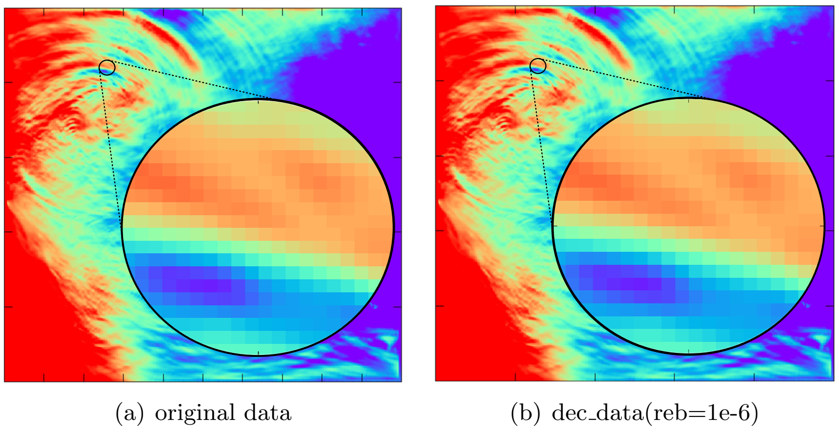

(a) original data (b) dec_data(reb=1e-6)

Figure 2.13: Visualization of Hurricane (Uf48)

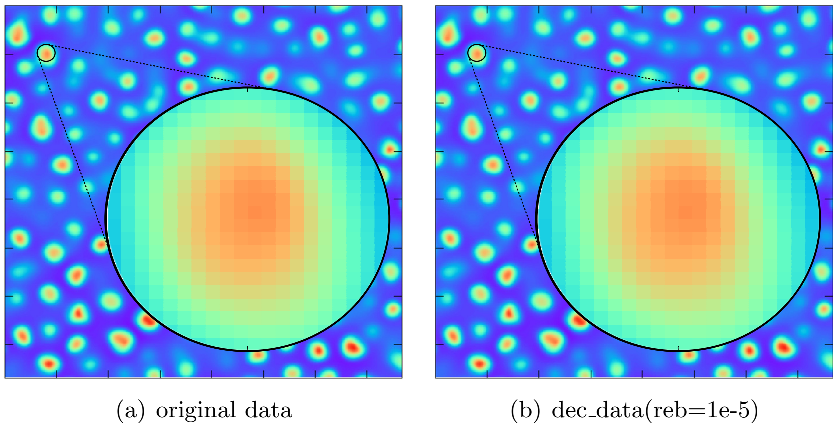
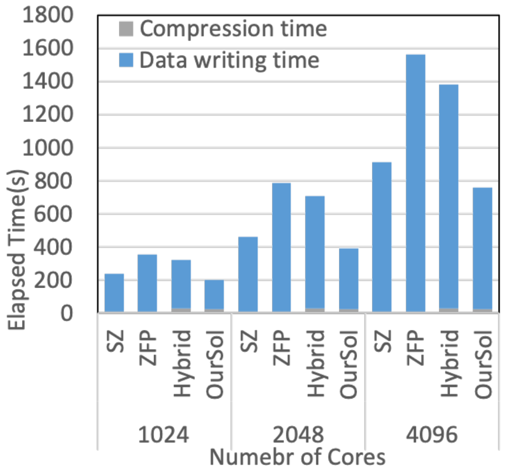
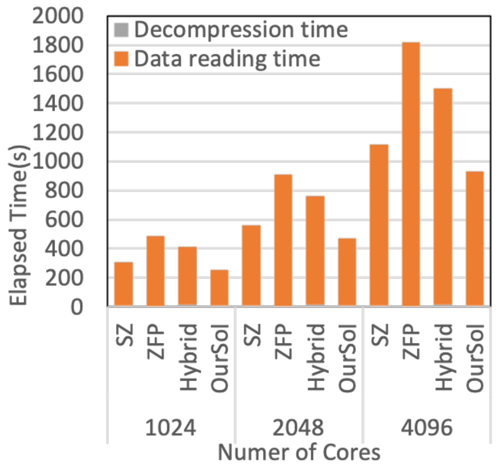

(a) original data (b) dec_data(reb=1e-5)

Figure 2.14: Visualization of Miranda (velocityz)

We present the parallel I/O performance evaluation results in Figure 2.15 and Figure 2.16. Without any compression techniques, it took 6,141 s and 4,881 s to store the original data and 7,274 s and 5,891 s to read the original data (using 4,096 processes) because of limited I/O bandwidth. The figures clearly show that the parallel I/O performance with compression techniques is always less than 1,800 seconds. In particular, our solution has the least overall elapsed times, which are 20%∼40% less than the times when using the second-best lossy compressor (SZ). This is due to the significantly reduced data sizes achieved by our compressor. Such a performance gain can benefit the applications significantly. On the one hand, for the applications suffering a bottleneck in I/O cost, the overall runtime can be reduced significantly. On the other hand, the storage requirement would be decreased for each application, enabling more applications to run on supercomputers.

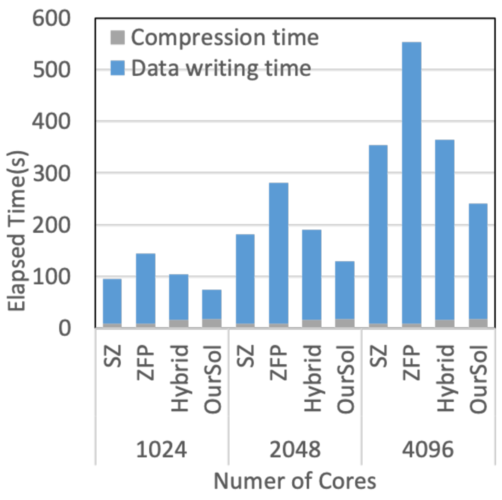


(a) Data dumping performance

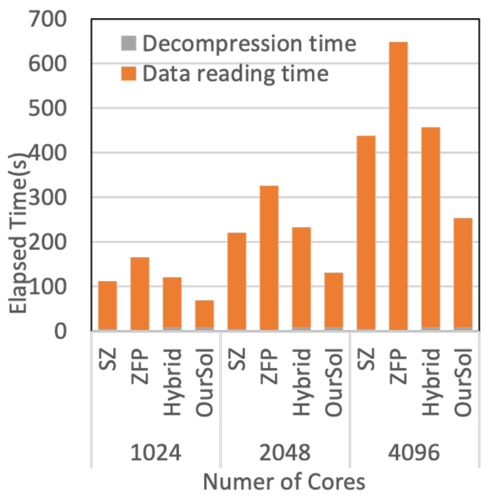


(b) Data loading performance

Figure 2.15: Parallel performance on Hurricane



(a) Data dumping performance



(b) Data loading performance

Figure 2.16: Parallel performance on Miranda

Chapter 3

SZ3: Error-Bounded Lossy Compression for Scientific Data by Dynamic Spline Interpolation

3.1 Introduction

In the scientific research domain, the users often adopt scientific data libraries such as NetCDF [82] and HDF5 [38] to manage the scientific data due to their performance advantage and better support of multidimensional objects over traditional database management systems. Those scientific data libraries have database features including metadata, data indexing, data manipulation, and data visualization tools [3, 22]. In particular, due to the vast amount of data to deal with, these data management libraries also support integrating different data compressors. For example, HDF5 offers a filter mechanism [96] to allow users to call various compressors (including SZ [25], ZFP [64], Zlib [117], etc.) transparently when storing scientific data.

Error-bounded lossy compression techniques [56, 60, 91] have been developed for several years, and they have been widely recognized as an optimal solution to reduce the storage demand of scientific data management systems. For example, Liang et al. [61] showed that an error-bounded lossy compressor can improve the overall I/O performance by 60X, with no degradation of visual quality on the reconstructed data. Kukreja et al. [51] showed that using error-bounded lossy compression can get high compression ratios without affecting the convergence or final solution of the full waveform inversion solver clearly.

In this chapter, we significantly improve the error-bounded lossy compression quality for scientific datasets, by designing a dynamic best-prediction-selection strategy and proposing a novel, spline interpolation based prediction approach with a series of optimizations. This predictor completely eliminates the serious storage overhead compared with the linear-regression predictor used in SZ. Our contribution is threefold.

- We provide an in-depth analysis of the latest version of SZ and identify significant drawbacks of its prediction method; the analysis also sheds light on our new design. The critical challenge in the current design of SZ comes from its linear-regression prediction method, which has two significant drawbacks. On the one hand, it suffers from limited accuracy in predicting nonlinear varied datasets. Many scientific simulations (such as seismic wave simulation [48] and quantum chemistry research [32]) , however, may produce a vast amount of data with nonlinear features, such that SZ cannot work very effectively on them. On the other hand, the linear-regression-based prediction needs to store several coefficients (e.g., four coefficients per block for 3D compression) in each block of data, introducing significant overhead especially when a relatively high compression ratio is required.

- We propose a novel prediction method that can significantly improve the compression ratio compared with the linear-regression prediction method. On the one hand, cubic spline interpolation is included in our novel approach to represent high order data variation, which obtains much higher prediction accuracy over linear-regression for datasets with nonlinear data variation characteristics. On the other hand, we derive the constant coefficients in our interpolation approach such that the coefficient storage overhead can be completely eliminated. We further propose a dynamic optimization strategy to select the best predictor from between the spline interpolation approach and the multilevel Lorenzo predictor to improve the overall compression quality.

- We perform a comprehensive assessment of our solution versus five state-of-the-art lossy compressors, using multiple real-world simulation datasets across different scientific domains. Experiments show that our solution improves the compression ratio by 20%$\sim$460% over the second-best compressor with the same error bound and experiences no degradation in the post-analysis accuracy.

The rest of the chapter is organized as follows. In Section 3.2 we discuss the related work. In Section 3.3 we formulate the research problem. In Section 3.4 we offer an in-depth analysis of the pros and cons of SZ In Section 3.5 we describe our solution and detailed optimization strategies. In Section 3.6 we present the evaluation results compared with five other state-of-the-art lossy compressors using real-world applications.

3.2 Related Work

Data compression is becoming a critical technique for database management systems. For time series databases, Gorilla [76] is proposed to improve query performance using lossless compression techniques including XOR and variable-length encoding. AMMMO [111] utilizes machine learning to select the best lossless compression scheme for each data point in time series databases. SciDB [23] is a science-oriented database system that supports several lossless algorithms including run-length encoding and adaptive huffman encoding. Snappy [87] is a high-speed lossless compression framework used by many databases such as InfluxDB [42]. Zlib [117] and Zstandard [118] are another two state-of-the-art lossless compressors.

Although lossless compression techniques are widely used in database management systems, they are not suitable for scientific data. Lossless compressors suffer from very limited compression ratios (generally $\sim$2 or even less) on scientific data [113] since they rely on repeated byte-stream patterns whereas scientific data is often composed of diverse floating-point numbers. Thus, scientific lossy compression has been studied for years.

Unlike traditional lossy compression techniques (such as JPEG [107]) that were designed mainly for image data, the error-bounded lossy compression can not only get a fairly high compression ratio (several dozens, hundreds, or even higher) but also guarantee that the reconstructed data is valid for scientific post-analysis in terms of the user-defined compression error bound. Error-bounded lossy compression can be categorized as higher-order singular value decomposition (HOSVD)-based models such as TTHRESH [11], transform-based models such as ZFP [64], and prediction-based models including SZ [25, 91].

There are also some machine learning (ML) based lossy compressors such as LFZIP [18]. ML compressors have two drawbacks in terms of scientific data prediction. First, the ML models have non-negligible size and ML models need to be retrained for data in different scientific domains. As a result, the model weights should be stored together with compressed data and this brings significant storage overhead. Second, ML compressors involves the ML inference process which has much higher computational cost than traditional methods including interpolation based predictors that are linear time complexity.

In our work, we choose the prediction-based model because SZ has been recognized as the leading compressor in the scientific data compression community. In fact, how to leverage SZ to improve compression quality has been studied for more than two years. Tao et al. [93] developed a strategy that can combine SZ and ZFP to optimize the compression ratios based on a more significant metric, peak signal-to-noise ratio (PSNR). Liang et al. [58] further analyzed the principles of SZ and ZFP and developed a method integrating ZFP into the SZ compression model, which can further improve the compression quality. Zhao et al. [114] proposed to adopt second-order Lorenzo+regression in the prediction methods and developed an autotuning method to optimize the parameters of SZ. Liang et al. [62] accelerated the performance of MultiGrid Adaptive Reduction of Data (MGARD) [4] and used SZ to compress the nodal points generated by the MGARD framework [4], which can improve the compression ratios significantly.

All these existing SZ-related solutions have to rely on the linear regression prediction to a certain extent. This is a critical restriction to the compression quality improvement, which will be analyzed deeply in Section 3.4.

3.3 Problem Formulation

In this section we describe the research problem formulation. Given a scientific dataset (denoted by D) composed of N floating-point values (either single precision or double precision) and a user-specified absolute error bound (e), the objective is to develop an error-bounded lossy compressor that can always meet the error-bounding constraint at each data point with optimized compression quality and comparable performance (i.e., speed).

Rate distortion is arguably the metric most commonly used by the lossy compression community to assess compression quality. It can be converted to the commonly used statistical data distortion metric known as normalized root mean squared error, and it is a good indicator of visual quality. Rate distortion involves two critical metrics: peak signal-to-noise ratio (PSNR) 2.1 and bit rate .

Bit rate is used to evaluate the compression ratio (the ratio of the original data size to the compressed size). Specifically, bit rate is defined as the average number of bits used per data point in the compressed data. For example, suppose a single-precision original dataset has 100 million data points; its original data size is 100,000,000$\times$4 bytes (i.e., about 400 MB). If the compressed data size is 4,000,000 bytes (i.e., a compression ratio of 100:1), then the bit rate can be calculated as $32/100 = 0.32$ (one single-precision number takes 32 bits). Obviously, smaller bit rate means higher compression ratio.

Two other important compression assessment metrics are compression speed (denoted by s_c) and decompression speed (denoted by s_d). They are defined as the amount of data processed per time unit (MB/s).

In our research, we focus on the optimization of compression quality (i.e., rate distortion) with high performance, which can be formulated as follows:

$$\begin{aligned}
&Optimize rate\text{-}distortion \\
&subject to\, |d_i - d'_i| \leq e \\
&s_c(newsol.) \approx s_c(sz) \\
&s_d(newsol.) \approx s_d(sz),
\end{aligned} \tag{3.1}$$

where d_i and d'_i are referred to the value of the ith data point in the original dataset D and the decompressed dataset D' by the new compression solution, respectively. The notations $s_c(newsol.)$ and $s_d(newsol.)$ represent the compression speed and decompression speed of the new solution, respectively, and $s_c(sz)$ and $s_d(sz)$ represent the compression speed and decompression speed of the original SZ compressor, respectively. That is, we are trying to increase the compression ratio with the same level of data distortion and comparable compression/decompression performance compared with SZ as a baseline (because SZ has been confirmed as a fairly fast lossy compressor in many existing studies [61, 62]),

In our evaluation in Section 3.6, not only do we present the rate distortion results for many different datasets at different bit-rate ranges, but we also assess the impact of our lossy compressor on the results of decompressed-data-based post-analysis on one production-level seismic simulation research.

3.4 Deeply Understanding the Pros and Cons of SZ

In this section, we first give a review of the current SZ design and then provide an in-depth analysis of a serious problem in the latest version of the SZ compressor (SZ2.1) [60]. Understanding this problem is fundamental to understanding why our new solution can significantly improve the compression ratio.

3.4.1 Critical Features of SZ Compression Framework

First, SZ is a very flexible compression framework, in which the data prediction is the most critical step. More accurate data prediction will result in more quantization numbers being close to zero which leads to a better compression ratio during the encoding and lossless compression steps. Thus we have explored other more efficient predictors in the past two years (from version 0.1 through the latest released version 2.1, as well as a few recent prototypes [58, 63, 68]). Accordingly, we are still focused only on the data prediction stage in this chapter.

Second, SZ has to follow a necessary condition, in order to guarantee that the compression errors are always within the user-predefined error bound. For the same data point, its predicted value during the compression stage has to be exactly the same as the one predicted in the decompression stage. Otherwise, the compression errors would be accumulated easily during the decompression, causing totally uncontrolled compression errors. Thus, in the compression stage, SZ has to predict each data point by its nearby lossy decompressed values instead of the original values, which will in turn degrade the prediction accuracy (as exemplified in our prior work [91]). We proposed the linear-regression predictor in SZ2.1 [60], which can mitigate this issue to a certain extent. Such a predictor, however, has a significant drawback and may substantially inhibit the compressor from obtaining a high compression ratio in many cases. We analyze this drawback in detail in the following text.

3.4.2 Review of Linear Regression Predictor in SZ2.1

In what follows, we describe the linear regression predictor used in SZ2.1 and its serious drawback.

In SZ2.1, the whole dataset is split into equal-sized blocks (e.g., 6×6×6 for a 3D dataset) and performs a linear-regression-based prediction when the data inside the block is relatively smooth or the error bound is relatively high. The basic idea is to use linear regression to construct a hyperplane in each block, such that the data inside the block can be approximated by the hyperplane with minimized min squared error (MSE), as illustrated in Fig. 3.1. The details can be found in our prior work [60].

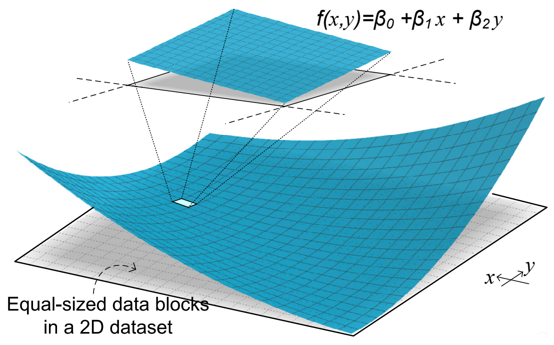

$f(x,y)=\beta_0+\beta_1 x+\beta_2 y$
x
y
Equal-sized data blocks in a 2D dataset

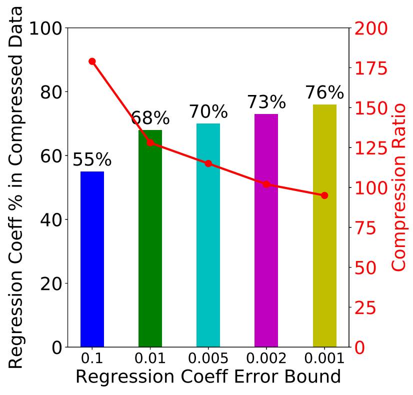
Figure 3.1: Illustration of linear-regression-based prediction (2D dataset)

3.4.3 Serious Dilemma of Linear-Regression Predictor in SZ2.1

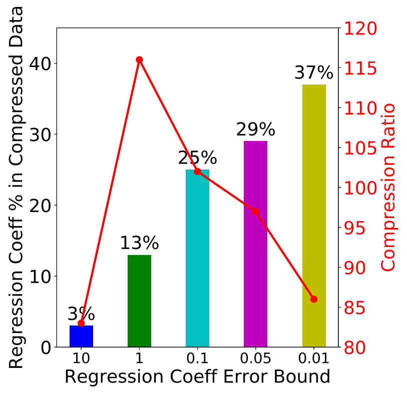
In order to get a high compression quality (i.e., a very good rate-distortion result), the four coefficients need to be compressed based on a certain error bound, which may introduce a serious dilemma: a higher error bound used on coefficient compression will decrease the overhead of storing the coefficients (to be demonstrated in Fig. 3.2) but also decrease the regression accuracy of the constructed hyperplane (to be demonstrated in Fig. 3.3). We confirm this issue by four real-world scientific simulations (QMCPack [50], RTM [48], Hurricane [40], and NYX [74]), which are commonly used by scientists in quantum structure research, seismic imaging for oil and gas exploration, climate research, and cosmology research, respectively. More details about these applications are given in Section 3.6. We exemplify the results using specific fields (e.g., time step 1500 of RTM data, the W field of Hurriane, and velocity z in NYX) because of the space limits and similar results in other fields.

Fig. 3.2 shows that the overhead always increases with decreasing error bounds used on the compression of coefficients. Specifically, we observe that when the error bound decreases from 0.1 to 0.01, the coefficient overhead in the compressed data increases from 55% to 68%, from 25% to 37%, from 40% to 53%, and from 60% to 70%, for the four test cases, respectively. The compression ratios (the red curve) thus degrade from 179 to 128, from 102 to 86, from 114 to 90, and from 152 to 118, respectively.

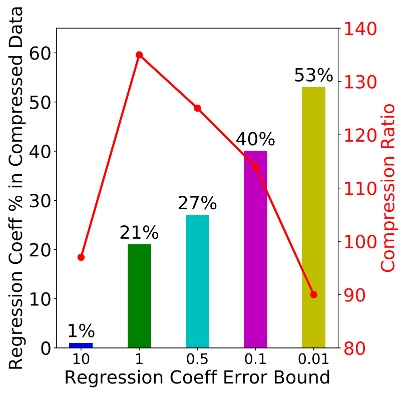
We present a slice segment of the four application datasets in Fig. 3.3 to illustrate that the error bounds of the coefficient compression would significantly affect the prediction accuracy of the constructed linear-regression hyperplane. For instance, when the coeffi-



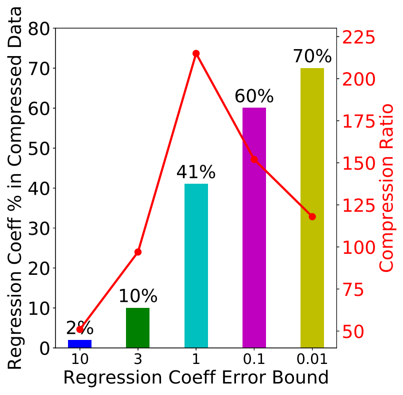
(a) QMCPack



(b) RTM (time step 1500)



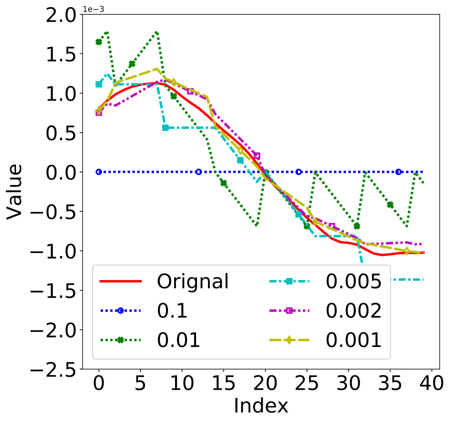
(c) Hurricane(W)



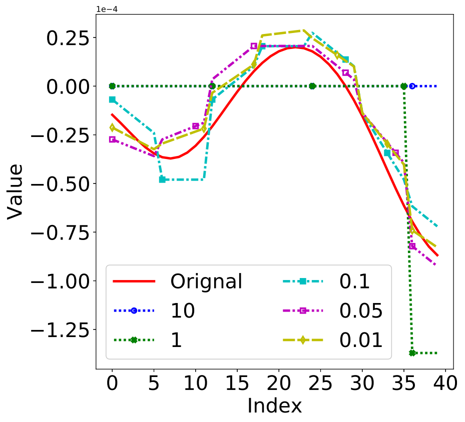
(d) NYX(velocityz)

Figure 3.2: Overhead of linear regression coefficients

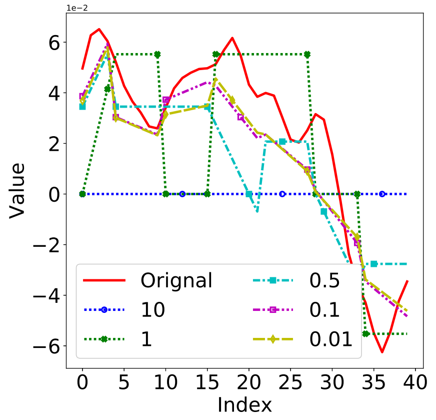
cients' compression error bound is set to 0.001 for QMCPack and 0.01 for RTM (time step 1500), the constructed hyperplane (the yellow curve) can fit the real data (the red curve) well, but the fitting will be much worse with increasing error bounds. In the case with a relatively large error bound (e.g., 0.1 in QMAPack), the hyperplane will downgrade to a simple horizontal line (see blue lines in the figures), because simply using the neighbor data value is "accurate" enough for the large error-bounded compression of the coefficients. This will definitely result in large prediction errors (the difference between predicted value and raw value) significantly degrading the final compression ratios.



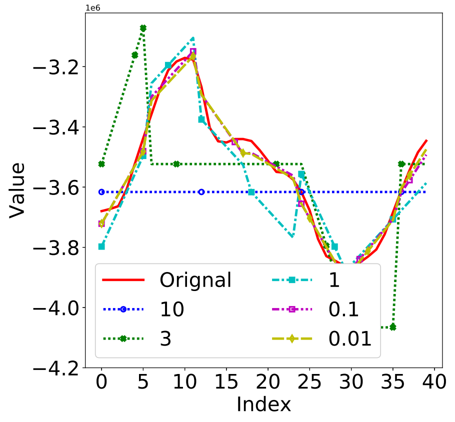
(a) QMCPack



(b) RTM (time step 1500)



(c) Hurricane(W)

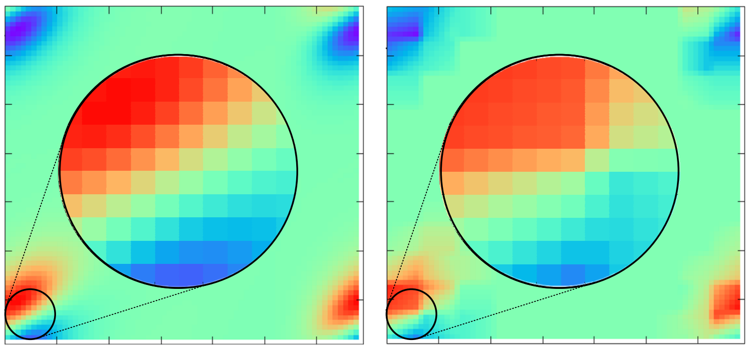


(d) NYX(velocityz)

Figure 3.3: Linear regression prediction hyperplane with different error bound settings of coefficients

In Fig. 3.4, we demonstrate that the latest version of SZ (v2.1) may cause significant loss of the data visualization, especially when the compression ratio (CR) is relatively high (e.g., 196 and 568 for the two test cases). We observe that SZ suffers from a significant undesired block texture artifact, resulting from its blockwise linear-regression design.

To address the serious issue of the linear regression predictor, we developed a novel efficient predictor based on a dynamic spline interpolation, such that compression quality


(a) Original data (QMCPack) (b) Decompressed data (QMC-Pack)

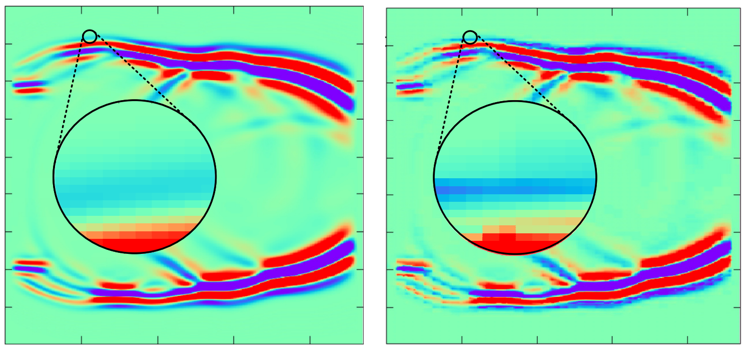

(c) Original data (RTM) (d) Decompressed data (RTM)

Figure 3.4: Visualization of SZ decompressed data based on two applications: (1) QMCPack – PSNR=56.2, CR=196, and (2) RTM – PSNR=50.7, CR=316

(rate distortion) can be significantly improved for almost all application datasets, with little performance overhead.

3.5 Error-Bounded Lossy Compression With a Dynamic Multidimensional Spline Interpolation

We present the design overview in Fig. 3.5, with yellow rectangles indicating the differences between our design and the classic SZ compressor and with highlighted rectangles indicating the critical steps. The fundamental idea is to develop a **dynamic**

multidimensional spline interpolation-based predictor (i.e., solution P2 shown in Fig. 3.5) to replace the linear-regression-based predictor such that the coefficient overhead can be completely eliminated while still keeping a fairly high prediction accuracy. Our newly designed interpolation-based predictor starts with one data point and performs interpolation and linear-scale quantization alternatively along each dimension recursively until all data points are processed. Two alternative approaches can be used to perform the interpolation in the multidimensional space. We can build a multidimensional curve to fit all the already-processed data points, or we can build multiple 1D curves to do the interpolation. We choose the latter because the former is much more expensive.

In what follows, we introduce the background of spline interpolation (Section 3.5.1), followed by our design of dynamic multidimensional spline interpolation based predictor (Sections 3.5.2, 3.5.3, and 3.5.4).

3.5.1 Introduction to Spline Interpolation

Interpolation is widely used in the field of engineering and science to construct new data points with a set of known data points. Interpolation techniques attempt to build a curve that goes through all the known data points. It differs from regression analysis, which usually seeks a curve that most closely fits the known data points according to a specific mathematical criterion such as mean squared error. The curve generated by regression may not go through all known points.

The most popular interpolation methods can be categorized into three types: piecewise constant interpolation, polynomial interpolation, and spline interpolation. Piecewise

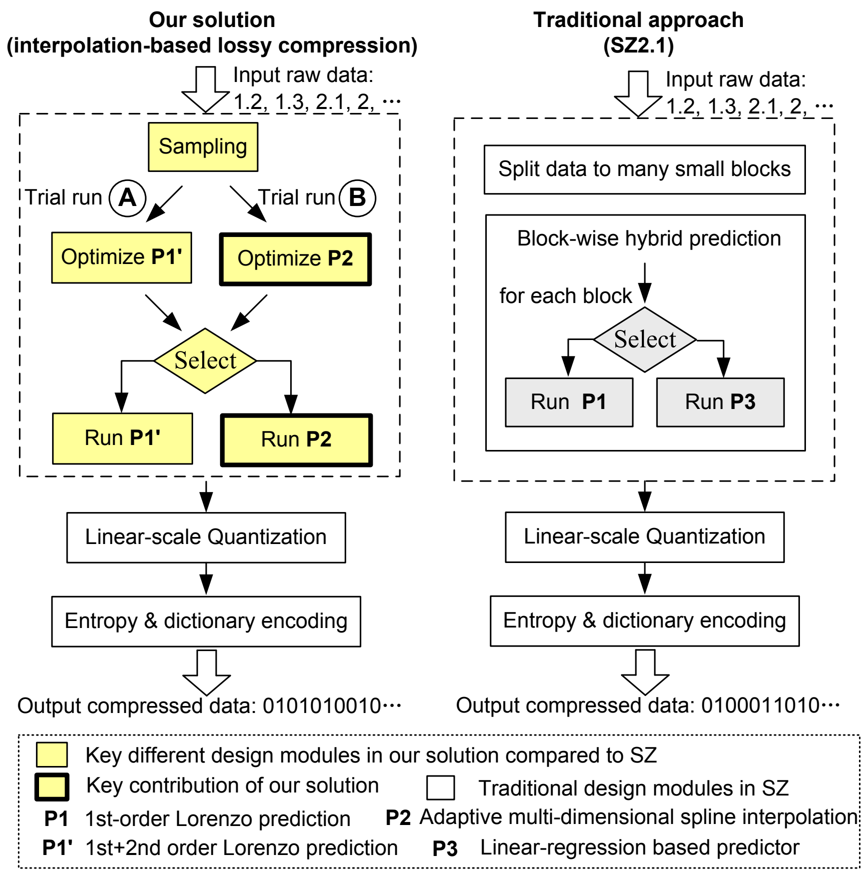

Our solution
(interpolation-based lossy compression)
Input raw data:
1.2, 1.3, 2.1, 2, ⋯
Sampling
Trial run A
Trial run B
Optimize P1'
Optimize P2
Select
Run P1'
Run P2
Linear-scale Quantization
Entropy & dictionary encoding
Output compressed data: 0101010010⋯
Traditional approach
(SZ2.1)
Input raw data:
1.2, 1.3, 2.1, 2, ⋯
Split data to many small blocks
Block-wise hybrid prediction
for each block
Select
Run P1
Run P3
Linear-scale Quantization
Entropy & dictionary encoding
Output compressed data: 0100011010⋯
Key different design modules in our solution compared to SZ
Key contribution of our solution
Traditional design modules in SZ
P1 1st-order Lorenzo prediction
P2 Adaptive multi-dimensional spline interpolation
P1' 1st+2nd order Lorenzo prediction
P3 Linear-regression based predictor

Figure 3.5: Design overview of our solution

constant interpolation always uses the nearest known data points to estimate the new data point, so it has a simple implementation and fast speed. However, its ability to estimate complex curves is limited because it does not consider the surrounding data points. Polynomial interpolation is designed to find a polynomial with the lowest possible degree that passes through all the known data points. If the number of known data points is large, the polynomial may suffer highly inaccurate oscillation between the data points. This issue is well known as *Runge's phenomenon* and could be mitigated by spline interpolation. Spline interpolation uses piecewise polynomials to define the estimation curve. If the degree of the polynomials is 1, the spline interpolation turns to **linear interpolation**. If the degree of polynomials is 3, it is known as **cubic spline interpolation**. Cubic spline polynomials have different restrictions. In this chapter, we use not-a-knot restriction for cubic spline interpolation.

3.5.2 Spline Interpolation Designed for Scientific Data

In this sub-section, we introduce a basic interpolation method and derive closed-form formulas with the optimal coefficients, which is a fundamental work to the development of our following multi-dimensional interpolation predictor.

We propose a light-weight cubic interpolation based prediction method for each unknown data point by only using its four surrounding data values, to address the drawbacks of the conventional interpolation methods. The accuracy of polynomial interpolation could be affected significantly by Runge's phenomenon when interpolating across multiple regions with different locality features. Cubic spline interpolation can prevent large oscillation, but it has high computational cost as it needs to solve a huge linear system. To avoid high computation cost, we precompute a closed-form interpolation formula based on four specific neighbor data points (e.g., using the data points $i-3$, $i-1$, $i+1$ and $i+3$ to predict data point i as shown in Figure 3.6). In what follows, we mainly use a 1D example to illustrate how we derive the interpolation formula, but the formula can be extended to multidimensional cases easily.

Lemma 1 *Denote the dataset as $d = (d_1, d_2, ..., d_n)$ with n as the total number of elements. The prediction values are denoted as p=$(p_1, p_2, ..., p_n)$. We consider all elements in odd-index positions as preknown and use them to predict the elements in even-index positions.*

The prediction formulas of linear and cubic spline interpolation are shown in Table 3.1.

Table 3.1: Spline estimations

Spline method	Prediction Value p_i
Linear spline	$p_i = \frac{1}{2}d_{i-1} + \frac{1}{2}d_{i+1}$
Cubic spline	$p_i = -\frac{1}{16}d_{i-3} + \frac{9}{16}d_{i-1} + \frac{9}{16}d_{i+1} - \frac{1}{16}d_{i+3}$

The linear formula is easy to derive, so we prove only the cubic spline formulas as follows. In our designed cubic spline interpolation, the known data points d_{i-3},d_{i-1},d_{i+1} and d_{i+3} are used to predict the data point p_i. Three spline curves correspond to the known data points:

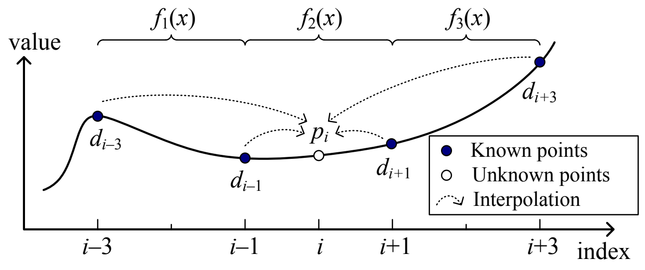


Figure 3.6: Illustration of cubic spline interpolation

$$\begin{aligned}
f_1(x) &= a_1(x-(i-3))^3 + b_1(x-(i-3))^2 + c_1(x-(i-3)) + \delta_1 \\
f_2(x) &= a_2(x-(i-1))^3 + b_2(x-(i-1))^2 + c_2(x-(i-1)) + \delta_2 \\
f_3(x) &= a_3(x-(i+1))^3 + b_3(x-(i+1))^2 + c_3(x-(i+1)) + \delta_3
\end{aligned} \tag{3.2}$$

The scope of f_1, f_2, and f_3 is [i−3,i−1], [i−1,i+1], and [i+1,i+3] (as shown in Fig. 3.6). The spline curves should pass through the known data points.

$$\begin{aligned} &f_1(i-3) = d_{i-3};\ f_1(i-1) = d_{i-1} \\ &f_2(i-1) = d_{i-1};\ f_2(i+1) = d_{i+1} \\ &f_3(i+1) = d_{i+1};\ f_3(i+3) = d_{i+3} \end{aligned} \tag{3.3}$$

The first derivatives of $f_1(x)$ is $f_1^{'}(x) = 3a_1(x-(i-3))^2 + 2b_1(x-(i-3)) + c_1$. The second derivative is $f_1^{''}(x) = 6a_1(x-(i-3)) + 2b_1$. The third derivative is $f_1^{'''}(x) = 6a_1$. Derivatives of f_2 and f_3 are similar with f_1.

To have a smooth curve, we should let the adjacent spline functions have the same first derivatives and the same second derivatives on the joint data points.

$$\begin{aligned} &f_1^{'}(i-1) = f_2^{'}(i-1);\ f_2^{'}(i+1) = f_3^{'}(i+1) \\ &f_1^{''}(i-1) = f_2^{''}(i-1);\ f_2^{''}(i+1) = f_3^{''}(i+1) \end{aligned} \tag{3.4}$$

The not-a-knot restriction requires the third derivative of f to be equal on locations $i-1$ and $i+1$.

$$f_1^{'''}(i-1) = f_2^{'''}(i-1);\ f_2^{'''}(i+1) = f_3^{'''}(i+1) \tag{3.5}$$

Using the system of Equations (3.3), (3.4), and (3.5), we can derive

$$\begin{aligned} &a_2 = -\tfrac{1}{48}d_{i-3} + \tfrac{1}{16}d_{i-1} - \tfrac{1}{16}d_{i+1} + \tfrac{1}{48}d_{i+3} \\ &b_2 = \tfrac{1}{8}d_{i-3} - \tfrac{1}{4}d_{i-1} + \tfrac{1}{8}d_{i+1} \\ &c_2 = -\tfrac{1}{6}d_{i-3} - \tfrac{1}{4}d_{i-1} + \tfrac{1}{2}d_{i+1} - \tfrac{1}{12}d_{i+3} \\ &\delta_2 = d_{i-1}. \end{aligned} \tag{3.6}$$

Thus the prediction value of pi will be

$$p_i = f_2(i) = -\tfrac{1}{16}d_{i-3} + \tfrac{9}{16}d_{i-1} + \tfrac{9}{16}d_{i+1} - \tfrac{1}{16}d_{i+3}. \tag{3.7}$$

Equation (3.7) is the cubic formula in Table 3.1.

We discuss why we adopt only four known data points in our interpolation instead of six or more data points. If we use six data points d_{i-5}, d_{i-3}, d_{i-1}, d_{i+1}, d_{i+3}, and d_{i+5} to predict p_i, the formula by not-a-knot spline turns out to be

$$p_i = \tfrac{d_{i-5}}{80} - \tfrac{d_{i-3}}{10} + \tfrac{47}{80}d_{i-1} + \tfrac{47}{80}d_{i+1} - \tfrac{d_{i+3}}{10} + \tfrac{d_{i+5}}{80}. \tag{3.8}$$

Compared with Equation (3.7), Equation (3.8) involves two additional data points d_{i-5} and d_{i+5}, but the weight of the two data points is only 1/80, which means a very limited effect on the prediction. Moreover, it has 50% higher computation cost compared with Equation (3.7). Hence, we choose to use four data points for prediction, as shown in Table 3.1.

In addition, we note that the linear spline interpolation may exhibit better prediction accuracy than the cubic spline does when setting a relatively large error bound (as shown in Table 3.2). The reason is that our interpolation method relies on the reconstructed data values generated after a linear-scale quantization step, so that the reconstructed data is lossy to a certain extent. When the error bound is relatively large, the loss of these reconstructed data would degrade the prediction accuracy, and the more data points used in the interpolation, the higher the impact on the accuracy. Since linear spline adopts fewer data points, it could be superior to cubic spline especially when the error bound is relatively large. This possibility motivated us to design a dynamic method selecting the better interpolation type (linear or cubic) in practice.

Table 3.2: Comparison of spline methods prediction error

Dataset	$\epsilon = 1E-2$		$\epsilon = 1E-4$	
	Linear Spline	Cubic Spline	Linear Spline	Cubic Spline
RTM (time step 1500)	**1.20E-4**	1.27E-4	2.0E-5	**8.3E-6**
Miranda (velocityz)	**0.0026**	0.0025	0.0061	**0.0020**
QMCPACK	**0.05**	0.06	0.008	**0.004**
SCALE (QS)	**0.076**	0.078	**0.040**	0.041
NYX (velocityz)	**123486**	134820	22453	**19978**
Hurricane (W)	**0.04**	0.05	0.023	**0.022**

3.5.3 Multilevel Multidimensional Spline Interpolation

The previous derivation works in the 1D case with 50% of preknown data points, based on which we predict the other 50%. In this section, we extend this interpolation method to support data prediction on the entire multidimensional dataset.

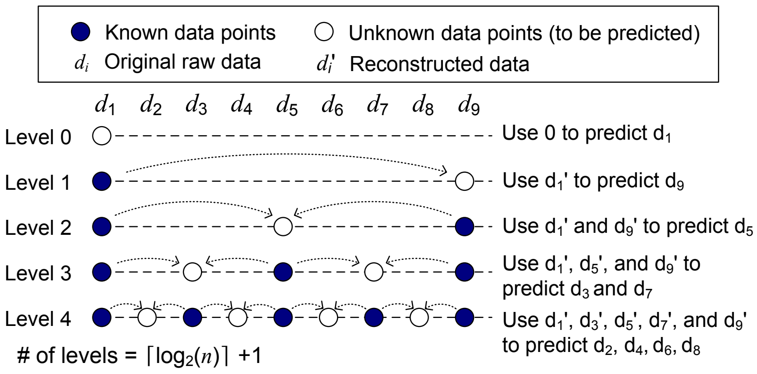

● Known data points
○ Unknown data points (to be predicted)
d_i Original raw data
d_i' Reconstructed data
d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9
Level 0
Use 0 to predict d_1
Level 1
Use d_1' to predict d_9
Level 2
Use d_1' and d_9' to predict d_5
Level 3
Use d_1', d_5', and d_9' to predict d_3 and d_7
Level 4
Use d_1', d_3', d_5', d_7', and d_9' to predict d_2, d_4, d_6, d_8
# of levels = $\lceil \log_2(n) \rceil + 1$

Figure 3.7: Illustration of multilevel linear spline interpolation

We use Fig. 3.7 to demonstrate the multilevel solution with linear interpolation; cubic interpolation has the same multilevel design. Suppose the dataset has n elements in one dimension. The number of levels required to cover all elements in this dimension is $l = 1 + \lceil log_2 n \rceil$. At level 0, we use 0 to predict d_1, followed by the error-bounded linear-scale quantization. We perform a series of interpolations from level 1 to level $l-1$ recursively, as shown in Fig. 3.7. At each level, we use error-bounded linear-scale quantization to process the predicted value such that the corresponding reconstructed data must be within the error bound. We denote the reconstructed data as $d'_1, d'_2, ..d'_n$, as shown in the figure.

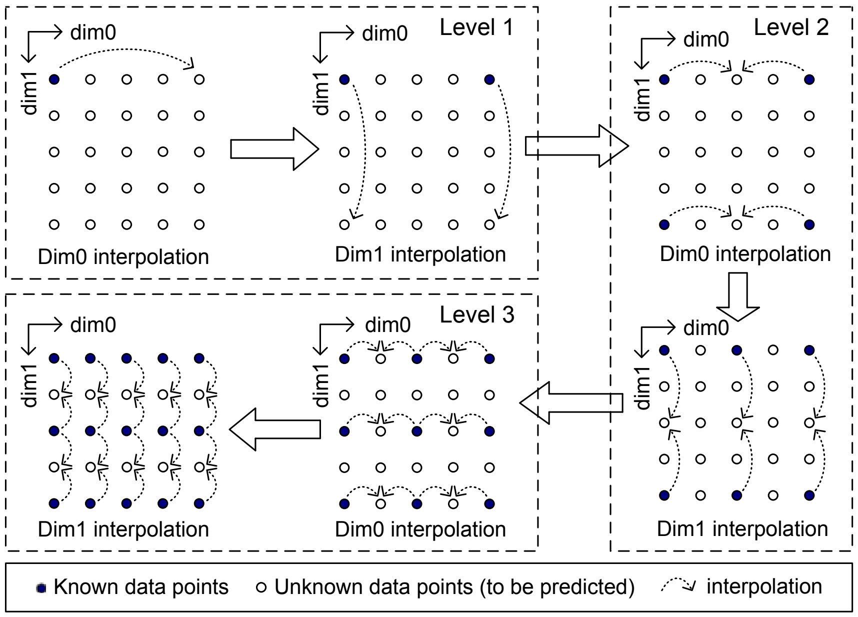


Figure 3.8: Illustration of multidimensional linear spline interpolation

Such a multilevel interpolation is applied on a multidimensional dataset, illustrated in Fig. 3.8 with a 2D dataset as an example. We perform interpolation separately along all dimensions at each level, with a fixed sequence of dimensions. A 2D dataset, for example, has two possible sequences: $dim_0 \rightarrow dim_1$ and $dim_1 \rightarrow dim_0$. A 3D dataset has 6 possible sequences. In our solution, we propose to check only two sequences, $dim_0 \rightarrow dim_1 \rightarrow dim_2$ (sequence 1) and $dim_2 \rightarrow dim_1 \rightarrow dim_0$ (sequence 2) instead of all 6 possible combinations.

On the one hand, the last interpolation dimension involves about 50% of the data points (much more than other dimensions), so which dimension to be put in the end of the sequence determines the overall prediction accuracy. On the other hand, we note that either the highest or lowest dimension in scientific datasets tends to be smoother than other dimensions without loss of generality, as confirmed by the first three columns of Table 3.3 (with all 6 applications), which presents the autocorrelation (AC) of each dimension (higher AC means smoother data). Accordingly, putting either dim0 or dim2 in the end of the sequence at each level will get lower overall prediction errors, as validated in Table 3.3. Hence, we also develop a dynamic strategy to select the best-fit sequence of dimensions from among the two candidates, as detailed in the next subsection.

Table 3.3: Autocorrelation and prediction error of cubic spline interpolation with different sequences of dimension settings, ϵ=1E−3

Dataset	Autocorrelation (Lag=4)			Prediction Error		
	dim2	dim1	dim0	0→1→2	0→2→1	2→1→0
RTM (time step 1500)	**0.88**	0.58	0.45	**2.17E-5**	2.32E-5	2.51E-5
Miranda (velocityz)	0.84	0.82	**0.96**	0.004	0.004	**0.003**
QMCPACK	**0.83**	0.83	0.75	**0.010**	0.010	0.013
SCALE (QS)	**0.987**	0.986	0.872	**0.0447**	0.0448	0.10
NYX (velocityz)	0.9818	0.99	**0.99**	31668	29903	**28975**
Hurricane (W)	0.19	0.027	**0.86**	0.024	0.025	**0.016**

3.5.4 Dynamic Optimization Strategies

In this section we propose a dynamic design with two adaptive strategies: (1) automatically optimizing the spline interpolation predictor (Trial run Ⓑ in Fig. 3.5) by selecting the best-fit interpolation type (either linear or cubic) and optimizing the sequence of interpolation dimensions and (2) automatically selecting the better predictor between the Lorenzo-based predictor (Trial run Ⓐ in Fig. 3.5) and the interpolation predictor.

We use a uniform sampling method to determine the best interpolation settings for the input dataset. There are two settings to optimize for the multidimensional interpolation predictor: the interpolation type and the dimension sequence. We adopt a uniform sampling method with only 3% total data points to select the better interpolation type with the higher compression ratio.

We note that the spline interpolation predictor does not work as effectively as the multilayer Lorenzo predictor [91, 114] on the relatively nonsmooth dataset, especially when the user's error bound is relatively small (as shown in Table 3.4). As a result, our final solution is selecting the better predictor from our spline interpolation method and Lorenzo method.

Table 3.4: Prediction error of multidimensional spline interpolation predictor (S), regression predictor (R), and Lorenzo predictor (L)

Dataset	$\epsilon = 1E-2$			$\epsilon = 1E-7$		
	S	R	L	S	R	L
RTM (time step 1500)	**1.2E-4**	1.3E-4	2.0E-4	6.9E-6	1.0E-4	**1.8E-7**
Miranda (velocityz)	**0.02**	0.03	0.05	0.001	0.02	**6E-5**
QMCPACK	**0.05**	0.06	0.13	0.004	0.03	**6E-4**
SCALE (QS)	**0.07**	0.16	0.11	0.04	0.15	**0.01**
NYX (velocityz)	**121436**	132071	410083	**15237**	51963	16965
Hurricane (W)	**0.04**	0.05	0.06	0.01	0.04	**0.004**

3.6 Experimental Evaluation

In this section we present the experimental setup and discuss the evaluation results and our analysis.

3.6.1 Experimental Setup

Execution Environment

We perform the experiments on the Argonne Bebop supercomputer. Each node in Bebop is driven by two Intel Xeon E5-2695 v4 processors with 128 GB of DRAM.

Applications

We perform the evaluation using six real-world scientific applications from different domains:

- QMCPack: An open source ab initio quantum Monte Carlo package for the electronic structure of atoms, molecules, and solids [50].
- RTM: Reverse time migration code for seismic imaging in areas with complex geological structures [48].
- NYX: An adaptive mesh, cosmological hydrodynamics simulation code.
- Hurricane: A simulation of a hurricane from the National Center for Atmospheric Research in the United States.
- Scale-LETKF: Local Ensemble Transform Kalman Filter (LETKF) data assimilation package for the SCALE-RM weather model.
- Miranda: A radiation hydrodynamics code designed for large-eddy simulation of multicomponent flows with turbulent mixing.

Detailed information about the datasets (all using single precision) is presented in Table 3.5. Some data fields are transformed to their logarithmic domain before compression for better visualization, as suggested by domain scientists.

Table 3.5: Basic information about application datasets

App.	# files	Dimensions	Total Size	Domain
RTM	3600	449×449×235	635GB	Seismic Wave
Miranda	7	256×384×384	1GB	Turbulence
QMCPACK	1	288×115×69×69	612MB	Quantum Structure
Scale-LETKF	13	98×1200×1200	6.4GB	Climate
NYX	6	512×512×512	3.1GB	Cosmology
Hurricane	48×13	100×500×500	58GB	Weather

State-of-the-Art Lossy Compressors in Our Evaluation

In our experiment we compare our new compressor with five other state-of-the-art error-bounded lossy compressors (SZ2.1 [60], ZFP0.5.5 [64], SZ(Hybrid) [58], SZ(SP+PO)[1] [114] and MGARDx [62]), which have been recognized as the best in class (validated by different researchers [60, 70]).

State-of-the-Art Lossless Compressors in Our Evaluation

We also evaluate six lossless compressors including Google Brotli [6], Google Snappy [87], Facebook Zstandard [118], LZMA [1], Zlib [117], and Fpzip [66] as a comparison with lossy compressors. Brotli, Snappy and Zstandard are deployed in many industrial data management systems. LZMA is the default compression method of 7-Zip. Zlib [117] is one of the most widely used compressor in operating systems. Fpzip [66] is a compressor targeted at floating-point data.

[1]SZ(SP+PO) represents the SZ compression model in chapter 2.

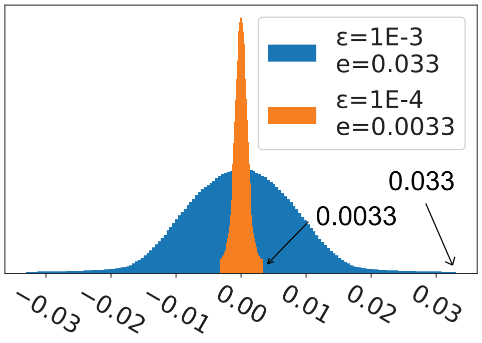
Evaluation Metrics

We evaluate the six lossy compressors based on five critical metrics, as described below.

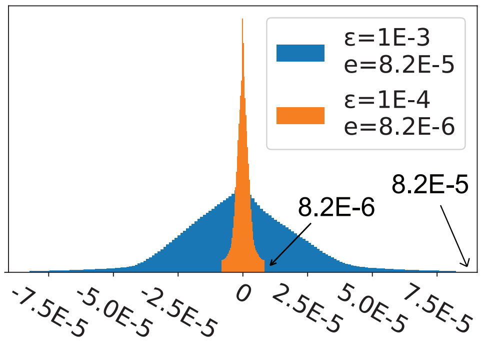
- Compression ratio (CR) based on the same error bound: The descriptions of CR and absolute error bound are defined in Section 3.3. Without loss of generality, we adopt value-range-based error bound (denoted as ϵ), which takes the same effect with absolute error bound (denoted e) because $e = \epsilon(\max(D) - \min(D))$.
- Compression speed and decompression speed: $\frac{originalsize}{compressiontime}$ (MB/s) and $\frac{reconstructedsize}{decompressiontime}$ (MB/s).
- *Rate-distortion*: The detailed description is in Section 3.3.
- Visualization with the same CR: Compare the visual quality of the reconstructed data based on the same CR.
- Precision of final execution results of RTM data with lossy compression.

3.6.2 Evaluation Results and Analysis

First, we verified the maximum compression errors for all six lossy compressors based on all the application datasets with different error bounds. Experiments confirm that they all respect the error bound constraint very well. Fig. 3.9 shows the distribution of compression errors of our solution on two error bounds (ϵ=1E-3 and ϵ=1E-4, in other words, e=0.033&0.0033 for QMCPACK and e=8.2E-5&8.2E-6 for RTM). We can clearly see that the compression errors are 100% within the absolute error bound (e) for all data points.



(a) QMCPack



(b) RTM (time step 1500)

Figure 3.9: Compression error distribution of our solution

Table 3.6 presents the compression ratios of the six lossy compressors based on the six real-world applications with the same error bounds. We can clearly observe that our solution always exhibits the highest compression ratio in all cases. In particular, the compression ratio of our solution is higher than other compressors by 20%∼460% in most cases. For example, when setting the error bound to 1E-3 for compressing RTM data, the second-best compressor ((SZ(SP+PO)) gets a compression ratio of 114.4, while our compressing ratio reaches up to 397.6 (with a 247.5% improvement). The key reason our solution can get a significantly higher compression ratio is twofold: (1) we significantly improve the prediction accuracy by a dynamic spline interpolation, and (2) some other compressors such as ZFP and MGARDx suffer from the precision-overpreservation issue (i.e., the actual maximum errors are smaller than the required error bound, as verified by prior works [25, 62, 91].

Table 3.7 compares the compression/decompression speed among all six lossy compressors for all six applications. It clearly shows that our solution exhibits compression performance similar to that of SZ2.1 and MGARDx, and its decompression performance is

Table 3.6: Compression ratio comparison based on the same error bound

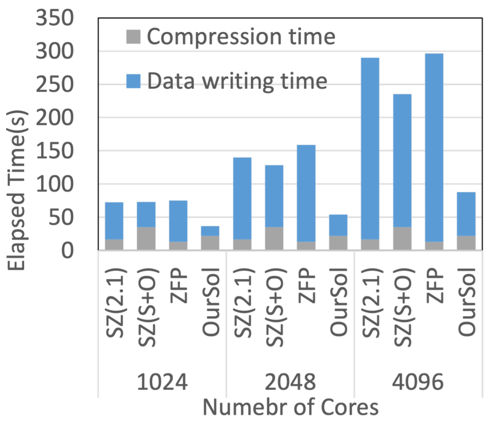
Dataset	ϵ	SZ 2.1	SZ (Hybrid)	SZ (SP+PO)	ZFP	MGARDx	OurSol	OurSol Improve %
RTM	1E-2	271.7	195.7	358.1	111.0	229.7	**1997.5**	457%
	1E-3	109.8	101.4	114.4	59.3	78.1	**397.6**	247%
	1E-4	57.3	44.4	63.0	34.9	38.3	**116.3**	84%
Miranda	1E-2	125.6	130.4	188.4	46.6	113.7	**582.1**	209%
	1E-3	59.9	55.4	58.4	25.6	38.0	**160.7**	168%
	1E-4	30.6	23.4	33.9	14.5	20.0	**47.1**	39%
QMCPack	1E-2	196.2	144.8	174.5	39.4	159.8	**675.5**	244%
	1E-3	51.1	53.4	68.0	21.2	47.1	**204.3**	200%
	1E-4	18.9	24.9	23.6	10.4	14.9	**63.7**	155%
SCALE	1E-2	84.3	94.2	108.2	14.5	52.8	**157.0**	45%
	1E-3	26.6	27.1	31.8	7.8	20.2	**40.5**	27%
	1E-4	14.0	13.2	14.1	4.6	10.4	**14.9**	5%
NYX	1E-2	43.6	33.2	48.7	12.0	24.7	**59.4**	22%
	1E-3	16.8	16.3	17.4	6.0	11.2	**21.1**	21%
	1E-4	7.6	8.0	8.1	3.7	5.5	**9.1**	12%
Hurricane	1E-2	49.4	44.6	65.4	11.3	28.1	**69.3**	6%
	1E-3	17.6	17.9	19.8	6.7	12.7	**22.5**	14%
	1E-4	9.8	10.1	10.5	4.3	7.4	**10.8**	3%

also comparable to that of SZ2.1 and is about 30% higher than that of MGARDx.

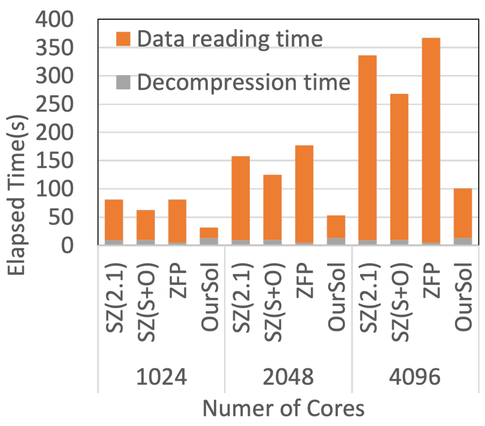
Table 3.7: Compression/decompression speeds (MB/s) with ϵ=1E-3

Type	Dataset	SZ 2.1	SZ (Hybrid)	SZ (SP+PO)	ZFP	MGARDx	OurSol
Compression	RTM	207	76	97	549	128	149
	Miranda	125	73	91	201	140	128
	QMCPack	146	63	78	158	136	133
	SCALE	145	59	75	101	122	128
	NYX	123	81	86	131	117	110
	Hurricane	115	63	78	115	122	131
Decompression	RTM	385	299	298	984	173	276
	Miranda	285	221	206	531	177	232
	QMCPack	327	232	282	367	168	241
	SCALE	271	184	192	295	164	215
	NYX	222	172	215	244	145	136
	Hurricane	222	186	200	257	163	193

Data smoothness and error bound settings are two key factors that affect the compression ratio and speed. In the SZ framework, datasets with better local smoothness or with larger error bound settings will result in smaller quantization data value range and more close-to-zero quantized data values. In general, this will get higher compression ratios and speed because such quantized data turns much easier to be compressed by the succeeding encoding steps. This analysis also applies to other lossy compressors such as ZFP and MGARDx that utilize the coding stage.



(a) Data dumping performance



(b) Data loading performance

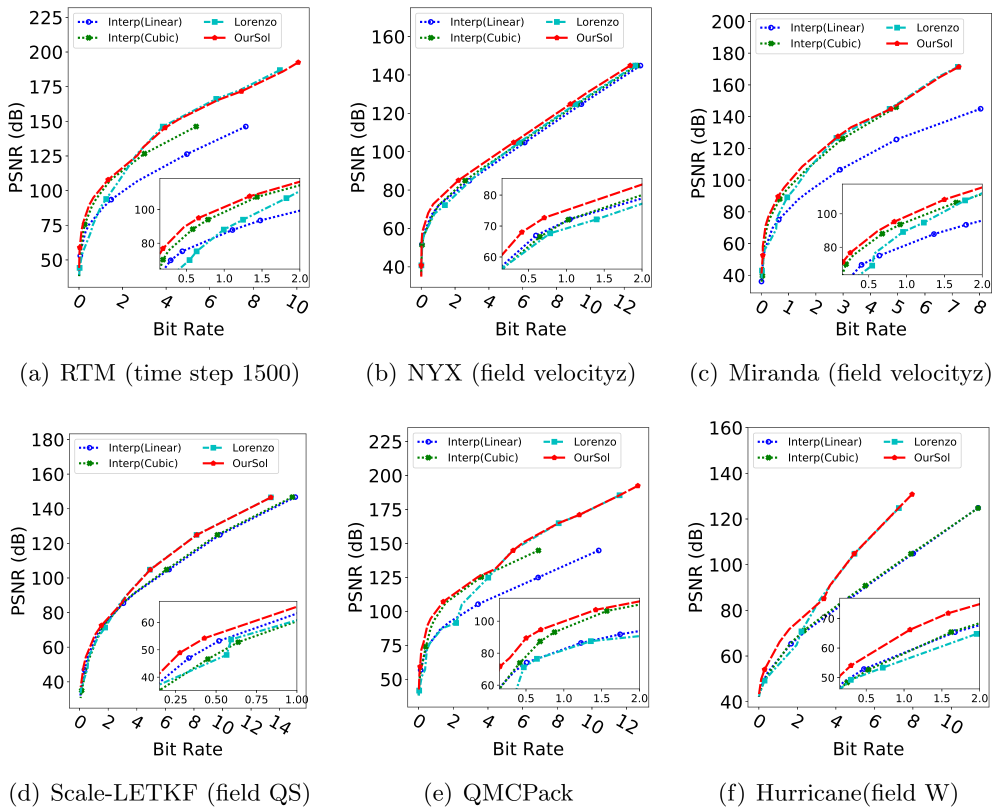
Figure 3.10: Parallel performance evaluation of QMCPack simulation (SP(S+O) stands for SP(SP+PO))

We evaluate the data dumping and loading performance of the QMCPack simulation when using lossy compressors to demonstrate the performance impact of lossy compressors on scientific simulations. SZ2.1, SZ(SP+PO), ZFP, and our solution are assessed under the same level of data distortion (PSNR fixed to 70). The evaluation uses up to 4096 cores and each core processes 3.4GB of data. Fig. 3.10 shows that our solution leads to the highest data dumping and loading performance. In the scale of 4096 cores, QMCPACK simulation needs more than 3 hours to dump the data to disk without the help of lossy compressors. Our solution reduces the elapsed time to less than 100 seconds and it is 1.7X faster than the second-best one.

Table 3.8 demonstrates the compression ratios of the six lossless compressors. It confirms our statement in Section 3.2 that lossless compressors have limited compression ratios on scientific datasets. Lossy compressors, on the other hand, can achieve much higher compression ratio as shown in Table 3.6.

Table 3.8: Compression ratio comparison of lossless compressors

Dataset	Brotli	Zstd	Snappy	Fpzip	Zlib	LZMA
RTM	2.04	2.02	1.87	2.62	2.04	2.18
Miranda	1.21	1.21	1.11	1.86	1.21	1.30
QMCPack	1.19	1.19	1.01	1.75	1.20	1.51
SCALE	1.45	1.39	1.17	2.60	1.39	1.80
NYX	1.19	1.11	1.00	1.37	1.11	1.25
Hurricane	1.52	1.49	1.26	2.28	1.49	1.78

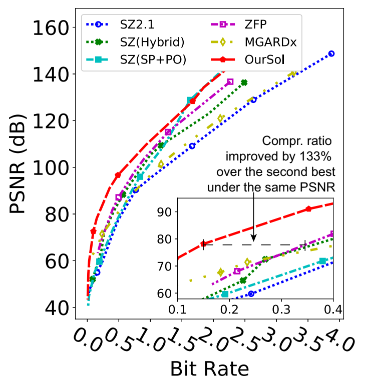


(a) RTM (time step 1500) (b) NYX (field velocityz) (c) Miranda (field velocityz)

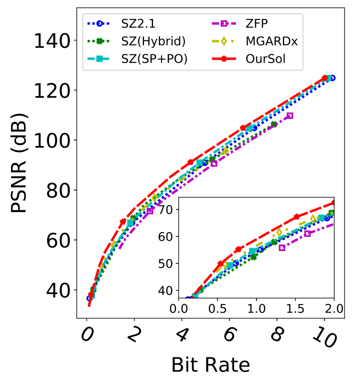
(d) Scale-LETKF (field QS) (e) QMCPack (f) Hurricane(field W)

Figure 3.11: Our solution compared with interpolation and Lorenzo

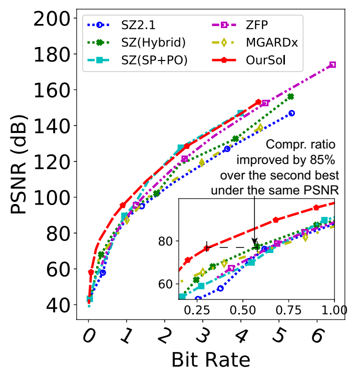
As discussed in Section 3.5.4, we designed a dynamic strategy to optimize the compression quality throughout the entire bit-rate range. Fig. 3.11 demonstrates that the dynamic strategy has a critical effect in the compression quality improvement. For instance, as shown in Fig. 3.11 (a), our solution always exhibits the best compression quality when the bit rate is lower than 2.5 because it adopts a dynamic interpolation method with optimized dimension sequences on a multilevel interpolation, whereas both linear interpolation and tricubic interpolation (shown in the figure) use a fixed sequence. ($z \rightarrow y \rightarrow x$). On the other hand, Fig. 3.11 (a) shows that our solution also keeps the best rate-distortion level when the bit rate is higher than 2.5, a result that is attributed to our accurate predictor selection algorithm (selecting a better predictor between interpolation and Lorenzo at runtime).



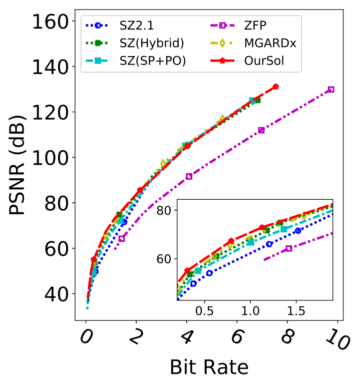
(a) RTM



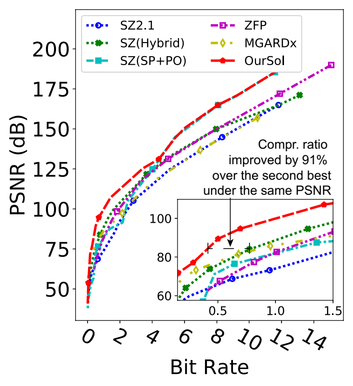
(b) NYX



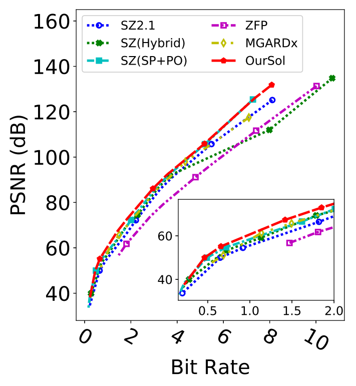
(c) Miranda



(d) Scale-LETKF



(e) QMCPack



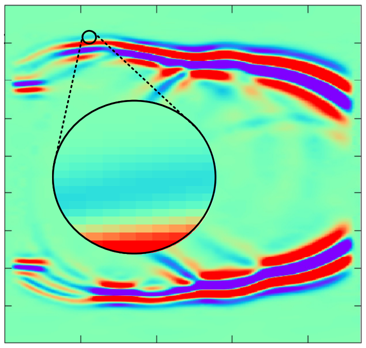
(f) Hurricane

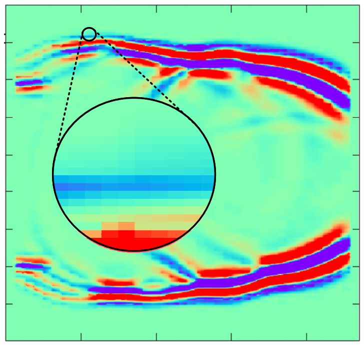
Figure 3.12: Overall evaluation (lower Bit Rate / higher PSNR $\rightarrow$ better quality)

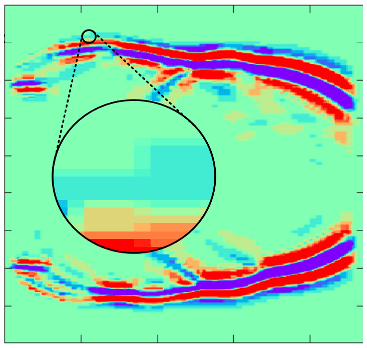
Fig. 3.12 presents the overall compression quality (i.e., rate distortion). One can see that our solution is the best in class from among all the related works for all six applications. In particular, with the same data-distortion level (PSNR), the compressed data size under our solution is about 50% of the compressed data size under the second-best compressor in most of the cases for RTM, Miranda, and QMCPack.

We demonstrate the visual quality of the decompressed data of four error-bounded lossy compressors in Fig. 3.13, using one slice image (slice 340) in the RTM dataset. The original visualization is shown in Fig. 3.4. The figure clearly shows that our solution keeps an excellent visual quality in the decompressed data with a compression ratio even up to 315. In contrast, other compressors suffer from prominent degradation in visual quality to different extents with the same compression ratios. In particular, SZ and ZFP suffer from undesired blockwise texture artifacts.

We show in Fig. 3.14 and Fig. 3.15 that the final RTM image for a single shot is not degraded at all using our lossy compressor with very high compression ratios (about 2$\sim$4$\times$ higher than that of other compressors). We use value-range-based error bound 1.25E-3 in our solution for each time step. The RTM application requires propagating waves generated by a source signal, in a given subsurface model. At the beginning of the propagation the compression ratios are very high (10k+) when the waves are close to the source locations. Over time, the waves are propagating further in the model, resulting in more complex images and compression ratios dropping to about 70. The overall compression ratio is 274 because the compression ratio at most time steps can reach 300+ (e.g., CR=315 at time step 1500 as shown in Fig. 3.13 (a)). In this simulation we used one shot to generate the


(a) OurSol (PSNR:69.3,CR:315)


(b) SZ (PSNR:50.7,CR:315)


(c) ZFP (PSNR:51.7,CR:258)

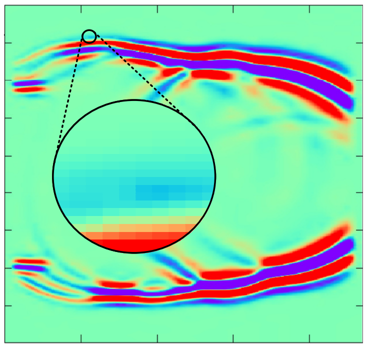

(d) MGARDx (PSNR:62.5,CR:310)

Figure 3.13: Visualization of decompressed snapshot data (RTM)

final image in Fig. 3.15. One can see a very good preservation of amplitudes and main structures of the lossy-compression-based final result, which is acceptable for post-analysis as confirmed by the seismic researchers. Our lossy compressor dramatically decreases the size of the RTM snapshots while not increasing the computation time compared with SZ 2.1. This can significantly lower the I/O throughput requirements and enable either faster turnaround or higher-fidelity simulations for production-level seismic imaging.

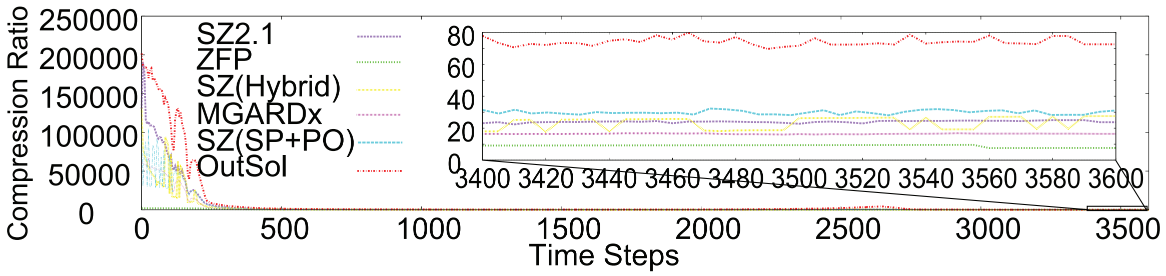
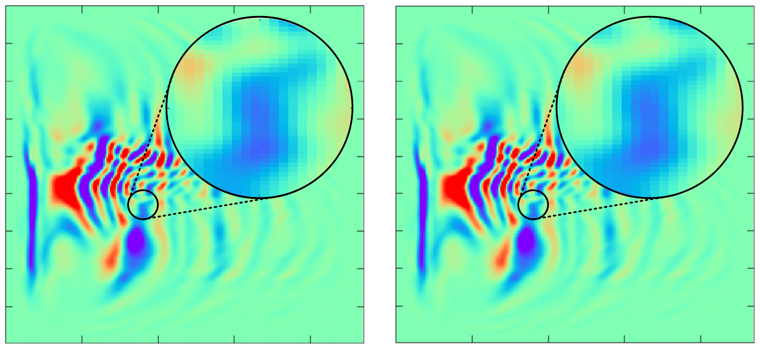


Figure 3.14: Compression ratio of RTM data for different time steps (with value-range-based error bound 1.25E-3)



(a) Original Final Result (b) Compression-based Final Result

Figure 3.15: Visualization of RTM image for one shot

Chapter 4

MDZ: An Efficient Error-Bounded Lossy Compressor for Molecular Dynamics

4.1 Introduction

Molecular dynamics simulations have become one of the most commonly-used methods to study the physical movements of atoms and molecules. For instance, MD simulations are often used to refine 3D structures of proteins and macro-molecules in terms of experimental constraints in X-ray crystallography or nuclear magnetic resonance (NMR) spectroscopy. In physics, MD simulations can be used to study the dynamics of atomic-level phenomena, such as thin-film growth and ion implantation (the atomic-scale details of which are very difficult to observe directly) or to investigate physical properties of nanoscale devices. In biophysics and structural biology, MD simulations are often applied to examine the motions of macromolecules (e.g., proteins and nucleic acids), for interpreting the results of some biophysical experiments and modeling interactions between molecules.

Generally speaking, scientific data can be categorized into three distinct types, including particle data (e.g., locations of atoms), structured mesh (regular multidimensional grid in space), and unstructured mesh (irregular mesh such as triangular grid). MD simulation is one of the most significant/typical particle-based research in the community. As the computational scales at which MD simulations are carried out rapidly increases [94], so does the volume of data generated during the simulations. For example, an atomistic model of the SGLT membrane protein may consist of 240 million frames each with 90k particles, producing a total of $\sim$260 TB of raw trajectory data over a 480 ns simulation [41]. The most recent MD simulations [94] are able to simulate 20 trillion particles in a long trajectory, generating 10 PB of data if there are hundreds of frames to store.

The explosive growth of data volume has brought major challenges to the storage systems designed for saving and managing scientific datasets [20, 34, 49]. For scientific applications, the vast amount of data are generally stored in the form of files [5], for the purpose of convenient post hoc analysis, management, and transfer. How to efficiently store and transfer the large amount of data becomes a serious concern. In fact, for today's supercomputers, a research project generally is allocated only dozens of terabytes of storage space (e.g., 50 TB by default on ORNL Summit [89]) or a few hundreds of terabytes upon requests. Obviously, efficiently reducing the volume of generated data can substantially lower the burden on storage, management and transfer.

In this chapter, we aim at designing an efficient error-bounded lossy compressor for MD datasets, which presents a series of challenges. (1) In MD simulations, each snapshot may contain a large number of particles, so that only a limited number of snapshots can be held in memory and the compression should be done in batches. Therefore, compressors that rely on the time series patterns [54, 71, 110] will have sub-optimal performance, and a practical and effective compressor for MD data should involve both efficient time-based compression and efficient snapshot-based compression. (2) It is very challenging to develop an efficient snapshot-based compression method because the adjacent data values in a snapshot may not be smooth (shown in Section 4.5.2), while existing state-of-the-art lossy compressors substantially depend on the high smoothness of the data in space. (3) Unlike some existing compressors [54] optimized for cosmological N-body simulations, MD compressors could not exploit velocities to help compress position data in most cases, because MD particle often quickly vibrate around their equilibrium positions and velocities are only predictive of future positions for a few femtoseconds in the future (a fraction of a typical vibrational period).

With all the above challenges in mind, we propose a novel error-bounded lossy compressor that is particularly efficient for MD simulations. The key contributions are listed as follows:

- We carefully characterize a variety of MD simulation datasets and exploit some of the key patterns identified in the MD data to significantly improve compression ratios.
- We design an adaptive error-bounded lossy compressor for MD datasets which fully leverages the specific characteristics in both spatial and temporal dimensions.

- We evaluate our solution with six state-of-the-art related works. Experiments show that MDZ can always lead to the best compression quality in various execution patterns. In absolute terms, our solution obtains up to 233% higher compression ratios than does the second best error-bounded lossy compressor.
- We integrate our solution into the MD package LAMMPS. Evaluation shows our solution has negligible time overhead in real-world MD simulations under different scales and settings.
- We discuss the generalizability of our solution and demonstrate MDZ has the best compression quality on datasets beyond MD simulations.

The rest of the chapter is organized as follows. In Section 4.2, we discuss the related work. In Section 4.3, we describe the research background. In Section 4.4, we formulate the research problem. In Section 4.5, we present an in-depth characterization of several MD simulation datasets, which motivates our design and optimization. In Section 4.6, we describe in details our developed MD data compressor - MDZ. In Section 4.7, we present and discuss the evaluation results.

4.2 Related Work

The compression of MD datasets is critical to the cost-effective data processing of MD simulations.

In general, compression techniques can be divided into two categories - lossless compression and lossy compression. Lossless compressors have been deployed in many fields.

For example, Google Brotli [6] and Facebook Zstandard [118] are widely used in industrial data management systems. Gorilla [76] and AMMMO [111] bring lossless methods to time series databases. However, lossless compressors suffer from very low compression ratios in the scientific domain, as demonstrated in Section 4.7.2. The reason is that scientific datasets are mainly composed of floating-point numbers each of which has very random ending mantissa bits so that it is very hard for lossless compressors to catch the repeated patterns during the encoding.

Lossy compression, unlike lossless compression, can reach a higher compression ratio with some information loss. Lossy compression has been adopted in some database systems. For example, ModelarDB [43, 44] is a time series management system with lossy compression built-in. It has three compression algorithms, including the PMC-mean [53], the linear Swing model [27], and the lossless method in Gorilla [76]. ModelarDB uses a window-based approach to find the best algorithm for each data segment. SummaryStore [2] is an approximate time-series store which merges the old data when the space limit is reached. Besides time series databases, there are also some lossy compression studies [16, 29, 57, 69] for GPS trajectory data systems.

Lossy compressors in database systems are not suitable for MD datasets for the following reasons. First, time series databases such as ModelarDB use simple data estimation methods and they do not have quantization or entropy coding process, thus they suffer from low compression ratios on MD datasets (demonstrated in Section 4.7.3). Second, GPS trajectory compressors are not suitable for MD datasets either because MD data is much more unconstrained than the GPS data (note that GPS devices follow direct lines while

MD particles move rather randomly). Third, many database systems such as SummaryStore do not have an error-bounded design such that they cannot guarantee the quality of the decompressed data would satisfy the users' requirements.

Even general lossy compressors for scientific applications such as ZFP [64] and SZ-Interp [112] exhibit sub-optimal results on MD datasets [54], because they are designed and optimized for three-dimensional data. While MD datasets are two-dimensional and are split into batches for compression.

Due to the above limitations, researchers are investigating lossy compressors that are specifically designed for MD datasets. HRTC [41] adopts a piecewise linear representation of trajectories, followed by an error-controlled quantization and a variable length integer representation. Li et al. [54] improved the compression ratio by employing velocity fields to assist the prediction of spatial coordinates. Note that, as mentioned in Section 4.1, this strategy may not be efficiently applied to MD datasets. PMC [26] utilizes information on atomic bonds in a molecule to predict atomic positions in each frame. This method, however, is not suitable for simulations with non-bonded interactions, where connectivity between neighboring atoms can dynamically change during the simulation.

4.3 Research Background

The background of MD simulation is important to the development of our novel error-bounded lossy compressor for MD simulation datasets.

4.3.1 MD Simulations

MD is a type of N-body simulations which is widely used to explore the behavior of materials at the nanoscale. As illustrated in Figure 4.1, a single MD simulation generally involves many time steps, in each of which the new position and velocity of each particle are predicted based on sophisticated calculations of interatomic forces. Force calculation typically consumes the overwhelming majority of the computing time. After adjusting atomic positions based on the calculated forces (shown as the highlighted arrow), boundary conditions are applied and coordinates or physical quantities of interest care calculated and written out.

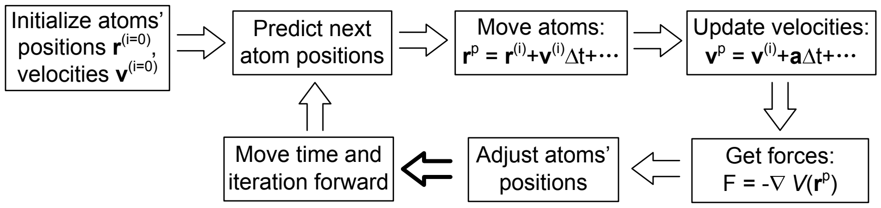

Initialize atoms' positions $\mathbf{r}^{(i=0)}$, velocities $\mathbf{v}^{(i=0)}$
Predict next atom positions
Move atoms: $\mathbf{r}^p = \mathbf{r}^{(i)}+\mathbf{v}^{(i)}\Delta t+\cdots$
Update velocities: $\mathbf{v}^p = \mathbf{v}^{(i)}+\mathbf{a}\Delta t+\cdots$
Get forces: $F = -\nabla V(\mathbf{r}^p)$
Adjust atoms' positions
Move time and iteration forward

Figure 4.1: Illustration of classic MD simulation

A typical MD output is dominated by the storage of coordinate information along the trajectory. Each particle's position is composed of three axes (x, y, z). This is why most of the existing lossy trajectory compressors [26, 41, 72, 86] focus on the positions rather than velocities. As such, in the following, the targets for compression are the particles' positions (x, y, z) alone.

4.4 Problem Formulation

In this section, we formulate the research problem by classifying the input and output of error-bounded lossy compressors in the context of MD simulation datasets.

The research problem can be formulated as follows. Suppose an MD simulation dataset (denoted by $\boldsymbol{D}$) is composed of M snapshots each containing N particles. Atomic positions (represented as three axes values {x, y, z}) need to be stored during the simulation.

In general, compression time should be negligible compared with the time to execute hundreds to thousands of timesteps. In most MD simulations, the stiffness of the equations of motion entails very short timesteps on the order of femtoseconds, which is a small fraction of the vibration period of the fastest modes in the system. Hence, by construction, very little structural changes occur between neighboring timesteps. As transitions that change the topology are typically thermally-activated, simulation data need to be saved only occasionally (i.e., thousands to tens of thousands of timesteps). For applications to estimate systems with fast relaxation processes, e.g., to estimate the viscosity of liquids, or the vibrational properties of solids, a higher frequency might be required (e.g., hundreds of timesteps).

Accordingly, our research target can be summarized as maximizing the compression ratio while keeping the compression and decompression speed fast enough for the MD simulations, and processing the M snapshots in batches instead of compressing the entire dataset D at once.

Based on the above problem definition, traditional pure trajectory compression methods [41, 54, 86] are not suitable, since they need to collect a large number of snapshots for the compression, and decompressing any one snapshots needs to decompress all its preceding snapshots as well. Moreover, single-snapshot based compression [75, 90] is not an ideal solution either, in that it will suffer from low compression ratios because of the non-smooth nature of the spatial particle data. To address these issues, we propose MDZ which makes full use of the characteristics of MD datasets in both spatial and temporal dimensions to significantly improve compression ratios.

4.5 Investigation of MD Datasets

In this section, we identify key characteristics and patterns from a number of MD datasets. Specifically, we first analyze the spatial patterns present in MD datasets and then investigate their temporal features.

4.5.1 MD Simulations Used in Our Work

Table 4.1 summarizes the eight MD simulation datasets. For Copper and Helium datasets, we include two broad execution modes, noted A and B. In mode A, each snapshot involves a relatively large number of atoms (generally more than 100K atoms). These are typical of conventional large-scale MD simulations. In mode B, each simulation involves a large number of timesteps and a relatively small number of atoms (such as 1k atoms). This mode is more typical of long-timescale simulations, e.g., using methods such as Parallel Trajectory Splicing [77]. The eight datasets are described as follows.

Copper (Mode A&B): The data comes from the study of the influence of strong electric fields on copper in the context of particle accelerators. The mode A sample contains

Table 4.1: MD simulation dataset in our study

Application	**State**	**Code**	**Snapshots**	**Atoms**
Copper-A	Solid	LAMMPS	83	1077290
Copper-B	Solid	LAMMPS	5423	3137
Helium-A	Plasma	LAMMPS	2338	106711
Helium-B	Plasma	EXAALT	7852	1037
ADK	Protein	CHARMM	4187	3341
IFABP	Protein	CHARMM	500	12445
Pt	Solid	LAMMPS	300	2371092
LJ	Liquid	LAMMPS	50	6912000

1077290 atoms and the mode B sample has 3137 atoms. The time evolution was obtained by molecular dynamics method using the LAMMPS code [79] in the canonical ensemble at a temperature of 800K. The simulation was run on up to 30 nodes (1024 cores) of the Grizzly [35] supercomputer at Los Alamos National Laboratory (LANL).

Helium (Mode A): This dataset contains simulations of the growth of helium bubbles embedded in a body-centred cubic tungsten matrix. The simulation cell contains 106711 atoms. Helium atoms are gradually inserted in the bubble as the simulation proceed, mimicking the agglomeration of helium atoms incoming from the plasma into the first wall of a fusion reactor. The simulations were carried out with the Parallel Replica Dynamics method [106] using the LAMMPS code [79]. Simulations were carried out on up to 1000 nodes of the Trinity supercomputer [102] at LANL.

Helium (Mode B): This dataset contains simulations of small vacancy/helium clusters in a body-centred cubic tungsten matrix. The simulation cells contain 1037 atoms. Long-time simulations were carried out with the Parallel Trajectory Splicing methods [77] to investigate the mobility of these defects formed by helium atoms incoming through the plasma in contact with the first wall of fusion reactors [78]. These simulations were carried out on up to 2000 nodes of the Trinity supercomputer at LANL.

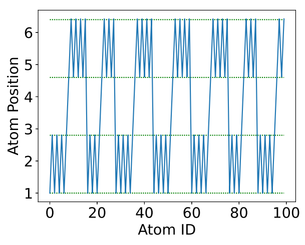
ADK: This dataset is from the simulation of adenylate kinase (ADK) which is the critical enzyme controlling the energy balance in cells. According to Seyler [85], ADK was simulated with explicit water and ions in isothermal–isobaric ensemble settings with temperature being 300 K and pressure being 1 bar. The experiment was conducted on the Anton supercomputer [8] at Pittsburgh Supercomputing Center. The snapshots contains 3341 atoms and were saved every 240 picoseconds for a total runtime of 1.004 µs.

IFABP: The data comes from an MD simulation with 12445 atoms of intestinal fatty acid-binding protein in water. Fatty acid-binding proteins affect the transfer of fatty acids between cell membranes while their mechanism are largely unknown. The simulation data is valuable for studying protein dynamics, protein-ion, and protein-water interactions [13]. The experiment was running for 500 picoseconds using CHARMM [14]. The timestep is set to 2 femtoseconds and the snapshots are saved every 1 picosecond.

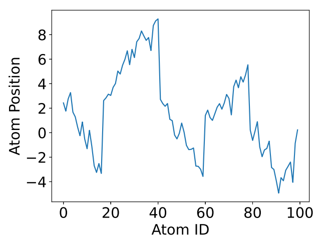
Pt: The data corresponds to an MD simulation of surface diffusion and adatom clustering on a platinum surface. The model had 2371092 atoms and was run for 32M timesteps using the local hyperdynamics methodology. More details on the method and simulation analysis are given in [80]. The simulation was run on 64 KNL nodes (4096 cores) of the Theta supercomputer at ALCF [97].

LJ: This simulation dataset was generated by the Lennard-Jones liquid benchmark [67]. The Lennard-Jones potential estimates the potential between particles based on the particle distance. LAMMPS includes the Lennard-Jones potential as one of the simulation benchmarks. The simulation cell contains 6912000 atoms. The simulation was run on up to 500 cores of the Bebop supercomputer [12] at Argonne National Laboratory.

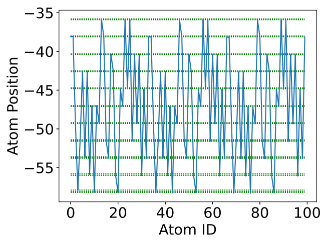
4.5.2 Characterization of Spatial Features



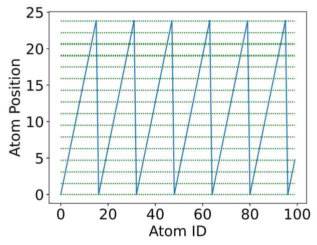
(a) Copper-B



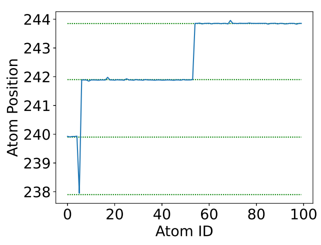
(b) ADK



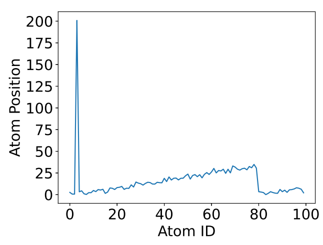
(c) Helium-A



(d) Helium-B



(e) Pt

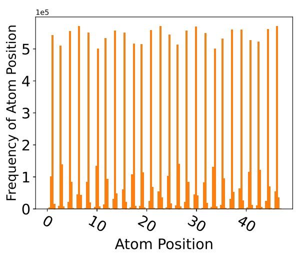


(f) LJ

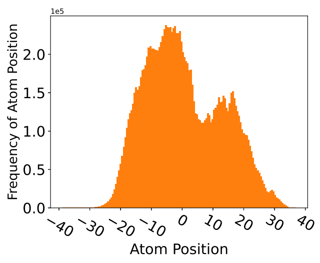
Figure 4.2: Demonstration of spatial correlations in atom position data

Takeaway 1: Our first critical observation is that in many cases, the MD datasets exhibit various patterns in the spatial domain. Due to the space limit, we give six typical examples (including Copper-B, ADK, Helium-A, Helium-B, Pt and LJ) to demonstrate the diverse spatial patterns in Figure 4.2. As illustrated in the figure, the dataset may exhibit a stable zigzag pattern (Figure 4.2 (a) (d)), an erratic zigzag pattern (Figure 4.2 (c) (f)), a stair-wise pattern (Figure 4.2 (e)), or a random pattern (Figure 4.2 (b)).

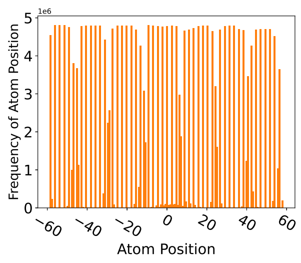
Takeaway 2: We also observe from Figure 4.2 (a) (c) (d) that in many cases, the data are clustering into several equal-distant discrete levels in the whole value range. In fact, for all the data points that are clustering at a specific level, their positions actually vibrate in a small range and are not strictly constant. These regular patterns emerge from



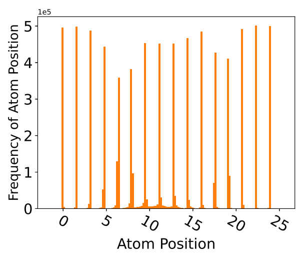
(a) Copper-B



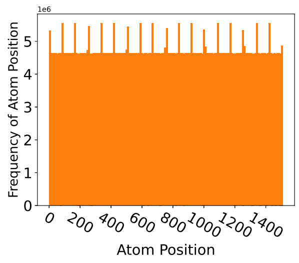
(b) ADK



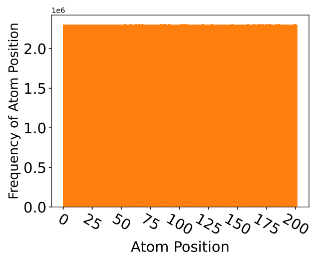
(c) Helium-A



(d) Helium-B



(e) Pt



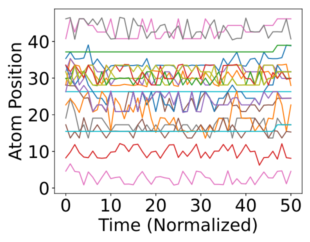
(f) LJ

Figure 4.3: Demonstration of frequencies of atom position data

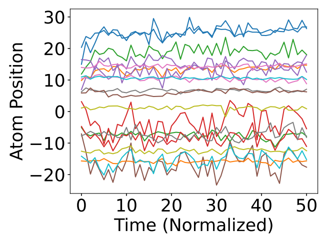
the crystalline structure of the underlying materials. The observed zig-zag patterns are also typical of how crystalline samples are usually created. Such patterns can change with time, as the structure of the materials evolve.

Takeaway 3: Based on Figure 4.2, we also learn that the atom's coordinate may jump from one discrete level to another nearby level, point by point throughout the whole dataset. Since many prediction-based compressors such as SZ simply predict each data point based on its preceding data points without explicitly using the discrete levels, it would definitely suffer from relatively low prediction accuracy in this situation, leading to low compression ratios (as discussed in Section 4.3).

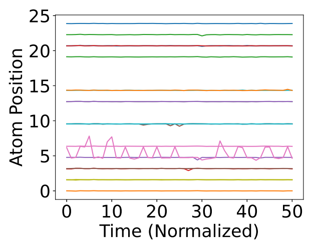
As mentioned above, we observe that the data values often cluster and vibrate around a number of different discrete levels, which can be verified by the distribution of the data (as presented in Figure 4.3). As shown in the figure, the distribution of any MD



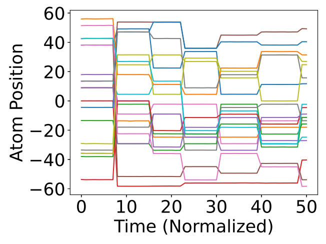
(a) Copper-B



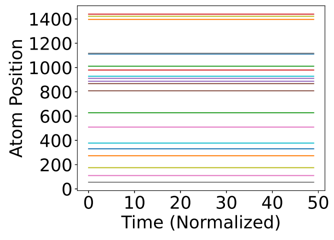
(b) ADK



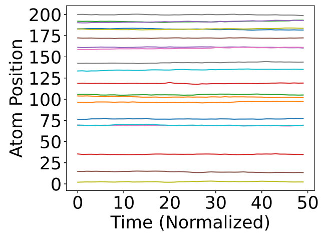
(c) Helium-A



(d) Helium-B



(e) Pt



(f) LJ

Figure 4.4: Demonstration of temporal correlations in atom position data (time is normalized to 0-50)

dataset can be split into two categories - multiple-peak-dominated distribution (see Figure 4.3 (a) (c) (d)) and rather uniform distribution (see Figure 4.3 (b) (e) (f)). The former clearly indicates the strong clustering feature of the data in many cases, which is consistent with our analysis based on Figure 4.2.

4.5.3 Characterization of Temporal Features

Datasets which have no prominent spatial patterns often exhibit particular temporal correlations that can be used to achieve very high compression ratios. Figure 4.4 presents the position data in the time dimension (i.e., trajectories of atomic positions) for six datasets. It is clearly observed that the data value always exhibit more or less correlations in the time dimension. Basically, there are two correlation levels as summarized below. (1) The data may change relatively largely and frequently for some datasets (such as Copper-B, ADK, and Helium-B). (2) The data may change slightly in some situations (such as Helium-A, Pt, and LJ).

Takeaway 4: One very useful observation is that for the datasets which exhibit low spatial patterns, for example Pt and LJ, they often have extremely strong data smoothness in the time dimension and a large majority of the data are extremely close in the time dimension throughout the whole simulation.

Based on the four important takeaways explored in our characterization, we develop an adaptive error-bounded lossy compressor for the diverse MD datasets which can significantly improve the compression ratios over the existing state-of-the-art MD compressors.

4.6 MDZ: An Adaptive Error-bounded Lossy Compressor for MD Datasets

The basic design idea is selecting the best method from among three compressors best suited to diverse data features in both the spatial and temporal dimension.

Figure 4.5 summarizes the design of MDZ. Basically, the datasets are generated by the data source such as the MD simulation, as illustrated in Figure 4.1. As mentioned in Section 4.1 and Section 4.4, the MD applications need to perform the compression operation periodically in order to avoid out-of-memory issue. The snapshots to be compressed are stored in a buffer and the buffer size (BS) is defined as the maximum number of snapshots to be kept in the buffer. Finally, the compressed data will be stored into the parallel file systems (PFS).

As illustrated in Figure 4.5, the entire compression pipeline involves four critical steps: prediction, optimized quantization, encoding and Zstd, following the classic SZ compression framework [25, 91]. Our key contribution involves improvements to the prediction and quantization stages.

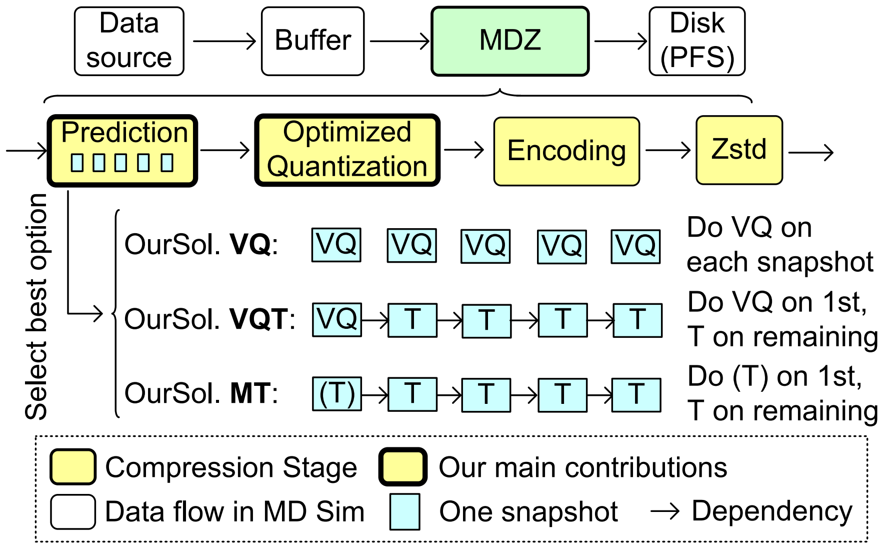


Figure 4.5: Design overview (VQ,VQT,and MT are described in Section 4.6.1 and 4.6.2)

Specifically, we design three efficient MD data prediction strategies to adapt to diverse data patterns in the MD datasets:

- Vector-quantization-based compressor (abbreviated as **VQ**): The VQ compressor predicts the data values totally based on the spatial information, thus the data prediction for any one snapshot has no dependencies on any other snapshots, such that any snapshot data can be decompressed very quickly without a need in decompressing other snapshots. This is particularly effective on the datasets with very low smoothness in time dimension (see Figure 4.4 (a) (b)).

- Vector-quantization-time-based compressor (abbreviated as **VQT**): The VQT compressor adopts the VQ predictor on the first snapshot in the buffer, and adopts a time-based predictor (i.e., predict each data point using the corresponding data values in the previous snapshot) for all the remaining snapshots in the buffer. This method is designed particularly for datasets that have smooth time dimension and also have strong multi-peak-distribution patterns in space (see Figure 4.4 (c) and (d)).
- Multi-level-time-based compressor (abbreviated as **MT**): The MT compressor adopts a particular data prediction method - called *initial-time-based prediction* (shown as the notation (T) in Figure 4.5)), and applies the ordinary time-based predictor on all other remaining snapshots in the buffer. This method is particularly effective on the datasets with very high smoothness in the time dimension (see Figure 4.4 (e) (f)).

We describe in detail the compressors with optimized prediction and quantization methods in the following subsections.

4.6.1 Vector-Quantization-Based Compression (VQ and VQT)

The basic idea of the VQ algorithm is to leverage the spatial patterns characterized in Section 4.5.2 (i.e., takeaway 2 and takeaway 3). Takeaway 2 indicates that the data are clustering into different roughly equal-distant discrete levels (as shown in Figure 4.2 and Figure 4.3), which motivates us to use the centroid (a.k.a., center) of each cluster to predict all the data values within this cluster.

We present the pseudo-code of the VQ algorithm in Algorithm 2. The first step is computing the level distance λ and the initial level value μ (line 1), based on which every level value can be retrieved easily. Line 2∼8 is is the main compression procedure, including data prediction (line 4∼5), computation of level index (line 6) and quantization (line 7).

Algorithm 2 VECTOR-QUANTIZATION-BASED COMPRESSION (VQ)

```
Input: raw MD data $D$ (single snapshot)
Output: compressed data (byte stream)

 1: Compute the level distance $\lambda$ and the initial level value $\mu$ by the sampling-based KMeans;
 2: Store $d_o$ as it is;
 3: for $d_i \in D$, where $i$=1, 2, $\cdots$, $N$ do
 4:     $L_i \leftarrow$ Round($\frac{d_i - \mu}{\lambda}$); /*Compute level*/
 5:     $V_i \leftarrow \mu + \lambda \cdot L_i$; /*Get the level's centroid value - VQ based predictor*/
 6:     $j_i \leftarrow L_i - L_{i-1}$; /*Compute relative level index*/
 7:     $b_i \leftarrow (\frac{d_i - V_i}{e} + 1)/2$; /*linear-scale quantization, where $e$ is error bound*/
 8: end for
 9: $\hat{B} \leftarrow$ HUFFMAN$_{b_i \in B}(B)$; /*Huffman encoding on quantization codes*/
10: $\hat{J} \leftarrow$ HUFFMAN$_{j_i \in J}(J)$; /*Huffman encoding on level index codes*/
11: ZSTD($\hat{B} + \hat{J}$); /*Compress Huffman output by Zstd [118]*/
```

We illustrate the key steps (data prediction and quantization) of the VQ compression algorithm in Figure 4.6. The figure shows a snippet of the dataset (i=10 → 26). As we mentioned previously, the data values are clustered at different levels with a small vibration, so we use the corresponding level's centroid value to predict each data point. As such, the quantization bin (see the red number 3 in the figure) is calculated based on the prediction error (i.e., $d_i - V_i$). The vector B is used to hold the quantization bins, and J is used to hold the relative index numbers. Both of them will be compressed by Huffman encoding later on (line 9-10 in Algorithm 2).

As mentioned above, we develop an efficient sampling-based 1D K-means clustering algorithm to identify the level distance and initial level value. In what follows, we first describe the basic K-means algorithm and then discuss how we boost its performance in the context of compression.

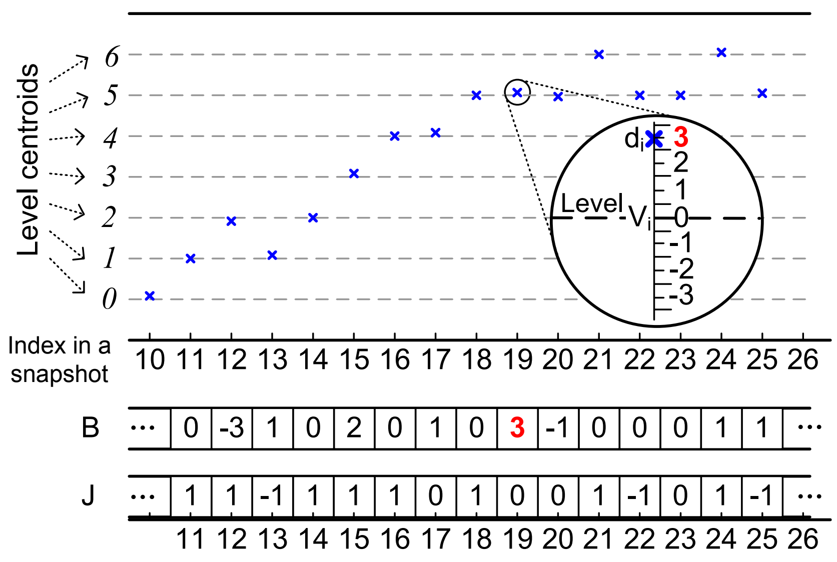


Figure 4.6: Illustration of VQ-based prediction + quantization

Unlike the time-consuming 2D K-means problem, optimally partitioning N sorted 1-dimensional data points to K groups has polynomial time complexity solutions. Define the sorted data points as $d_1, d_2, .., d_N$, and the cost of clustering as the summation of the distance between the data points and their centroid points. In Formula (4.1), we define $Cost(l, r)$ as the optimal cost of clustering $d_l, .., d_r$ to one group, $F(n, k)$ as the optimal cost of clustering $d_1, .., d_n$ to k groups, and $H(n, k)$ as the argument that minimizes F.

$$\begin{aligned}
&Cost(l, r) = \textstyle\sum_{i=l}^{r}(d_i - \frac{\sum_{j=l}^{r} d_j}{r-l+1}) \\
&F(n, k) = \min(F(i-1, k-1) + Cost(i, n), \forall 0 < i <= n) \\
&H(n, k) = \arg\min(F(i-1, k-1) + Cost(i, n), \forall 0 < i <= n)
\end{aligned} \tag{4.1}$$

The boundaries of clusters can be restored from H iteratively. The naïve implementation to solve $F(N, K)$ has $O(KN^2)$ time complexity, and we adopt a solution [36] that optimizes the computational cost to $O(KN)$.

In our case, the number of clusters K is unknown and the data points are unsorted. To boost the performance, on the one hand, we compute F only once during the whole simulation, and we compute it on a sampled dataset that has 10% data points from the first single snapshot. We observe the snapshots have unchanged level patterns during the simulation thus the result on the first snapshot is applicable for the following snapshots. On the other hand, note that the value of $F(N,1), F(N,2), .., F(N,K)$ are computed in order when computing $F(N,K)$. Let $G(k) = \frac{F(N,k)}{F(N,k-1)}$; we stop the computation of F at κ if $G(\kappa)$ decreases significantly than $G(\kappa-1)$. The maximum test value of K is set to 150 as a higher number of clusters will harm the compression ratio of the vector quantization indexes. The level distance λ and initial level value μ are computed using the boundaries obtained from H.

For the VQ compression method, we adopt the VQ algorithm on each snapshot, as illustrated in Figure 4.5. By comparison, the VQT compression method applies the VQ algorithm only on the first snapshot in each buffer, and all other snapshots in the buffer will be compressed by the classic time-based compression. Specifically, each subsequent data point will be predicted using the corresponding data value in the previous timestep. This may significantly improve the compression ratio especially in situations with relatively smooth data in the time dimension (see Figure 4.4 (c)-(f)).

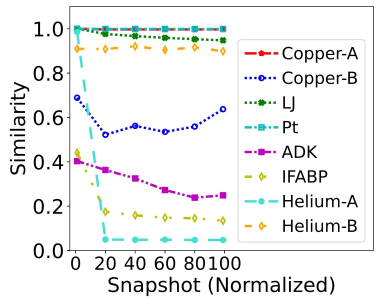
4.6.2 Multilevel Time-Based Compression (MT)

We propose an additional error-bounded compression method - called multi-level-time-based compression (MT), which is particularly effective for the datasets with extremely high smoothness in the time dimension.

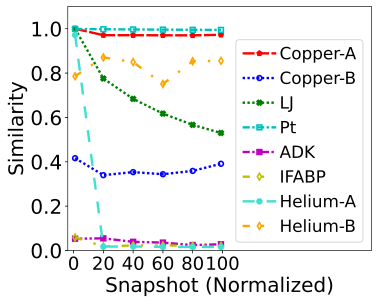
The MT compression algorithm also adopts the prediction-based compression model. The particular design of MT is that the first snapshot in the buffer will be predicted based on the initial snapshot of the whole dataset, which is motivated by the very strong correlation between all the simulation snapshots and the initial snapshot in some datasets. Figure 4.7 shows the similarity of all the snapshots compared with the initial snapshot (i.e., snapshot 0). The similarity is defined in Formula (4.2).

$$Similarity(\tau, i) = \frac{Count(|\frac{S_i[j]-S_0[j]}{S_i[j]}|<\tau, \forall j)}{Count(S_i)} \tag{4.2}$$

where τ refers to a threshold, S_i refers to snapshot i, $S_i[j]$ refers to the jth data point in the snapshot S_i. The similarity formula calculates the percentage of the "unchanged" data points based on threshold τ. The figure demonstrates that succeeding snapshots in some datasets such as Copper-A and Pt are always extremely similar to the initial snapshot.



(a) Threshold = 10%



(b) Threshold = 1%

Figure 4.7: Snapshots similarity with snapshots-0 (snapshots normalized to 0-100)

Using the snapshot-0-based prediction, the prediction error could be much lower than using the Lorenzo predictor, as presented in Table 4.2.

Table 4.2: Prediction errors for the first snapshot in buffer

Method	Pt			LJ			Helium-B		
	x	y	z	x	y	z	x	y	z
Lorenzo+Regression	3.46	5.50	3.88	5.87	3.87	3.75	5.83	1.14	0.83
MT(Snapshot-0-based)	0.13	0.13	0.01	1.37	1.37	1.38	0.08	0.07	0.09

4.6.3 Linear-Scale Quantization Optimizations

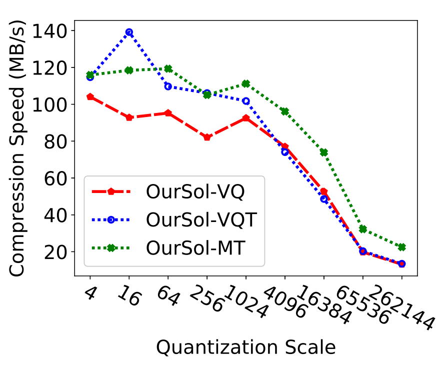
In this section, we optimize the linear-scale quantization step by tuning two quantization settings to further improve the overall compression performance and quality.

Optimization of Quantization Scale

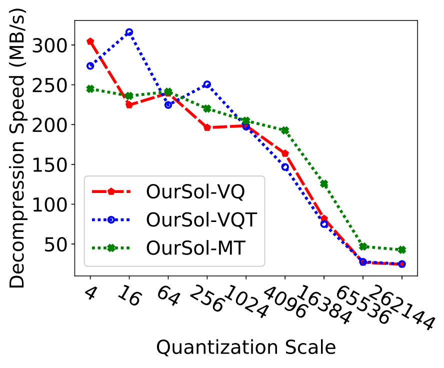
The quantization scale controls the value-range of the quantized integers. The data points that are out-of-scope will be marked with reserved integer value and stored separately. A smaller scale will increase the number of out-of-scope data points which impacts the compression ratio, while a larger quantization scale leads to a bigger Huffman tree such that the Huffman coding will be slower. In Figure 4.8, we illustrate the compression/decompression speed with different quantization scale settings. The compression speeds of VQ, VQT, and MT decrease from 95MB/s, 109MB/s, 119MB/s to 19MB/s, 20MB/s, 32MB/s respectively when the quantization scale changes from 64 to 65536. As such, in our solution, we set the optimal quantization scale to 1024, which can always keep a high compression performance while preserving a high compression ratio.

Optimization of Quantization Sequence

The quantization sequence controls how the integers from multiple snapshots are stored together as 1D array for the Huffman coding and dictionary coding. We denote Seq-



(a) Compression



(b) Decompression

Figure 4.8: Compressor performance affected by quantization scale on Helium-B dataset (value-range-based error bound (ϵ) = 1E-3, BS = 10)

1 as storing one snapshot first, then storing the following snapshots. Seq-2 is denoted as storing one particle in all snapshots first, then storing the following particles. We observe that Seq-2 is better than Seq-1 in terms of compression ratio, especially when the data is stable in time (as shown in Figure 4.4 (c) (e) (f)). When many data points remain unchanged in the time dimension and if they are put together as required by Seq-2, the dictionary coder will have better compression results. Table 4.3 demonstrates compression ratios of the two sequences on Helium-B dataset. The second row of the table is the value-range-based error bound (ϵ), and the corresponding absolute error bound is $value\_range \times \epsilon$. The table shows Seq-2 improves the compression ratio by 37.8%, 37.6%, and 39.7% over Seq-1 on axis x, y, and z respectively. As a result, we adopt Seq-2 in our solution.

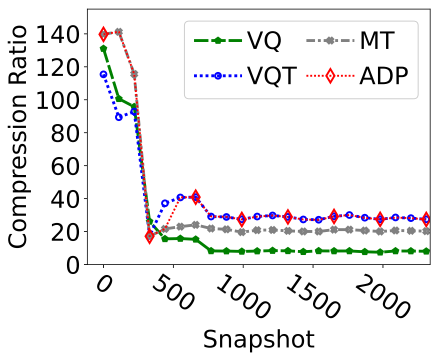
Table 4.3: Compression ratio (CR) of Helium-B dataset with difference sequence settings, buffer size (BS) = 10 (method=MT)

Axis	X			Y			Z		
ϵ	1E-1	5E-2	1E-2	1E-1	5E-2	1E-2	1E-1	5E-2	1E-2
Seq-1	156	97	46	176	101	47	146	97	46
Seq-2	**215**	**132**	**53**	**236**	**139**	**54**	**204**	**133**	**53**

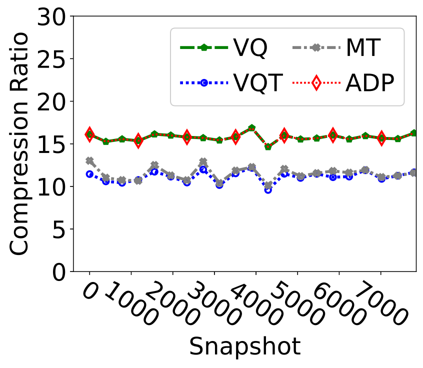
4.6.4 Adaptive Selection of Best Compressor (ADP)

In this section, we propose our adaptive solution (ADP) that can select the best compressor (VQ, VQT or MT) dynamically at runtime. MDZ uses ADP by default to simplify the compression configuration, while manually choosing VQ, VQT, or MT as the compressor is also supported in MDZ.

We notice that during the simulation, the data patterns stay the same in a short term and the patterns (either spatial or temporal) may change prominently in the long term. Furthermore, the best compressor keeps its advantage across some snapshots, but it may not be the best one on all the snapshots. As illustrated in Figure 4.9 (a), MT has the highest compression ratio before snapshot 400 and VQT becomes the best compressor after that snapshot. As a result, we propose to evaluate the three compressors (VQ, VQT, and MT) periodically by using them to compress the same data batch independently and selecting the one with the best compression ratio for the following snapshots. The evaluation will be invoked every 50 compression operations. This time interval ensures that the best compressor is updated in time, while keeping the updating overhead low (less than 6% of the total compression time). Figure 4.9 confirms the effectiveness of our adaptive solution (ADP). All other datasets exhibit similar results (i.e., our ADP algorithm can always select the best solution accurately).



(a) Helium-A, ϵ=1E-3



(b) Helium-B, ϵ=1E-3

Figure 4.9: Illustration of smooth CR in short term and diverse CR in long term (BS=10). ADP can pick up the best compressor throughout all the snapshots.

4.7 Experimental Evaluation

In this section, we present the experimental settings and the evaluation results of our solution on eight MD simulation datasets.

4.7.1 Experimental Setting

Execution Environment

The experiments are executed on the Bebop supercomputer [12] at Argonne National Laboratory with up to 216 cores. Each node in Bebop is equipped with two Intel Xeon E5-2695 v4 processors and 128GB memory.

Datasets

The experiments are evaluated on eight real-world MD simulation datasets. The detailed information about the datasets is presented in Section 4.5.1 and Table 4.1.

State-of-the-Art Lossless Compressors in Our Evaluation

We evaluate six lossless compressors as a comparison with lossy compressors. We include Zstd, Brotli, and Zlib which are widely used in databases and file systems. We also include ZFP, Fpzip, and FPC which specifically target the floating-point data format and are the state-of-the-art lossless compressors for scientific datasets.

State-of-the-Art Lossy Compressors In our Evaluation

We compare our solution with two MD data compressors, two widely used scientific data compressors, as well as two state-of-the-art time series compressors.

- TNG [71]: a MD compressor that uses quantization, delta coding, and a set of integer compressors to compress the trajectory data. TNG is supported by the MD simulation package GROMACS [105].

- HRTC [41]: a lossy compressor targets on MD trajectory compression. HRTC relies on piecewise linear function to approximate data points.

- ASN [54]: a scientific compressor designed for N-body simulation that utilizes the time dimension for prediction.

- SZ2 [59]: a prediction based error-bounded lossy compressor. SZ is widely used in many scientific domains.

- MDB: a full C++ implementation of ModelarDB's compression solution. ModelarDB is described in Section 4.3. ModelarDB tightly couples its compressor with many database features which are useless for scientific data and introduce extra overhead. As such, we eliminate the overhead caused by those features for a fair comparison.

- LFZip [18]: a lossy compressor designed for multivariate floating-point time series data. LFZip is a prediction-based lossy compressor. We evaluate LFZip with its normalized least mean square (NLMS) predictor and skip its neural network (NN) predictor because the NN predictor requires training and is 2000X slower than the NLMS predictor according to the authors [18].

SZ supports both 1D mode and 2D mode. Table 4.4 presents the compression ratios of SZ in the two modes. We can observe from the table that the 2D mode has up to 200% higher compression ratios than 1D, because 2D mode can utilize the data continuity in the space and time dimension at the same time. In our experiments, we use 2D mode for SZ.

Table 4.4: Compression ratios of SZ in 1D and 2D modes (BS=10, ϵ=1E-3)

Method	Mode	Pt			LJ			Helium-A		
		x	y	z	x	y	z	x	y	z
SZ2	1D	150.5	139.6	38.9	6.35	6.45	6.53	7.18	7.28	6.36
	2D	356.5	371.6	32.1	12.26	12.40	12.44	11.11	12.03	11.58

Excluded Cases

HRTC has runtime exceptions on Copper-A, Helium-A, Pt, and LJ datasets. TNG has runtime exceptions on Pt and LJ datasets. A possible reason is the number of atoms is larger than their upper limit. As a result, no HRTC or TNG results are shown on those datasets.

4.7.2 Evaluation Results and Analysis of Lossless Compressors

We first evaluate the six state-of-the-art lossless compressors. Table 4.5 shows the compression ratios of the lossless compressors on four of the MD datasets (results are similar on other datasets). It is clear that all the lossless compressors have extremely low compression ratios (around 1∼2). The results confirms our statement in Section 4.2 that lossless compressors are not suitable for scientific applications.

Table 4.5: Compression ratio comparison of lossless compressors

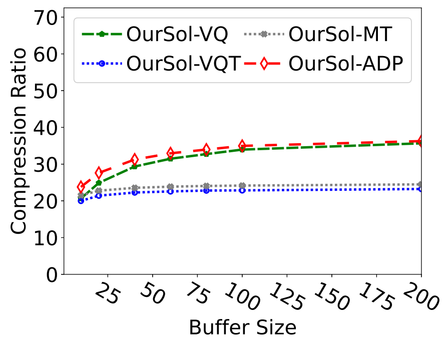
Dataset	Zstd	Zlib	Brotli	Fpzip	FPC	ZFP
Copper-A	1.13	1.15	1.14	1.41	1.18	1.47
Helium-B	1.38	1.33	1.37	1.29	1.22	1.30
ADK	1.08	1.07	1.08	1.26	1.09	1.21
LJ	1.23	1.31	1.24	1.44	1.16	1.39

4.7.3 Evaluation Results and Analysis of Lossy Compressors

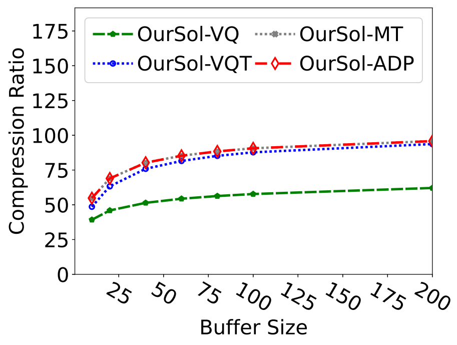
The evaluation involves two aspects - the compression quality and performance. On the one hand, the evaluations of compression error, compression ratio, and rate-distortion demonstrate that our solution has superior compression quality over other state-of-the-art lossy compressors. On the other hand, the performance evaluation reveals that our solution has near the top compression and decompression throughput.

Compression Ratio

Figure 4.10 demonstrates the compression ratio of our solutions. We can observe that ADP has the highest compression ratio among our solutions under different datasets and buffer size settings. It further confirms our claim in Section 4.6.4 that ADP can always



(a) Copper-B, ϵ=1E-2

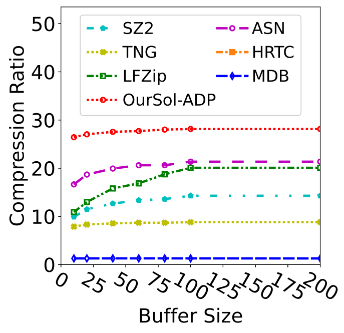


(b) Helium-B, ϵ=1E-2

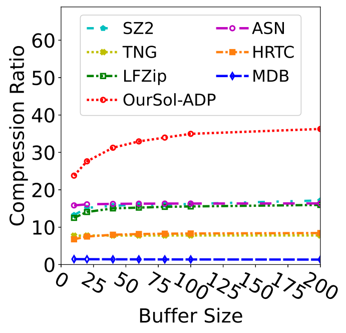
Figure 4.10: Our adaptive solution (ADP) has the highest compression ratio over VQ, VQT, and MT under different datasets and buffer size (BS) settings, because ADP can always select the best compression method accurately.

select the best compressor from VQ, VQT and MT accurately during runtime. As such, we focus on ADP in the following evaluation section.

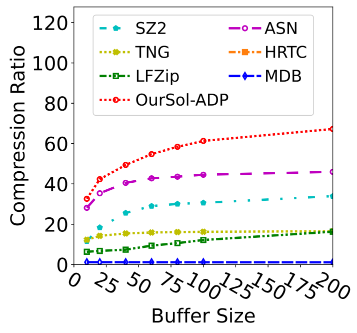
Figure 4.11 compares the compression ratio of the lossy compressors in different buffer size settings. It clearly shows that our solution always has the highest compression ratio on all the eight datasets with any buffer settings. In particular, when buffer size is 100, our solution has 31%, 114%, 38%, 84%, 6%, 27%, 96%, 233% compression ratio improvements over the second-best on Copper-A, Copper-B, Helium-A, Helium-B, ADK, IFABP, Pt, and LJ datasets respectively. MDB has extremely low compression ratios (1∼6) on all the datasets, as shown in Figure 4.11. The result confirms our statement in Section 4.3 that simple data estimation methods and the lack of quantization and entropy coding make ModelarDB suffer from low compression ratios on MD datasets. As a comparison, LFZip, which is also a time series compressor, has comparable results with other lossy compressors, because LFZip has the adaptive linear predictor as well as quantization and entropy coding steps. However, LFZip is still not as good as our solution. The key reason



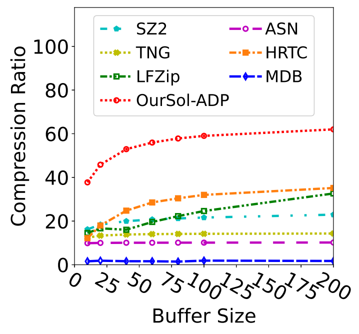
(a) Copper-A (ϵ=5e-4)



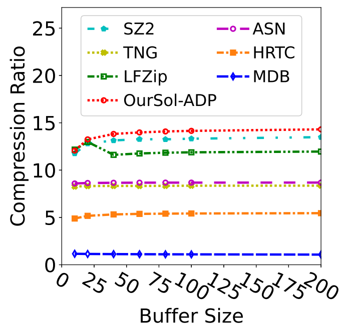
(b) Copper-B (ϵ=1e-2)



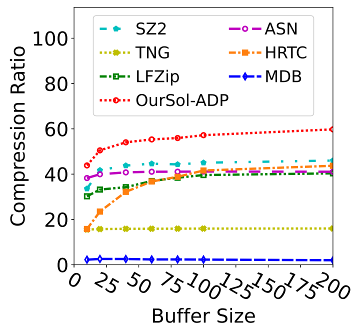
(c) Helium-A (ϵ=1e-3)



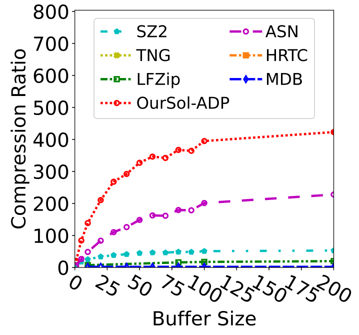
(d) Helium-B (ϵ=5e-3)



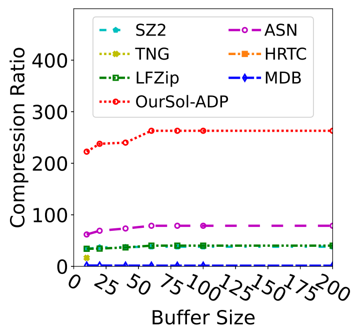
(e) ADK (ϵ=5e-3)



(f) IFABP (ϵ=1e-1)



(g) Pt (ϵ=1e-6)

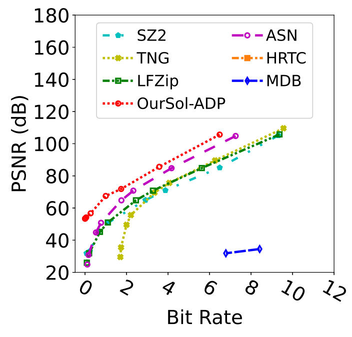


(h) LJ (ϵ=1e-2)

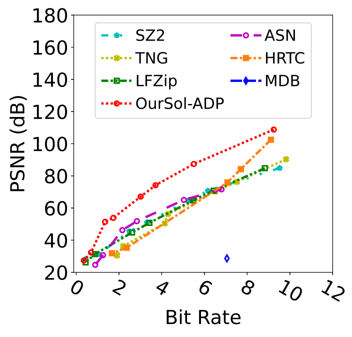
Figure 4.11: Our solution has the highest compression ratio on all datasets and under different buffer size settings, HRTC and TNG fail to run on some datasets.

why our solution has such a high compression ratio is that we investigate and utilize the MD data features in both spatial and temporal dimensions (as shown in Figure 4.2, Figure 4.3, Figure 4.4, and Table 4.2).

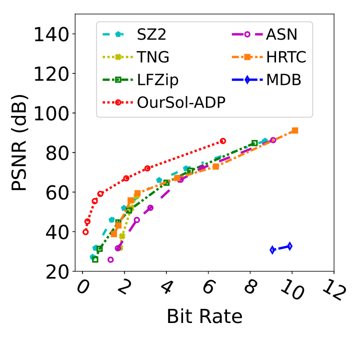
Rate-Distortion



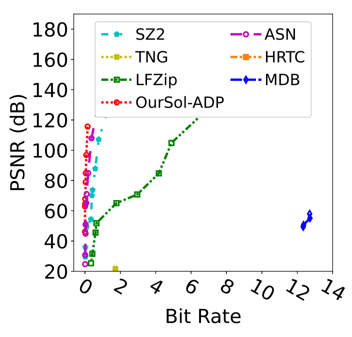
(a) Copper-A



(b) Copper-B



(c) Helium-B



(d) Pt

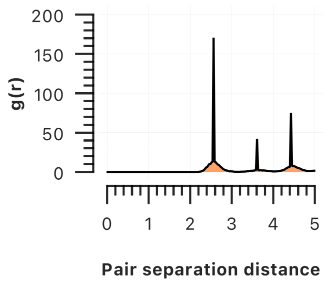
Figure 4.12: Rate-distortion graphs show our solution has the best compression quality. Lower bit rate and higher PSNR indicate better compression quality.

Rate-distortion graph is one of the main assessment metrics of lossy compression quality. Rate-distortion involves bit rate and PSNR. The bit rate is defined as the average bits per data point of the compressed data. PSNR is the peak-signal-to-noise ratio and it is inversely proportional to mean squared error. Lower bit rate or higher PSNR indicts better compression quality. Figure 4.12 presents the rate-distortion results of all the lossy compressors. It is clear that our solution has the best PSNR given the same bit rate (about 20dB improvement in most cases), and also has the lowest bit rate given the same PSNR (about 50% reduction in size in most cases).

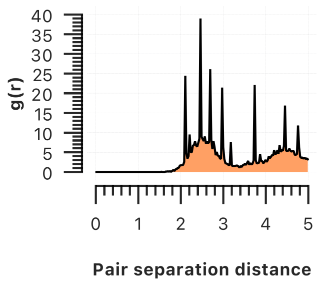
Compression Error

In the domain of lossy compression, compression error is defined as the differences between the decompressed data and the original data. The maximum of the compression error (MaxError) and the normalized root-mean-square error (NRMSE) are two key metrics to evaluate the compression quality of lossy compressors. As an example, we present in Table 4.6 the two error metrics for all the lossy compressors (excluding MDB) based on the Copper-B dataset. Other datasets exhibit the similar results. MDB is excluded from this section because it could not achieve a compression ratio of 10. In this example, the MaxError of ADP always matches the lowest one from VO, VQT, and MT because VQ is always the best on x/y-axis, and MT is always better than the others on z-axis. Thus ADP chooses VQ for x/y-axis and MT for z-axis all the time. In other cases when no compressor is always better than the others, ADP will have even lower MaxError and NRMSE than any of the three compressors. We can observe from Table 4.6 that our solution has the lowest MaxError and NRMSE on all axes. Specifically, the MaxError of our solution is 87%, 87%, 60% lower than the second-best compressor on x, y, and z axis, respectively.

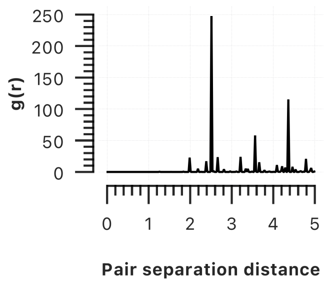
To further demonstrate that our solution upholds the physical characteristics of the data after compression, we present the radial distribution functions (RDFs) of the



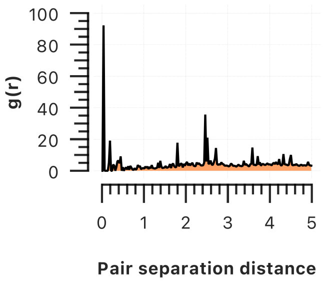
(a) Original Data



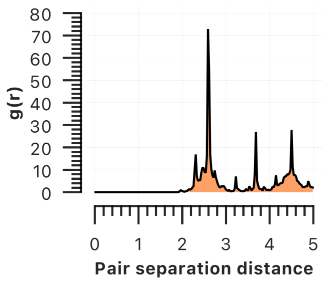
(b) SZ2



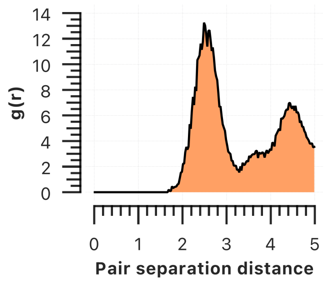
(c) TNG



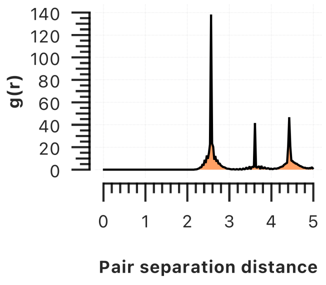
(d) HRTC



(e) ASN



(f) LFZip



(g) Oursol-ADP

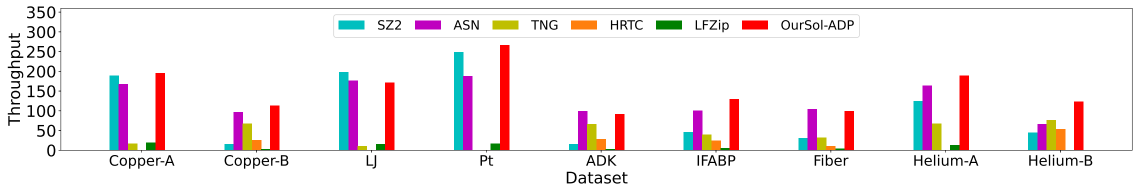
Figure 4.13: Only our solution yields the correct radial distribution function (RDF) on decompressed data (Copper-B, CR=10, BS=10)

original data and decompressed data in Figure 4.13. RDF, denoted as $g(r)$, is a critical analysis metric, which represents the possibility of finding a particle from the base particle at distance r. RDF is proportional to the local density of the particle systems. Figure 4.13 reveals that only our solution could yield the correct RDF on Cooper-B dataset under the same compression ratio. Therefore, only our solution delivers the decompressed data with unaltered local density to downstream applications. In order to get the same RDF as ours, other compressors need to significantly reduce their compression ratios. In summary, the RDF result proves that, with suitable compression ratio, our solution maintains the physical characteristics of the data accurately.

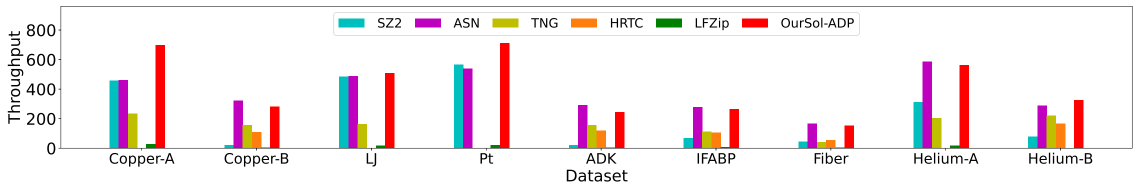
Table 4.6: MaxError and NRMSE of decompressed Copper-B dataset, CR=10, BS=10

Type	Axis	SZ2	ASN	TNG	HRTC	LFZip	OurSol			
							VQ	VQT	MT	**ADP**
Max Error	X	0.37	0.23	0.44	2.06	0.35	0.03	0.10	0.10	**0.03**
	Y	0.32	0.23	0.44	1.94	0.35	0.03	0.10	0.09	**0.03**
	Z	0.16	0.10	0.44	1.13	0.17	0.11	0.05	0.04	**0.04**
NRMSE (×1E-4)	X	45.5	27.4	61.8	133	43.2	3.10	9.00	11.9	**3.10**
	Y	40.0	28.5	61.9	128	43.3	3.10	8.92	10.0	**3.10**
	Z	20.6	9.41	45.2	74.1	21.4	15.3	8.18	5.32	**5.32**

Compression/Decompression Throughput



(a) Compression throughput



(b) Decompression throughput

Figure 4.14: Our solution is the only one that always has high compression/decompression throughput (MB/s) on all datasets. As a comparison, ASN is slow on Pt and Helium-B. TNG and HRTC fail to run on some datasets and LFZip is very slow due its intermediate disk operations.

We present the throughput comparison among all lossy compressors in Figure 4.14. It is clear that our solution is one of the fastest among all the lossy compressors on all datasets. As a comparison, ASN is slower than some compressors on Pt and Helium-B datasets. There are no results for TNG and HRTC on datasets such as Pt due to runtime exceptions, as explained in Section 4.7.1. The results of LFZip are barely visible in this figure because LFZip has intermediate disk operations which bring significant runtime overhead. The excellent performance of our solution is attributed to both effective prediction methods and our optimized quantization settings (see Section 4.6.3 for details).

4.7.4 Integration with LAMMPS

We integrate MDZ into the MD simulation software LAMMPS. To enable MDZ, LAMMPS users only need to adjust the data dumping option in the configuration file.

We executed the Lennard-Jones benchmark in LAMMPS with different settings to evaluate the overhead of our solution in real-world MD systems. The simulation lasts 1 million timesteps and is executed in three different scales, with the number of atoms ranging from 64K to 4096K. We choose data saving frequencies of 1 per 100 timesteps and 1 per 5000 timesteps, which is the range of the typical data saving frequency of MD simulations, as discussed in Section 4.4. The runtime breakdown is shown in Table 4.7. It is clear that enabling MDZ does not affect the output portion of the runtime or the total runtime. MDZ even improves the output performance when the data saving frequency is 100 because the I/O time is significantly reduced due to the reduced file size by MDZ. In general, MDZ has negligible overhead to the MD simulation, and it can improve the simulation performance if a large mount of data needs to be saved and the I/O speed is the bottleneck.

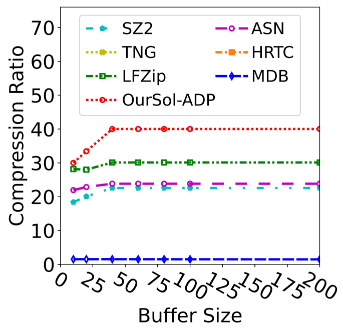
Table 4.7: Runtime breakdown of LJ simulation (F: Data saving frequency, Comp: computation time, comm: communication time, output: data saving time including compression)

F	# Atoms	Option	Duration (minutes)	Runtime Breakdown		
				Comp	Comm	Output
100	64K	w/o MDZ	329	96.4%	1.4%	**2.2%**
		w MDZ	322	98.3%	1.4%	**0.3%**
	512K	w/o MDZ	428	93.9%	3.5%	**2.6%**
		w MDZ	418	95.5%	3.6%	**0.9%**
	4096K	w/o MDZ	516	82.7%	12.8%	**4.5%**
		w MDZ	513	83.0%	12.7%	**4.3%**
5000	64K	w/o MDZ	322	97.0%	2.9%	**0.06%**
		w MDZ	312	98.5%	1.5%	**0.01%**
	512K	w/o MDZ	415	96.2%	3.7%	**0.07%**
		w MDZ	425	93.7%	6.2%	**0.03%**
	4096K	w/o MDZ	474	90.0%	9.8%	**0.14%**
		w MDZ	480	88.9%	10.9%	**0.16%**

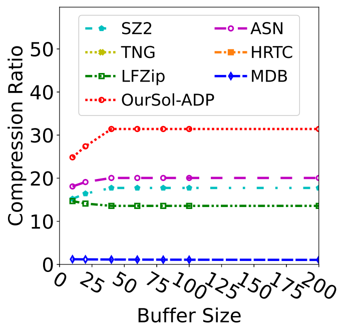
4.7.5 Generalizability of Our Solution Beyond MD Simulations

In this section, we discuss the generalizability of our solution beyond MD simulation datasets. As we mentioned in Section 4.1, scientific data can be categorized into particle data, structured mesh, and unstructured mesh. In order to reach as high compression quality as possible, developers need to design specific solutions for each of the three categories of data. For example, our previous work [112] proposes a customized interpolation-based compressor for structured mesh data (the second category). In comparison, our solution leverages both spatial and temporal data characteristics that exist in many domains. It can be applied to the first category of datasets (all kinds of particle data instead of only MD simulation data).

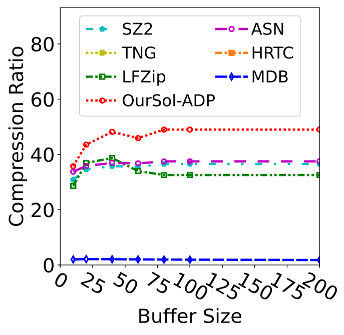
We present the compression ratio evaluation on the Hardware/Hybrid Accelerated Cosmology Code (HACC) datasets in Figure 4.15 to demonstrate the effectiveness of our solution in domains other than MD simulation. HACC is an extreme-scale cosmological simulation code that studies the structure formation in the Universe. HACC saves the positions and velocities of the particles periodically. We include two HACC datasets in this evaluation (HACC-1: 30 snapshots × 15767098 atoms, HACC-2: 80 snapshots × 13131491 atoms). It is clear that our solution is the best among all the compressors on both of the datasets, and it has 30%∼56% higher compression ratios than the second-best compressor.



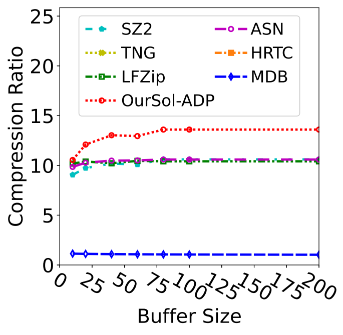
(a) HACC-1 position (ϵ=1e-4)



(b) HACC-1 velocity (ϵ=1e-3)



(c) HACC-2 position (ϵ=5e-4)



(d) HACC-2 velocity (ϵ=5e-4)

Figure 4.15: Our solution has the best compression ratios on HACC datasets

Chapter 5

Conclusions

In this thesis, we present three efficient error-bounded lossy compressors for scientific applications. The key findings are summarized below.

- We extend the SZ2 lossy compressor by adding 2nd-order prediction methods based on Lorenzo prediction and regression prediction. We also develop an efficient algorithm that can select the best-fit predictors and optimized parameter settings at runtime. Experiments with 5 real-world scientific simulations show that the 2nd-order prediction can improve the compression ratio by 50+% and the parameter optimization can further improve the compression by 4%∼50% in most cases. Moreover, our solution has the least overall elapsed I/O times, which are 20%∼40% less than the times when using the second-best lossy compressor.

- We propose a dynamic spline interpolation based compressor with adaptive optimization strategies to replace SZ2. Our analysis shows that the linear regression predictor in SZ2 has a significant problem because its coefficient overhead is non-negligible

(25%∼70% in compressed data). Our dynamic spline interpolation solution can improve the compression ratio by 457%, 244%, and 209% compared with the second-best compressor on RTM, QMCPACK, and Miranda datasets, respectively. Our solution keeps an extremely high visual quality in the decompressed data, whereas other lossy compressors suffer from prominent degradation in visualization with the same compression ratios.

- we develop an error-bounded lossy compressor MDZ for molecular dynamics simulation applications. The key idea is to improve the prediction accuracy based on the regularities and correlations of the data in both spatial and temporal dimensions. MDZ contains vector-quantization-based compressor VQ and VQT, and multilevel-time-based compressor MT. MDZ is also equipped with our adaptive solution ADP which can select the best compressor (VQ, VQT, or MT) dynamically at runtime. MDZ improves the compression ratio by up to 233% compared with the second-best compressor on eight real-world MD datasets. We integrate MDZ to the MD software LAMMPS. MDZ shows negligible overhead in real-world MD simulations under different scales and settings.

Bibliography

[1] 7-Zip. `https://www.7-zip.org`, 2022. Online.

[2] Nitin Agrawal and Ashish Vulimiri. Low-latency analytics on colossal data streams with summarystore. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 647–664, 2017.

[3] Anastasia Ailamaki, Verena Kantere, and Debabrata Dash. Managing scientific data. *Commun. ACM*, 53(6):68–78, June 2010.

[4] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science*, 19(5):65–76, Dec 2018.

[5] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. Nodb: Efficient query execution on raw data files. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, page 241–252, 2012.

[6] Jyrki Alakuijala, Andrea Farruggia, Paolo Ferragina, Eugene Kliuchnikov, Robert Obryk, Zoltan Szabadka, and Lode Vandevenne. Brotli: A general-purpose data compressor. *ACM Trans. Inf. Syst.*, 37(1), December 2018.

[7] Francesc Alted. Blosc, an extremely fast, multi-threaded, meta-compressor library. `https://www.blosc.org/`, 2022.

[8] Anton supercomputer. `https://www.psc.edu/resources/anton`, 2022. Online.

[9] Allison Baker, Dorit Hammerling, Sheri Mickelson, Haiying Xu, Martin Stolpe, Phillipe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, Francesco Carbone, Christian Gencarelli, John Dennis, Jennifer Kay, and Peter Lindstrom. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development*, 9(12):4381–4403, 2016.

[10] Allison Baker, Haiying Xu, Dorit Hammerling, Shaomeng Li, and John Clyne. Toward a multi-method approach: Lossy data compression for climate simulation data. In *High Performance Computing*, pages 30–42, 2017.

[11] Rafael Ballester-Ripoll, Peter Lindstrom, and Renato Pajarola. TTHRESH: Tensor compression for multidimensional visual data. *IEEE Transactions on Visualization & Computer Graphics*, 26(09):2891–2903, sep 2020.

[12] Bebop supercomputer. `https://www.lcrc.anl.gov/systems/resources/bebop`, 2019. Online.

[13] Oliver Beckstein. Molecular dynamics trajectory of I-FABP for testing and benchmarking solvent dynamics analysis. `https://figshare.com/articles/dataset/Molecular_dynamics_trajectory_of_I-FABP_for_testing_and_benchmarking_solvent_dynamics_analysis/7058030`, 9 2018.

[14] Bernard R. Brooks, Charles L. Brooks, Alexander D. MacKerell, Lennart Nilsson, Robert J. Petrella, Benoit Roux, Youngdo Won, Georgios Archontis, Christian Bartels, Stefan Boresch, Amedeo Caflisch, Leo Simon Dominic Caves, Qiang Cui, Aaron R. Dinner, Michael Feig, Stefan Fischer, Jiali Gao, Milan Hodoscek, Wonpil Im, Krzysztof Kuczera, Themis Lazaridis, Jianpeng Ma, Victor Ovchinnikov, Emanuele Paci, Richard W. Pastor, Carol Beth Post, Jingzhi Pu, Michael Schaefer, Bruce Tidor, Richard M. Venable, Henry L. Woodcock, Xiongwu Wu, Wei Yang, Darrin M. York, and Martin Karplus. Charmm: The biomolecular simulation program. *Journal of Computational Chemistry*, 30, 2009.

[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, Jan 2009.

[16] Hu Cao, Ouri Wolfson, and Goce Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15:211–228, 01 2006.

[17] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications*, 33(6):1201–1220, 2019.

[18] Shubham Chandak, Kedar Tatwawadi, Chengtao Wen, Lingyun Wang, Juan Aparicio, and Tsachy Weissman. LFZip: Lossy compression of multivariate floating-point time series data via improved prediction. In *2020 Data Compression Conference (DCC)*, pages 342–351, 2020.

[19] 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*, pages 733–744, 2014.

[20] Yu Cheng and Florin Rusu. Parallel in-situ data processing with speculative loading. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, page 1287–1298, 2014.

[21] John Clyne, Pablo Mininni, Alan Norton, and Mark Rast. Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. *New Journal of Physics*, 9(301):1–29, 2007.

[22] Shirley Cohen, Patrick Hurley, Karl W. Schulz, William L. Barth, and Brad Benton. Scientific formats for object-relational database systems: A study of suitability and performance. *SIGMOD Rec.*, 35(2):10–15, June 2006.

[23] Philippe Cudre-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Roman Simakov, Emad Soroush, Pavel Velikhov, Daniel Wang, Magdalena Balazinska, Jacek Becla, David DeWitt, Bobbi Heath, David Maier, Samuel Madden, Jignesh Patel, Michael Stonebraker, and Stan Zdonik. A demonstration of SciDB: A science-oriented DBMS. *Proc. VLDB Endow.*, 2(2):1534–1537, August 2009.

[24] L Peter Deutsch. GZIP file format specification version 4.3. `https://datatracker.ietf.org/doc/html/rfc1952`, 1996.

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

[26] Jan Dvořák, Martin Maňák, and Libor Váša. Predictive compression of molecular dynamics trajectories. *Journal of Molecular Graphics and Modelling*, 96:107531, 2020.

[27] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. Online piece-wise linear approximation of numerical streams with precision guarantees. *Proc. VLDB Endow.*, 2(1):145–156, August 2009.

[28] EXAALT. `https://www.exascaleproject.org/research-project/exaalt/`, 2022. Online.

[29] Ziquan Fang, Yuntao Du, Lu Chen, Yujia Hu, Yunjun Gao, and Gang Chen. E2dtc: An end to end deep trajectory clustering framework via self-training. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 696–707, 2021.

[30] Ian Foster, Mark Ainsworth, Bryce Allen, Bessac Julie, Franck Cappello, Jong Youl Choi, Emil Constantinescu, Philip Davis, Sheng Di, Wendy Di, Hanqi Guo, Scott Klasky, Kerstin Dam, Tahsin Kurc, Qing Liu, Abid Malik, Kshitij Mehta, Klaus Mueller, Todd Munson, and Shinjae Yoo. Computing just what you need: online data analysis and reduction at extreme scales. In *European Conference on Parallel Processing*, pages 3–19, 2017.

[31] Globus. `https://www.globus.org`, 2022. Online.

[32] Ali Murat Gok, Sheng Di, Yuri Alexeev, Dingwen Tao, Vladimir Mironov, Xin Liang, and Franck Cappello. PaSTRI: A novel data compression algorithm for two-electron integrals in quantum chemistry. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11, 2018.

[33] Leonardo A. Bautista Gomez and Franck Cappello. Improving floating point compression through binary masks. In *2013 IEEE International Conference on Big Data*, pages 326–331, Oct 2013.

[34] Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. Scientific data management in the coming decade. *SIGMOD Rec.*, 34(4):34–41, December 2005.

[35] Grizzly supercomputer. `https://www.lanl.gov/org/ddste/aldsc/hpc/index.php`, 2021. Online.

[36] Allan Grønlund, Kasper Green Larsen, Alexander Mathiasen, and Jesper Sindahl Nielsen. Fast exact k-means, k-medians and bregman divergence clustering in 1d. `https://arxiv.org/abs/1701.07204`, 2017. Online.

[37] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, and Katrin Heitmann. HACC: extreme scaling and performance across diverse architectures. *Communications of the ACM*, 60(1):97–104, 2016.

[38] HDF5. `http://www.hdfgroup.org/HDF5`, 2022. Online.

[39] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[40] Hurricane ISABEL simulation data. `https://www.earthsystemgrid.org/dataset/isabeldata.html`, 2022. Online.

[41] Jan Huwald, Stephan Richter, Bashar Ibrahim, and Peter Dittrich. Compressing molecular dynamics trajectories: Breaking the one-bit-per-sample barrier. *Journal of Computational Chemistry*, 37(20):1897–1906, 2016.

[42] InfluxDB. `https://github.com/influxdata/influxdb`, 2022. Online.

[43] Soren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Modelardb: Modular model-based time series management with spark and cassandra. *Proc. VLDB Endow.*, 11(11):1688–1701, July 2018.

[44] Soren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. Scalable model-based management of correlated dimensional time series in modelardb+. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1380–1391, 2021.

[45] Sian Jin, Sheng Di, Xin Liang, Jiannan Tian, Dingwen Tao, and Franck Cappello. DeepSZ: A novel framework to compress deep neural networks by using error-bounded lossy compression. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, pages 159–170, 2019.

[46] Sian Jin, Pascal Grosset, Christopher M. Biwer, Jesus Pulido, Jiannan Tian, Dingwen Tao, and James Ahrens. Understanding gpu-based lossy compression for extreme-scale cosmological simulations. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 105–115, 2020.

[47] Jennifer Kay, C. Deser, A Phillips, A Mai, Cecile Hannay, G. Strand, J. Arblaster, Susan Bates, G. Danabasoglu, James Edwards, M. Holland, Paul Kushner, Jean-François Lamarque, D. Lawrence, Keith Lindsay, A Middleton, Ernesto Munoz, R. Neale, Keith Oleson, and Mariana Vertenstein. The community earth system model (CESM), large ensemble project: A community resource for studying climate change in the presence of internal climate variability. *Bulletin of the American Meteorological Society*, 96(8):1333–1349, 2015.

[48] Suha Kayum, Thierry Tonellot, Vincent Etienne, Ali Momin, Ghada Sindi, Maxim Dmitriev, and Hussain Salim. GeoDRIVE - a high performance computing flexible platform for seismic applications. *First Break*, 38(2):97–100, 2020.

[49] Martin L. Kersten, Stratos Idreos, Stefan Manegold, and Erietta Liarou. The researcher's guide to the data deluge: Querying a scientific database in just a few seconds. *Proc. VLDB Endow.*, 4(12):1474–1477, August 2011.

[50] Jeongnim Kim, Andrew Baczewski, Todd Beaudet, Anouar Benali, Michael Bennett, Mark Berrill, Nick Blunt, Michele Casula, David Ceperley, Simone Chiesa, Bryan Clark, III Clay, Kris Delaney, Mark Dewing, Kenneth Esler, Hongxia Hao, Olle Heinonen, Paul Kent, Jaron Krogel, and Luning Zhao. 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.

[51] Navjot Kukreja, Jan H uuckelheim, Mathias Louboutin, John Washbourne, Paul H.J. Kelly, and Gerard J. Gorman. Lossy checkpoint compression in full waveform inversion: a case study with zfpv0.5.5 and the overthrust model. *Geoscientific Model Development*, 15(9):3815–3829, 2022.

[52] Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, Choong-Seock Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013.

[53] Iosif Lazaridis and Sharad Mehrotra. Capturing sensor-generated time series with quality guarantees. In *Proceedings 19th International Conference on Data Engineering (ICDE)*, pages 429–440, 2003.

[54] Sihuan Li, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Optimizing lossy compression with adjacent snapshots for n-body simulation data. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 428–437, 2018.

[55] Sihuan Li, Sheng Di, Kai Zhao, Xin Liang, Zizhong Chen, and Franck Cappello. Towards End-to-end SDC detection for HPC applications equipped with lossy compression. In *2020 IEEE International Conference on Cluster Computing*, pages 326–336, 2020.

[56] Sihuan Li, Sheng Di, Kai Zhao, Xin Liang, Zizhong Chen, and Franck Cappello. Resilient error-bounded lossy compressor for data transfer. In *Proceedings of the*

International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21, pages 1–14, 2021.

[57] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 617–628, 2018.

[58] Xin Liang, Sheng Di, Sihuan Li, Dingwen Tao, Bogdan Nicolae, and Franck Cappello. Significantly improving lossy compression quality based on an optimized hybrid prediction model. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–26, 2019.

[59] Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, and Franck Cappello. An efficient transformation scheme for lossy data compression with point-wise relative error bound. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 179–189, 2018.

[60] 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*, pages 438–447, 2018.

[61] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Bogdan Nicolae, Zizhong Chen, and Franck Cappello. Improving performance of data dumping with lossy compression for scientific simulation. In *2019 IEEE International Conference on Cluster Computing*, pages 1–11, 2019.

[62] Xin Liang, Ben Whitney, Jieyang Chen, Lipeng Wan, Qing Liu, Dingwen Tao, James Kress, David Pugmire, Matthew Wolf, Norbert Podhorszki, and Scott Klasky. Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction. *IEEE Transactions on Computers*, PP(01):1–1, jul 5555.

[63] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. SZ3: A modular framework for composing prediction-based error-bounded lossy compressors. `https://arxiv.org/abs/2111.02925`, 2021. Online.

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

[65] Peter Lindstrom. Error distributions of lossy floating-point compressors. *Joint Statistical Meetings*, 1(1):2574–2589, 2017.

[66] 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.

[67] Lennard-Jones liquid benchmark on LAMMPS. `https://lammps.sandia.gov/bench.html#lj`, 2022. Online.

[68] Jinyang Liu, Sheng Di, Kai Zhao, Sian Jin, Dingwen Tao, Xin Liang, Zizhong Chen, and Franck Cappello. Exploring autoencoder-based error-bounded compression for scientific data. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 294–306, 2021.

[69] Cheng Long, Raymond Wong, and H.V. Jagadish. Trajectory simplification: On minimizing the directionbased error. *Proceedings of the VLDB Endowment*, 8:49–60, 01 2014.

[70] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Mathew Wolf, Tong Liu, and Zhenbo Qiao. Understanding and modeling lossy compression schemes on HPC scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium*, pages 348–357, 2018.

[71] Magnus Lundborg, Rossen Apostolov, Daniel Spangberg, Anders Gardenas, David van der Spoel, and Erik Lindahl. An efficient and extensible format, library, and api for binary trajectory data from molecular simulations. *Journal of computational chemistry*, 35, 01 2014.

[72] Patrick Marais, Julian Kenwood, Keegan Carruthers Smith, Michelle M. Kuttel, and James Gain. Efficient compression of molecular dynamics trajectory files. *Journal of Computational Chemistry*, 33(27):2131–2141, 2012.

[73] Miranda. `https://wci.llnl.gov/simulation/computer-codes/miranda/papers`, 2019. Online.

[74] NYX. `https://amrex-astro.github.io/Nyx`, 2019. Online.

[75] Andrey Omeltchenko, Timothy J. Campbell, Rajiv K. Kalia, Xinlian Liu, Aiichiro Nakano, and Priya Vashishta. Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm. *Computer Physics Communications*, 131(1):78–85, 2000.

[76] Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proc. VLDB Endow.*, 8(12):1816–1827, August 2015.

[77] Danny Perez, Ekin D. Cubuk, Amos Waterland, Efthimios Kaxiras, and Arthur F. Voter. Long-time dynamics through parallel trajectory splicing. *Journal of Chemical Theory and Computation*, 12(1):18–28, 2016.

[78] Danny Perez, Luis Sandoval, Sophie Blondel, Brian Wirth, Blas Uberuaga, and Arthur Voter. The mobility of small vacancy/helium complexes in tungsten and its impact on retention in fusion-relevant conditions. *Scientific Reports*, 7, 05 2017.

[79] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995.

[80] Steven Plimpton, Danny Perez, and Arthur Voter. Parallel algorithms for hyperdynamics and local hyperdynamics. *The Journal of Chemical Physics*, 153(5):054116, 2020.

[81] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. Fast lossless compression of scientific floating-point data. In *Data Compression Conference (DCC'06)*, pages 133–142, 2006.

[82] Russell Rew and Glenn Davis. NetCDF: an interface for scientific data access. *IEEE Computer Graphics and Applications*, 10:76–82, 1990.

[83] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. Exploration of lossy compression for application-level checkpoint/restart. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium*, IPDPS '15, pages 914–922, 2015.

[84] SCALE-LETKF weather model. `https://github.com/gylien/scale-letkf`, 2019. Online.

[85] Sean Seyler and Oliver Beckstein. Molecular dynamics trajectory for benchmarking MDAnalysis. `https://figshare.com/articles/dataset/molecular_dynamics_trajectory_for_benchmarking_MDAnalysis/5108170`, 6 2017. Online.

[86] Ardita Shkurti, Ramon Goni, Pau Andrio, Elena Breitmoser, Iain Bethune, Modesto Orozco, and Charles A. Laughton. pypcazip: A pca-based toolkit for compression and analysis of molecular simulation data. *SoftwareX*, 5:44–50, 2016.

[87] Snappy. `https://google.github.io/snappy`, 2022. Online.

[88] Seung Woo Son, Zhengzhang Chen, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. Data compression for the exascale computing era-survey. *Supercomputing Frontiers and Innovations*, 1(2):76–88, 2014.

[89] Summit supercomputer. `https://www.olcf.ornl.gov/summit/`, 2019. Online.

[90] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. In-depth exploration of single-snapshot lossy compression techniques for N-body simulations. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 486–493, 2017.

[91] 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*, pages 1129–1139, 2017.

[92] Dingwen Tao, Sheng Di, Hanqi Guo, and Franck Cappello. Z-checker: A framework for assessing lossy compression of scientific data. *The International Journal of High Performance Computing Applications*, 33, 06 2017.

[93] 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*, 30(8):1857–1871, 2019.

[94] Nikola Tchipev, Steffen Seckler, Matthias Heinen, Jadran Vrabec, Fabio Gratl, Martin Horsch, Martin Bernreuther, Colin Glass, Christoph Niethammer, Nicolay Hammer, Bernd Krischok, Michael Resch, Dieter Kranzlmüller, Hans Hasse, Hans-Joachim Bungartz, and Philipp Neumann. Twetris: Twenty trillion-atom simulation. *The International Journal of High Performance Computing Applications*, 33(5):838–854, 2019.

[95] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, IOPADS '99, pages 23–32, 1999.

[96] The HDF Group. H5Z: Filter and Compression Interface. `https://support.hdfgroup.org/HDF5/doc1.8/RM/RM_H5Z.html`, 2017. Online.

[97] Theta supercomputer. `https://www.alcf.anl.gov/theta`, 2019. Online.

[98] Jiannan Tian, Sheng Di, Xiaodong Yu, Cody Rivera, Kai Zhao, Sian Jin, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. Optimizing error-bounded lossy compression for scientific data on GPUs. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 283–293, 09 2021.

[99] Jiannan Tian, Sheng Di, Chengming Zhang, Xin Liang, Sian Jin, Dazhao Cheng, Dingwen Tao, and Franck Cappello. waveSZ: A hardware-algorithm co-design of efficient lossy compression for scientific data. In *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 74–88, 2020.

[100] Jiannan Tian, Sheng Di, Kai Zhao, Cody Rivera, Megan Fulp, Robert Underwood, Sian Jin, Xin Liang, Jon Calhoun, Dingwen Tao, and Franck Cappello. CuSZ: An efficient gpu-based error-bounded lossy compression framework for scientific data. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, page 3–15, 2020.

[101] Jiannan Tian, Cody Rivera, Sheng Di, Jieyang Chen, Xin Liang, Dingwen Tao, and Franck Cappello. Revisiting huffman coding: Toward extreme performance on modern gpu architectures. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 881–891, 2021.

[102] Trinity supercomputer. `https://www.lanl.gov/projects/trinity/`, 2021. Online.

[103] Andy Turner. Parallel I/O performance. `https://www.archer.ac.uk/training/virtual/2017-02-08-Parallel-IO/2017_02_ParallelIO_ARCHERWebinar.pdf`, 2019. Online.

[104] Robert Underwood, Sheng Di, Jon 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, 05 2020.

[105] David Van Der Spoel, Erik Lindahl, Berk Hess, Gerrit Groenhof, Alan E Mark, and Herman JC Berendsen. Gromacs: fast, flexible, and free. *Journal of computational chemistry*, 26(16):1701–1718, 2005.

[106] Arthur F. Voter. Parallel replica method for dynamics of infrequent events. *Physical Review B*, 57, 1998.

[107] Gregory K Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

[108] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004.

[109] Brent Welch. POSIX io extensions for HPC. In *4th USENIX Conference on File and Storage Technologies (FAST05)*, page 1, 2005.

[110] Dow Yung Yang, Ananth Grama, and Vivek Sarin. Bounded-error compression of particle data from hierarchical approximate methods. In *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, SC '99, page 32–es, 1999.

[111] Xinyang Yu, Yanqing Peng, Feifei Li, Sheng Wang, Xiaowei Shen, Huijun Mai, and Yue Xie. Two-level data compression using machine learning in time series database. In *36th IEEE International Conference on Data Engineering*, pages 1333–1344, 2020.

[112] Kai Zhao, Sheng Di, Maxim Dmitriev, Thierry-Laurent D. Tonellot, Zizhong Chen, and Franck Cappello. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1643–1654, 2021.

[113] Kai Zhao, Sheng Di, Xin Lian, Sihuan Li, Dingwen Tao, Bessac Julie, and Franck Cappello. SDRBench: Scientific data reduction benchmark for lossy compressors. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2716–2724, 2021. Online.

[114] 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, 2020.

[115] Kai Zhao, Sheng Di, Danny Perez, Xin Liang, Zizhong Chen, and Franck Cappello. Mdz: An efficient error-bounded lossy compressor for molecular dynamics. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 27–40, 2022.

[116] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[117] Zlib. `http://www.zlib.net/`. Online.

[118] Zstandard. `https://github.com/facebook/zstd`, 2021. Online.