THE UNIVERSITY OF CHICAGO

HYBRID LOSSY COMPRESSION METHODS CAN CONFIDENTLY OPTIMIZE WIDE NETWORK TRANSFER OF COMPLEX DATASETS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCE
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
YUANJIAN LIU

CHICAGO, ILLINOIS
JUNE 2025

ABSTRACT

Large volumes of data generated by scientific simulations, genome sequencing, and other applications need to be moved among clusters for data collection/analysis. Data compression techniques have effectively reduced data storage and transfer costs. However, users' requirements on interactively controlling both data quality and compression ratios are non-trivial to fulfill. Lossy compression methods need to respect several data constraints to be useful in a realistic data transfer scenario. In this thesis, I propose a novel Compression-as-a-Service (CaaS) platform called GlobaZip with five important contributions: (1) a multi-interval/multi-region based compression algorithm that supports several data constraints to further limit the distortion in data fidelity even though the compression is lossy; (2) a layer-by-layer compression technique that allows much higher parallel compression rate in HPC systems and can coordidate CPU cores on multiple compute nodes to compress extremely large files without out-of-memory errors; (3) a decision tree-based compression performance prediction model that allows users to use very limited computation overhead to estimate compression characteristics including compression ratio, time and data fidelity; (4) an optimized reference-based genome sequence compression algorithm that exeeds the performance of state-of-the-art algorithms by using more fine-grained sequence alignment procedure, reordering reads, a novel dominant bitmap method for quality score compression, and a few other small optimizations; (5) a Qt5-based user-facing app that utilizes Globus Compute and Globus Transfer to provide users with a universal interface to orchestrate remote data compression and transfer. Experiments on multiple real-world datasets on geographically distributed computers show that GlobaZip can significantly improve data transfer efficiency with a performance gain of more than 10x in computing clusters with relatively slow networks.

To my partner Yongli and my beloved parents.

"I am enough of the artist to draw freely upon my imagination. Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world." — Albert Einstein

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I foremost thank my advisor Dr. Ian Foster and Dr. Kyle Chard for guiding me through the process of coming all the way from China to the United States to pursue computer science research and writing elements of this Master's thesis. My grad school is packed with support and understanding and I owe a big thank you to you both.

I thank Dr. Sheng Di for providing me lots of knowledge, ideas, and time to discuss every detail of the research topic. You helped me a lot in forming the thesis' main idea and writing skills that will certainly manifest in my future efforts.

I acknowledge the senior Ph.D. student Kai Zhao for providing easy-to-read and well-written C++ coding and explanation, which helps me a lot when developing algorithms used in this thesis.

I am also grateful for the current and former students Jiajun Huang, Sian Jin, Xin Liang, Matt Baughman, Greg Pauloski, Maksim Levental, Tyler Skluzacek for their kind support when I was confused about things from academics to life.

I would like to express my heartfelt gratitude to my girlfriend, Yongli Zhu, for her unwavering love and support. Despite the vast distance separating us across the Pacific Ocean, she has remained a constant source of encouragement and connection, keeping us closely bonded throughout this journey.

Finally, I want to thank my parents and other family members for their strong support of my decision to go abroad and pursue academic achievement.

CHAPTER 1

INTRODUCTION

Large amounts of data are being produced by high performance computing (HPC) simulations and advanced instruments such as the Advanced Photon Source (APS) [26] and LCLS-II [58]. Researchers use many types of compression algorithms to reduce sizes for these data. Lossless compression algorithms can keep 100% data fidelity but the storage reduction is very marginal (1.5x-2x). However, lossy compression can significantly reduce the data sizes (10x-1000x+) with configurable error thresholds. Theoretically, there is no limit to the compression ratio in lossy compression algorithms if users allow a very high error bound, but the data will become unusable after a certain point. Due to uncertainty about the 'certain point', users may not be confident about using lossy compression, fearing it may lead to inaccuracies in downstream analysis. How to obtain both high compression ratios provided by lossy compression algorithms and certainty about data usability remains an open question.

The application of lossy compression for data transfer optimization has seen significant success in the video and photography industries with compression algorithms including JPEG [92], H.264 [15], and H.265 [84]. For example, an 80 GB video file can be compressed to around 800 MB using H.264, reaching a compression ratio of almost 2000. Video distribution platforms like YouTube can deliver similar quality content to users with 1/2000 of the original data. This data reduction technology saves billions of dollars. If similar approaches can be applied to scientific data, we can expect similarly large reductions in bandwidth and storage.

In addition, these data are typically needed to be shared for analysis, storage, publication, and archival purposes, often between multiple research institutions. However, data transfer over a wide area network (WAN) can be time consuming and significantly delay research progress. Tools such as Globus [13, 67], widely adopted in computing facilities (e.g.,

supercomputers), can accelerate transfer, but with such enormous datasets, transfer times can still be very long.

A large number of scientific datasets are floating point number-based[46][76][41][72], and often allow a certain level of imprecision in the data for the application to yield the same or similar results. Thus, it is possible to reduce the data size by reducing the precision in a controlled manner. Error-bounded lossy compression exploits this fact and offers the potential to significantly reduce data sizes. However, the optimal tuning of the compression process (i.e., for performance and quality) remains an open problem, and thus such methods are rarely used in data transfer situations. Although error-bounded lossy compression can substantially reduce data volume with user-tolerable data distortion, existing studies focus on conventional use cases such as reducing storage space [106], lowering I/O cost [55], or reducing memory capacity requirements [94, 43]. Li et al. [51] studied how to make the error-bounded lossy compressor SZ resilient to soft errors during data transfers and evaluated their approach by using a numerical analysis / simulator, but did not systematically model and optimize data transfer performance with respect to lossy compression techniques.

Many other lossless and lossy compressors have been developed to address the big data issue. However, they cannot be used directly or effectively for remote parallel data transfer tasks due to the following substantial drawbacks.

Limited/missing support for multi-node parallel compression. Off-the-shelf error-bounded lossy compressors lack efficient support for using processors on different compute nodes, which may significantly limit their parallel compression performance. On the one hand, the vast volume of data produced by many applications necessitates parallel compression across multiple nodes to complete the process within a reasonable time frame. On the other hand, the memory capacity of a single node is typically inadequate for most existing compression algorithms to handle some extremely large data files. To enable large file compression on multiple nodes, the algorithm should work with an efficient method that can

split the files appropriately and coordinate processors on different compute nodes.

Unable to meet users' diverse requirements for compression quality. Although error-bounded lossy compressors have been effective in many applications, existing compressors are developed/driven based on relatively simple error control methods such as absolute error bound, inevitably leaving a significant gap in user requirements on the reconstructed data quality. Users often need to visualize the reconstructed data to determine whether they are valid in practice. Error-bounded lossy compressors, however, may introduce undesired artifacts in the reconstructed data, which is a non-trivial issue as the artifacts are related to many factors such as datasets, compressor design, error-bound types, and values. As such, enabling users to interactively check the quality of compression in real time and adjust compression parameters timely is a substantial feature to guarantee user requirements on compression quality.

Limited/missing support for diverse types of datasets/files. In the data repositories, there are diverse types of scientific datasets. While the existing error-bounded compressors are suitable for many simulation datasets, they are not suitable for other types of data. For example, genome sequence data consist of text-based sequence identifiers (e.g. ATCGGC...), which cannot be well compressed by conventional error-bounded lossy compression or binary-level lossless compression. Based on biological DNA similarity, dedicated reference-based compression algorithms can be much more efficient in this situation. Moreover, we note some inefficiency in the sequence alignment process and the compression of the quality score of existing genome sequence compressors such as Genozip [48]. On the other hand, existing error-bounded compressors such as SZ [53, 57] and ZFP [59] are not qualified for parallel (de)compression of very large files each of which is composed of one single tensor (or variable).

The primary goal of this work is to research and create a workflow for scientists to transfer data between computing clusters faster while maintaining data usability when applying lossy

compression in the process. The thesis is organized into the following five research questions:

1. How to ensure data constraints are met when compressing data?
2. How to parallelize compression/decompression when more computation resources are available?
3. How to preview the decompressed data quality without doing full-length compression/decompression?
4. How to compress genome sequence data that does not follow the traditional characteristics of error bounded compression?
5. How to orchestrate the compression/decompression and transfer on multiple computing clusters?

1.1 Thesis statement

Hybrid lossy compression methods can improve overall data transfer efficiency while maintaining downstream application performance. In this thesis, my goal is to provide an optimized scientific data transfer solution that considers performance and accuracy tradeoffs of dynamic lossy compression.

I first explore ways to ensure certain data constraints are met for the reconstructed data by designing lossy compression algorithms that preserve multiple error bounds on various value ranges and regions. To help users understand characteristics of the decompressed data, I explore methods to predict the compression performance (compression ratio, time, peak signal-to-noise ratio, etc.) with minimal overhead costs. To aid the selection of the most suitable compressor and make users more confident about the compressed data quality for a given dataset, I explore an interactive compression paradigm, where users can preview a certain part of the raw data as well as the decompressed data generated by multiple

compressors with a visualization method. For extremely large datasets, I explore ways to utilize multiple compute nodes to collaboratively compress a single large file without exceeding resource constraints. I also explore compression methods for some special types of data that cannot be compressed well by error-bounded compression algorithms, including genome sequence data.

Finally, I combine all the proposed algorithms and mechanisms into an intelligent and scalable Compression-as-a-Service (CaaS) framework, GlobaZip, to be shared with the scientific community. This thesis expands upon substantial prior work [64, 65, 66, 63].

1.2 Contributions

The primary contribution of this thesis is the development and evaluation of a CaaS framework, GlobaZip, that builds on Globus data transfer and Function-as-a-Service (FaaS) capabilities. GlobaZip provides users with a universal graphical user interface for interactive control of compression/transfer of diverse datasets with multiple compression methods among computing clusters. In building, optimizing, and evaluating different compression algorithms over the past five years, I discovered and addressed research gaps in the transfer of large amounts of scientific data using lossy compression data. This thesis offers key insights into the design and evaluation of compression/transfer systems, both with regard to the overall usability and system performance. The findings are presented in the context of GlobaZip, comparing each individual component with the state-of-the-art methods and introducing my novel design and software stacks. In more detail, my contributions for each of the five research questions are as follows.

1. A constraint-based error-bounded lossy compression model The requirement of data fidelity for an application often involves more characteristics than a simple error rate for each data point. Users may have more sophisticated and complicated requirements for certain datasets. Therefore, we first define and apply several data constraints that the com-

pression model needs to respect and propose a lossy compression model. This compression model is the first of its kind, to the best of my knowledge. The data constraints include (A) isolating irrelevant values; (B) preserving global value range; (C) preserving multi-interval-based error bounds; (D) preserving multiregion-based error bounds; and (E) using a bitmap to mask complicated regions and apply different error bounds on each region. I also develop an efficient mechanism allowing users to interactively set multiple error bounds at different value ranges or regions for floating-point tensors, which is critical to meeting diverse user requirements.

2. An efficient multinode compression method with a layer-by-layer technique To compress extremely large tensors that cannot fit in the memory, I design a layer-based compression technique that allows thread-type and MPI-type parallelization. The thread-type parallelization is simple to run on any cloud servers which can utilize the available CPU cores on a single machine, while the MPI-type parallelization allows users to involve multiple compute nodes in a high-performance computing (HPC) system to further reduce the compression time of extremely large files.

3. A lightweight decision tree-based compression performance prediction model I use data-based, config-based, and compressor-based features to train a decision tree model to predict compression ratio, compression time, and peak-to-noise ratio. I also provide an interface for users to preview the data quality of selected layers and allow users to compress only certain layers to get an overview of the decompressed data quality without doing full-length compression/decompression.

4. An optimized reference-based genome sequence compression algorithm As there are various types of data stored and processed on computing clusters, not all data can be compressed using error-bounded compression methods. Genome sequence data are one of this kind. I propose a novel reference-based genome sequence compression algorithm with an improved sequence alignment technology and a bitmap-based quality score compression

method, the performance of which exceeds the state-of-the-art algorithm Genozip.

5. A Qt5-based GlobaZip app with remote orchestration capabilities I develop the app with Globus Transfer and Globus Compute support and allow users to run compression/transfer/decompression across multiple remote computing clusters. I conduct experiments and benchmarks on multiple real-world datasets and demonstrate GlobaZip's powerful capability to efficiently orchestrate data compression/transfer across heterogeneous supercomputers/clusters over WAN. Experiments show that performance improvement exceeds 10x when transferring data from supercomputers to typical cloud computing clusters.

1.3 Thesis Organization

Chapter 2 presents the research background and fundamental related work in data compression, transfer, remote orchestration, and addressing users' demands. Chapter 3 focuses on describing the methods I propose to solve the aforementioned research questions. There are 6 sections, each solving a dedicated problem: section 3.1 defines the data contraints and methods to preserve them; section 3.2 explores ways to preview data quality and feature-based solution to predict compression performance; section 3.3 elaborates the design of my novel reference-based genome sequence compression algorithm; section 3.4 introduces my layer-by-layer compression technique and the corresponding implementations; section 3.5 and section 3.6 report on my GlobaZip framework design and methods for optimizing data transfer via error-bounded lossy compression. Chapter 4 presents the evaluation results for each of the proposed methods. Chapter 5 discusses potential future work. In Chapter 6, I summarize and conclude the thesis with some remarks.

CHAPTER 2

BACKGROUND AND RELATED WORK

The proposed lossy compression methods mainly deal with floating-point number tensors and genome sequence data. Floating point tensors are the major outputs of scientific applications and also represent the most substantial storage demand within deep learning models. Error-bounded lossy compression can significantly reduce the sizes of floating-point tensor data. However, there is an increasing demand for storing and transferring human genome sequences within computational facilities, with the prospect of affordable DNA sequencing technologies. To achieve high compression ratios, it is not likely to propose a general lossy compression method to fit all types of data, as do lossless compression algorithms. Dedicated compression algorithms are needed for different types of data. Therefore, to optimize data transfer performance, we need a mechanism to orchestrate multiple types of compressors in the pipeline. This section explores the latest advancements in compression technologies for the mentioned data types, alongside recent initiatives to streamline the orchestration of remote tasks.

2.1 Error-bounded Lossy Compression

Data compression is widely used in scientific research, for example, to reduce the size and cost of data storage and transfer. Data compressors are typically divided into two classes: lossless compression [110, 33, 11, 8] and lossy compression [53, 57, 59, 60, 100, 32, 55, 99, 62, 88, 52, 91]. The former does not introduce data loss during compression, but suffers from very low compression ratios (generally 1.1-2 [83, 78]). The latter can achieve very high compression ratios (such as 100+) [53, 57, 59, 54], but potential data loss can distort the results of the analysis.

To address data loss concerns, researchers have studied error-bounded lossy compressors

[22] for scientific data, which can be split into two main categories: the prediction-based compression model and the transform-based compression model. SZ [53, 57, 54] is a typical prediction-based lossy compression model, which is composed of four key stages: data prediction, linear-scale quantization, Huffman encoding, and lossless compression. ZFP [59] is a typical compressor designed based on the transform-based model, which includes four key steps: splitting the dataset into fixed-size blocks, exponent alignment in each block, orthogonal data transform for each block; and embedded encoding for each block. These techniques play a crucial role in minimizing storage requirements, as evidenced by their applications in various domains such as molecular dynamics simulations [86], quantum computing [94, 93], and supercomputing environments [55]. By efficiently compressing data while ensuring that the error introduced during compression remains bounded, these methods not only can mitigate memory demands but also alleviate I/O expenses in high-performance computing settings. Furthermore, they can eliminate the need for costly data recomputation [32].

Existing error-bounded lossy compressors offer different types of error bounds to address diverse user demands. The most common error-bounding approach involves using an absolute error bound, which ensures that the pointwise difference between the original raw data and reconstructed data is confined within a constant threshold. Many compressors such as SZ [53, 57, 54], ZFP [59, 24], and MGARD [4] support absolute error bounds. Other error-bounding approaches have been explored to adapt to diverse user requirements. For example, SZ supports pointwise relative error bounds [101, 23]; and Digit Rounding [20], Bit Grooming [100], zfp [59], and FPZIP [60] support a precision mode that allows users to specify the number of bits to be truncated at the end of the mantissa, to control data distortion at different levels.

To satisfy user demands on a specific quality of interest, researchers have recently studied how to respect some specific metrics. For example, Tao et al. [85] developed a formula that can link the target peak signal-to-noise ratio (PSNR) metric to an absolute error bound

setting in SZ such that the data can be compressed based on a user-specified PSNR metric. MGARD [4] supports various norm error metrics and linear quantities of interest in its multigrid compression method. However, none of the existing error-bounded lossy compressors allow users to set particular constraints in the error-bounded compression, and hence impose a significant impediment on the practical use of such compressors. In fact, users often have diverse precision demands for various data value intervals or specific requirements on different spatial regions, which are determined by their sophisticated post hoc analysis purposes and quantities or features of interest.

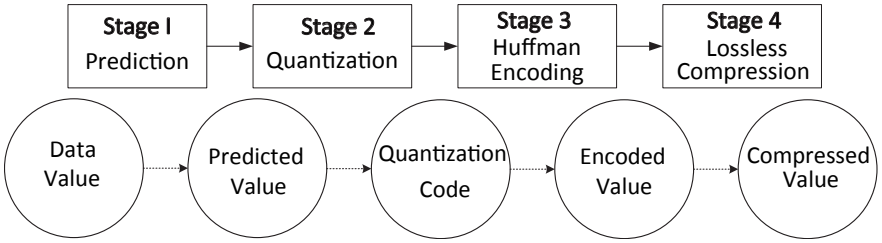

Figure 2.1: General procedure of constraint preserving error bounded lossy compression: Constraint (A) is handled before the prediction step; constraint (B) is handled primarily in both the prediction and quantization stage by replacing data points with Lorenzo-predicted values; constraints (C), (D), and (E) are addressed by designing a new quantization method.

2.1.1 SZ Compression Model

Our floating-point tensor compression methods use the prediction-based error-bounded compression framework similar to SZ, SZ3. Therefore, we give a more detailed introduction of this framework, which is illustrated in Figure 2.1. As shown in the figure, the compression model is made up of four key stages: prediction, quantization, Huffman encoding, and lossless compression. Given a set of raw data, SZ scans the whole dataset (pointwise [21, 87] or blockwise [54]) to predict the data values. In a 1D dataset, the prediction method is simply a first-order Lorenzo predictor [87], which uses only the preceding value to approximate the current data point. In a 2D or 3D dataset, SZ adopts a hybrid data prediction method combining the first-order Lorenzo predictor (using three nearby values in the 2D Lorenzo and 7

nearby values in the 3D Lorenzo) and a linear regression-based predictor [54]. Such a hybrid predictor can significantly improve the accuracy of data prediction, which in turn can substantially increase the compression ratio, especially when the error bound is relatively large. SZ3 proposes a newer interpolation-based predictor[104] that reaches a higher compression ratio for most datasets. The second stage uses a linear quantization method to convert the distance between the predicted value and the original value to an integer number (called the quantization code or quantization number) for each data point. A custom Huffman encoder is then applied to compress the integer quantization codes, followed by a lossless dictionary coding (using Zstd [110] by default in SZ3).

2.1.2 Prediction Algorithms

Lorenzo Prediction

The easiest prediction method is called Lorenzo prediction. According to Tao et al.[87], the value of (i_0, j_0) on the prediction surface can be expressed by the linear combination of the data values in the neighborhood, as shown in Figure 2.2. Because Tao et al.[87]'s evaluation results show that the first- and second-layer prediction yield the best results in most datasets, we only give the formula for the first two layers here.

Let $V(i, j)$ be a function that returns the recontructed data of point (i, j). Let $f(i, j)$ be the prediction function for point (i, j). For the 1-layer lorenzo prediction, the formulae is

$$f_1(i,j) = V(i,j-1) + V(i-1,j) - V(i-1,j-1) \tag{2.1}$$

And for the 2-layer lorenzo prediction, the prediction formulae is

$$\begin{aligned} f_2(i,j) =& 2V(i-1,j) + 2V(i,j-1) - 4V(i-1,j-1) - V(i-2,j) - V(i,j-2) + \\ & 2V(i-2,j-1) + 2V(i-1,j-2) - V(i-2,j-2) \end{aligned} \tag{2.2}$$

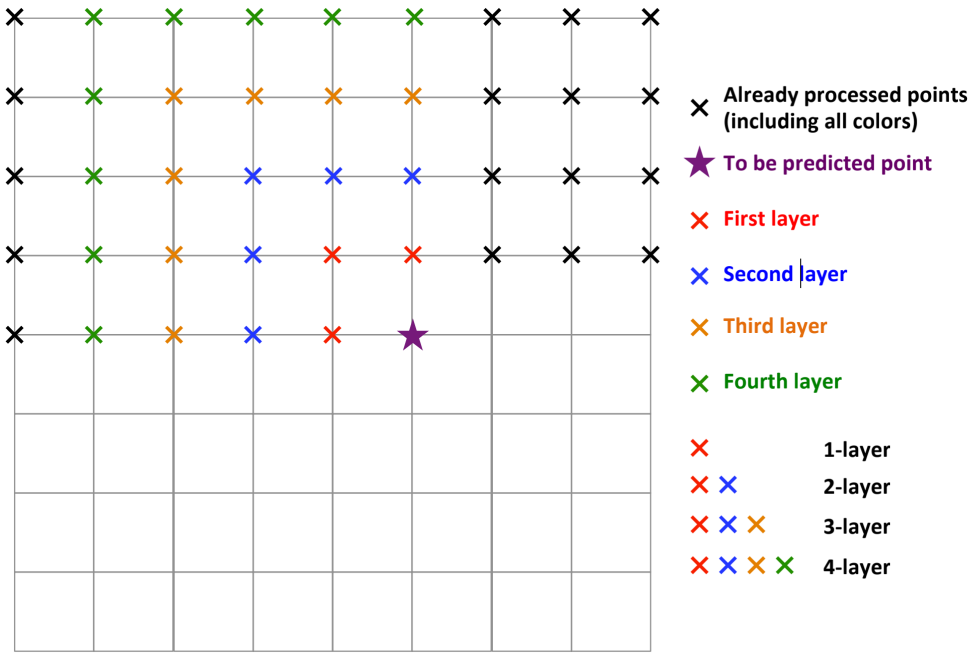

Figure 2.2: (Figure extracted from [87]) Example of Lorenzo Prediction in two-dimentional data: the point to be predicted can be predicted by its surrounding data points.

Note that V does not return the original data but the reconstructed data, meaning the prediction of previous data points will affect the prediction result for the succeeding points.

Linear Regression

Zhao et al.[104] proposed a linear regression method to predict data points, as illustrated in Figure 2.3. During compression, each data block is approximated by a linear hyperplane with four coefficients ($f(x,y,z)=\beta_1 x + \beta_2 y + \beta_3 z + \beta_0$), and thus only four coefficient values need be stored to substitute all the data in that block. During the decompression, each data block would be recovered by the hyperplane reconstructed with the four regression coefficients. Without loss of generality, suppose the data block size is $n_3 \times n_2 \times n_1$, then its regression coefficients can be calculated as the following formula (d_{ijk} refers to the data values in the data block at the location $\{i,j,k\}$).

$$\begin{cases} \beta_1 = \frac{6}{n_1 n_2 n_3 (n_1+1)} (\frac{2V_x}{n_1-1} - V_0) \\ \beta_2 = \frac{6}{n_1 n_2 n_3 (n_2+1)} (\frac{2V_y}{n_2-1} - V_0) \\ \beta_3 = \frac{6}{n_1 n_2 n_3 (n_3+1)} (\frac{2V_z}{n_3-1} - V_0) \\ \beta_0 = \frac{V_0}{n_1 n_2 n_3} - (\frac{n_1-1}{2}\beta_1 + \frac{n_2-1}{2}\beta_2 + \frac{n_3-1}{2}\beta_3) \end{cases} \tag{2.3}$$

$$\text{where } V_0 = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} d_{ijk}, \quad V_x = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} i * d_{ijk},$$
$$V_y = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} j * d_{ijk}, \quad V_z = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} k * d_{ijk}.$$

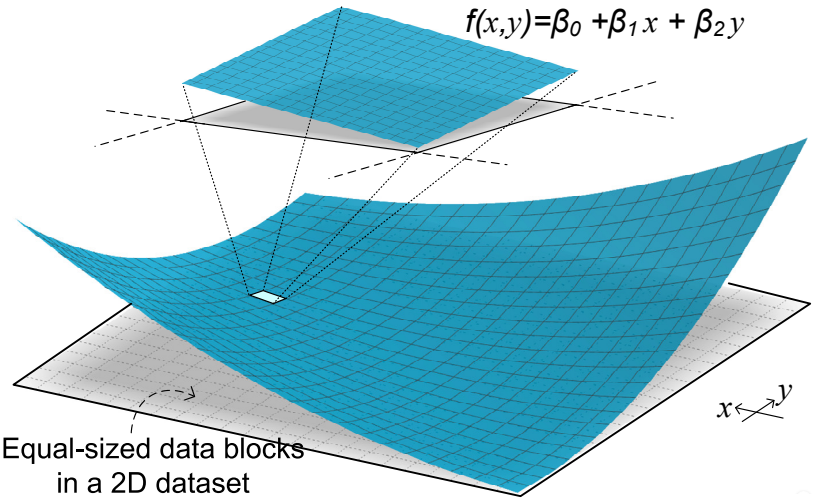

Figure 2.3: (Figure extracted from [104]) Illustration of linear regression model

In this compression method, the compression ratio can be estimated based on the following formula. For instance, when the block size is 4×4×4, the compression ratio is 64/4=16; when the block size is 3×3×3, the compression ratio is 27/4=6.75.

$$CR = \frac{n_3 n_2 n_1}{4} \tag{2.4}$$

Interpolation

Zhao et al.[104] also proposed a lightweight cubic interpolation-based prediction method for each unknown data point by only using its four surrounding data values. This method is claimed to outperform previous methods in terms of compression ratio (the prediction is more accurate) but have higher computational cost. The authors reduced the computational costs by precomputing a closed-form interpolation formula based on four specific neighbor data points (using the data points $i-3$, $i-1$, $i+1$ and $i+3$ to predict data point i as shown in Figure 2.4). There are two spline methods to do the prediction: (1) Linear spline; (2) Cubic spline. The closed form formula are equation 2.5 and 2.6 respectively.

$$p_i = \frac{1}{2}d_{i-1} + \frac{1}{2}d_{i+1} \tag{2.5}$$

$$p_i = -\frac{1}{16}d_{i-3} + \frac{9}{16}d_{i-1} + \frac{9}{16}d_{i+1} - \frac{1}{16}d_{i+3} \tag{2.6}$$

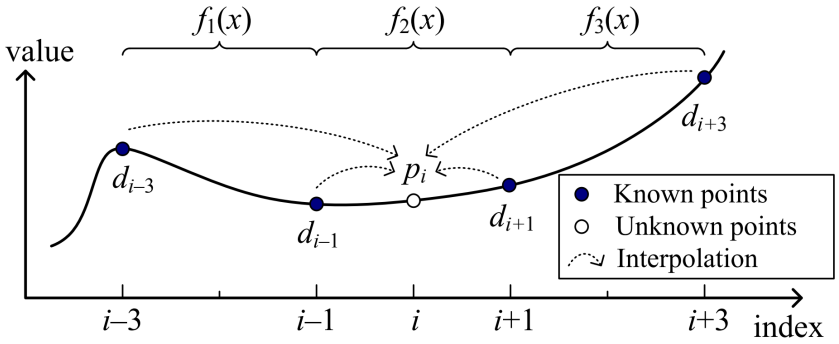

Figure 2.4: (Figure extracted from [104]) Illustration of cubic spline interpolation

To predict data points in the entire dataset, the interpolation method follows a level order pattern. Suppose that the dataset has n elements in one dimension. The number of levels required to cover all elements in this dimension is $l = 1 + ceil(\log_2 n)$. In the example shown in Figure 2.5, at level 0, the algorithm uses 0 to predict d_1; at level 1, it uses d_1 to predict d_9. For now d_9 and d_1 are not necessarily 0 because there are quantizations to drag

the predicted data back to an error-bounded range. After recursively doing the interpolation through each level, all data points would be predicted properly.

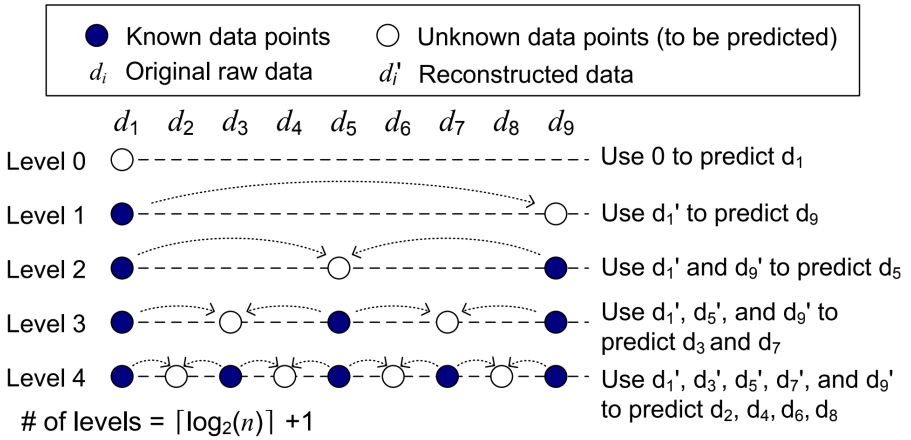

Figure 2.5: (Figure extracted from [104]) Illustration of multilevel linear spline interpolation

Such a multilevel interpolation can be applied on a multidimensional dataset. The idea is that, for each level, the interpolation will be performed along every dimension. The details are presented in [105]'s work and will be omitted here.

2.1.3 Quantization Algorithms

Prior work has used an integer quantizer with a constant error bound. This thesis focuses on quantization algorithms to improve compression performance. The following text briefly describes the prior error-controlled quantization [87].

The design of the quantization stage is shown in Figure 2.6. The first step is to use a prediction method to obtain a predicted value p. Often called p the "predicted value in the first phase" [87]. Then $2^m - 2$ values are expanded from the predicted value in the first phase by stacking the error bound linearly. And they are called the "second-phase predicted values". The quantization process is to add a certain integer (can be negative) times of double error bounds to p and make the predicted second-phase value within the error bound from the original value. Therefore, even if the prediction is not that accurate, the quantization

can always pull the value back to the original data, unless it is totally unpredictable.

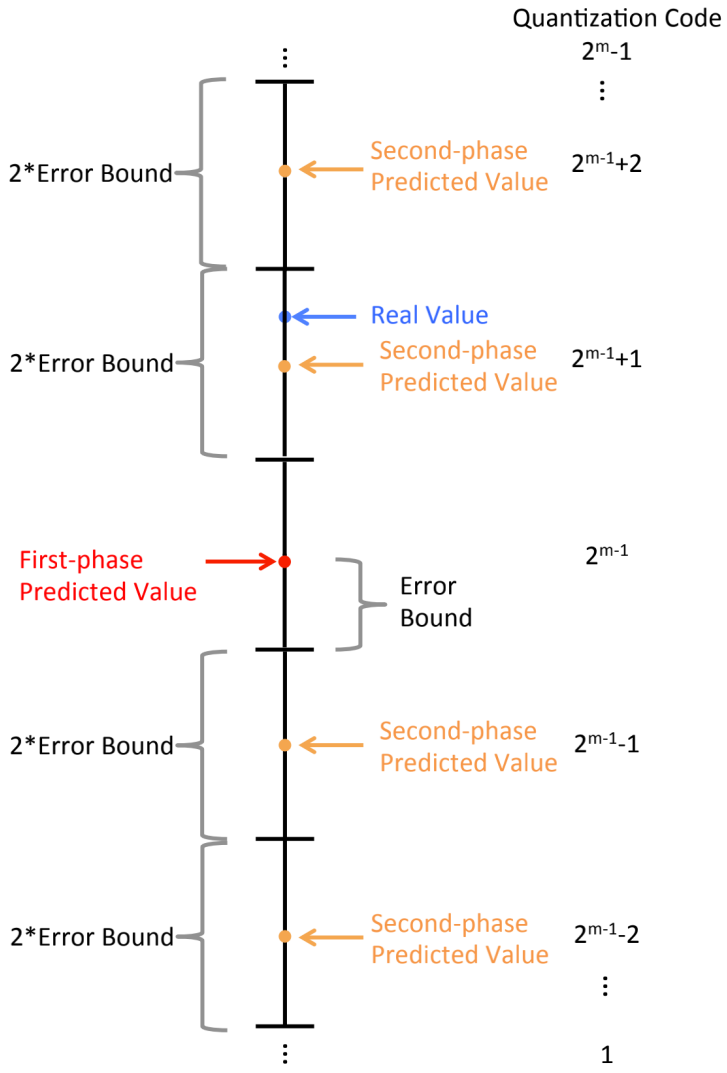

Figure 2.6: (Figure extracted from [87]) Illustration of the basic quantization algorithm

2.1.4 Encoding Algorithms

Two encoding algorithms are added in the SZ compression framework: (1) Huffman Encoding[40]; (2) Run-length Encoding[79]. As they are classic algorithms, the following text presents only a brief introduction.

Huffman Encoding is an Entropy-based lossless compression scheme which assigns each

symbol in the data stream a unique prefix-free code. SZ uses this encoding to compress the quantization bins. Because the prediction is supposed to be relatively precise, there would be a large number of 0 (or values near 0) in the quantization bins. The Huffman Encoding can significantly reduce the quantization bins' size in SZ.

Run-length encoding reduces size by eliminating the space taken by many repeating values. An illustrative example is to encode '111110000222222', the run-length encoding will be '5[1]4[0]6[2]', which records the number of repeating times for each consecutive pattern.

2.1.5 Lossless Compression Algorithms

Common lossless compressors include Gzip[33], LZ4[6], FPC[11], Fpzip[60]. Gzip[33] is a generic compression tool that can compress any type of data stream, such as video and graph files. It integrates LZ4 and Huffman encoding to perform compression. The LZ4 algorithm[6] makes use of a sliding window to find the same repeated sequences of the data and replace them with references to only one single copy existing earlier in the data stream. FPC[11] is a lossless floating-point data array compressor, which analyzes the IEEE 753 binary representations and leverages the finite context model. Fpzip[60] was proposed to compress HPC data by focusing particularly on floating-point data compression. Fpzip can obtain a higher compression ratio due to its more elaborate analysis of HPC floating data, such as predictive coding of floating-point data.

In the SZ compression framework, ZSTD[110] is used by default in the lossless compression stage due to its performance evaluated in [87][21]. As the focus of this thesis is not on the lossless compression, ZSTD is used to fairly compare with previous SZ versions.

2.2 Addressing Diverse User Demands

Existing error-bounded lossy compressors offer different types of error bounds to address diverse user demands. The most common error-bounding approach involves using an absolute

error bound, which ensures that the pointwise difference between the original raw data and reconstructed data is confined within a constant threshold. Many compressors such as SZ, ZFP, and MGARD support absolute error bounds. Other error-bound approaches have been explored to adapt to diverse user requirements. For example, SZ supports pointwise relative error bounds [101]; and Digit Rounding [20], Bit Grooming [100], ZFP, and FPZIP support a precision mode that allows users to specify the number of bits to be truncated in the end of the mantissa, in order to control the data distortion at different levels.

To satisfy user demands on a specific quality of interest, researchers have recently studied how to respect various metrics. For instance, Tao et al. [85] developed a formula that can link the target peak signal-to-noise ratio (PSNR) metric to an absolute error bound setting in SZ such that the data can be compressed based on a user-specified PSNR metric. MGARD supports various norm error metrics and linear quantities of interest in its multigrid compression method. However, no existing error-bounded lossy compressor allows users to set particular prerequisites (or constraints/conditions) in error-bounded compression. However, users often have diverse precision demands for various data value intervals or specific requirements on different spatial regions, which are determined by their sophisticated post hoc analysis purposes and quantities or features of interest.

2.3 Compression Performance Prediction

Knowing the expected compression ratio, quality, and time in advance can be very beneficial for scientific workflows. Many previous works tried either white-box or black-box methods to predict compression performance.

White-box methods usually do part of the compression and collect data from the compressor to predict the final compression performance. For example, Tao [89] developed a white-box method that samples data, estimates the probability density function of the data in the blocks and computes the entropy of the quantize values to derive a metric for com-

pressibility. Jin's method[44] collects the distribution of quantization bins and uses a fixed formula with two tunable parameters to calculate the predicted compression ratio for SZ3. The existing white-box methods can be efficient but cannot fit to all datasets and variations of the compressors.

The black-box methods use data-centric features and predictors that are not derived from the internal mechanism of certain compressors. For example, [77] extracts compressor-agnostic data features to determine the corresponding error bound for a target compression ratio. [28] and [90] compute statistical features derived from data including spatial diversity, spatial correlation, general distortion measurement, etc. to model the ease of compression on each dataset. Then they used the calculated features with a regression model to fit each compressor. The problem is that their selected features are quite expensive to compute. These methods are good in offline use cases but can be too heavy for optimizing the overall transfer time with compression. Therefore, we propose a method that combines compressor-related features, data-related features, and compressor configuration features for a fast and relatively accurate compression performance prediction for the SZ3 series of compressors.

2.4 Reference-based Sequence Compression

Raw sequencing data are typically stored in FASTQ format [18]. A FASTQ file consists of a separate entry for each short sequence, consisting of four lines: an identifier string, a nucleotide sequence (the read), the character '+', and quality scores. The identifier string contains information about the sequencing technology and other metadata obtained from the sequencing machine, which uniquely describes a read. The nucleotide sequence is a string of A, C, G, T, and N characters that represent the bases (base-pairs) of the DNA sequence. There are some other characters for storing protein sequences that do not exist in our datasets. Quality scores record the confidence of each base generated by the sequencing machine. Due to their higher entropy and larger alphabets, quality scores have proven to be

more difficult to compress than reads [36].

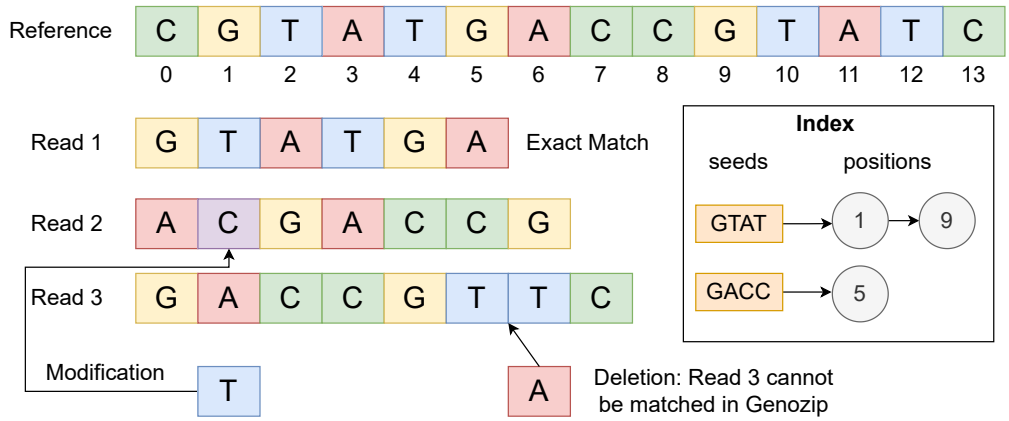

Figure 2.7: Reference-based sequence matching process: (1) use seeds to build an index for the reference sequence; (2) find matching locations for reads on the reference sequence; (3) for unmatched bases, store the difference. Our algorithm performs better matching by storing more seeds for a higher chance of matching, and local search for insertion and deletion detection.

To reduce the space required to store FASTQ files, researchers focus on compression of reads and quality scores, which consume most of the space and carry the most relevant information [36]. Traditional general-purpose compression algorithms such as gzip [33], and bzip2 [30] fail to obtain a high compression ratio for sequencing data. Thus, many specialized FASTQ compressors[17, 47, 96, 48] have been proposed. The most successful of these algorithms are so-called reference methods, which exploit the fact that more than 99% of human genomes are identical [42] to reduce greatly the storage space required.

Existing FASTQ sequence compression algorithms can be categorized into two classes: reference-based and reference-free algorithms. *Reference-based algorithms* map the nucleotide sequences in a FASTQ file to a reference genome and use the mapped positions to encode the sequences, as shown in Figure 2.7. Examples include LW-FQZip[102], LW-FQZip2[39], GTZ[95], and genozip[48]. *Reference-free algorithms* are used when a reference genome is not available. For example, Leon[7] and Quip[45] use assembly-based algorithms. Generally, reference-based compressors perform better in terms of both compression time and ratio than reference-free compressors, and thus we focus on developing a reference-based algorithm for

FASTQ compression.

2.5 Remote Task Orchestration

The cloud-based Function-as-a-Service (FaaS) paradigm supports transparent remote function execution and data staging. Researchers have shown the efficiency and scalability benefits of deploying applications as sets of functions that execute in response to events [5]. The FaaS paradigm has been extended as a general model for remote computing across federated resources. AWS Lambda is one widely adopted, serverless, event-driven platform that allows businesses to reduce infrastructure costs. It aims to run code without thinking about servers or clusters. However, in our scenario, users need to run compression on specific sites instead of letting Amazon choose a computation server. Globus Compute [14] is another platform that uses FaaS as an interface for remote function execution across federated computing infrastructure. In the Globus Compute model, users can deploy endpoints on arbitrary computers. These endpoints are registered with the cloud-hosted Globus Compute platform and may then be used to execute functions. The cloud platform manages the secure and reliable execution of these functions on the selected endpoints. This model suits our remote compression tasks very well.

Data transfer is essential in modern scientific computing. Globus Transfer is a widely used research data management platform that enables high-performance, secure, and reliable third-party data transfers. Globus Transfer builds on the GridFTP protocol for data movement and adopts several optimization techniques such as parallel streams [35, 98], which can significantly improve data transfer performance. Transferring big data files with Globus Transfer, however, may still be time-consuming due to limited network paths and underprovisioned data transfer nodes (DTNs). We aim to improve data transfer performance with GlobaZip by applying dynamic compression methods.

CHAPTER 3

METHODOLOGY

3.1 Preserving Diverse Data Constraints

Based on the mechanism of prediction-based compression algorithm SZ3, I identified five different data constraints that need to be handled properly. We need to modify one or more stages in both compression and decompression to ensure the data constraints are met. To clarify the meaning of terminologies including "data constraints", "compression", "data quality", I present a formalized description in the following section 3.1.1.

3.1.1 Formalization

Given a scientific dataset D composed of N floating-point values (either single precision or double precision), the objective is to develop an error-bounded lossy compressor that can respect a set of user-defined constraints such as preserving global value range or preserving multiple error bounds based on value intervals or different regions in the dataset.

Three assessment metrics are considered. The first two are compression speed s_c and decompression speed s_d. They are usually measured in megabytes per second: in other words, the size (in MB) of the original dataset processed (either compressed or decompressed) per time unit. The third metric is compression ratio (denoted by ρ), which is defined as follows:

$$\rho = \frac{N \cdot sizeof(dataType)}{Size_{compression}}, \tag{3.1}$$

where $dataType$ can be either float or double and $Size_{compression}$ is the total size after compression.

Table 3.1: Examples of user-required constraints applied to scientific simulation datasets

No.	User-Required Constraints	Examples	Science Domains
(A)	Isolating irrelevant value	Hurricane Isabel [41], Katrina [3]	Climate, Weather, etc.
(B)	Preserving global value range	CESM [46]	Climate, etc.
(C)	Preserving value-interval-based error bounds	Katrina [3], NYX	Weather, Cosmology, etc.
(D)	Preserving multiregion-based error bounds	CESM [46]	Weather, Seismic imaging, etc.
(E)	Preserving irregularly shaped regions	QMCPACK, Miranda, CESM [46]	Hydrodynamics, Weather, etc.

Our goal then can be formulated as Formula (3.2).

$$\begin{aligned} &\text{Maximize } \rho \\ &\textit{subject to } \textit{ user-required constraint} \end{aligned} \tag{3.2}$$

The *user-required constraint* refers to additional requirements applied to the lossy compression beyond the traditional error-bounding constraint. We formulate the five constraints listed in Table 3.1 as follows:

$$\text{CONSTRAINT (A): } \textit{Preserve and isolate } d_i \notin [R_{min}, R_{max}] \tag{3.3}$$

$$\text{CONSTRAINT (B): } \textit{Preserve} \begin{cases} \max(\widehat{d_i}) = high(r(D)) \\ \min(\widehat{d_i}) = low(r(D)) \end{cases} \tag{3.4}$$

$$\text{CONSTRAINT (C): } |d_i - \widehat{d_i}| \leq e(d_i) \tag{3.5}$$

$$\text{CONSTRAINT (D, E): } |d_i - \widehat{d_i}| \leq e(LOC(d_i)), \tag{3.6}$$

where $d_i \in D$ denotes the ith data point in the original dataset D, $\widehat{d_i}$ is its corresponding decompressed value, $low(r(D))$ and $high(r(D))$ are the boundaries of the dataset D's value range $r(D)$, $e(d_i)$ denotes the user-required error bound in terms of data point d_i's value (i.e., user-specified error bound in terms of the value interval that covers d_i), $LOC(d_i)$ refers to the spatial location of the data point d_i, and $e(LOC(d_i))$ denotes the user-specified error bound for the specific region covering $LOC(d_i)$. Constraints D and E have identical formulas: the key difference is that E allows irregular shapes, whereas D focuses on a regular shape

defined by a rectangular box or cube. We summarize all the notation in Table 3.2.

Table 3.2: List of Symbols

Notation	**Description**
d_i	original data value at position i
p_i	predicted value of d_i
$\hat{d}_i$	reconstructed data value after decompression
$r(x)$	value interval of data point x (d_i)
$e(x)$	specified error bound based on a value interval (x=$r(d_i)$)
$e(LOC(x))$	specified error bound based on the location ($x = d_i$)
$l(x)$	length of some value range ($x = r(d_i)$)
$low(r(x))$	lower boundary of value interval $r(x)$
$high(r(x))$	higher boundary of value interval $r(x)$
q	quantization index (i.e., quantization code)
q_s	shifted quantization number

We give an example to further illustrate how the research problem is formulated in our work. As described above, researchers using the Hurricane Katrina dataset to track the path of the hurricane are concerned only with water surface values above 1m. Based on Formula (3.2) and Formula (3.5), the target is to maximize the compression ratio while ensuring that the relatively higher values have lower error bound (e.g., if $d_i \geq 1$, then $e(d_i) = 0.01$; otherwise, $e(d_i)$=0.1). Another example is the Nyx cosmological simulation with a specific quantity of interest, namely, dark matter halo information. According to the Nyx analysis code [74], the dark matter halo cells are computed based on a threshold located in the interval of [80,85], which means that for any data point d_i in [80,85], their error bounds $e(d_i)$ should be lower than $e(d_j)$, where d_j refers to the data points that fall outside of the critical interval [80,85]. Such a multi-interval-based error bound setting can eliminate the distortion of halo cells calculated by the reconstructed data with the same compression ratios.

3.1.2 Isolating the irrelevant data

Certain values in the dataset may have special meaning which are outside a normal range. For example, in the Katrina dataset, there is a portion of data points with value -99999

representing the contour boundaries that are not in the range of normal data points. They can significantly lower the rate of valid prediction using neighbouring data points and thus lower the overall compression ratio of prediction-based compression algorithms. We can design a preprocessing step in the compression algorithm for these datasets to smoothen the so called "irrelevant data" but record the positions and reconstruct it during the decompression process.

In order to handle the irrelevant values correctly and efficiently, the first three stages in SZ (i.e., prediction, quantization, and Huffman encoding) all need to be modified. The details are as follows.

In stage 1 (data prediction), the key problem is to fill the missing values for the irrelevant data points such that the smoothness of the data will not be destroyed by irrelevant values. This strategy can maintain a high prediction accuracy at each data point throughout the whole dataset. To this end, we use the Lorenzo predicted values [57] to replace irrelevant values. More specifically, for a 1D dataset, the irrelevant data will be replaced by the values of their preceding data points ($d_i \leftarrow d_{i-1}$); for a 2D dataset, $d_{i,j} \leftarrow d_{i,j-1} + d_{i-1,j} - d_{i-1,j-1}$; and for a 3D dataset, $d_{i,j,k} \leftarrow d_{i-1,j,k} + d_{i,j-1,k} + d_{i,j,k-1} - d_{i-1,j-1,k} - d_{i-1,j,k-1} - d_{i,j-1,k-1} + d_{i-1,j-1,k-1}$. Figure 3.1 illustrates how irrelevant values are modified in the prediction stage for a 2D dataset. As shown in the figure, the irrelevant value is 1E35. When encountering an irrelevant data point during compression, the values will be estimated based on the Lorenzo predictor: for example, $1.29 \leftarrow 1.25+1.27-1.23$; $1.33 \leftarrow 1.31+1.29-1.27$).

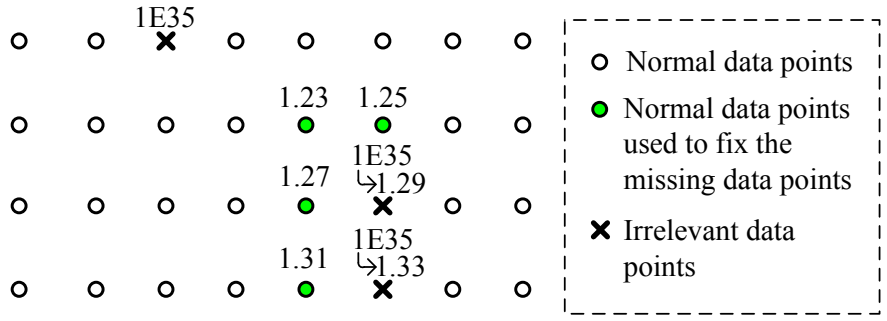

Figure 3.1: Illustration of how irrelevant data values are cleared in a 2D dataset.

After modifying the "irrelevant" data points, we propose two strategies to preserve the irrelevant values during the second stage of the compression pipeline.

- **Strategy A:** Since the irrelevant value is often a single floating-point number (such as 1E35), we use a 1-bit array to mark whether this is an irrelevant value for each data point (1 indicates irrelevant value, and 0 indicates normal data).

- **Strategy B:** Use one quantization bin (such as bin #1) from the quantization range to mark whether the data point is an irrelevant value. Thus, there are three types of quantization bins in this case: (1) quantization bin #0 records the unpredicted data value as usual [57], (2) quantization bin #1 marks the irrelevant data, and (3) the remaining quantization bins are used to record the distance between the predicted value and original value.

Each of the two strategies has its own advantages and disadvantages. Strategy A has no impact on the distribution of quantization codes, so it can maintain high Huffman-encoding efficiency on the quantization codes; but it suffers from an overhead of storing the extra bit array. Strategy B does not have such an overhead; but it may affect the distribution of quantization codes to a certain extent, which will inevitably lower the effectiveness of compressing the quantization codes by Huffman encoder.

In the third stage (Huffman encoding), if the solution adopts strategy A, we compress the 1-bit array using Huffman encoding. This compression may significantly lower the overhead because irrelevant data points are generally a small portion of the whole dataset and therefore the 1-bit array is composed mainly of 0s.

3.1.3 Preserving the global value range

Scientific data have important physical constraints regardless of the precision of the data. For instance, water's temperature at the standard atomosphere pressure should be within

the range of 0°C to 100°C. If after the compression, certain data points are reconstructed to be outside the boundary, they need to be found and further modified even if their values are within the error bound.

The simplest, yet suitably efficient, strategy for preserving the global value range is to include the original value range information as metadata in the compressed data. During decompression, when a reconstructed data value outside the "original value range" is found, the algorithm will replace it with either the minimum value or maximum value of the value range. This strategy introduces little computation overhead in the compression stage because we need only to scan the dataset to find the maximum and minimum values, a process we refer to as "preprocessing" in our evaluation. During decompression, a small computation overhead (generally ∼10% in our experiments) may be introduced by this strategy, because the algorithm needs to check each data point to determine whether the reconstructed value falls outside of the original dataset's value range. If so, it would be substituted by either the maximum or minimum value.

3.1.4 Preserving value-interval-based error bounds

Different value-intervals may require different precisions. The existing lossy compression methods usually can only set a global error bound for all data, leading to inefficient compression when the error bound is set too low or inacurate values for important intervals when the error bound is set too large. I design an algorithm that allows different value intervals to have different error bound while still maintaining the integrity of the compression/decompression pair.

As illustrated in Figure 3.2, the proposed multi-interval quantization method calculates the total number of quantization bin indices based on the varied-length quantization bins, followed by other compression techniques including Huffman encoding and dictionary encoding (Zstd). Algorithm 1 presents the pseudocode of the multi-interval quantization in the

compression stage.

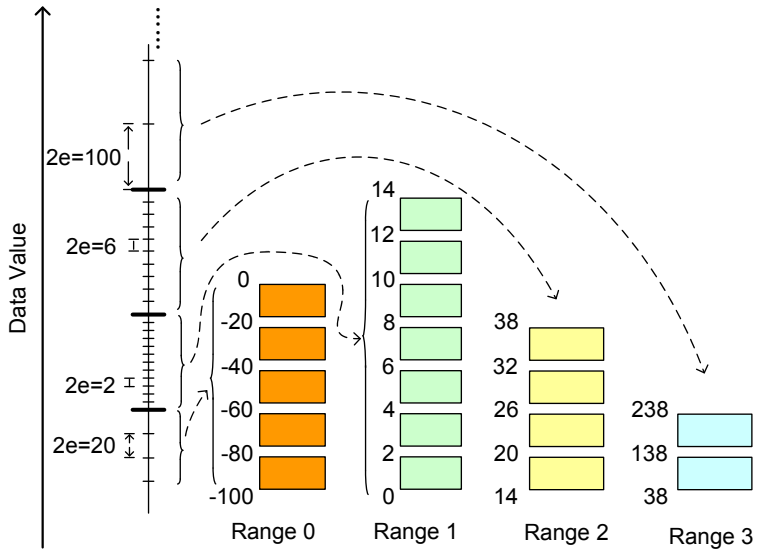

Figure 3.2: Multi-interval error bound model. This example shows a few exaggerated error bounds in each range for simplicity of description: each rectangle represents twice the error bound in that range, and the ranges are tightly connected. The error bound will usually be smaller in practice, and each range may contain hundreds or thousands of error bounds.

For each data point, we must deal with three relationships between the original raw value d_i and its predicted value p_i: (1) $r(d_i) = r(p_i)$: they fall in the same interval; (2) $r(d_i) < r(p_i)$: the predicted data are in some range ahead of the original data; and (3) $r(d_i) > r(p_i)$: the predicted data are in some range before the original data.

Situation 1 (lines 5∼6): If the original raw value d_i and the predicted value p_i fall in the same interval (*i.e.*, $r(d_i) = r(p_i)$), the quantization problem falls back to the traditional linear-scale quantization [57]. Specifically, we can use the following formulas to compute the logic quantization code and decompressed data.

$$q = round(\frac{d_i - p_i}{2e(r(d_i))}) \tag{3.7}$$

$$\hat{d_i} = p_i + 2e(r(d_i)) \cdot q \tag{3.8}$$

We use an example to illustrate how the linear-scale quantization works. Suppose the error

Algorithm 1: Multi-interval Quantization in Compression Stage

Input: user-specified intervals and error bounds ε
Output: compressed data stream in form of bytes

for each data point d_i **do**
  Use the composed prediction that combines Lorenzo predictor and linear regression predictor to obtain a prediction value p_i.
  $I_p \leftarrow r(p_i)$. /*Obtain interval index of p_i*/
  $I_d \leftarrow r(d_i)$. /*Obtain interval index of d_i*/
  if $I_d == I_p$ **then**
    $q \leftarrow round(\frac{(d_i - p_i)}{2e(I_d)})$./*Quantized distance between d_i & p_i.*/
  else if $I_d > I_p$ **then**
    $t = \sum_{i=I_p+1}^{I_d-1} \frac{l(i)}{2e(i)}$. /*Count bins for middle intervals.*/
    $t_p = round(\frac{high(I_p) - p_i}{2e(I_p)})$. /*Get quantized distance for I_p.*/
    $t_d = round(\frac{d_i - low(I_d)}{2e(I_d)})$. /*Get quantized distance for I_d.*/
    $q = t + t_p + t_d$. /*Get the logic quantization code.*/
  else
    $t = \sum_{i=I_d+1}^{I_p-1} \frac{l(i)}{2e(i)}$. /*Count bins for middle intervals.*/
    $t_p = round(\frac{high(I_d) - d}{2e(I_d)})$. /*Get quantized distance for I_d.*/
    $t_d = round(\frac{p_i - low(I_p)}{2e(I_p)})$. /*Get quantized distance for I_p.*/
    $q = t + t_p + t_d$. /*Get the logic quantization code.*/
  end if
  $q_s \leftarrow q + R$. /*Shift quantization code.*/
end for

bound (i.e., $e(r(d_i))$) is 20 and we have d_i=−74, p_i=−95. Then $d_i - p_i = 21$ and $q = round(21/40) = 1$. The decompressed value $\hat{d_i}$ is −75, whose distance to the raw value is less than the error bound.

Situations 2 and 3 (lines 7∼17): These correspond to the situation where the raw value d_i and its predicted value p_i fall in different value intervals (i.e., $r(d_i) \neq r(p_i)$). In the following text, we describe the situation with $r(d_i) > r(p_i)$ (i.e., lines 7∼11 shown in the algorithm); the other situation is similar.

The fundamental idea in handling this situation is to adjust the quantization policy to use various bin lengths or sizes in different value intervals. Specifically, we count the quantized distance (i.e., the number of quantization bins) from the predicted value to the original raw value. Whenever the counter crosses a different interval, we continue to add the quantization bins from the boundary of the new interval. As illustrated in Figure 3.2, suppose the predicted value is located at -10 and the original value is 100. Then the calculation of the quantization bins involves all the value intervals, and the quantization code is 1+7+4+1=13. The decompressed data would be (38+238)/2=138. Obviously, the decompressed data value is determined mainly by the last value interval and its quantization bin size. The formula for reconstructing the decompressed value is given below (we assume the raw data valuw is greater than the predicted value, without loss of generality):

$$q_t = round(\frac{d_i - low(r(d_i)) - e(r(d_i))}{2e(r(d_i))}) \tag{3.9}$$

$$\hat{d_i} = low(r(d_i)) + e(r(d_i)) + 2e(r(d_i)) \cdot q_t. \tag{3.10}$$

Now we describe the decompression in Algorithm 2. The algorithm proceeds by executing similar operations to the compression process but in reverse order to obtain the decompressed data from a predicted data value and the corresponding quantization bins. As shown in the pseudocode, we first calculate the number of quantization bins for each value interval (line

3∼5). We then decompress each data point based on the multi-interval quantization (lines 6∼34). If the raw data value is lower than the predicted value (i.e., $q_j < 0$), the code will scan all the involved value ranges downward (lines 10∼29). Lines 25∼29 refer to the situation where the predicted value and original raw data value fall in the same interval. Lines 13∼23 deal with the other situation where the two data values fall in different intervals.

Note that we need to deal with the edge situation carefully. For instance, when the original data are near the high or low bound of an interval, the quantization value in this final interval might be equal to $quantRange[i]$, causing the decompressed value to be in the next interval unexpectedly. In this case we shift the quantization by 1 in the compression stage to ensure that the decompressed data and original data are in the same interval.

3.1.5 Preserving multiregion-based error bounds

The positions of data points in some datasets have geolocation meanings and thus different regions may require different precisions for certain applications. Figure 3.3 illustrates our approach enabling users to mark interesting regions that we then use to apply a tighter error bound on each region according to the requirement and preknowledge of the data distribution.

To reduce the overhead in (de)compression time, we do not assign a region to each data point; instead, we consider each intrablock of data in the same region. To make the algorithm simpler, we adopt the intrablock of size 6×6×6 for 3D data, which is consistent with SZ's linear regression prediction block size [54]. The undesired side-effect of this method is that the user-customized region (a regular box) may cut through some intrablocks. Since the data have to be compressed/decompressed in the unit of blocks (e.g., 6×6×6 for 3D data), some storage overhead occurs at the edge of the customized region. We consider this storage overhead acceptable because the region of interest is relatively large in practice (at a scale of several thousands) while the block size is far smaller (such as 6×6×6). Keep in mind that

Algorithm 2: Multi-interval Quantization in Decompression

Input: compressed data stream
Output: decompressed data stream in the form of bytes

Read value intervals and error bounds in the header and initialize multi-interval quantizer.
Read the quantization bins and unpredictable data.
for each interval index I_i **do**
    $\hat{l}_i = \frac{l(I_i)}{2e(I_i)}$. /*Calculate # quantization bins for each interval*/
end for
for each decompressed data position j **do**
    Use the composed prediction that combines Lorenzo predictor and linear regression predictor to obtain a prediction value p_j.
    $q_j = q_s - R$. /*Get the logic quantization code q_j*/.
    $I_p \leftarrow r(p_j)$. /*Obtain range index of p_j*/
    if $q_j < 0$ **then**
        $\Delta \leftarrow p_j - low(I_p)$ /*Compute p_j's distance to the low boundary*/
        $\hat{\Delta} \leftarrow round(\Delta/2(e(I_p)))$ /*Compute quantized distance*/
        if $q_j + \hat{\Delta} < 0$ **then**
            for i from $I_p - 1$ to 1 **do**
                if $q_j + \hat{l}_i \geq 0$ **then**
                    $\hat{d}_j \leftarrow high(i) - e(i) + (q_j + 1) \cdot (2 \cdot e(i))$. /*Get decompressed data*/
                    if $\hat{d}_j < low(i)$ **then**
                        $\hat{d}_j \leftarrow low(i) + e(i)$. /*Correct decompressed data*/
                    end if
                else
                    $q_j \leftarrow q_j + \hat{l}_i$. /*Add quantization length for further search*/
                end if
            end for
        else
            $\hat{d}_j \leftarrow p_j + q_j \cdot 2 \cdot (e(I_p))$. /*Compute decompressed value*/
            if $\hat{d}_j < low(I_p)$ **then**
                $\hat{d}_j \leftarrow low(I_p) + e(I_p)$. /*Perform correction to avoid undesired boundary-crossing*/
            end if
        end if
    else if $q_j == 0$ **then**
        $\hat{d}_j \leftarrow p_j$. /*The prediction is accurate, directly use the predicted value*/
    else if $q_j > 0$ **then**
        Calculate the decompressed data using similar methods. /*For brevity we do not include details here. It is similar to the case when $q_j < 0$, with just a few changes to the low and high bounds and some calculation differences.*/
    end if
end for

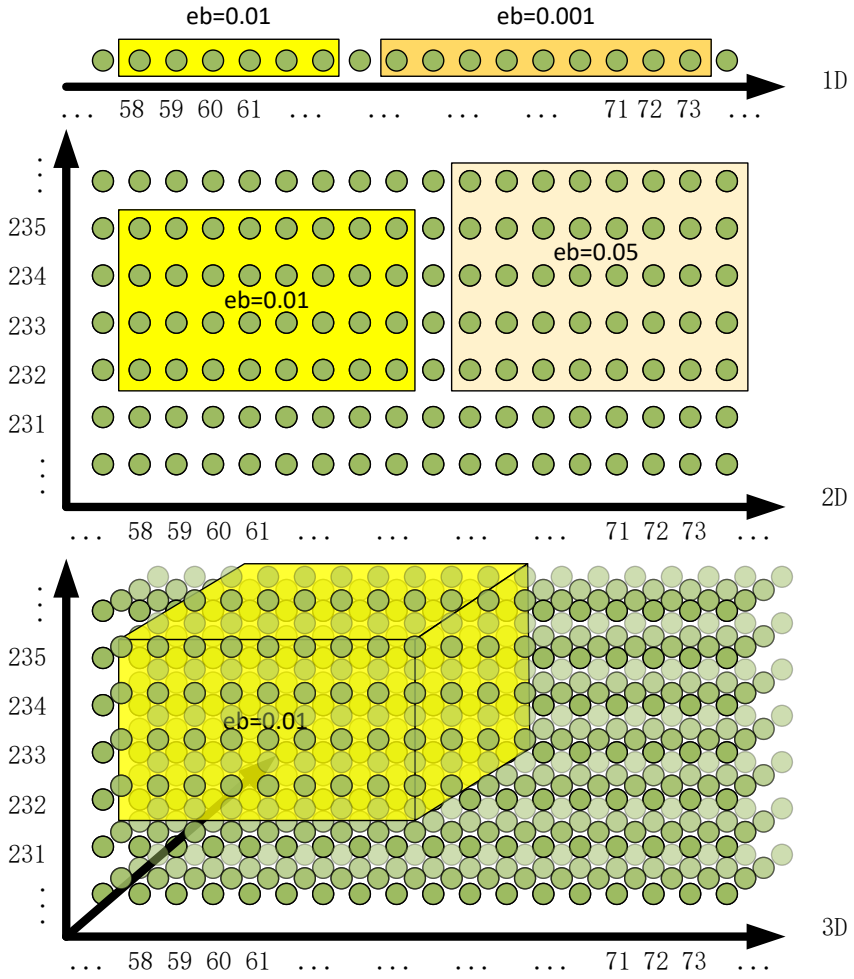

Figure 3.3: Constraint(D) region selection for 1D, 2D, and 3D data: In 3D cases, each region can be specified with seven parameters: the starting positions (3 parameters), the length of each direction (3 parameters), and the error bound (1 parameter).

the purpose of proposing this region-based algorithm is to reduce the compressed data size while preserving precision for post hoc analysis.

The whole process can be done in the quantization stage if the predictor is fixed, since the varied error bounds will take effect only when calculating the quantization code. However, when we compose the linear regression predictor and Lorenzo predictor together, the data sampling process will need a correct error bound to select an optimal predictor for the current block. The varying error bounds can cause the predictor selection to yield a bad result. This challenge exists in all kinds of blockwise compression where predictors may change according to the error bound for each block.

3.1.6 Preserve irregular regions with a bitmap

To satisfy complex, customized regions of error bounds (rather than just rectangles or cubes), we introduce a bitmap error bound array (as shown in Figure 3.4). It contains a set of integer values that indicate different data distortion levels, each of which corresponds to a specific error bound value. Such a method allows users to specify an error bound for each data point. However, it is not realistic to manually assign each data point an error bound, since there are usually millions of data points. Instead, users can use third-party software to mark a customized shape in a picture or apply computer vision techniques to obtain contours that distinguish regions (e.g., land and ocean). Such a customized-marking option is more accurate and flexible in practice especially in geolocation-related research (to be demonstrated later).

Although using bitmaps supports the most complex error bound settings—allowing each data point to have its own error bounds—cases rarely require many different error bounds to coexist in one dataset in practice. Most requirements are limited to a few different error bounds in total, because of coherence of data in space; for example, "higher precision may be required near the hurricane center" or "land areas need higher precisions than ocean areas."

Therefore, we use one byte to represent all different types of error bounds. That is, we use a byte array to store the index of error bound for each data point and apply Huffman coding and lossless compression to compress the bitmap array if needed. In the extreme case, the original single error bound would be equivalent to an all-zero bitmap, which would bring almost zero overhead after proper compression. The overhead of using a bitmap array will be presented in Section 4.6.

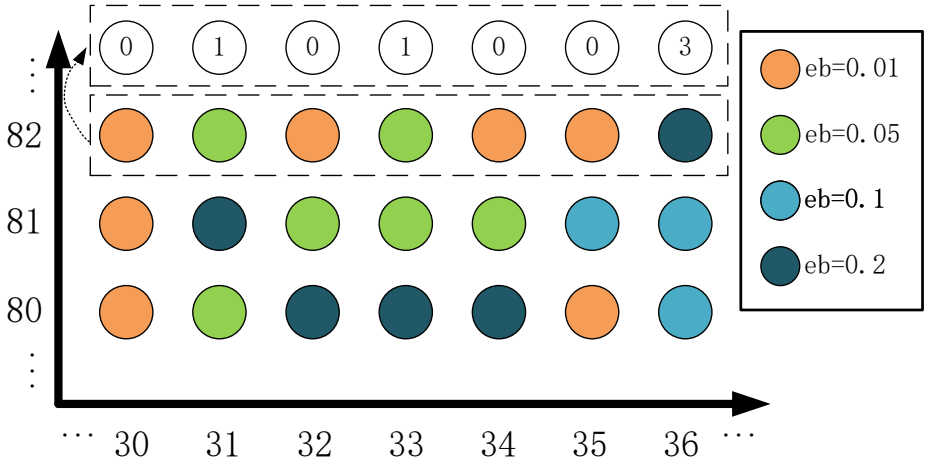

Figure 3.4: Illustration of bitmap error bound setting: Use an index to represent the error bound for each data point, and use a separate array to store all possible error bounds.

The bitmap solution solves a complicated error bound requirement (actually, all possible error bound requirements) and presents an opportunity for automated error bound selection, which may relieve scientists of having to configure advanced bitmap generation algorithms. This solution can also have additional global advantages compared with the region-based method when different error bounds are distributed evenly across the dataset. By setting a fixed proportion of data points with some certain error bounds, we can achieve higher compression ratio, lower root mean squared error, and comparable visual quality.

3.2 Compression Performance Preview

To ensure confidence in the quality and speed of data compression, we need a mechanism to visualize at least part of the decompressed data to verify that the distortion remains within an acceptable range. Additionally, we require an estimate of the compression performance in advance to confirm that the compressed size is within an acceptable range and that the compression speed meets the application's requirements.

3.2.1 Data Quality Preview

The data preview mechanism provides users with both a visualization and a histogram of data value distribution ahead of the compression of full dataset: see Figure 3.5 (C) and (D). Users can decide which value range to focus on by looking at the value distribution, and determine which regions to focus on by selecting rectangular regions on the visualized image. Setting the error bound in this way enables user to pay more attentions on data characteristics. For an extremely large file, users can select a layer to preview as shown in Figure 3.6, which helps guide their region/range-based compression configuration. This preview does not involve compression and only transfers an image from the remote machine to the local machine. The compute and data transfer overhead is very low, and thus the preview is very responsive. Users can easily select a few layers from different parts of the data to have a good understanding of the data characteristics.

3.2.2 Feature-based Performance Prediction

In general, it is impossible for users to predict compression quality (such as compression ratio and data distortion level) for a particular error-bounded lossy compressor without performing the compression on the given dataset. This is because the effect of data prediction/transform and coding in the compressor varies with diverse data features. However, certain compression

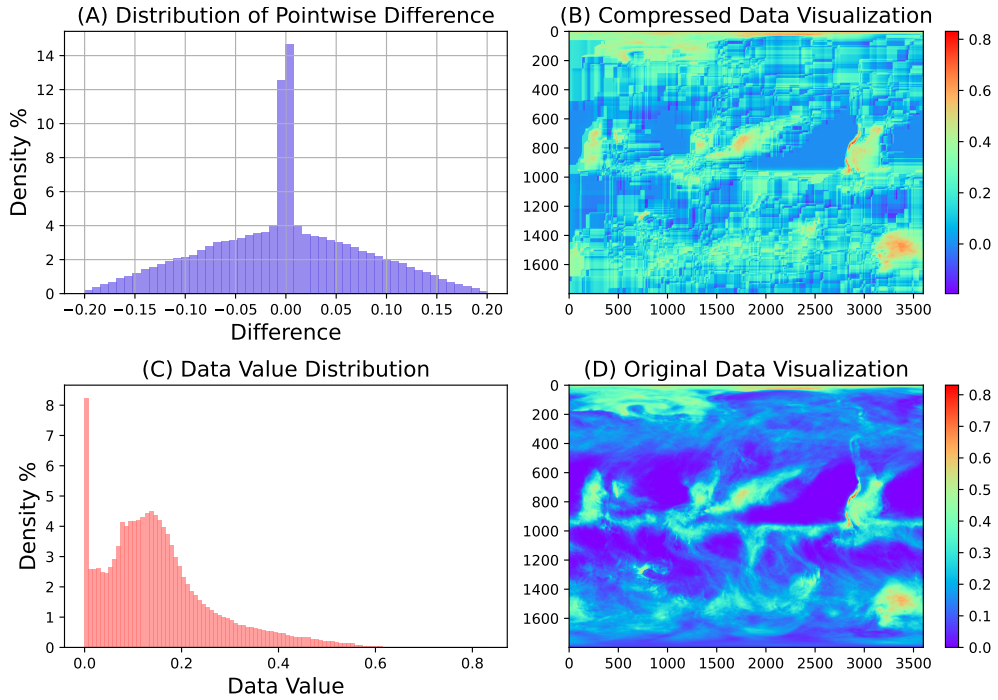

Figure 3.5: CESM CLDMED multi-range compression distortion

performance characteristics can be predicted with features extracted from the first few stages of the compression. I propose a prediction model to estimate the lossy compression ratio, compression speed, and peak-to-noise ratio (PSNR). With this prediction model, users can quickly test multiple compression settings and choose the one that best matches their use case.

We train a machine learning (ML) model on masses of sample datasets, with the aim to build a relationship between the compression-related features and the compression quality. The model can then be used to estimate compression quality accurately based on the features extracted from the given datasets at runtime.

We derive many features as input to our model, as illustrated in Figure 3.7. Identifying a set of useful features is challenging, because (1) the extraction of each feature should have low computation cost, and (2) the features should form an accurate indicator of the compression quality. We consider features in one of three categories: (1) config-level features, (2) data-based features, and (3) compressor-level features.

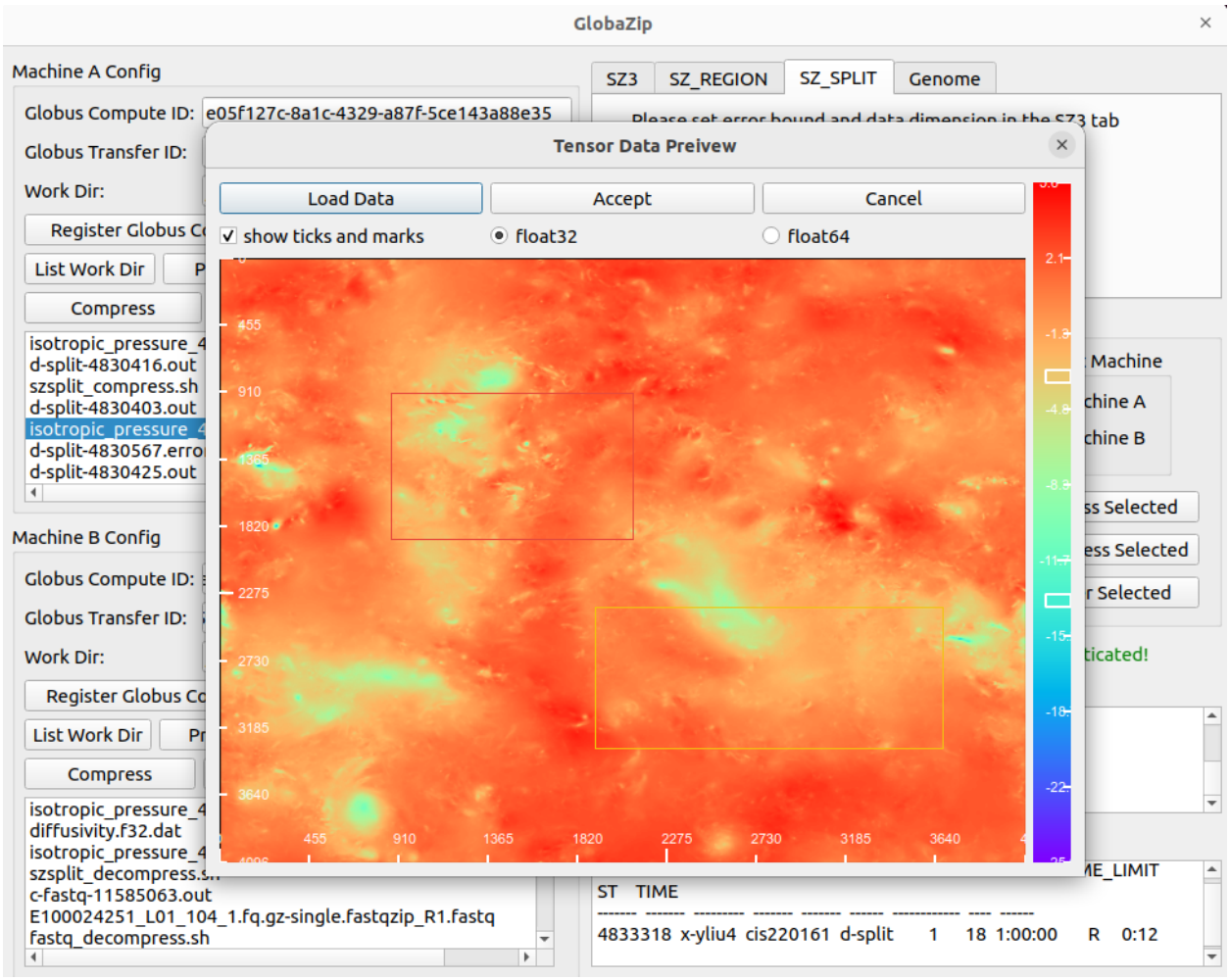

Figure 3.6: The UI to preview one layer of the data in a large file. It also allows users to visually select different regions and ranges for compression settings.

Config-based features are configuration settings (including error bound values and compression pipeline) specified by users. Different error bounds can yield largely different compression quality (e.g., compression ratios and compression speed). Compression quality also depends on specific compressors each with distinct designs. The prediction-based compressors[53, 57], for example, may adopt various predictors which may exhibit different performances. We enable our model to recognize the characteristics of compressors by treating the compressor-type feature as a discrete classification variable and feeding it with profiling data.

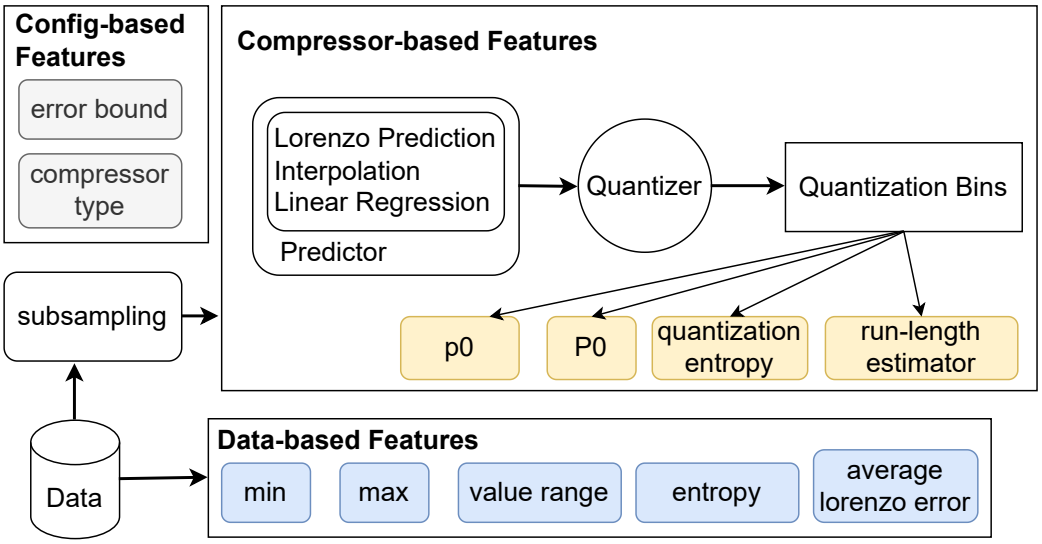

Figure 3.7: The features used to predict compression quality are categorized into three types: config-based, compressor-based, and data-based features, which are shown as colored boxes.

Data-based features describe the characteristics of datasets, which is also a key factor to distinguish the compressibility. As shown in Table 3.3, even for the same application, different datasets can have very different properties such as min, max, and value range. In addition, we also use byte-level information entropy as one feature, because it reflects the "chaos-level" of a dataset. The entropy is defined as

$$H(X) = -\sum_{x \in S} p(x) \log p(x) = E[-\log p(X)]$$

where S is the set of byte values (0-255) and p denotes the probability/frequency of an element in S. In general, the higher entropy a dataset exhibits, the more difficult it is to compress that dataset. As verified in Figure 3.8 (a) and (b), the entropy value projects a positive correlation against the compression time, especially when the error bound is relatively low. It is worth noting that when the error bound is relatively high, the entropy would lose its effect (as shown in Figure 3.8 (c)), because the large error bound would diminish the data variation. Moreover, we use the average Lorenzo error (i.e., the difference between

the true data value and Lorenzo-predicted value[53]) as a feature to shape the "easiness of prediction" for a dataset. If the average Lorenzo error is high, the prediction stage tend to be imprecise, leading to low compression ratio.

Table 3.3: Examples of the basic data-based features in different datasets: CLDHGH, FLDSC, PCONVT are three fields in the CESM[46] dataset. HACC-VX and HACC-VY are two fields in the HACC[34] dataset.

Dataset	**CLDHGH**	**FLDSC**	**PCONVT**	**HACC-VX**	**HACC-XX**
min	0.00	92.84	39025.27	-3846.21	0.00
max	0.92	418.24	103207.45	4031.25	256.00
value range	0.92	325.40	64182.18	7877.46	256.00

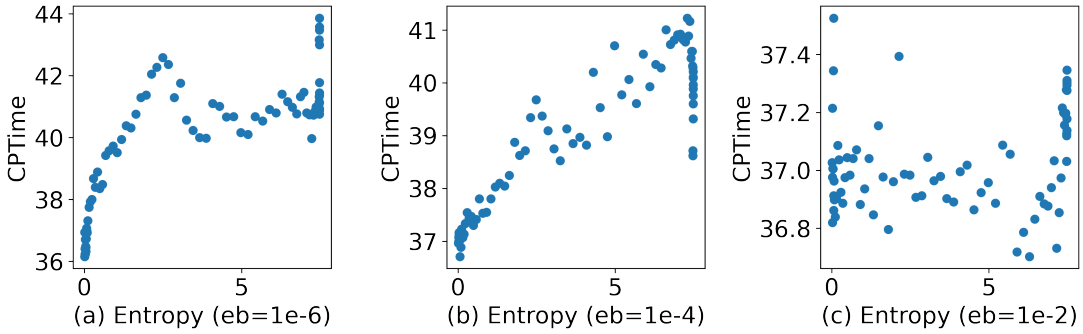

Figure 3.8: Data entropy vs compression time in Reverse Time Migration (RTM) [72] application with three error bound settings

Compressor-based features are the properties of the intermediate data used in the course of lossy compression, which generally have the highest prediction ability for compression quality. Specifically, we focus on the quantization bins, as shown in Figure 3.7. Since the quantization bins are encoded by the subsequent lossless encoders, its characteristic closely correlates to the final compression quality. In order to control the execution overhead, the quantization bins are computed based on the sampled data points. As demonstrated in Figure 3.7, we develop four compressor-based features, including p_0, P_0, quantization entropy, and run-length estimator. (1) p_0 denotes the percentage of the 0-value bins over all quantization bins. In general, large p_0 tends to yield a high compression ratio and compression speed, because a large majority of predictions should be accurate in this situation. (2) P_0

denotes the fraction of '0'(encoded) taken in Huffman coding in the regard of the full Huffman encoded data size. (3) Quantization entropy is the entropy of quantization bins. If the prediction is accurate, quantization bin values will mostly be near 0, and the quantization entropy will be low. (4) Run-length estimator (denoted R_{rle}) is derived from P_0 and p_0 by the following equation: $R_{rle} = 1/((1-p_0)P_0 + (1-P_0))$.

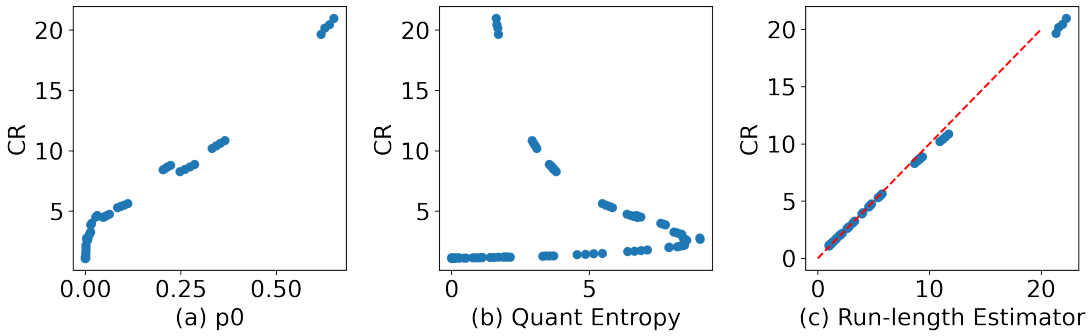

Figure 3.9: The relationship between p_0, quantization entropy, run-length estimator and compression ratio for Nyx application.

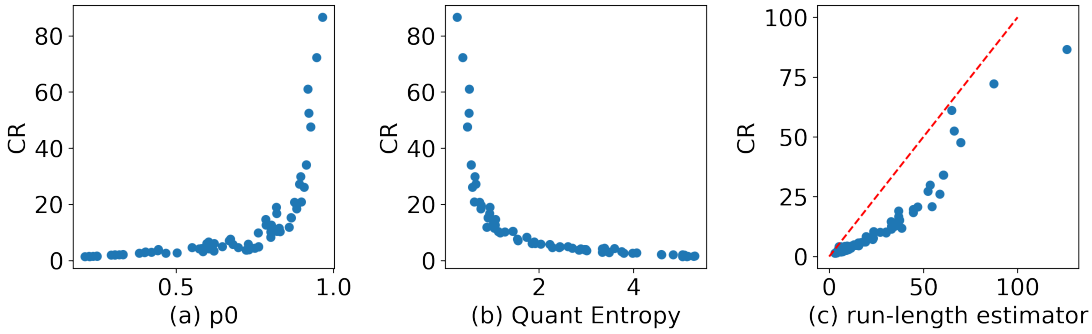

Figure 3.10: Run-length estimator alone fails to predict the compression ratio for Miranda application while the three features together form a correlation to the compression ratio which can be learned by a machine-learning model.

Although p_0 and P_0 are also used in related work [44], our solution is much more accurate in compression quality estimation in general cases. The estimation of compression ratio in [44] depends on the following formula: $\hat{CR} = 1/(C_1(1-p_0)P_0 + (1-P_0))$, where C_1 is an ad-hoc tuning parameter which varies with different applications. As shown in Figure 3.9

(c), almost all data points are located on the line $y = x$ (red line in the figure), which means the estimated compression ratio $\hat{CR}$ under that formula could be very accurate in this case. This is due to the fact that this formula happens to form a linear function with compression ratio for the Nyx[74] application. However, that formula is sensitive to the tuning of the C_1 parameter, which may cause unexpected large compression quality estimation errors in other applications. For instance, the estimator's value does not form a linear relationship with the compression ratio for the Miranda[73] application (as shown in Figure 3.10 (a) and (b)), which leads to bad compression quality estimation in turn (see Figure 3.10 (c)). In comparison, our R_{rle} formula does not depend on the C_1. In fact, R_{rle} serves as a feature and we feed it into the ML model along with other features (including p_0 and P_0), and thus the model can automatically fine-tune the coefficients applied on those features, thus being able to keep an accurate estimation in most of cases (to be shown later).

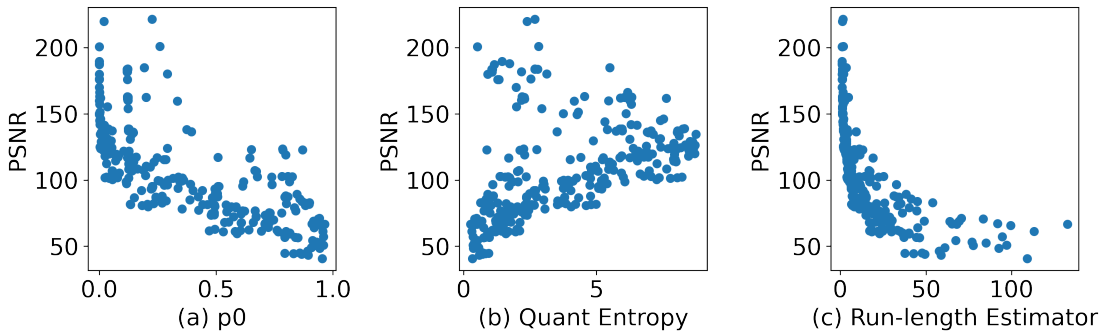

Figure 3.11: CESM dataset — PSNR versus compressor-level features

Our compressor-based features can also be used to predict the reconstructed data distortion. This is because these features are also closely correlated to the data distortion metrics such as PSNR, as verified in Figure 3.11. Based on the observations above, we use a decision tree model to perform the compression quality estimation.

3.3 Genome Sequence Compression

Genome sequence data format is completely different from the aforementioned datasets, and the traditional lossless compression or error bounded lossy compression methods do not contribute much to the file size reduction. However, the biological fact of DNA sequence similarity allows reference-based compression algorithms to significantly compress the files. I propose a novel reference-based genome sequence compression algorithm for FASTQ files. The contribution mainly lies in an improved sequence alignment approach, lossy quality score compression, and GlobaZip's remote orchestration capability for such compression.

3.3.1 Formalization

We provide a formal definition of our reference-based genome compression problem. A FASTQ file contains information about a set of reads, R_i, each defined by three components: a target sequence, a set of quality scores, and an identifier. In a raw FASTQ file, the target sequence and quality scores each take equally around 49% of the storage space, while the identifiers take the rest $\sim$ 2%. Each component can be compressed independently. The target sequence has to be lossless, but the order can be relaxed. We use the reference-based matching algorithm for the target sequence while using some traditional lossy/lossless algorithms for the quality scores and the identifiers.

We denote N as the target sequence length, M as the reference sequence length, and K as the compressed byte sequence length. For a single read R_i with target genome sequence X^N, then given Y^M as the reference information, we define an encoder $f(\cdot,\cdot)$ by mapping X^N to a byte sequence B^K with relationship $B^K = f(X^N, Y^M)$. One measure of a successful compressor is that it yields a B^K for which $K < N$. The decoder $g(\cdot,\cdot)$ should then recover B^K to X^N with a function $g(B^K, Y^M)$. Thus the encoder-decoder pair together recreate the original sequence:

$$X^N = g(f(X^N, Y^M), Y^M) \tag{3.11}$$

We preserve the losslessness for each read but relax the order restriction for a group of reads in one FASTQ file. Given K reads, each of length N, then after compression and decompression we have:

$$\begin{bmatrix} X^N_{i_1} \\ X^N_{i_2} \\ \vdots \\ X^N_{i_K} \end{bmatrix} = G\left(F\left(\begin{bmatrix} X^N_1 \\ X^N_2 \\ \vdots \\ X^N_K \end{bmatrix}, Y^M\right), Y^M\right) \tag{3.12}$$

where F, G are the corresponding functions of f, g that can handle a vector of X, and each X^N_k should be identical to one $X^N_{i_j}$, i.e., $\forall k \in [1, K], X^N_k, \exists i_j \in [1, K]$ such that $X^N_k = X^N_{i_j}$.

Moreover, there is a computation cost for $F(\cdot, \cdot)$ and $G(\cdot, \cdot)$. We use T_f and T_g to denote the time cost for the encoding and decoding process. Our algorithm should consider both the compression time cost and compression ratio, and therefore we define $s(CR,\ T)$ as a score function, where CR is the compression ratio and T is the (de)compression time.

The goal is to construct an encoding/decoding pair that maximizes $s(N/K, T_f + T_g)$ while preserving Equations 3.11 and 3.12.

For clarification, I will first describe the reference-based genome sequence compression problem briefly. Genome sequence matching is a process of comparing the genetic information (DNA sequences) of different organisms to identify similarities or differences. The ideal scenario is that each read is just a subsequence of the reference (the exact match), so we only need to mark the matching position for each read. However, the sequence can have modification, insertion, and deletion that complicate the matching process. The existing algorithms often cannot match sequences with insertions and deletions well, resulting in a lower compression ratio. I aim to improve this alignment process.

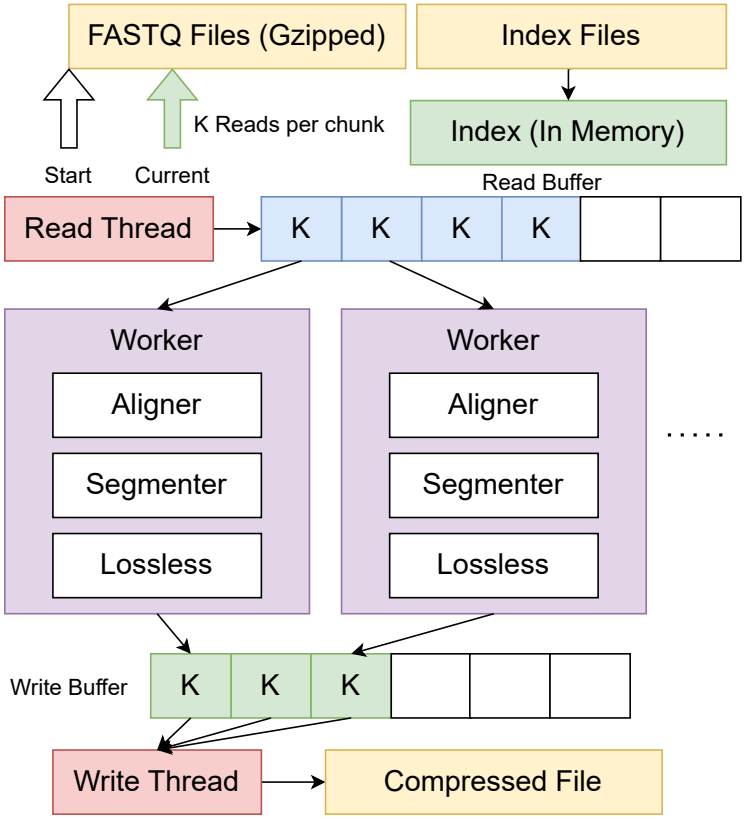

Figure 3.12: Genome sequence compression architecture: The read thread must be sequential, but workers can proceed in parallel. The read buffer and write buffer allow maximum parallelism for the whole pipeline.

3.3.2 Parallel Compression Architecture

We propose the compression architecture shown in Figure 3.12. The architecture employs one read thread, one write thread, and several worker threads to perform compression. These threads are synchronized by read and write buffers (in the program, the synchronization is performed by condition variables and mutexes). The read thread continues reading data from disks and stores them in the read buffer in memory. The worker will try to read a chunk from the read buffer. When the data is ready, one worker will retrieve it and remove the entry in the read buffer so that the read thread can read the next chunk. After the worker finishes compression, it puts the compressed data in the write buffer and tries to retrieve the next chunk from the read buffer. If the write buffer is full or the read buffer is

empty, the worker thread waits. This design allows FastqZip to compress extremely large FASTQ files without breaking memory limits and to achieve high degrees of parallelism.

Before everything, the index loader loads the index into memory and creates one if the index does not exist on the disk. During compression, the read thread reads the FASTQ files sequentially, decompresses them into strings, and puts them in the read buffer, where each chunk takes up one buffer space. Multiple workers contend to get one chunk's data from the read buffer and start the compression procedure. Each worker goes through the three main parts of our compression procedure, produces a compressed chunk, and puts the result into the write buffer. The write thread monitors the write buffer and constantly writes the data into the compressed file when there is data in the write buffer.

This architecture supports parallelism by allowing multiple workers to compress each chunk independently and write to the compressed file without waiting for any other workers. Moreover, we propose a compressed file structure that allows parallel reading which enhances the parallel decompression performance. However, there is only one read thread because gzip can only be decompressed sequentially from the start. When there are more workers, the read thread can soon be too slow to fill up the read buffer. We will evaluate the parallelism and scalability in Section 4.2.

3.3.3 Indexing Method

We employ a key-value map as an index for an efficient aligment process because naive long-string comparison is slow. We only need to build the index once for each reference sequence; once built, it can be loaded into memory rapidly during compression. As depicted in Figure 3.13, the short seed sequences function as keys, and seed positions in the reference sequence serve as values. To simplify the storage of the index file, we propose three concepts: (1) forward sequence, (2) range index, and (3) forward index. The *forward sequence* connects the reference sequences to form one long sequence, and replaces all non-ACGT bases with 'A'.

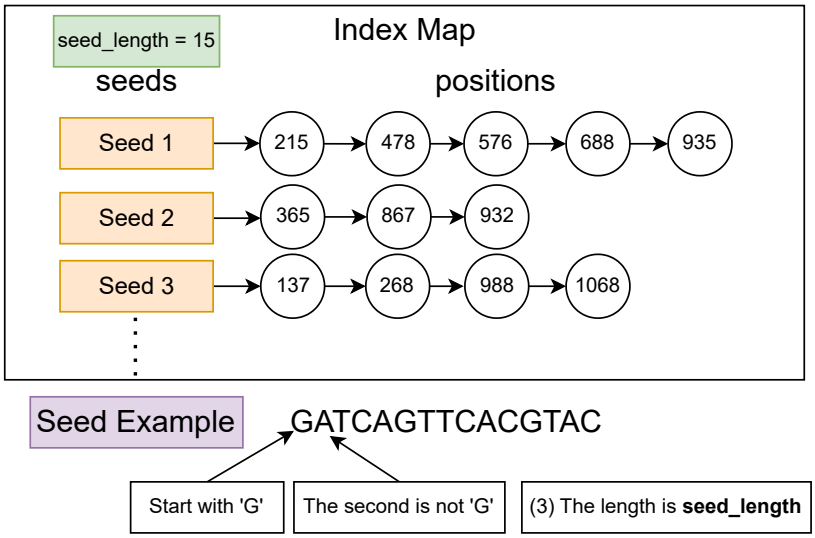

Figure 3.13: Index concept: we look for all valid seeds in the reference sequence and record their positions. There are multiple positions because the same seed may appear multiple times in different locations on the reference sequence.

The *range index* is a fixed-length array used to store the cumulative number of repetitions for seeds, as shown in Figure 3.14. The *forward index* stores the reference positions in seed-converted integer order. For example, in Figure 3.14, seed1 and seed2 appear once each in the reference sequence, at positions 59 and 98, respectively, and seed3 appears three times, at positions 180, 340, and 790.

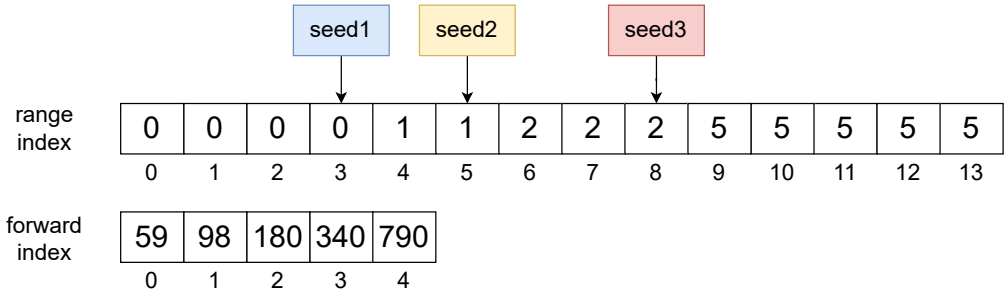

Figure 3.14: Index storage: The range index and forward index arrays together store the reference positions for all seeds. A seed can be uniquely mapped to an index i in the range index array. The value in range index$[i]$ is the starting index in the forward index array, and the value in range index$[i+1]$ is the index after the ending index in the forward index array.

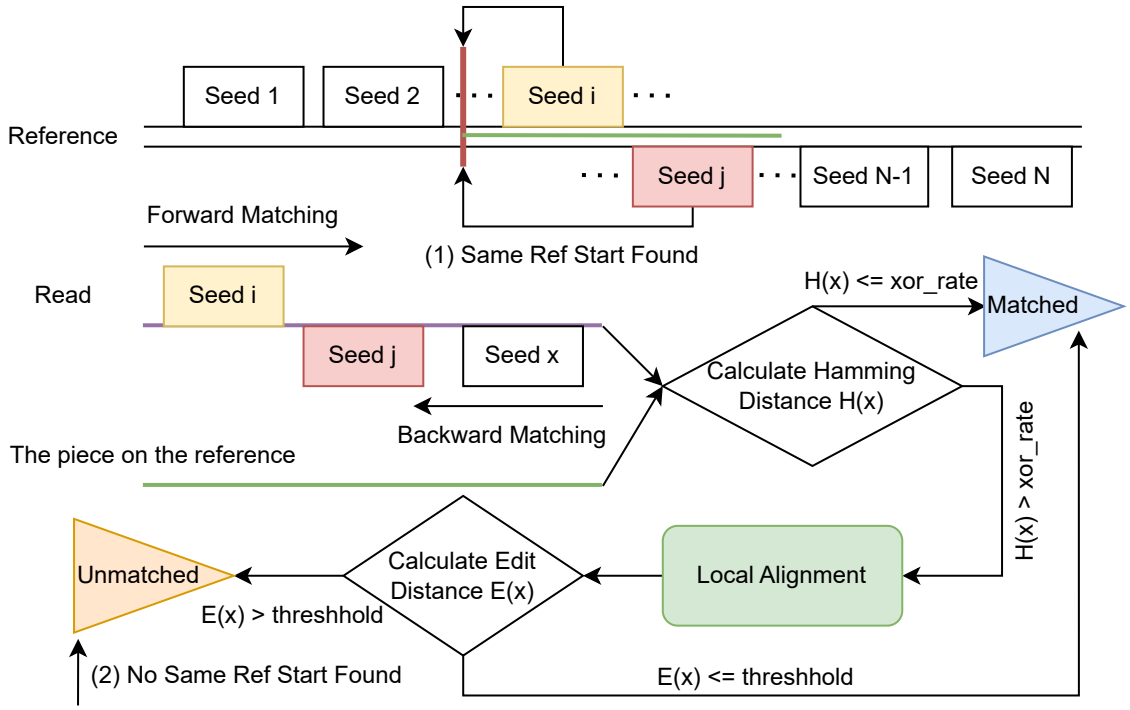

Figure 3.15: Alignment procedure: when multiple seeds exist on a single read, if a match exists, two seeds should match to the same starting position on the reference. If the candidate sequence on the reference has a very low Hamming Distance against the read, it is a match. If there are the same starting positions, but the Hamming Distance is large, we use our proposed local alignment to find a match with insertion or deletion.

3.3.4 Alignment Procedure

The core stage that improves the alignment capability in our algorithm is the alignment procedure, illustrated in Figure 3.15. For each read, we iterate through the seeds in both forward and backward directions and calculate the starting position of the read in the reference sequence. If two seeds appear to have the same starting position, likely, the read is indeed cut from the reference at that position. We consider this read a matchable candidate when the same reference start is found. We need also to verify whether the match is exact. To make this process fast, we calculate the Hamming Distance[9], an XOR between two sequences. When there is no difference or only a few base modifications, the Hamming Distance will be small, and we can consider that a match is found.

We improve sequence alignment capability by further conducting a local search to calculate the Edit Distance[103] when the same reference start is found but the Hamming Distance is large. A matchable sequence with a large Hamming Distance is usually caused by insertion or deletion. Prior works[48, 81, 29] consider such cases unmatchable sequences. We use the WFA-2 algorithm[70, 71] to obtain the Edit Distance and the alignment CIGAR[50] to reconstruct the original read with insertions or deletions.

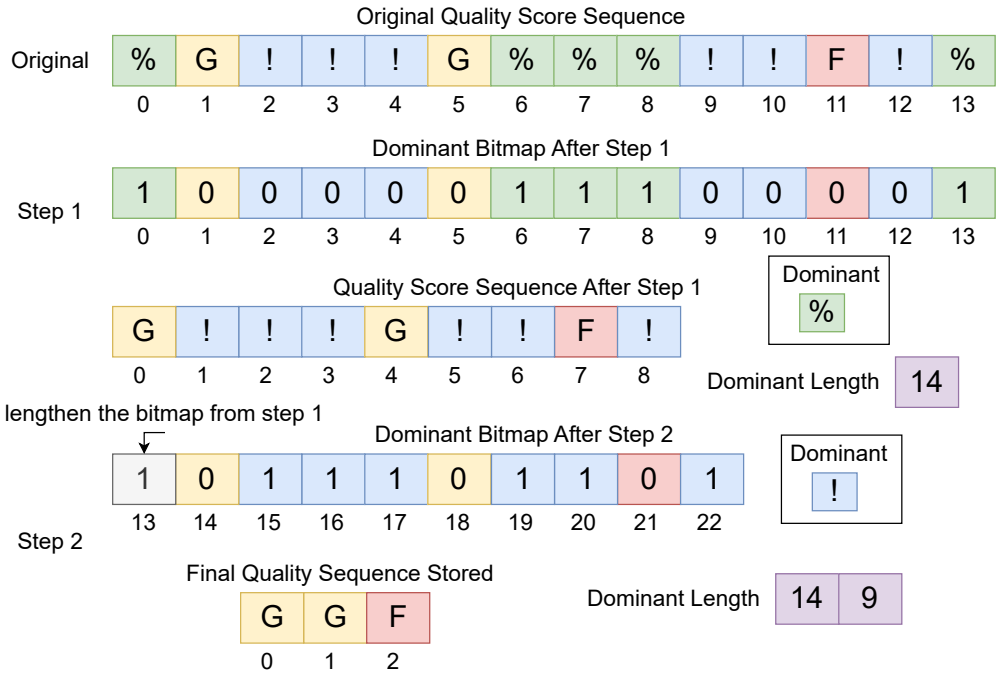

Figure 3.16: Dominant quality bitmap generation: when a quality score is dominant over others, we use 1 to mark them and remove them in the quality score sequence. We continue to find a dominant quality score in the remaining sequence and repeat the process. In the end, only a few non-dominant qualities will remain in the sequence. We store a bitmap, a dominant length array, a dominant quality array, and the remaining quality sequence.

3.3.5 Segmentation Process

The segmentation process connects short map results to form a single aggregated segment for better lossless compression. The sequence and quality will be handled separately.

In sequence segmentation, we can further reduce the reference position storage by using

the difference between positions (delta) when possible. For example, when two reads have quite close reference positions—the first read's reference position is 14340909 and the second read's position is 14340997—the delta is just 88, and therefore we can use fewer bits to store the delta compared to the primary position. Moreover, for paired FASTQ files, each Map Result stores two related reads $r1$ and $r2$, which usually form a reverse complement pair. We will switch the forward read's result to $r1$ so there is a higher chance for delta to be valid in the following segmentation process.

For quality segmentation, we use bin-quantization, dominant bitmaps, or Huffman coding to reduce their size. In each Map Result, the quality scores are a string of the read's length. According to the FASTQ format, there are 94 possible characters (from 0x21 to 0x7e) in total for each score. We propose a dominant bitmap solution as illustrated in Figure 3.16 to handle this situation. The idea is to use a bit instead of a byte to store each dominant quality and let the dominant quality be further compressed by a lossless compression algorithm such as the run-length algorithm. Moreover, it is possible to cluster the scores together to form fewer quality scores if the user allows a less fine-grained quality. We call this method bin-quantization. In reality, sequencing platforms such as Illumina[25] only provide fewer than 10 different quality scores. In this case, Huffman coding has excellent compression performance and is fast to complete.

3.3.6 Lossless Compression

Many fields in our segmentation process can be further compressed by general-purpose lossless compression algorithms such as Zstd[110] and Zpaq[69]. The lossless compression mainly deals with repeated patterns such as a long sequence of 1s or 0s in our bitmaps. Since these compressors compress a stream of bytes, we consider them as a black box to reduce field sizes. However, it is worth noting that these compressors have to store some additional header information during compression and thus do not necessarily reduce the sizes for certain fields.

Therefore, we need to be wise in selecting compressors or ignoring any compressors when dealing with different fields.

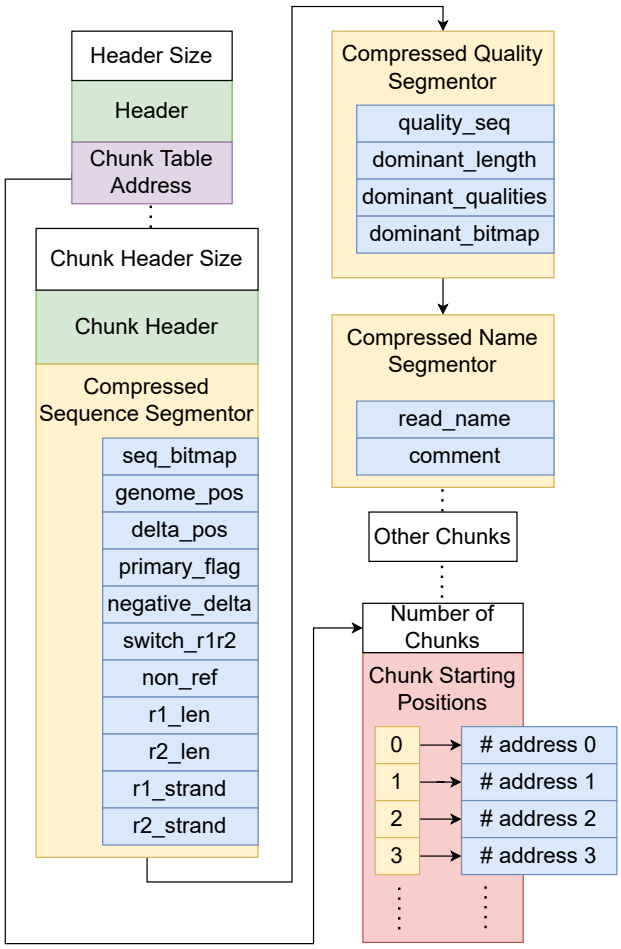

Figure 3.17: Compressed file structure: each chunk independently compresses its content, and provide a chunk total size to the main thread. The main thread will record each chunk's starting position, and save a table at the end of the file.

We design a compressor selection process on a test chunk to determine which compressor with which level best fits a certain field. The selection process takes both the compression time and compression ratio into consideration. It calculates a score based on each compressor's performance on the test chunk and selects the best compressor for each field. This selection process is pure overhead for the overall compression, and thus, we use a relatively small test chunk and fix the compressor selection for the actual compression process. Dur-

ing our evaluations, we found that, in general, Zstd[110] is most suitable for the sequence segmenter's field, and BSC[80] is most suitable for the quality segmenter's field when considering both compression time and ratio. Zpaq[69] is the best at compression ratio, but it is several times slower compared to other compressors.

We illustrate our final compressed file structure in Figure 3.17. The file header has a fixed length and will be written at the beginning of the compression. It stores metadata such as the FastqZip version, whether the read names are kept, whether Gzip[33] will be used in decompression, the sequence mode (single or paired), and so on. The chunk table address can only be known after all chunks have been compressed and written to disk. The writer thread will write the chunk table and then move the file pointer back to write the chunk table address. After the chunk table address is written, the whole compressed file is successfully stored on disk. Each chunk has its own header so that it can be independent of other chunks for better parallelism. The chunk header stores the number of elements and the compressed size for each field. The sequence segment has 11 fields, while the quality segment has four fields or just one field if Huffman coding is used to replace the dominant bitmap solution. The name segment stores the read names and comments if the read names are selected to be kept.

3.4 Layer-by-layer Compression

When there are many CPU cores in a machine, there needs to be a way to separate the data independently for each core to compress without mitigating the overall compression ratio. Moreover, the memory limit can be a huge problem to parallelize compression/decompression because a large amount of data needs to be loaded and the compressors also need certain additional memory and may require copying the raw data from time to time. Most existing compressors including ZFP, SZ3 need to load the entire dataset into memory for compression and are not directly applicable for extremely large single-file datasets. I explore ways to

cut the data with a layer-by-layer compression mechanism, allowing the data to be loaded consecutively while distributed to multiple CPU cores on multiple nodes.

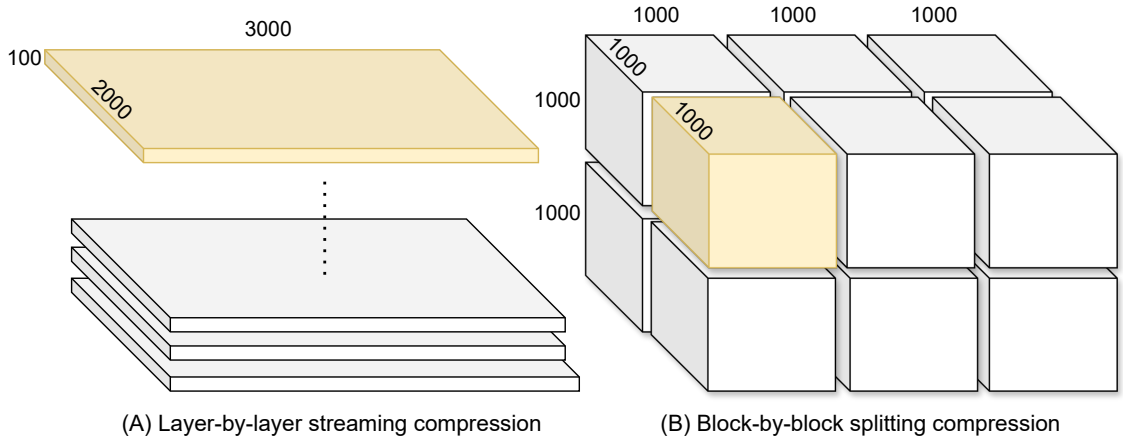

Figure 3.18: Techniques to split large files into smaller blocks or layers to resolve the out-of-memory problem.

The proposed layer-by-layer compression technology to compress exceedingly large files is shown in Figure 3.18 (A). While data can be split in other ways, as shown in Figure 3.18 (B), we favor layer-by-layer because (1) with other methods, the data points of a 'block' do not sit in continuous disk space and thus would require multiple seek operations that slow down I/O; and (2) the simplicity of layers makes it easy to parallelize the compression. A layer-by-layer streaming compression method transforms a 3D tensor into a sequence of 2D layers or slim 3D tensors. This technique can divide the files into smaller sections and then compress each section independently, avoiding out-of-memory errors when compressing large files.

3.4.1 MPI-based compression for HPC systems

To parallelize compression on supercomputers, we design an MPI-based architecture, as shown in Figure 3.19. For compression, as the original file is huge, we can utilize multiple read processes to read the tensor in parallel. The read processes send each layer's information

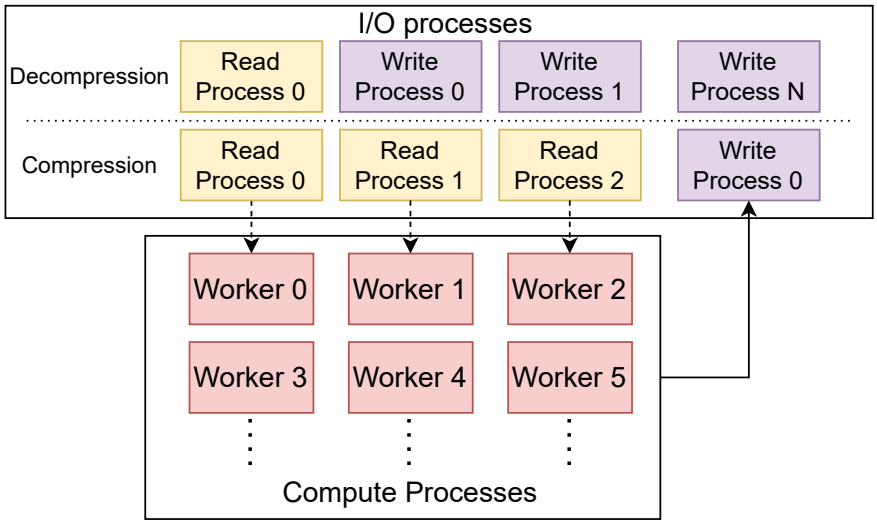

Figure 3.19: Parallel layer-by-layer compression architecture for multiple processors on single/multiple compute nodes.

to preassigned worker processes for compression. We only use one write process because the compression ratio of lossy floating-point tensor compression is usually quite high. For decompression, as the decompressed data are relatively large, we assign more processes to be writers. The advantage of this architecture allows processors on different nodes to compress a single file collectively. Moreover, parallel I/O is very suitable for our layer-by-layer compression method as each layer's offset can easily be calculated in advance.

3.4.2 Multi-threading compression for cloud servers

The MPI-program requires dedicated commands including 'mpirun' to call CPU cores on different compute nodes, but many users may only use one single computer or a cloud server that does not have multi-node computation. In this scenario, we provide a multi-threading implementation of the layer-by-layer algorithm. The advantage is that this can be easily integrated into other programs without the need of MPI-related commands. We also created a Python binding for this mechanism so that users can directly run the parallel compression in a Jupyter Notebook or other Python environment. The architecture for this multi-threading

implementation is very similar to Figure 3.12.

3.5 Compression and Transfer Orchestration

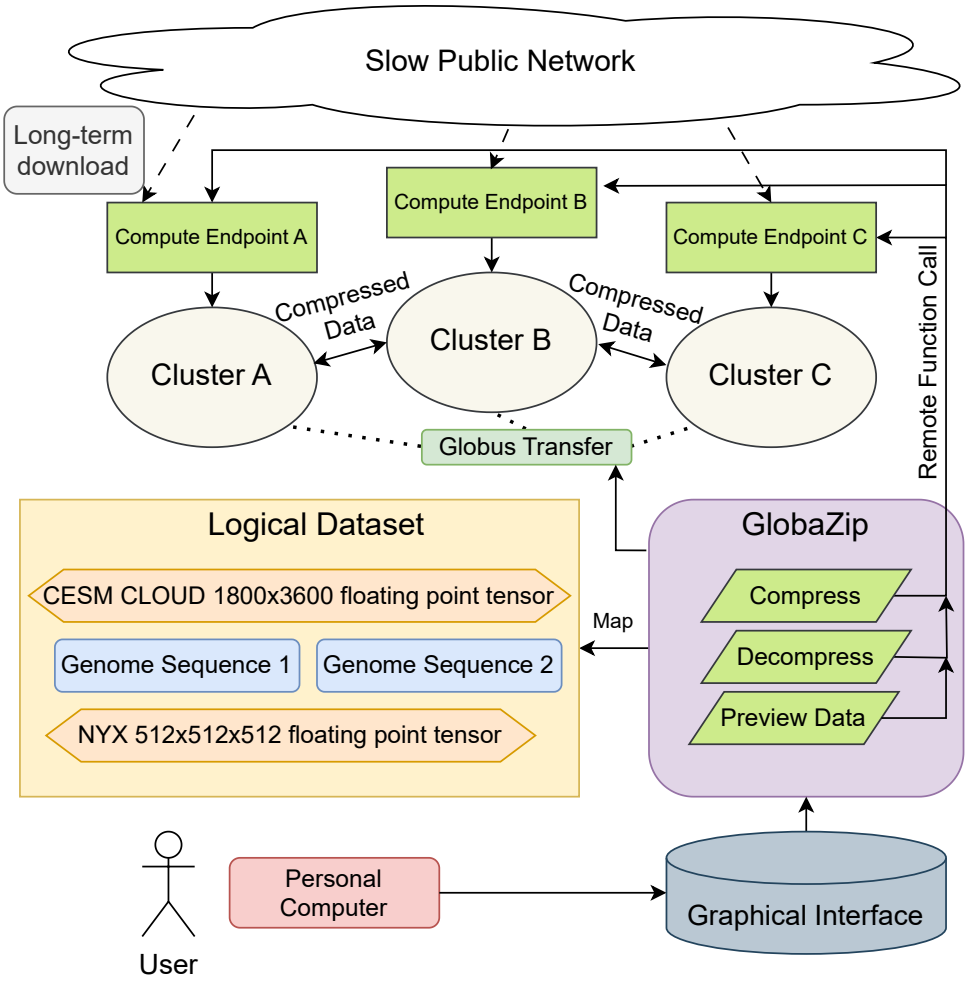

Figure 3.20: GlobaZip decouples the task execution from task manipulation and provides a universal data management interface for users to compress, transfer, analyze, and store various types of data.

To allow an easy control of compression and transfer for different types of data, I present an overview of the system GlobaZip in Figure 3.20, an orchestrator that interacts with the compressors, datasets, and transfer service on remote endpoints. The users interact with GlobaZip through a graphical interface, as shown in Figure 3.21. GlobaZip provides a fine-

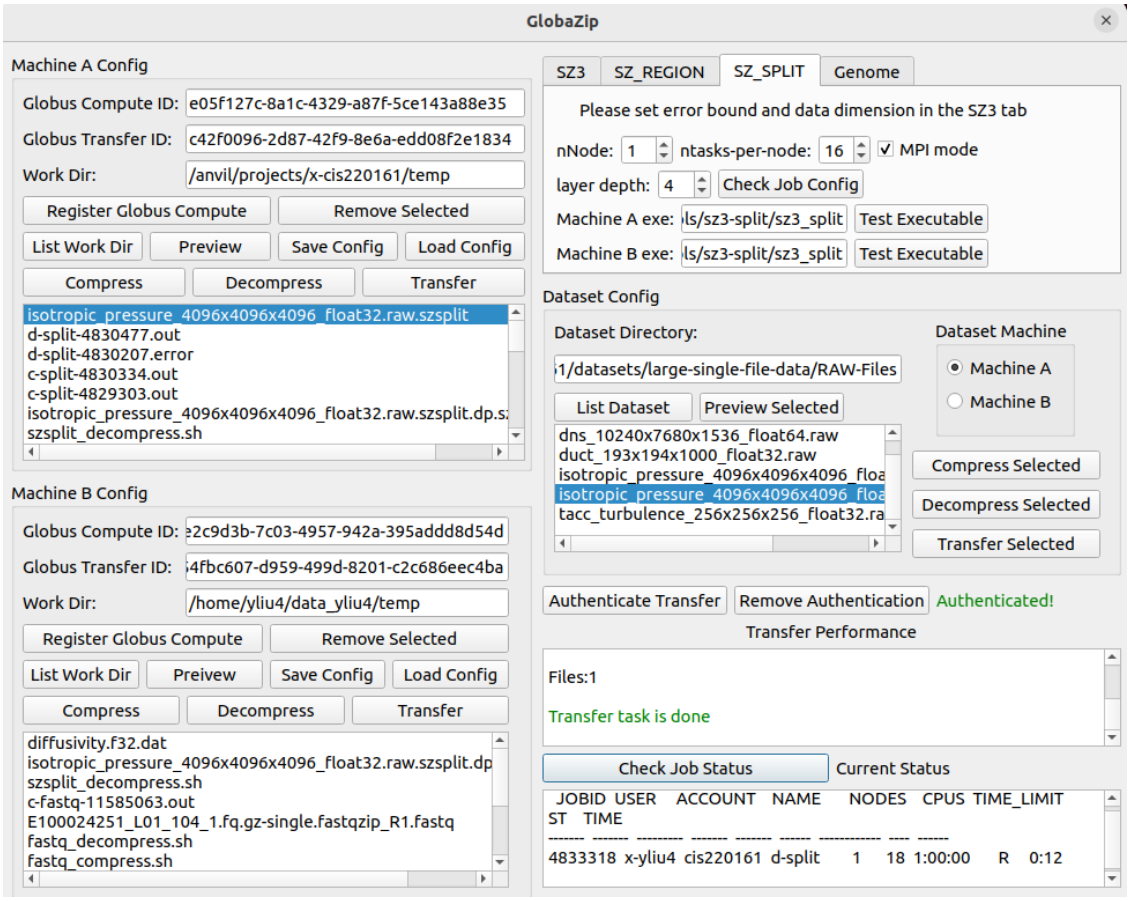

Figure 3.21: GlobaZip graphical user interface: users can control data transfer and compression with different algorithms between configured computing clusters. All the authentications are done when configuring the endpoints. Users no longer need to repeatedly authenticate when running jobs.

grained control over user-defined machines and allow users to choose different compressors against various datasets. GlobaZip can map the datasets from multiple clusters to a logical directory so that users do not need to worry about the exact physical location of their desired datasets. I use color maps to offer a data preview option for floating-point tensors, which allow users to set multiple error bounds according to visual results. Users can also easily develop plugins to add their data analysis program to the GlobaZip framework. With GlobaZip, data scientists no longer need to repeatedly log in to each cluster to perform data compression/analysis tasks and transfer the results back to their personal computers for visualization.

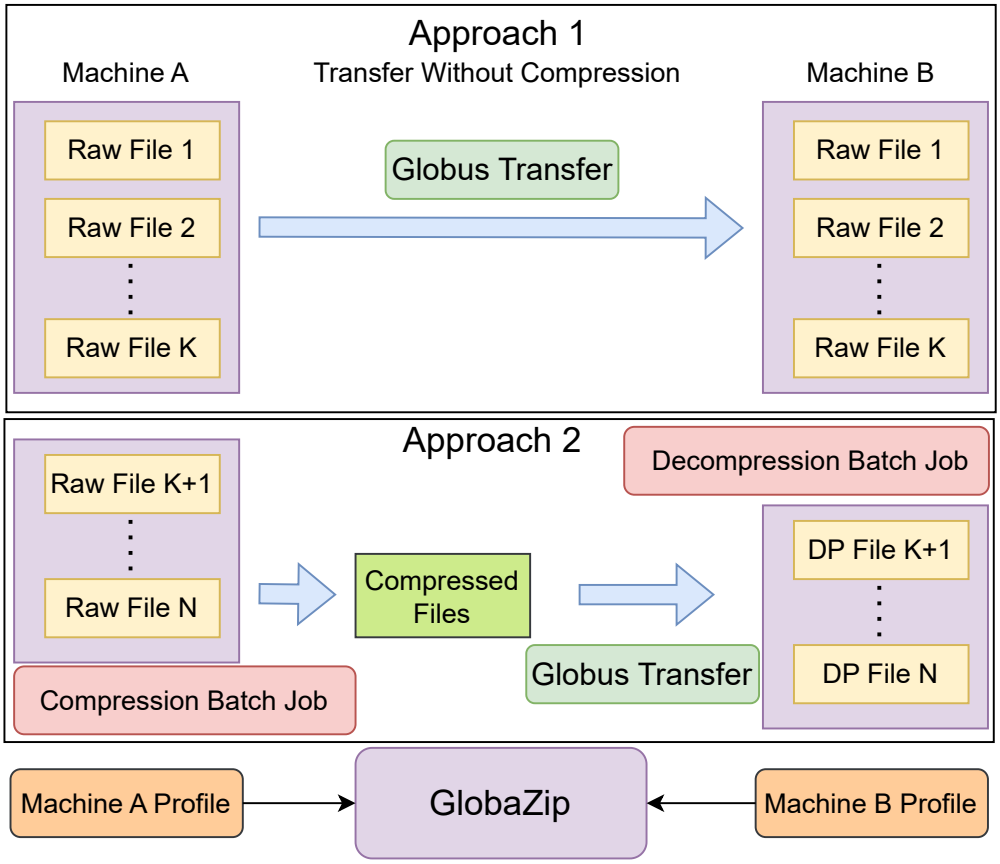

Figure 3.22: GlobaZip employs historical compute node wait times and compression/transfer time estimates in deciding whether to compress a transfer.

GlobaZip deploys long-term running endpoints on computing clusters, which gather information about datasets and compression performance. The information helps GlobaZip predict the future compression time and determine the busyness of each cluster. The estimated compression time helps GlobaZip decide the preallocated time for a batch job on shared systems. The benefit of this approach compared to the previous compression performance prediction work is that it does not have an additional cost when users perform a compression task. Moreover, the endpoint serves as a convenient stable downloader that can download datasets from the public network with a very slow transfer rate. The long-term running endpoint avoids the stable connection requirement to users' devices and thus can successfully download files for several days without encountering an SSH connection pipeline

broken error or an accidental network drop that often occurs when users try to download datasets through their laptop.

GlobaZip features a high adaptability to the execution environment as shown in Figure 3.22. GlobaZip refers to its historical data about cluster load and estimates the time to acquire a compute node. If the estimated wait time is over long, GlobaZip directly initiates a transfer task via Globus Transfer. If the estimated compression/decompression time plus the transfer time of compressed files can benefit the overall throughput, GlobaZip will do compression. If a cluster cannot run the desired software, GlobaZip will automatically perform a roundtrip approach: transfer the data to another cluster for compression and then transfer compressed data to the destination. We show in the following that due to high network bandwidths among supercomputers, the roundtrip approach can still reduce the total time for transferring data to cloud computing servers or personal computers which suffer slow networks. On supercomputers, data transfer is conducted on previously allocated Data Transfer Nodes (DTNs) and is managed through the Globus Transfer API. For personal computers and cloud computing servers, users can manually set up the endpoints and include it in the GlobaZip.

3.6 Optimization of Data Transfer with Error-bounded Lossy Compression

The compression performance prediction model described above provides a fast and automatic way to determine appropriate compressor settings. However, compression remains a computational expensive process, especially with large data. In GlobaZip, we utilize multiple cores/nodes to compress files in parallel.

Nonetheless, it is worth noting that there are two issues that may impede the "compress and transfer" performance. First, for large datasets the compression task may exceed the capacity available on DTNs or login nodes, and thus require provisioning of compute nodes

via batch scheduler. Such requests may not be scheduled immediately. Second, the number and size of files significantly influence the transfer speed because (1) each file transfer has an inevitable data handling cost in addition to data transfer time, and many small files may significantly lower the overall transfer throughput; (2) transfers with too few files cannot utilize the available concurrent transfer threads.

We describe our transfer performance optimization strategies in this section. To address the first issue, we need a strategy to transfer files when compute nodes are not immediately available. For the second issue, we need an efficient file grouping method to counter issues with many compressed small files.

3.6.1 Parallel Compression/Decompression

Our fundamental approach to reduce the transfer time is using compression methods to reduce the file sizes. However, each compression suffers a certain time cost, thus if we compress thousands of files sequentially, the total compression time may surpass the transfer time. We utilize parallel computing to significantly accelerate the compression process. We investigate the performance of different levels of parallelization: as shown in Figure 3.23 (left), the increase in the number of CPU cores significantly reduces the time needed to compress these datasets because they consist of many independent files. To address this issue, we let each core handle the compression of a set of files in parallel. The compression time cannot be further reduced when the number of cores reaches the number of files to be compressed because of the saturation of the parallelism.

Our experiments show that decompression performance does not increase monotonically with the number of CPU cores. For instance, decompressing the CESM [46] dataset on Cori takes 68.7s on four nodes but more than 5 minutes on 16 nodes. We conduct a more thorough test for parallel decompression on the Purdue Anvil machine, and the result is shown in Figure 3.23 (right). We see in this experiment that performance degrades with

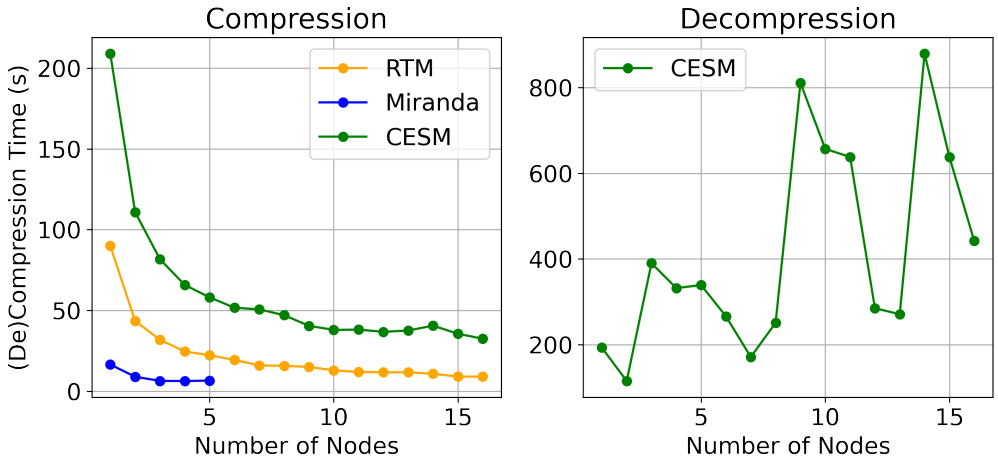

Figure 3.23: Parallel compression and decompression times vs. number of nodes, as measured on Purdue Anvil. Each node has 128 CPU cores.

more nodes. We believe this to be due to I/O contention on a shared file system. We can avoid the slow-down by tuning the number of cores to the parallel file system.

3.6.2 Optimization for Node Waiting Time

The uncertain wait time on compression tasks and transfer tasks may degrade transfer performance when involving compression. In most systems there are infrequently sufficient nodes available immediately to do the compression when users submit the data compression tasks. If the compression tasks are stuck too long in the scheduler queue, the overall transfer performance would be even slower than transferring without compression.

In order to counter the node waiting time, we run a sentinel program to monitor and schedule the transfer/compression task. As shown in Figure 3.24, when a user submits a transfer request (with lossy compression option turned on) which is not assigned compute nodes immediately, we start transferring the files in groups without compression. Once a file transfer is complete, we write their filenames in a meta file so that the compression scheduler knows which files no longer need compression. When the compute nodes are assigned, the sentinel program notifies the transfer tasks to stop and let the parallel compression scheduler

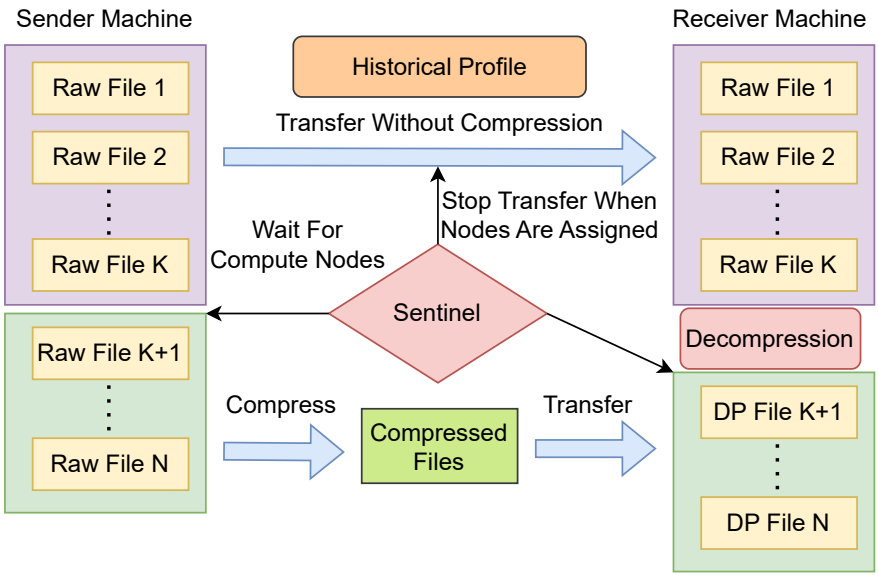

Figure 3.24: Transfer without compression during node waiting time: the monitoring program submit the compression task; while waiting for nodes, the transfer service is already transferring the data without compression.

take over the remaining files. In this way, the data transfer is not be suspended because of waiting for nodes, and the worst-case is that all data are transferred without compression (when the nodes are not assigned through the whole period). In production deployments, we anticipate that the Ocelot service could be deployed on dedicated cluster nodes (e.g., DTNs) with the approval of system administrator (similar to Globus service). In this case, wait time would be only dependent on other Ocelot transfers sharing those resources.

3.6.3 File Grouping for High Data Transfer Throughput

We propose a file grouping strategy to improve the data transfer throughput based on our observation that the number of files and file sizes may significantly affect the transfer speed (as shown in Table 3.4). Although the effective transfer speed fluctuates due to network and I/O contention, we generally see that the effective network speed decreases as the number of files increases, when transferring the same amount of data. This motivates us to optimize

the file transfer speed by grouping small files together.

Table 3.4: File Transfer Patterns between two supercomputers: Nersc Cori and Argonne Bebop

Total size	File size	# Files	Speed (MB/s)	Duration (s)
300GB	1M	300000	247.0	1235
300GB	10M	30000	921.1	325
300GB	100M	3000	1120.0	267
300GB	1000M	300	1060.0	281

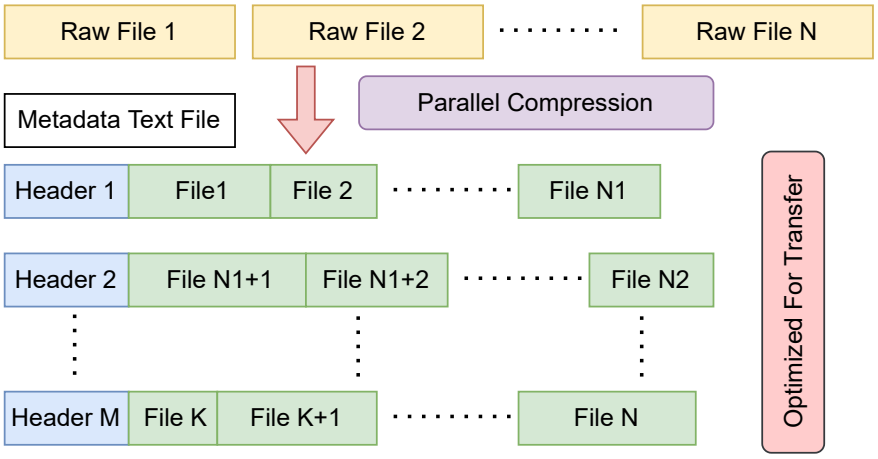

Figure 3.25: Parallel compression optimization by grouping small compressed files to achieve higher transfer speed.

Grouping small compressed files can increase a single file's size and reduce the number of files, and thus improve transfer speed. As shown in Figure 3.25, we compress files in parallel and group many compressed files to achieve a better size for transfer. We use MPI to communicate the compressed sizes among CPU cores to determine the file offset for each core to write. Each grouped file has a header and a body of connected compressed data. The header contains information about the number of compressed files in this group, the starting offset, and the size of each file. The metadata text file contains human-readable information about the number of files, grouping strategy, and the original filenames that are useful for decompression. The default strategy is to group files by the "world_size", i.e., the available number of cores for compression, because they run in parallel and can usually finish the

compression at a similar time. According to the profiling test and information provided by the administrator, we know in advance the preferred size for each file to achieve the fastest transfer speed. Thus, the compression scheduler can also determine the number of files to put in one group based on the file sizes.

CHAPTER 4

EVALUATION

In this chapter, we examine GlobaZip's performance in terms of data constraints perservation, prediction accuracy, scalability, compression speed, compression ratio, and compare multiple proposed algorithms with other state-of-the-art methods. The sections are adapted from the following papers: DRBSD-7[64], HiPC 2021[65], TPDS 2022[66], and ICDCS 2023[63].

4.1 Experimental Settings

We collect performance data on 2 ACCESS supercomputers (Purdue Anvil, Rockfish), 2 Alibaba ECS machines, and 1 Argonne supercomputer with specifications in Table 4.1. The network bandwidth for all the supercomputers are 100 Gbps, while for Alibaba ECS, the default is 1 MB/s, with a maximum configuration of 100 MB/s—highlighting a significant network disparity between supercomputers and cloud clusters.

Table 4.1: Machine Specifications: Cores and Memory indicate a single compute node's total.

Machine	**CPU**	**Cores**	**Memory**
Rockfish	Intel Xeon Gold Cascade Lake 6248R	64	192 GB
Purdue Anvil	AMD EPYC 7543 32-Core Processor	128	256 GB
Argonne Bebop	Intel Xeon E5-2695v4 & Phi 7230	36	128GB
Alibaba ecs.c7se.4xlarge	Intel Xeon Platinum 8369B	16	32 GB
Alibaba ecs.g7.32xlarge	Intel Xeon Platinum 8369B	128	512 GB

For floating-point tensor compression, we employ 6 moderately sized datasets, QMCPACK[76], ISABEL[41], RTM, Miranda [73], CESM [46], and Nyx [74], and two large datasets, Forced

Table 4.2: Floating-point Tensor datasets

Application	**Description**	**Dimensions**	Sizes
QMCPACK	electronic structure calculations of molecular, periodic 2D, and periodic 3D solid-state systems	33120×69×69 (float32)	150MB
ISABEL	temperature, speed, etc.	100×500×500 (float32)	95MB
RTM	seismic imaging in complicated areas	235×449×449 (float32)	180MB
Miranda	Hydrodynamics code for large turbulence simulations	256×384×384 (float32)	144MB
CESM	cloud, temperature, pressure in climate simulation.	1800×3600 (float32)	25MB
Nyx	density, temperature in cosmology simulation	512×512×512 (float32)	512MB
Turbulent Channel Flow	Pressure field of a direct numerical simulation of forced isotropic turbulence.	4096×4096×4096 (float32)	256GB
Forced Isotropic Turbulence	A pressure field of a direct numerical simulation of fully developed flow	10240×7680×1536 (float64)	900GB

Isotropic Turbulence and Turbulent Channel Flow [49]: see Table 4.2. The two quantitative metrics we use to measure the data quality are RMSE and PSNR, and we will briefly introduce their meaning below. The root-mean-square error (RMSE) is a frequently used measure of the differences between values. We calculate the mean error between the decompressed data values and the original values to understand how much error the compression algorithm brings into the data. The term peak signal-to-noise ratio (PSNR) is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise. PSNR will not be severely affected by the data ranges, and we can have a universal understanding of how good the data is.

Table 4.3: Genome Sequence Datasets

Dataset	Platform	Symbol	Size
E100024251_L01_104[37]	DNBSEQ-T7	A	18+20 GB
CL100076243_L01[1]	BGISEQ-500	B	54+55 GB
E100030471QC960_L01[2]	DNBSEQ-T7	C	28+27 GB

For genome sequence compression, we evaluate the compressors on three datasets sequenced on different platforms and with various lengths: see Table 4.3 — we refer to these datasets as A, B, C in the following text.

4.2 Evaluation of Genome Sequence Compression

We compare the performance of our genome sequence compression algorithm against that of three modern genome sequence compression algorithms: Spring[12], GTZ[95], and genozip[48]. We measure compression ratio (CR), (de)compression CPU time, and (de)compression wall time. As shown in Figure 4.1, Spring compresses more slowly and achieves a lower compression ratio than the other methods, and thus we do not consider it further. GTZ[95] is fast but has a lower compression ratio. It has a bug in paired FASTQ file compression that we could not resolve and for which we could not get timely support. We conclude that the best existing genome compression algorithm is Genozip[48] in terms of project completeness, ease of use, compression ratio, and compression speed. We present a more detailed comparison with Genozip in Table 4.4. Note that the compression ratio is calculated based on the gzipped original file sizes.

Our findings indicate that our algorithm has demonstrated advantages in both compression ratio and compression time across the three paired sequence datasets. Our algorithm allocates additional time to sequence alignment to achieve a superior compression ratio. However, the overall (de)compression time is reduced due to our multi-threading architecture disregarding the order of reads and string identifiers. In contrast, Genozip ensures complete lossless compression. We assert the validity of our algorithm for the following

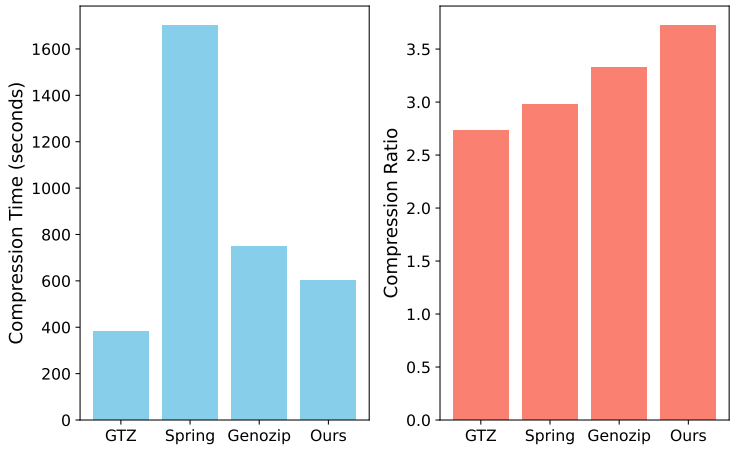

Figure 4.1: Compression time and ratio comparison on the first file of E100024251_L01_104. Each algorithm uses 16 threads on the **ecs.c7se.4xlarge** machine.

Table 4.4: Our algorithm vs. Genozip. CR: compression ratio; CPTime: CPU time in compression; DPTime: CPU time in decompression.

Compressor	**Dataset**	**CR**	**CPTime**	**DPTime**
Our Algorithm	A	**3.37**	**151m48s**	**94m20s**
	B	**2.44**	**417m17s**	**251m15s**
	C	**2.54**	**522m16s**	**245m31s**
Genozip	A	3.14	160m28s	100m14s
	B	2.33	572m5s	303m54s
	C	2.45	526m43s	281m14s

facts: in FASTQ files, each read is self-contained, and the string identifier typically pertains only to sequencing machine specifications, which are inconsequential for downstream analysis.

We analyze our algorithm's memory and CPU utilization. In Figure 4.2, memory usage ranges between 50% and 60% (∼19 GB) when all CPU resources are utilized. If memory is limited, say to 16 GB, reducing the number of threads or decreasing the read number can reduce the memory demand. On the other hand, our algorithm is capable of fully utilizing computing resources with an appropriate number of threads. Figure 4.2 (B) shows that by setting the thread number to 16 (the number of available CPUs), each worker can take up one CPU core and reach over 90% of CPU utilization. The slight oversubscription shown in

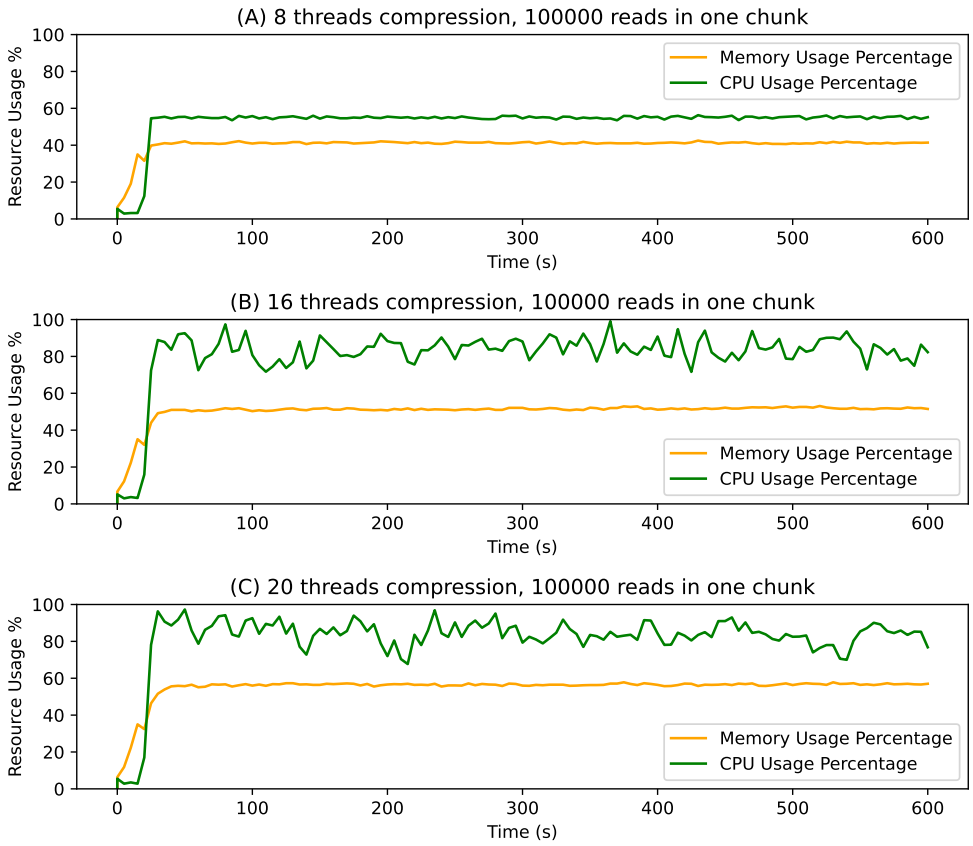

Figure 4.2: Memory and CPU usage in compression. Tests are run on a **ecs.c7se.4xlarge** cloud server with 16 CPUs and 32 GB memory; the dataset is E100024251_L01_104.

Figure 4.2 (C) does not increase the CPU utilization more while having some troughs that drop to 65% of utilization. We can safely conclude that setting the thread number to the number of CPU cores available can utilize the computing resources well enough.

Lastly, we analyze the scalability of our algorithm by recording compression wall time as we increase the number of threads. As shown in Figure 4.3, our algorithm scales well when there are fewer than 16 threads. The I/O wait computation time decreases to 0 when there are 20 threads, meaning the I/O has been too slow to provide sufficient data for so many threads to consume. If the I/O is faster than the computation, the read buffer will build up to full and the I/O has to wait for the computation to finish to continue reading data. The total compression time converges with fastqIO read time when enough threads

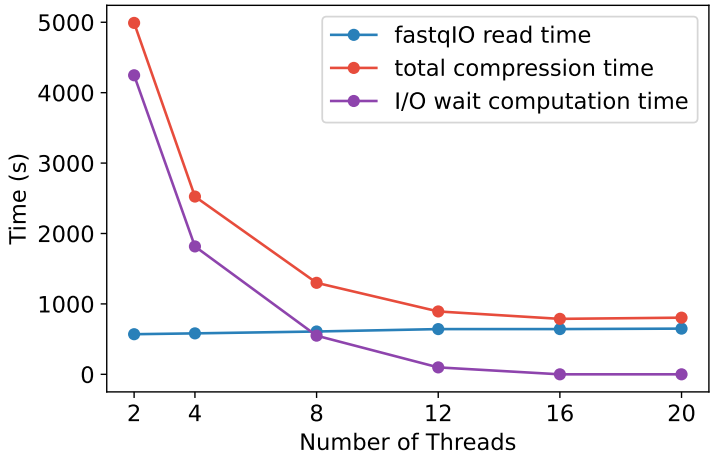

Figure 4.3: Scalability evaluation of our algorithm: The evaluation is performed on **ecs.g7.32xlarge** with sufficient CPUs and memory.

are provided. As the genome sequence data is currently distributed by gzip or plain text format, there is very little room to optimize for parallel I/O. However, we will show that our method to compress floating-point tensors can scale up further with faster parallel I/O on supercomputers in the next section.

4.3 Evaluation of Layer-by-Layer Parallel Compression Techniques

We first evaluate the relationship between layer depth and compression ratio for our proposed layer-by-layer compression algorithm on the four 3D floating-point tensor applications. As Figure 4.4 shows, the compression ratio increases when the layer gets thicker for most datasets. This is because the 3D tensor gives the compressor more information to predict nearby data values. The current SZ3 3D interpolation method can reach a higher compression ratio for 3D data. However, we also notice that after the thickness reaches 32 for Miranda and Nyx, the compression ratio does not increase clearly. That is, when applying such a thin layer, we can already reach a good compression ratio with a small amount of

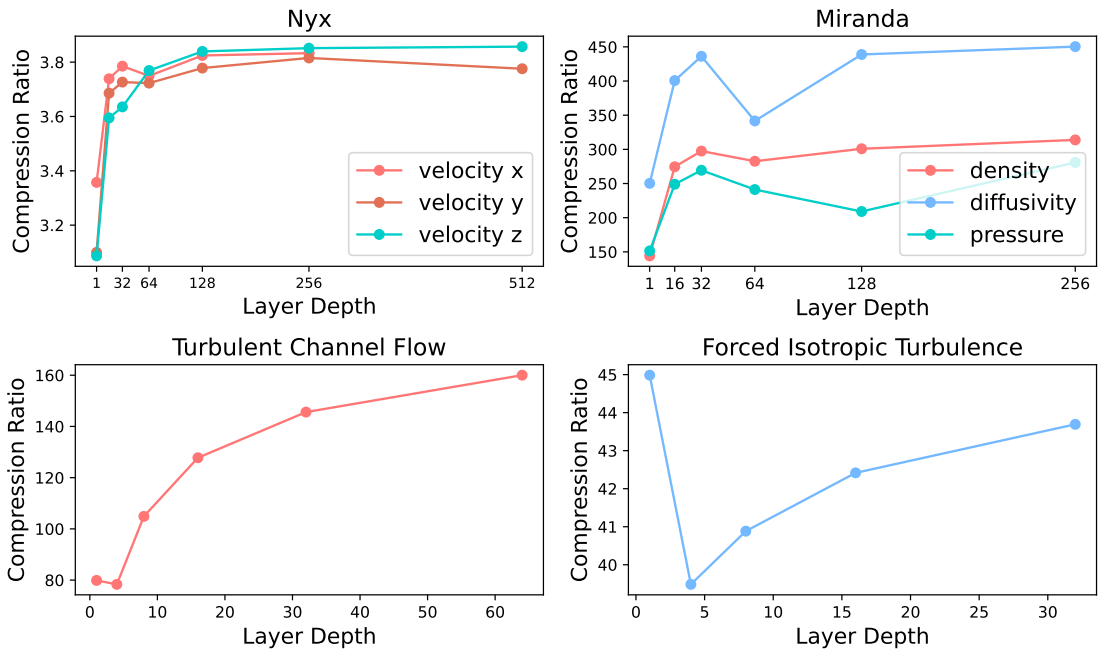

Figure 4.4: Compression rate vs. Layer Depth: Multiple fields in Nyx, Miranda and two large tensors Turbulent Channel Flow and Forced Isotropic Turbulence.

memory or a high level of parallelization. The Forced Isotropic Turbulence result shows that for certain huge datasets, the 2D layer already contains many data points for prediction, which might even outperform 3D compression in terms of compression ratio.

We then evaluate the compressed data quality with point-wise error distribution and visualization. The experiment result shows that our layer-by-layer compression method has comparable or even superior compression quality compared to the traditional all-together method. The pointwise error distribution changes when using layer-by-layer compression: see Figure 4.5. This approach appears to have a more concentrated error for both Miranda and Nyx datasets, although it has a slightly worse compression ratio due to the overhead of describing each layer. We can also verify the compression quality via the visualization method provided by GlobaZip in Figure 4.6, where there is no obvious difference between the original data and our compressed data.

Next, we evaluate the scalability of our multi-threaded layer-by-layer compression algo-

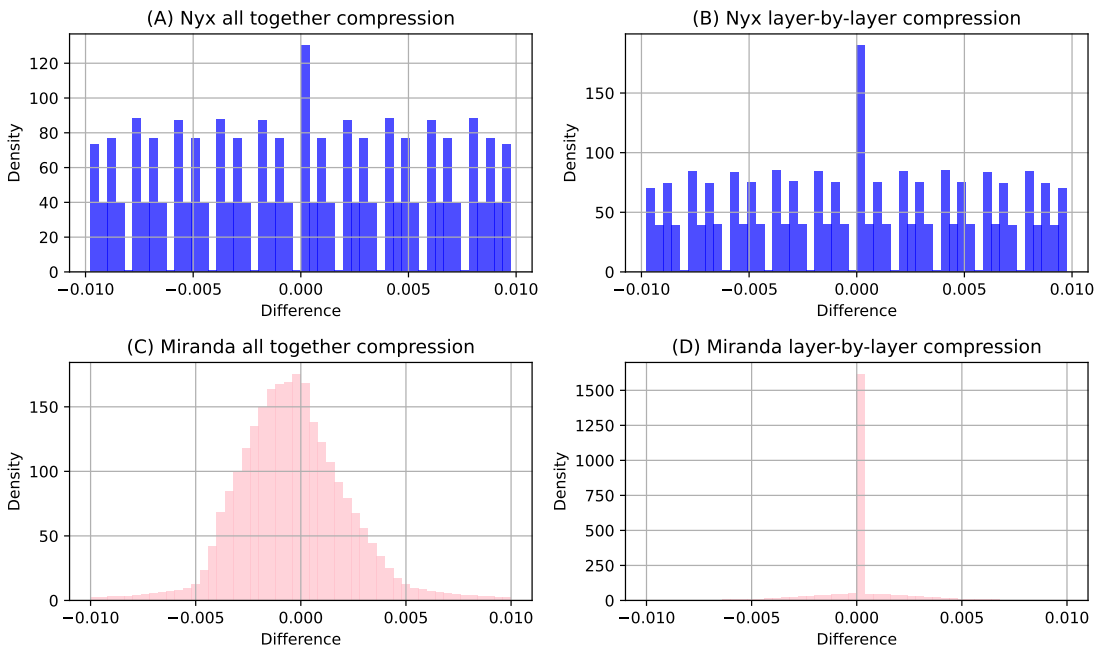

Figure 4.5: Nyx temperature and Miranda density pointwise error distribution. The error bound is set to 0.01 for all four configurations. The layer depth is set to 32 for layer-by-layer compression. The compression ratios are (A) 2.17 (B) 2.07 (C) 313, (D) 297.

rithm. Figure 4.7 shows that our algorithm can benefit from multiple CPU cores to compress a huge file in a much shorter time. The total compression wall time ceases to decrease after reaching 16 threads because the I/O has become the bottleneck. The program spends 160 seconds compressing 270GB of data, while the total amount of read time is 100s, the overhead of multi-thread is quite minimal. To further improve the performance, parallel I/O is needed as the read speed has become a bottleneck. Also, we note that the memory consumption is higher in decompression with more threads while the memory consumption is higher in compression with fewer than 16 threads. This is because the write I/O becomes a bottleneck in decompression and many worker threads hold the decompressed data to be written. This can cause an out-of-memory error with a mismatched I/O speed and number of threads.

We further improve the scalability by utilizing MPI programming with parallel I/O support and multi-node coordination. To avoid the out-of-memory problem for Forced Isotropic

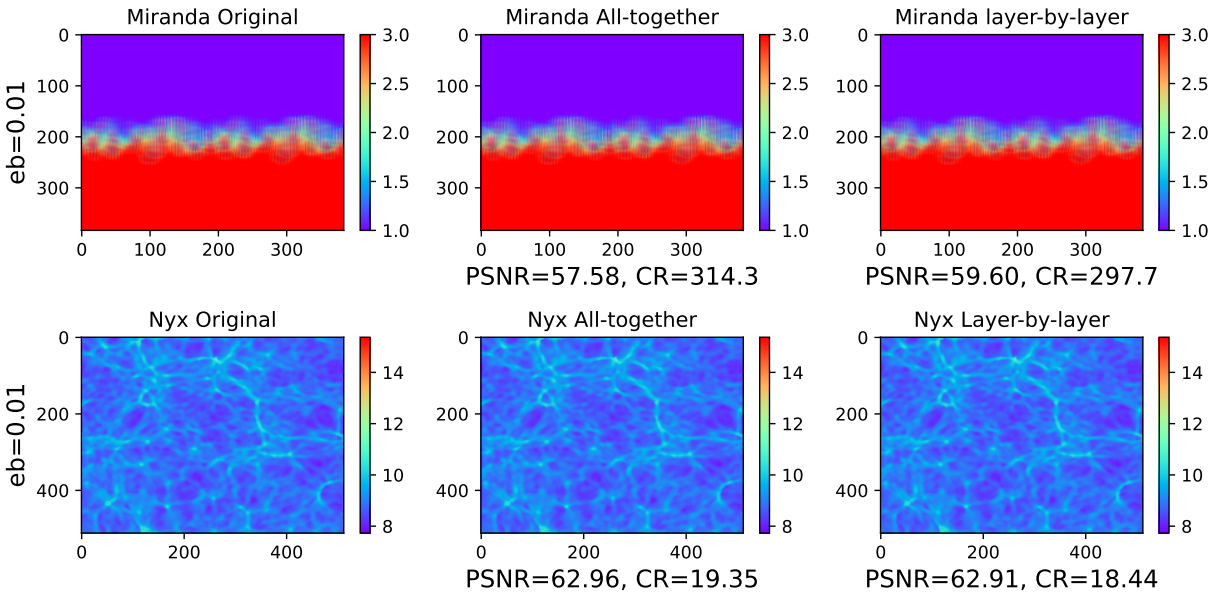

Figure 4.6: Miranda density and Nyx temperature visualization results. The compressed data with either method looks identical to the original data.

Turbulence and optimize (de)compression time, we limit the tasks-per-node parameter to 32 but increase the number of nodes to get more CPU cores, e.g. we used 8 nodes to get 256 processors. Also, we keep the ratio of the number of I/O processes and the number of worker processors 1:8 to reach near-optimal performance. As Figure 4.8 shows, the (de)compression time continues to decrease after using more CPU cores. We also notice that this approach's total time is much lower than the multi-threading method even with the same number of threads/processes. It is largely because we use a more proper ratio of I/O processes and worker processes. In the multi-threading model, when the worker threads are more than the read thread's ability to fill up the read queue, workers may contend for locks with a higher overhead time. Most existing error-bounded compressors have no such capability to parallelize to this scale. Our approach extends float-point tensor compression to non-uniform memory access systems.

Another advantage of our method is that it demands minimal memory for compression,

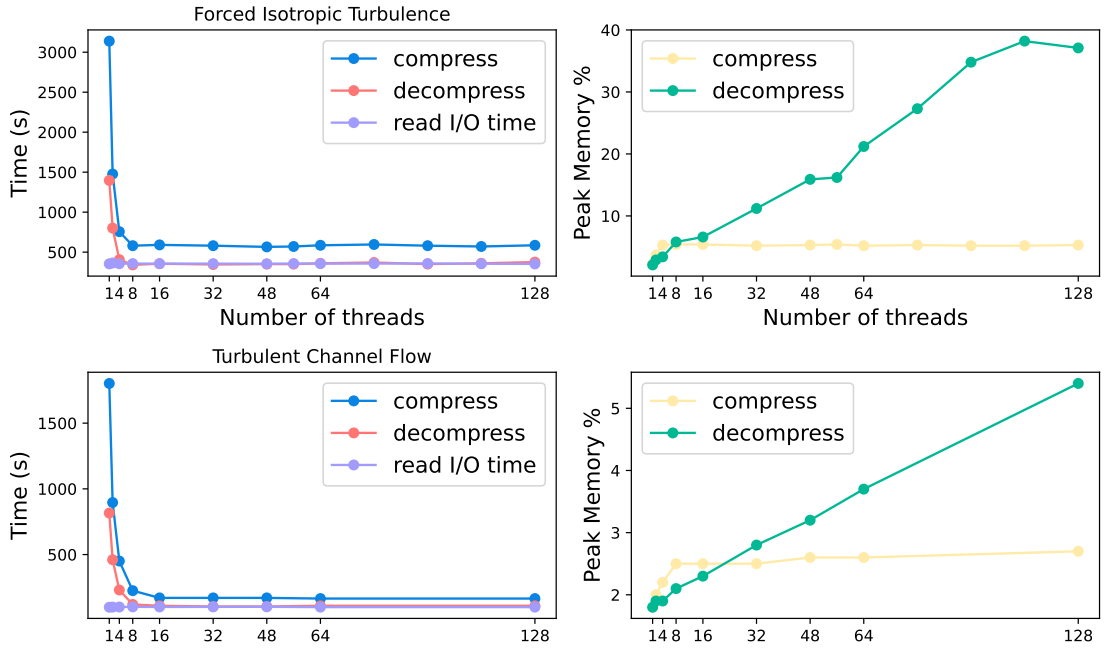

Figure 4.7: Compression ratio/time vs. number of threads for the two large datasets. The layer depth is set to 4.

particularly with thin layers. As shown in Figure 4.9, the layer depth determines the peak memory consumption. For a thin layer with depth 1, the program only needs 1%, 5 GB memory to compress a file of over 900GB. Other compressors like SZ3, SZ, ZFP, and MGARD face challenges with large tensors, primarily due to memory limitations. On the other hand, our program can run on multiple nodes to utilize memory on multiple nodes with thicker layers to obtain a better compression ratio.

4.4 Evaluation of Compression Performance Prediction

In this section, we evaluate the compression performance prediction on compression ratio, compression time, and PSNR for data quality. We focus on SZ2 [53], SZ3 [57] and their variants because our compression quality prediction method is based on the prediction-based compression model.

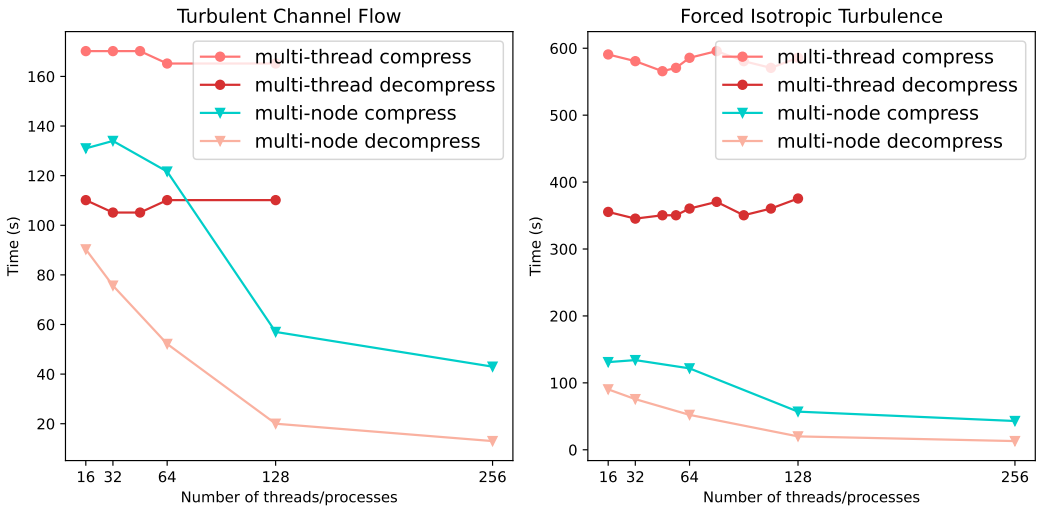

Figure 4.8: Compression performance comparison between single-node multi-threaded and multi-node multi-process methods. The multi-thread line stops at 128 threads because the memory is insufficient with more threads on a single node and the program would exit with an out-of-memory error.

4.4.1 Estimation of Compression Time and Ratio

To make an estimation of compression time and ratio, we apply a decision tree regressor model on 11 features described earlier, and train on 30% of files from each of the applications in Table 4.2 (the remaining 70% serves as testing data). We set 11 different error bounds from 1e-6 to 1e-1 to compress the data and collect the features for training.

The distribution of the difference between the predicted values and real values is shown in Figure 4.10. The green bounding box shows the 80% confidence interval, meaning 80% of prediction error falls into the green box. Thinner box means higher prediction accuracy. Figure 4.10 indicates our prediction method performs very well, as the differences between predicted and actual values are very close to 0.

The prediction has a negligible overhead (around 1.7%) compared with the total compression time when we sample 1% of data (using 1 data point every 100 data points). As shown in Figure 4.11 (A), the sampling helps reduce the overhead time from more than 70% to less than 5%. The extracted compressor-based features p_0 and P_0 are different from the

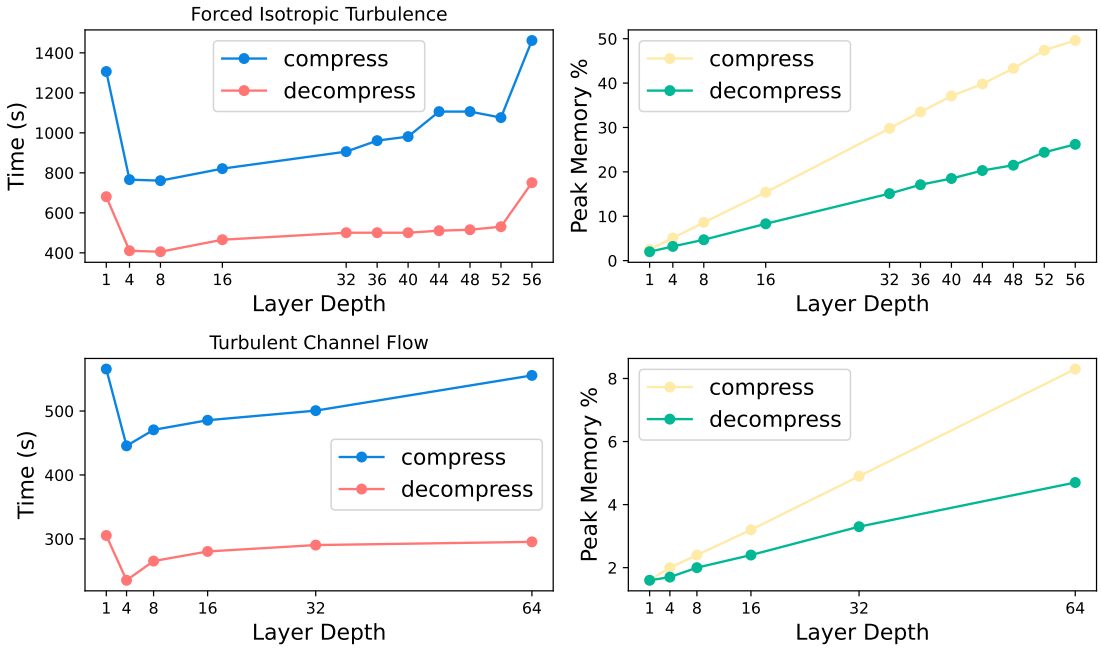

Figure 4.9: Compression time and memory consumption of 4-thread layer-by-layer compression on Turbulent Channel Flow dataset. Experiment on Purdue Anvil **shared** partition with 4 tasks per node and 512 GB memory.

actual percentage of the zero quantization code because we run the Lorenzo prediction with the real data values instead of the reconstructed data values.

Figure 4.12 shows a high correlation between compression time and the compressor-level features. In fact, the datasets' compression times are similar with each other as long as they have the same dimensions (usually because they belong to the same application) as shown in Figure 4.11 (B). This pattern helps us estimate the overall compression time accurately in parallel compression: the rough estimation would be the number of datasets divided by the number of cores then multiplied by the average compression time per one dataset.

Table 4.5 shows the prediction results for our datasets. We can observe from the values that the compression time is gathered into groups related to the application to which they belong. Moreover, we see that our model can always precisely predict the compression ratio and time at different error-bound settings. This is because the distribution of the quantization code changes according to error bounds, and our model captures this information with

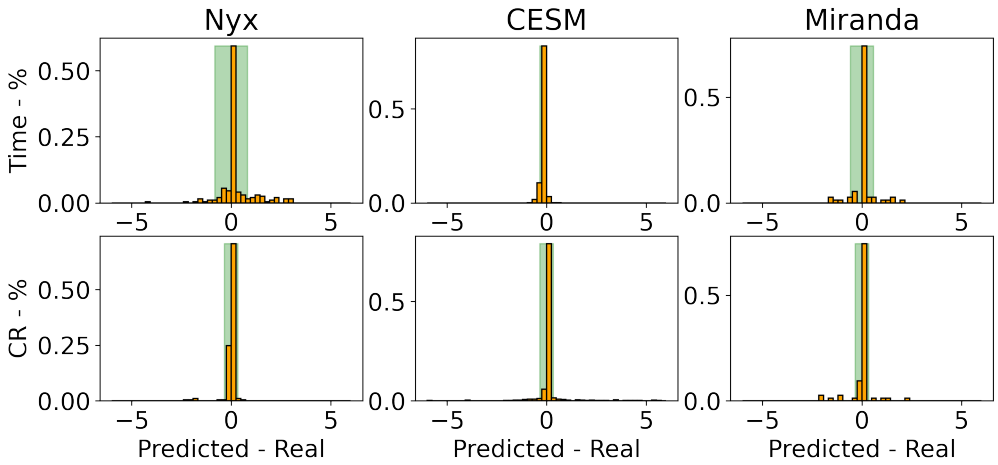

Figure 4.10: Nyx/CESM/Miranda application compression time and ratio prediction error distribution (measured on Bebop KNL partition): the X-axis is the difference between the predicted value and the real value, the Y-axis is the percentage for each small range of difference values.

p_0, P_0 and the quantization entropy effectively.

4.4.2 Estimation of Data Quality via PSNR

We use 50% of gathered data for training, and perform the compression quality prediction test for the remaining 50% of data. Table 4.6 shows the PSNR based on 10 data files randomly selected in the CESM application, where the root mean squared error of the PSNR prediction is 13.05. Table 4.7 shows a similar prediction result for the ISABEL application, and the corresponding root mean squared error of PSNR is 14.23. Unlike the prediction of compression ratio/time which is fairly accurate, the prediction of PSNR is good in most cases yet still suffers relatively high errors occasionally on a few datasets. We plan to improve it in our future work.

We explain the key reasons why the PSNR is predictable as follows. On one hand, if the quantization bins often gather around zero (especially when a relatively large error bound is used), the predicted values are likely unable to be corrected by quantization bins, leading to relatively low PSNR. On the other hand, if the zero quantization bin takes a tiny percentage,

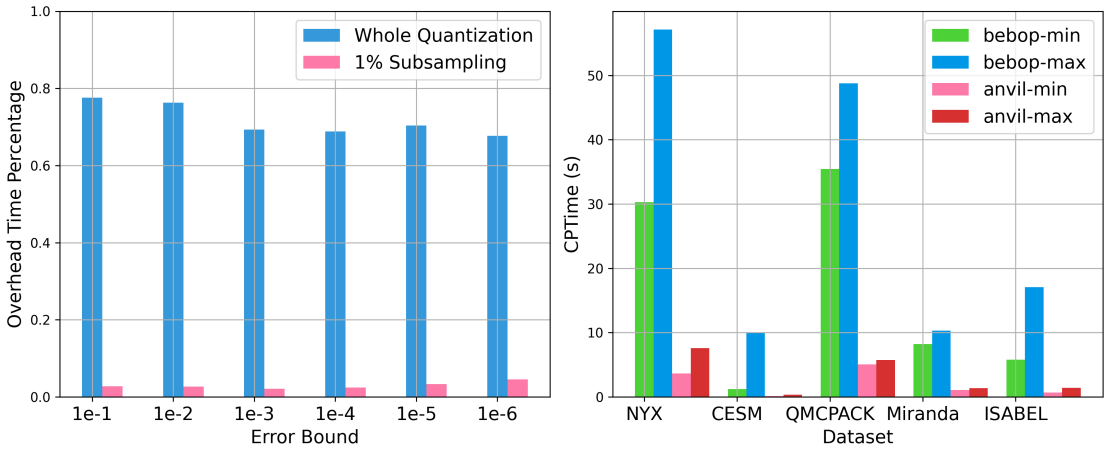

Figure 4.11: (A) Overhead time analysis on Nyx application; (B) Compression time range on Bebop and Anvil machines for multiple applications.

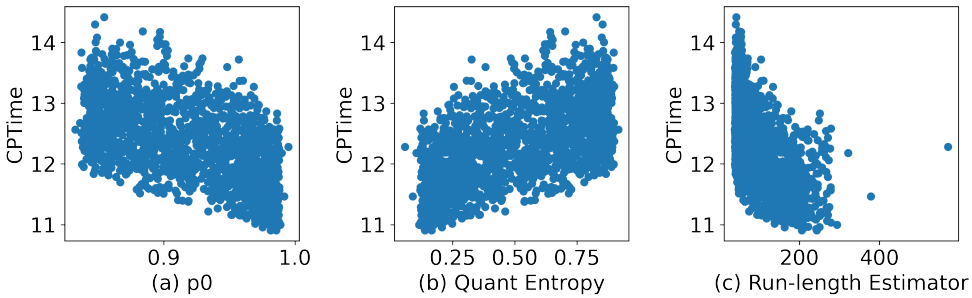

Figure 4.12: RTM application compression time versus compressor-level features

this means the quantization bins are likely very small because of the small error bounds used. In this situation, many data points would be corrected by the quantization bins or stored as they are based on the SZ compression model, thus leading to relatively high PSNR.

We explain why the prediction of PSNR may not be as precise as the compression ratio's prediction as follows. In fact, when p_0 and the quantization entropy are in the middle, most data points can still be the quantization-based reconstructed data, and it is unclear how far away these data points are from the original data values. They can be either an error bound away or quite close, therefore it is unclear how they will contribute to the final PSNR based on the selected features.

With the settings shown in Table 4.6, we visualize the original and compressed data of

Table 4.5: Compression Time and Ratio Prediction Examples: EB denotes error bound, CR denotes compression ration, CPTime denotes compression time. P-CR and P-CPTime denote predicted compression ratio and compression time, respectively. All time-related information is measured on Bebop machine in KNL partition.

Dataset	**EB**	**P-CR**	**CR**	**P-CPTime**	**CPTime**
Nyx Baryon Density	1e-6	1.19	1.18	35.9	35.6
	1e-4	3.15	3.10	32.3	33.3
	1e-2	10.40	10.20	30.3	30.3
CESM LHFLX	1e-6	1.139	1.135	1.459	1.456
	1e-3	2.56	2.49	1.97	1.59
	1e-2	5.25	4.43	1.55	1.50
CESM SNOWHICE	1e-6	5.36	6.97	1.61	1.85
	1e-4	21.0	21.9	1.55	1.58
	1e-3	48.0	52.8	1.40	1.48
RTM-1982	1e-6	4.78	4.80	13.85	13.32
RTM-1048	1e-4	24.72	24.89	13.1	13.3
RTM-0594	1e-4	83.15	84.99	12.13	11.43
Miranda Velocity-x	1e-2	18.99	16.74	9.57	9.31
	1e-3	7.11	7.67	10.17	9.7
	1e-1	9.11	9.43	52.05	52.49

three data files in Figure 4.13. From our experience, when PSNR is higher than 50, there is no visible visual difference between the original and compressed data. Therefore, when the predicted PSNR is high, we are confident that the compressed data will be of a good quality for post-analysis.

4.5 Evaluation of Data Transfer Performance

Data transfer performance is crucial in determining whether data compression is advantageous for achieving a shorter transfer time, considering the additional time costs incurred during compression and decompression. Our framework aims to be adaptable to various network conditions to meet users' requirements. Generally, supercomputers boast extensive network bandwidths, whereas cloud service providers like Alibaba Cloud may provide a more restricted network capability. We evaluate the data transfer performance of our GlobaZip

Table 4.6: Prediction of PSNR for CESM application

Filename	**eb**	**Real PSNR**	**Predicted PSNR**
TMQ_1_1800_3600.dat	1e-3	96.80	96.39
CLDMED_1_1800_3600.dat	1e-3	59.64	60.88
TROP_Z_1_1800_3600.dat	1e-3	146.05	141.45
ICEFRAC_1_1800_3600.dat	1e-5	102.43	98.65
PSL_1_1800_3600.dat	1e-1	99.10	117.11
FLNSC_1_1800_3600.dat	1e-2	85.07	92.02
ODV_ocar2_1_1800_3600.dat	1e-5	79.16	83.92
LHFLX_1_1800_3600.dat	1e-4	138.92	136.23
TREFHT_1_1800_3600.dat	1e-3	99.28	86.82
FSDTOA_1_1800_3600.dat	1e-6	184.85	184.86

Table 4.7: Prediction of PSNR for ISABEL dataset

Filename	**eb**	**Real PSNR**	**Predicted PSNR**
QSNOWf48_log10.bin.dat	1e-2	72.85	81.11
PRECIPf48_log10.bin.dat	1e-1	52.49	52.78
QVAPORf48.bin.dat	1e-6	88.01	128.52
PRECIPf48_log10.bin.dat	1e-6	160.18	160.26
CLOUDf48_log10.bin.dat	1e-2	62.07	88.00
Wf48.bin.dat	1e-2	69.19	67.96
QSNOWf48_log10.bin.dat	1e-3	93.14	93.48
CLOUDf48_log10.bin.dat	1e-6	144.24	123.23
Pf48.bin.dat	1e-2	98.23	81.21
QSNOWf48_log10.bin.dat	1e-6	160.12	165.35

framework on the machines listed in Table 4.1.

The findings suggest that our proposed compression algorithms offer significant advantages for systems with slow networks and rapid computation. As demonstrated in Table 4.8, the performance improvement exceeds 10x when transferring data from supercomputers to typical cloud computing clusters. While contemporary cloud computing platforms boast comparable single-node computational capabilities to supercomputers, many are constrained by network bandwidth limitations. Our framework holds the potential to substantially enhance transfer efficiency across these platforms.

On the other hand, the overall transfer time may not benefit from compression in supercomputers because of superior networks. Nonetheless, our GlobaZip framework effectively

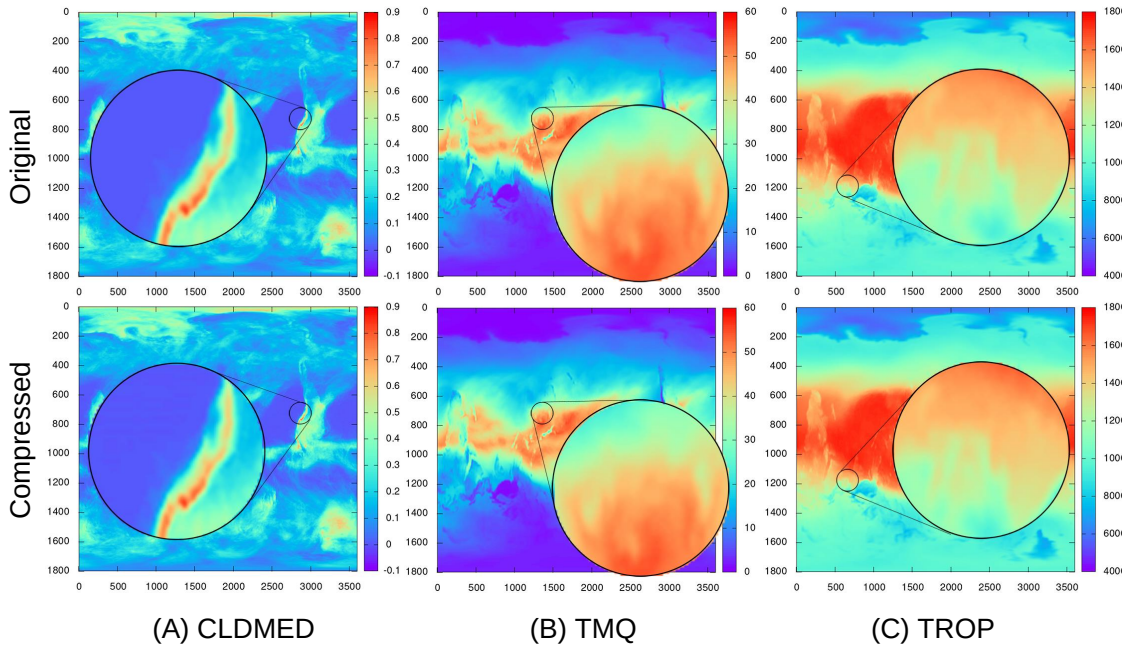

Figure 4.13: CESM data visualization comparison between original and compressed data: The PSNRs are 59.64, 96.80, and 146.05 respectively, and there is no obvious visual difference between the original and compressed data.

leverages specific characteristics to distribute computational tasks across different supercomputers. For example, we aim to transmit genome sequence data from Purdue Anvil to the Alibaba ECS. Due to the absence of Intel AVX512 optimization in the AMD Zen 3 processors utilized by Purdue Anvil, our genome sequence compression code cannot be compiled on this cluster. As a workaround, we transfer the data to Rockfish for compression before forwarding the compressed data to the ECS destination. This roundtrip approach ultimately optimizes overall time efficiency despite the intermediate steps involved.

4.6 Evaluation of Data Constraints Preservation

We first evaluate how our proposed methods can preserve the 5 data constraints elabotrated in the previous section. Each data constraint is preserved by an individual method and evaluated on multiple aforementioned floating-point number datasets.

Table 4.8: Data compression and transfer performance. T(NP): transfer time with no compression, Speed(NP): transfer speed with no compression, CPTime: time taken to compress the data, T(CP): transfer time for compressed data, Speed(CP): transfer speed for compressed data, DPTime: time taken to decompress data, Total: sum of compression time, transfer time, and decompression time; Reduced: total time reduced with compression. All times in seconds.

Dataset	Total Size	Direction	T(NP)	Speed(NP)	CPTime	T(CP)	Speed(CP)	DPTime	Total	Reduced
Nyx	3.22 GB	Anvil→Rockfish	6	547 MB/s	20	38	12.3 MB/s	20	78	-72
		Anvil→ECS	273	11.8 MB/s	20	42	9.49 MB/s	40	102	171
		ECS→Anvil	273	10.3 MB/s	60	45	10.3 MB/s	20	125	148
Turbulent Channel Flow	256 GB	Anvil→Rockfish	774	355 MB/s	175	55	338 MB/s	735	965	-191
		Anvil→ECS	25,303.00	10.4 MB/s	175	1740	11.0 MB/s	134	2049	23,254.00
		ECS→Anvil	23,198.00	11.3 MB/s	230	1709	11.2 MB/s	125	2064	21134
Genome A	39.27 GB	Rockfish→Anvil	64	611 MB/s	1095	36	316 MB/s	375	1506	-1442
		Anvil→(Rockfish)→ECS	3527	11.4 MB/s	**N/A**	1055	11.3 MB/s	874	**3088**	439
		ECS→(Rockfish)→Anvil	3904	10.3 MB/s	1100	1181	10.1 MB/s	**N/A**	**2692**	1212

4.6.1 Preserving Irrelevant Data (constraint A) and Global Value Range (constraint B)

The Hurricane Isabel dataset contains irrelevant data values marked as 1E35, which is well outside the normal value range. Table 4.9 shows the value range for five of 13 fields in the dataset which contain irrelevant (or missing) data points. The reason for the missing values is that the data simulates an actual event (a hurricane) and, in the locations where there is ground, no meaningful wind speed or pressure is recorded. More information about the dataset is available on the website.[1]

Figure 4.14 shows the distribution of data points in the Hurricane Isabel dataset. Because the actual value of the irrelevant data is far too large to be put in the same figure with normal data, we use a made-up value that is outside the range of each field to represent the irrelevant value. We can see that every field contains a non-negligible amount of irrelevant data, although not as many as normal data points. While the amount of irrelevant data is small, such data may severely harm the overall compression ratio because they are mixed among normal data points, destroying the continuity of normal data. We verify this statement by sampling a random continuous portion of the temperature field, as shown in Figure 4.15.

1. http://vis.computer.org/vis2004contest/data.html

Table 4.9: The 5 fields tested in the Hurricane Isabel dataset

Field	**Description**	**Value Range**
P	Pressure (weight of atmosphere above a grid point)	-5471.8579/3225.4257
TC	Temperature (Celsius)	-83.00402/31.51576
U	X wind speed (positive means winds from west to east)	-79.47297/85.17703
V	Y wind speed (positive means winds from south to north)	-76.03391/82.95293
W	Z wind speed (positive means upward wind)	-9.06026/28.61434

Figure 4.15 clearly shows that irrelevant data are distributed among normal data, destroying the smoothness of the data space. Obviously, if we predict a normal data point using the irrelevant data value, the prediction cannot be precise. As lossy compressor designers, we want to preserve irrelevant data values while mitigating their influence on the compression ratio. Note that even though they appear to be irrelevant for compression, they carry potentially useful information—in this case, they indicate ground locations.

In Figure 4.16, we investigate five different ways of handling irrelevant data. Time is measured on Bebop. The five strategies are: *Ignore* treats all irrelevant data as normal data; *Zero* replaces all irrelevant data by 0 for simplicity; *Clear* replaces all irrelevant data using the Lorenzo predictor based on their nearby values (our solution); *Quant* and *Bitmap* indicate the storage algorithm: Quant refers to using one additional quantization bin to mark irrelevant data, and Bitmap indicates that we use a bit array containing 1 and 0 to indicate whether each data point is an irrelevant value or not. Figure 4.16(A) shows that handling the irrelevant data may double the compression and decompression time. The overhead is due primarily to additional traversing of the whole dataset to find, clean, and recover irrelevant data. Moreover, constructing additional Huffman trees for irrelevant data will add additional time to the compression and decompression. Figure 4.16(B) shows that

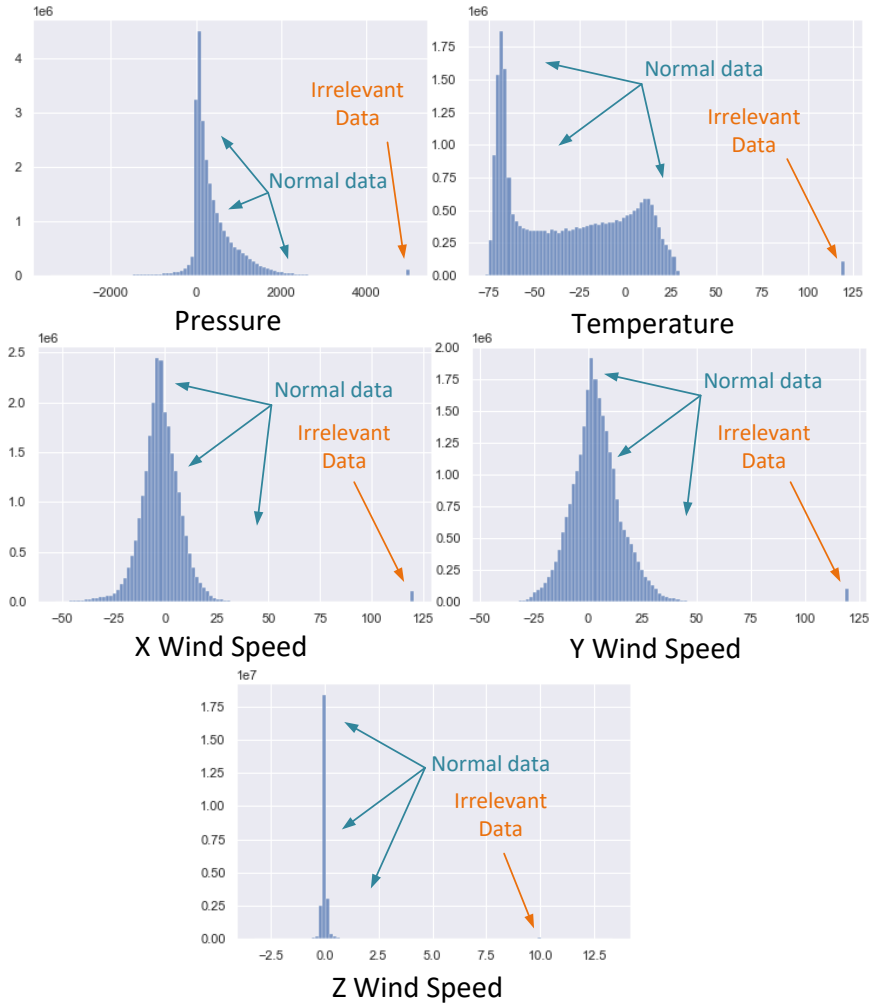

Figure 4.14: Data distribution of the five fields in the hurricane dataset. The irrelevant data value is 1E35. To visualize it in the distribution figure, we modify the big value to a made-up outside value that is not in the normal data range.

handling irrelevant data is generally better than ignoring them; however, it is difficult to determine whether it is better to clear them with the Lorenzo predictor or simply convert them to 0. Moreover, the simple bitmap method and quantization method exhibit similar performance. The likely reason is that irrelevant data are only a very small portion of the entire data and thus the methods are unable to demonstrate a huge difference in terms of the overall compression ratio. We conclude that in this scenario the quantization strategy slightly outperforms use of a bitmap.

The global range constraint requires only a scan in the preprocessing stage to obtain the

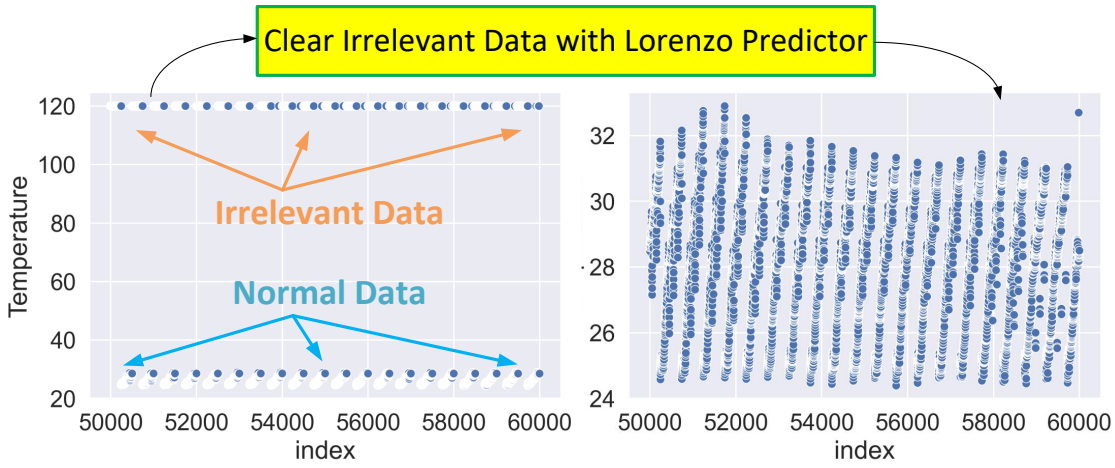

Figure 4.15: Temperature data points with (left) and without (right) irrelevant data. We show only a sample of 10,000 points (between index 50,000 and 60,000 in the original dataset). We observe the data points in the given index range and can see that the irrelevant data is mixed among the normal data points, harming data continuity; after clearing them with the Lorenzo predictor, the separating effect disappears.

max and min value. After the decompression, an additional traverse will be sufficient to pull back those few points whose values are beyond the min or max value. The time overhead is nearly negligible, as indicated in the gray bar in Figure 4.16 (A).

4.6.2 Multi-interval Error-Bounded Compression (constraint C) Based on Visual Quality and Post-hoc Analysis

Figures 4.17 and 4.18 show the substantial advantage of our multi-interval error bound-based compression over the traditional constant error-bounded compression, using two datasets (QMCPACK and Miranda). Specifically, the multi-interval-based compression preserves higher visual quality for the value intervals of interest, while achieving the same or even higher overall compression ratios by lowering precision on insignificant value intervals. For instance, in the QMCPACK dataset, over 90% of the data points are located around 0, but they are smooth and easy to be predicted by neighboring data points; however, the data points with values in the interval of $[-8, -5]$ are the sparse interesting values that are harder to be predicted accurately. That is, they are more important to preserve the overall visual

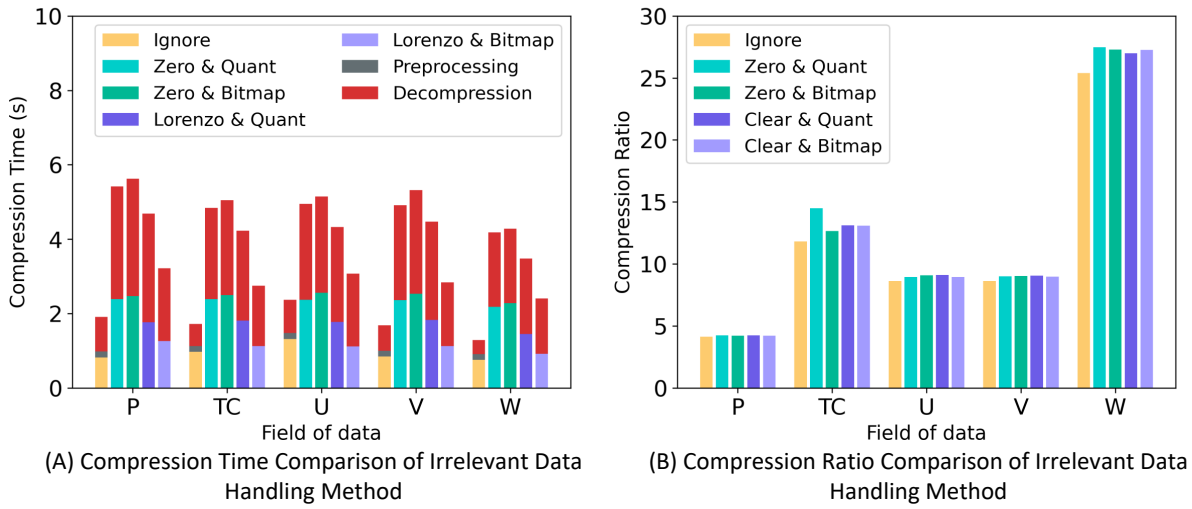

Figure 4.16: Performance of irrelevant data-handling methods: all methods slightly improve the compression ratio with a cost of longer compression time and decompression time.

quality because the distortion of their values is easier to observe in the visualization image.

Our method grants a tighter error bound and thus a higher precision in the more important value intervals, while allowing more distortion in insignificant value ranges, such that the overall compression ratio is not degraded. Detailed evaluation results are shown in Table 4.10 and Table 4.11. Given similar compression ratios, our method can achieve lower RMSE and higher PSNR in the critical value interval.

Table 4.10: QMCPACK RMSE & PSNR Comparison

Method	**Range**	**eb**	**RMSE**	**PSNR**
Global Range CR=210	[-17, -8]	0.4	0.232	43.067
	[-8, -5]		**0.233**	**43.041**
	[-5, 17]		0.051	56.159
Multi-Intervals CR=210	[-17, -8]	1.0	0.538	35.747
	[-8, -5]	0.15	**0.086**	**51.623**
	[-5, 17]	1.0	0.089	51.354

We now consider compression of the Nyx cosmological simulation with a specific quantity of interest (i.e., dark matter halo cell information). Dark matter halos play an important role in the formation and evolution of galaxies and consequently in cosmological simulations.

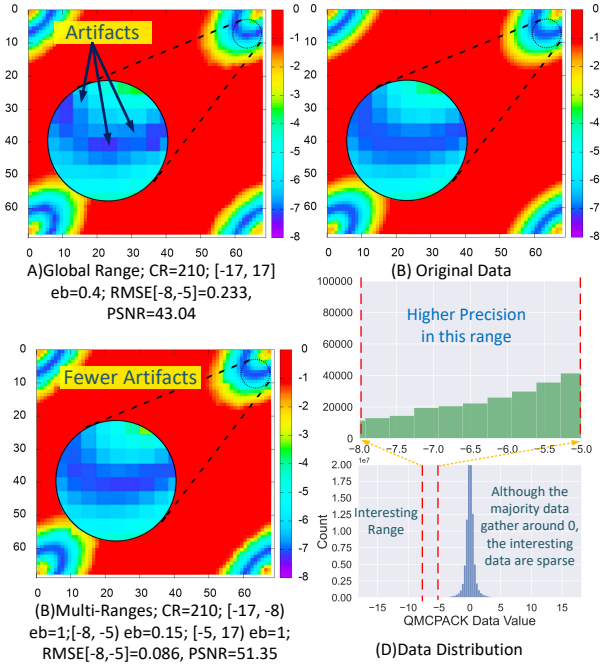

Figure 4.17: QMCPACK data: (A) The basic method is setting one error bound for the global range. We can see obvious artifacts in the blue area. Applying our multi-interval algorithm with a tighter error bound 0.15 in the interesting value range [-8,-5), we can see fewer artifacts in (C), while the compression ratio is kept the same as the global range method.

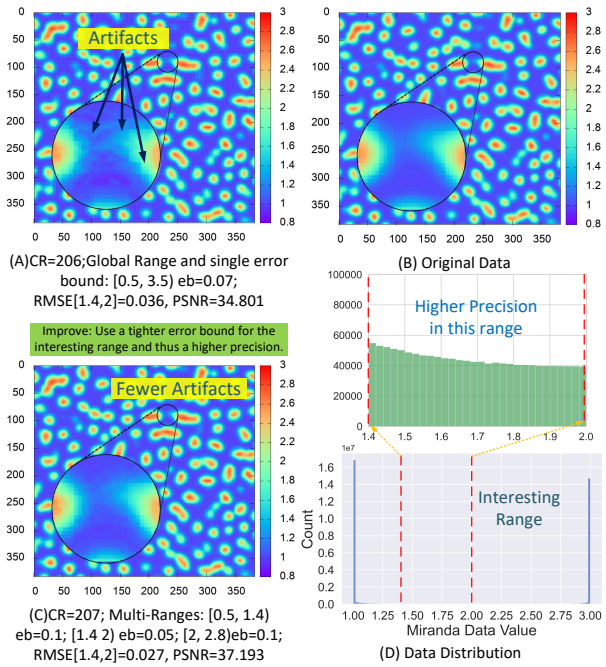

Figure 4.18: Miranda density slice No. 120. (A) The basic method is to set a single error bound for the global range. The artifacts are obvious compared to the original data in (B). When we give the interesting range [0.5, 1.4) a lower error bound, the artifacts in (C) are less significant and the data quality in the interesting range is higher due to lower error bound.

Halos are overdensities in the dark matter distribution and can be identified by using different algorithms; in this instance, we use the friends-of-friends algorithm [19]. For the Nyx simulation, which is an Eulerian simulation instead of a Lagrangian simulation, the halo-finding algorithm uses density data to identify halos [27]. For decompressed data, some of the information can be distorted from the original, such as halo cells and halo mass.

Figure 4.19 demonstrates that setting different error bounds for different value intervals in Nyx simulation datasets can preserve the features of interest (i.e., halo cells in this example) better than global-range error-bounded compression can. The key reason is that according to the Nyx halo analysis code, the values in the range of [81,83] need to be extremely precise (the reason is related to the sophisticated physics, and we ignore the details here). For our

Table 4.11: Miranda density RMSE & PSNR Comparison

Method	Range	eb	RMSE	PSNR
Global Range CR=206	[0.5, 1.4]	0.07	0.012	44.804
	[1.4, 2]		**0.036**	**34.801**
	[2, 3.5]		0.015	42.379
Multi-Intervals CR=207	[0.5, 1.4]	0.1	0.013	43.5813
	[1.4, 2]	0.05	**0.027**	**37.193**
	[2, 3.5]	0.1	0.018	40.682

compression task, we set three value ranges and assign a smaller error bound (0.1) to the data in the range of [81,83]. In this way the overall compression ratio will be higher with less distortion on the halo visualization result, as shown in Figure 4.19.

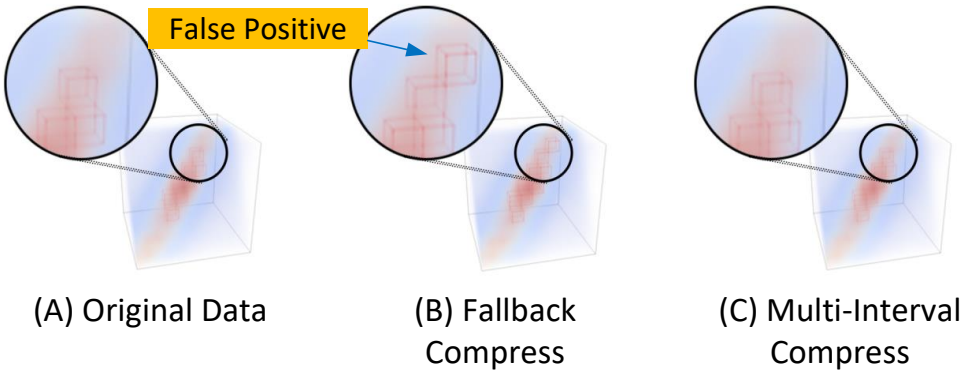

Figure 4.19: Nyx halo cell visualization: The fallback method sets a global error bound to be 0.5, and the compression ratio is 75. Our solution (C) sets three ranges: [min, 81) with error bound 1, [81, 83) with error bound 0.01, and [83, max) with error bound 1, and the compression ratio is 78. In the visualization, our multi-interval solution (C) has cells almost identical to the result using the original data, while the fallback method (B) shows some distortion, and the cells' position and number are not identical to (A).

Table 4.12 shows the substantially higher precision of our multi-interval error-bounded compression over global-range error-bounded compression. We use RMSE of cell number differences of halos and RMSE of mass differences of halos in comparison with original data as two main metrics to evaluate the results. Specifically, when passed through the post hoc analysis, our multi-interval solution can lead to significantly lower RMSE for cell number and halo mass, compared with the original RMSE under the global-range error-bounded

compression.

Table 4.12: Comparison of Different Range Settings. Fallback sets only a global error bound (here 0.01 and 0.5). Multi-interval uses our multi-interval error-bounded compression with three error bounds ([min, 81)=1, [81, 83)=0.01, and [83,max)=1)

Method	**RMSE of cell number**	**RMSE of halo mass**
Fallback-0.01:	0.089	125.84
Fallback-0.5:	2.820	429.26
Multi-interval:	0.198	135.41

We now investigate the combination of our methods on the Hurricane Katrina dataset. The combined methods include handling irrelevant data, multi-interval error bound settings, and different predictor settings (Lorenzo/linear regression).

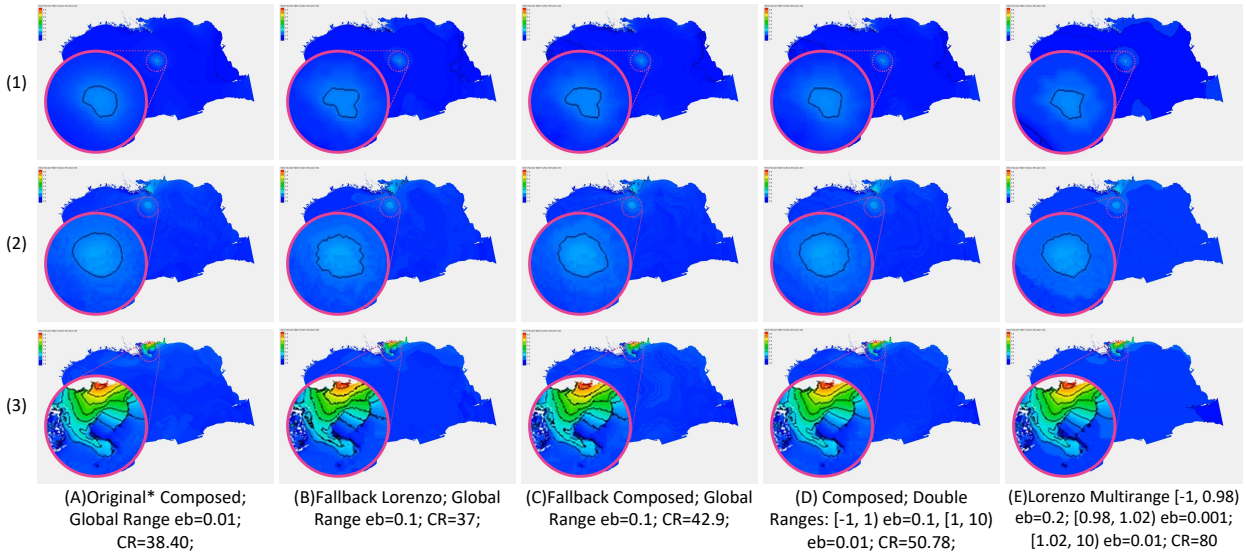

Figure 4.20: Hurricane Katrina data: Each row is a frame of the Katrina simulation: (1) is frame 120, (2) is frame 130, and (3) is frame 141. Each column represents a different setting of ranges and error bounds. Most of the blue data points in the graphs are close to zero.

Hurricane Katrina was one of the most devastating storms in the history of the United States because of its resulted significantly high storm surge (over 10 meters on the Mississippi coast) and high velocity. To model Katrina, the area was discretized into 417,642 nodes forming 826,866 unstructured meshes. The simulation was performed with a 1-second time step, from 18:00 UTC August 23 through 12:00 UTC August 30, 2005. The output hourly

water elevation data downloaded from the ADCIRC website (adcirc.org) was used in this study, and the water elevation contour map with a 1-meter interval at four times—3:00 am and 17:00 pm UTC August 28 and 3:00 am UTC and 14:00 pm UTC August 29—was plotted for illustrative comparison.

Katrina caused water elevation, and we wish to preserve more precisely the information about the elevation data that are above 1 meter (the multi-interval constraint). Moreover, some data points do not have meaningful values in this dataset and are represented by -99999 (irrelevant data). Therefore, we need to treat these values properly to mitigate their influence on the compression performance. By considering both irrelevant data and multi-interval error-bound constraints, the compression quality (as shown in Figure 4.20) can be improved significantly compared with the original compression quality under the state-of-the-art SZ 2.1. By applying a global range with error bound to be 0.01 with our solution, the visualization is almost identical to the original data's, and therefore we use one column (A) to demonstrate the visualization result as a reference. The fallback version shown in (B) is to use the original 1D SZ compressor, which has only the Lorenzo predictor and does not handle the irrelevant data; thus it has the lowest compression ratio even with a higher error bound 0.1. "Composed" in (C) and (D) means we use a composed Lorenzo and linear regression predictor to predict values. "Lorenzo" in (E) means we use only the Lorenzo predictor with no linear regression. Comparing (B) and (C), our solution wins on the global range test by handling the irrelevant data and using the composed predictor (both Lorenzo and linear regression). Comparing (C) and (D), our multi-interval solution wins in both the compression ratio and visualization result. Comparing (D) and (E), we can further improve the compression ratio by using the Lorenzo predictor only and allowing some distortion in the deep blue area.

4.6.3 Multiregion Error-Bounded Compression (constraint D) Based on Visual Quality

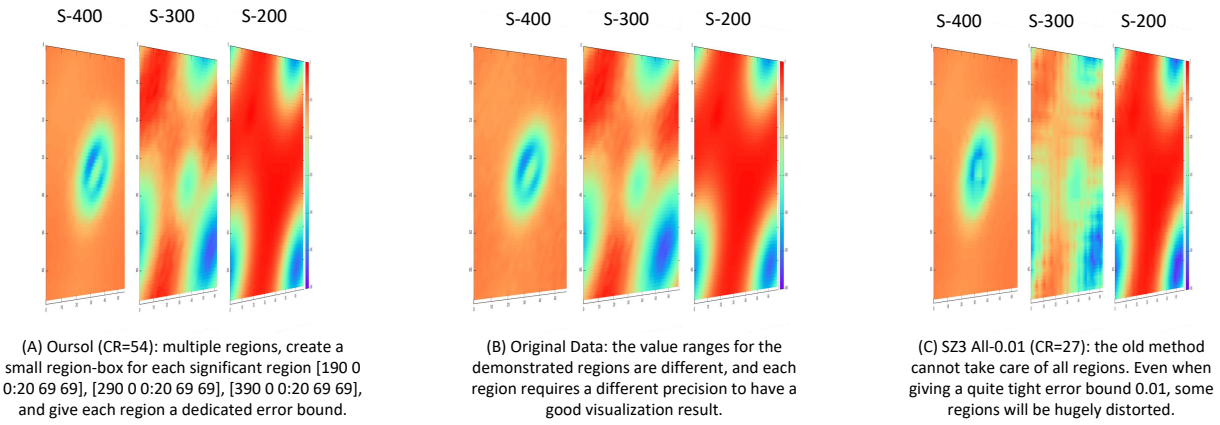

Figure 4.21: QMCPACK visual quality comparison: Each slice has 69×69 pixels. We select slice 200, 300, and 400 to observe the visual distortion because each has a different range: slice 200 has range [-0.06, 0], slice 300 has range [-0.0016, 0], and slice 400 has range [-0.0025, 0.0005].

To demonstrate the power of the region-based compression method, we perform a post hoc analysis of three regions in the QMCPACK dataset: slices 200, 300, and 400. Since each slice will usually be observed in one analysis step, it is better to set a suitable error bound for each slice instead of using a uniform error bound. For example, an error bound of 0.001 might be suitable for a slice with the data value range [-0.5, 0.5] but would be too large for a slice with range [-0.0025, 0.0005]. In Figure 4.21, we can see significant distortion in the selected regions in (C) even though the error bound is generally small (0.01) for the whole dataset. Our solution improves the quality by applying tighter error bounds on the three regions/slices. The compression ratio may not drop clearly, because the "tight-error-bound regions" are small compared with the global dataset.

In addition to addressing some chosen slices with specific regions, the region-based compression algorithm can achieve the effect of "different precisions for different areas" in each slice. As shown in Figure 4.22, the left-bottom corner has much better visual quality than

the other corners. With these two examples, we demonstrate the flexibility and locality of this region-based compression algorithm. In general, setting some small regions for some parts of the data that are of interest to the researchers will not influence the global compression quality. Moreover, researchers can set any number of regions in any parts of the dataset. Although it does not make sense to set hundreds of regions to select every possible interesting data points, the region-based algorithm offers the flexibility to accommodate complex requirements and demands.

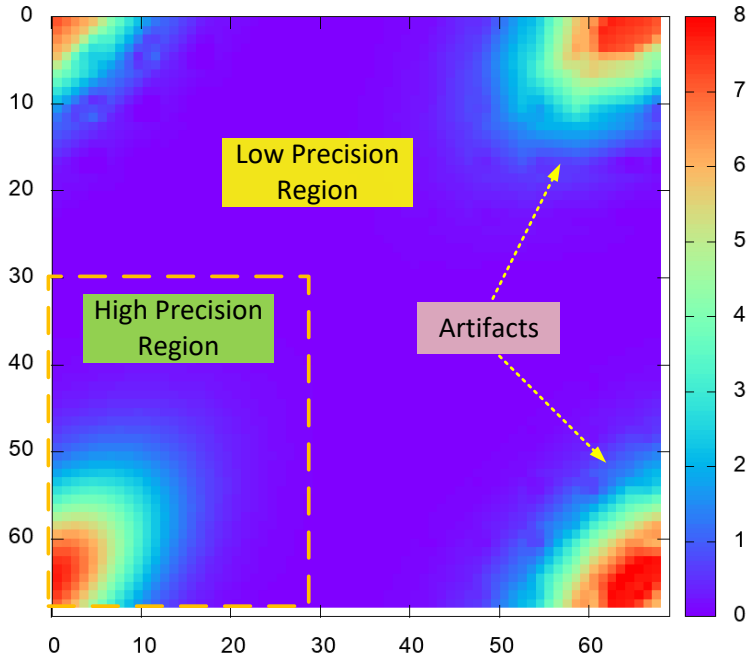

Figure 4.22: QMCPACK Slice 450, value range [0, 8]: A higher precision 0.001 for data in the area where $x \in [0, 30]$ and $y \in [30, 69]$, while keeping the error bound of other areas 0.5; The compression ratio is 242, and the SZ3 method with global error bound equal to 0.5 has a compression ratio 243. The region almost does not harm the compression ratio at all.

The feature of being able to set "different precisions for different areas" is extremely useful in climate data. Scientists and policy makers from different nations may share the same global climate data while focusing on their own country's details. We use the CLDHGH field in the CESM dataset to exemplify this feature. Since the dataset has a tight value range and the neighboring values are smooth, it is hard to visualize the difference directly between

the decompressed data and the original data in a small picture. We calculate the difference between each data point and visualize the difference instead. In Figure 4.23 (B), we can clearly see that the data inside the region (circled by a red rectangle) are much more precise than in the other areas since there are almost no artifacts in the difference image. In reaching the desired precision for the regions of interest, the region-based method clearly outperforms the traditional SZ compressor.

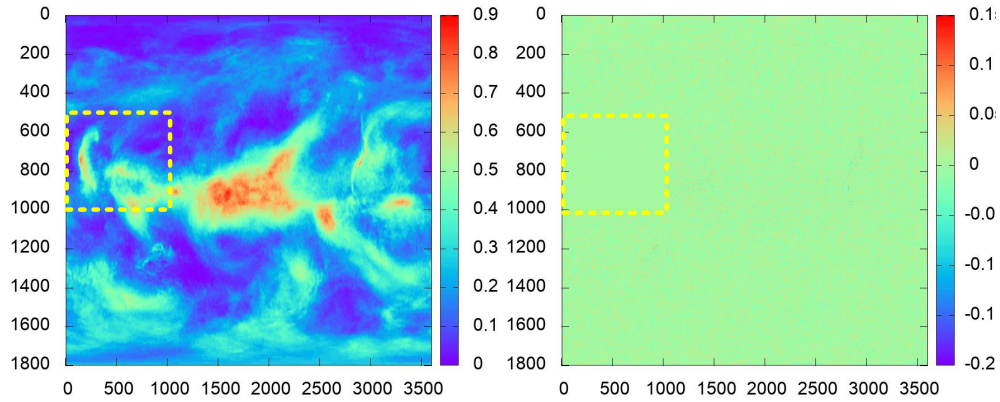

Figure 4.23: CESM with a region: while keeping the compression ratio high (CR=316), we make the interesting region more precise (eb=0.01). The error bound for the remaining regions is 0.02 in this example. If the SZ3's global error bound is used to reach eb=0.01 for the desired area, the compression ratio is 57.

We also evaluate the (de)compression time overhead of both multi-interval and multi-region methods. The overhead of the multiregion method is proportional to the number of regions, since each block needs to check the region list to find which region it belongs to. In contrast, the overhead of the multi-interval method is highly related to the precision of the prediction. To make the performance measurement as fair as possible, we use the same error bounds for all regions and value intervals on 6 datasets, and we set 5 different regions/intervals for each compression to guarantee that the overhead is observable.

The compression tasks are performed on the Bebop bdwall partition with a single node,

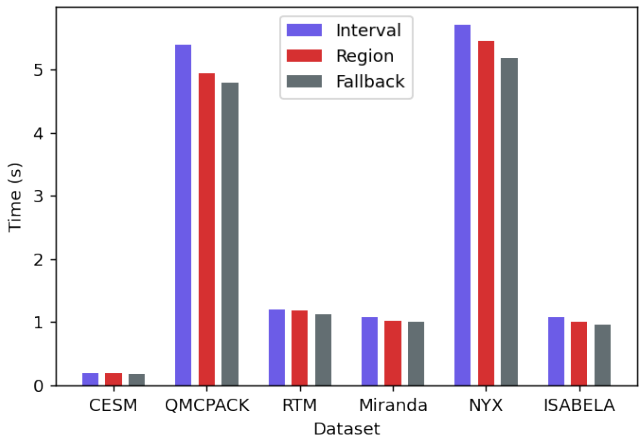

Figure 4.24: Comparison of compression time: The reference point is the *Fallback* version, which means using a uniform error bound for all data points. The overhead of the region-based method is slightly lower than that of the multi-interval method.

and we record the average of 10 runs for each compression configuration. As shown in Figure 4.24 and Table 4.13, the compression time overheads of both the multi-interval method and region-based method are not very high. The region-based method has slightly smaller overhead compared with the multi-interval method. The main reason is that our region-based method does not follow a point-to-point evaluation; instead, we stipulate each intrablock of the same region, cutting down considerable unnecessary computation. The same approach cannot be applied to the multi-interval method because we cannot assume neighboring points to be in the same value interval: actually, they are likely to be in two different value intervals specified by the user. To summarize, both methods lead to a certain compression time overhead, while the overheads are confined within an acceptable range.

Table 4.13: Compression Time and Overhead of Interval/Region/Fallback Methods

Method	**CESM**	**QMC**	**RTM**	**MIRAN**	**NYX**	**ISAB**
Interval(s)	0.20	5.39	1.20	1.08	5.70	1.08
Region(s)	0.19	4.94	1.18	1.03	5.46	1.01
Fallback(s)	0.18	4.80	1.12	1.00	5.18	0.96
Interval%	8.9%	12.3%	7.1%	6.8%	10.0%	13.0%
Region%	3.3%	3.0%	5.4%	1.9%	5.4%	5.7%

4.6.4 Bitmap-Specified Error Bound Compression (constraint E)

Table 4.14: Compression Setting Definition

Setting	Description
A	SZ2.1 [54]: Lorenzo & Linear Regression Predictor with one global error bound
B	Use SZ2.1's predictor, but adopt two error bounds set by a bitmap array
C	Interpolation-based compression with one uniform error bound [105]
D	Our developed region-based error-bounded compressor with two error bounds set by a bitmap

A bitmap defines the most concrete error bound information since it specifies an error bound for each data point. The overhead of storing a bitmap is non-negligible if not properly compressed. In the following, we evaluate two methods for storing the bitmap-specified error bounds: (1) the bitmap array is background information that is stored separately by users as metadata (e.g., the world map); and (2) the bitmap needs to be stored with compressed data so it must also be compressed.

Situation 1: We consider the CESM dataset as an example to evaluate the first bitmap method. Our bitmap solution can help users specify different precisions with fine granularity on irregular regions, in contrast with the other regular-region-based multierror-bounded compression method.

In the CESM dataset, we retrieve the bitmap array by using the LANDFRAC field, because it is a good match for separating the land and ocean area in a world map (as shown in Figure 4.25 (F)). Applying LANDFRAC as the bitmap, we test four different compression settings (described in Table 4.14) on the other five data fields, as shown in Table 4.15. In Table 4.14 we can see that the bitmap solution sacrifices precision in the red area and can obtain a higher compression ratio. The overall PSNR will decrease when enlarging the error bound for red areas, but the compression quality for the interesting areas (here, the blue

Table 4.15: Impact of compression settings on compression ratio (CR) and PSNR for the six CESM fields of Fig 18: P_0/P_1 are the PSNR in the bitmap separated blue/red area, respectively; CR' is the compression ratio that takes the bitmap into account

Data Field	Setting	CR	CR'	PSNR	P_0	P_1
CLDLOW	A: eb=0.01	21	-	44.94	46.74	49.59
min=-0.1	B: eb=0.01, 0.1	30	29.0	29.71	46.74	29.73
max=1	C: eb=0.01	138	-	47.14	49.23	51.26
	D: eb=0.01, 0.1	224	176.6	32.31	49.22	32.34
FREQSH	A: eb=0.01	16	-	44.73	46.76	48.97
min=0	B: eb=0.01, 0.1	22	21.4	28.67	46.76	28.67
max=1	C: eb=0.01	88	-	46.79	48.83	50.99
	D: eb= 0.01, 0.1	126	109.5	32.10	48.83	32.13
LHFLX	A: eb=1	30	-	60.27	62.28	64.55
min=-100	B: eb=1, 10	48	45.4	49.36	62.28	49.55
max=600	C: eb=1	106	-	62.41	64.58	66.40
	D: eb= 1, 10	216	171.6	47.81	64.63	47.84
PBLH	A: eb=5	37	-	53.04	55.20	57.07
min=0	B: eb=5, 15	45	42.7	47.72	55.20	48.55
max=1600	C: eb=5	107	-	55.03	57.24	58.99
	D: eb= 5, 15	169	140.5	49.23	57.26	49.93
TSMN	A: eb=1	66	-	44.78	47.04	48.64
min=200	B: eb=1, 10	191	155.4	36.19	47.04	36.51
max=310	C: eb=1	292	-	47.14	49.41	50.99
	D: eb= 1, 10	812	411.5	31.64	49.24	31.66

areas are considered interesting areas) remains the same—P_0 almost does not decrease, while P_1 decreases because of a larger error bound set in the corresponding area.

Table 4.15 demonstrates that our region-based multierror-bounded compression method significantly outperforms all other solutions in compression quality. The reason is twofold. (1) Our developed bitmap method can be used to fine-tune the precisions for different irregular regions, which can preserve the quality for regions of interest more effectively while reaching a high compression ratio. This can be verified by comparing the settings C and D in the table. (2) As we discussed ear;oer, the interpolation predictor is much more effective than the linear regression predictor used by SZ2.1. This can be verified by comparing settings A and C in the table.

Situation 2: In the second situation where the bitmap array needs to be stored together

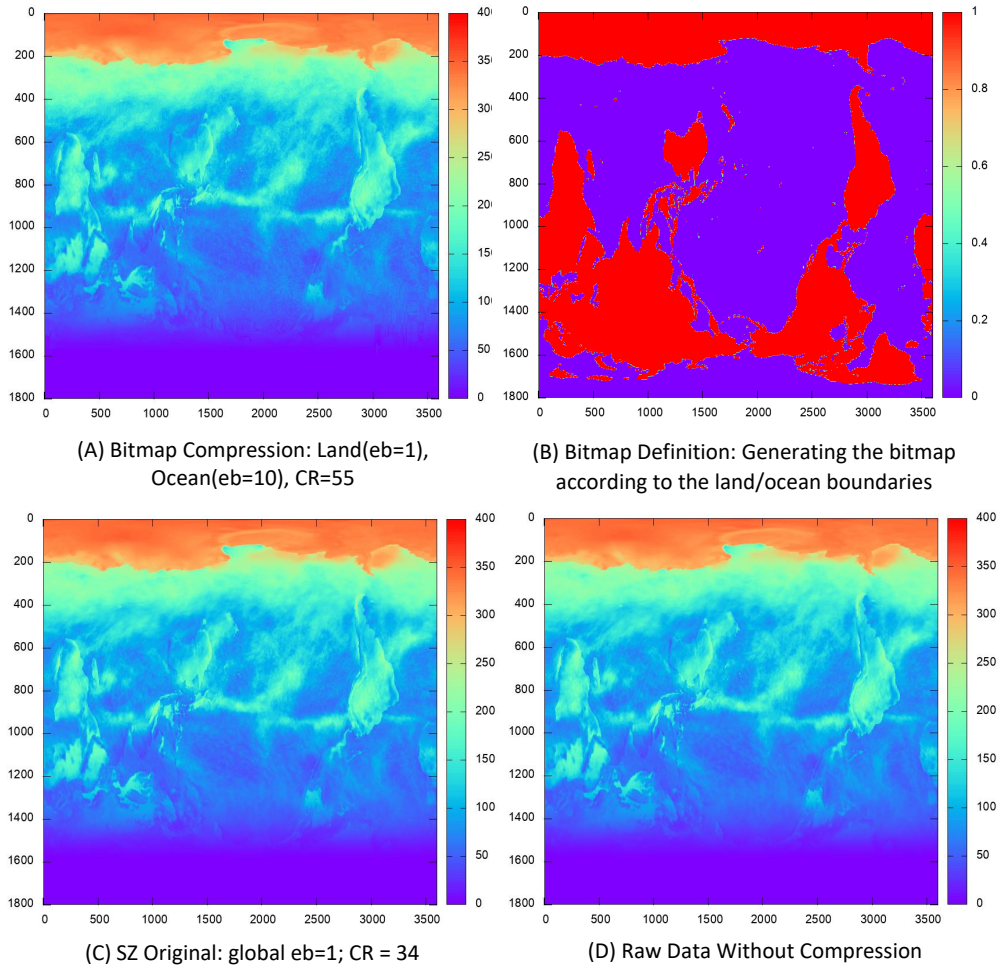

Figure 4.25: The visualization indicates that bitmap-separated precisions may be suitable to compress these fields.

with the compressed data, we compress the bitmap array by integer-based Huffman encoding [53] and Zstd [110]. Specifically, the input data is the integer bitmap array with the same number of elements as the original dataset.

Table 4.15 shows the compression ratio of our region-based multierror-bounded lossy compression method (denoted as CR') after embedding the bitmap into the compressed data. Since uniform error-bounded compression does not need to store the bitmap array, this column shows only the compression ratios for settings B and D. We observe that CR' is close to CR (i.e., the compression ratio without storing the bitmap array) in most cases.

The reason is that the bitmap array is fairly easy to compress with high ratios (reach ∼800 in this example) because of the limited number of error bounds. In fact, there are typically few error bounds in practice because of the limited number of value intervals of interest or regions of interest in general. Accordingly, the error level values would likely exhibit repeated patterns in the bitmap array, especially for the consecutive data points in space, leading to a very high compression ratio.

4.7 Case Study: High-energy diffraction microscopy workflow

High-energy diffraction microscopy (HEDM) is used to non-destructively probe the internal structure of polycrystalline materials. In a typical HEDM workflow, thousands of x-ray images will be captured while the object is subject to external mechanical loading to measure internal strains and stresses within individual grains – crucial to understanding material fatigue, failure, or deformation. The raw scans are collected on some edge devices and need to be sent to a supercomputer for compute-intensive training and inference of ML models. The computation on supercomputers generate analysis results as well some smaller ML/rule-based models suitable for edge devices. The updated models and analysis results will then be sent back to the edge device to guide the experiments. The whole workflow is illustrated in Figure 4.26.

One of such ML models is to calculate the rare event indicator (REI)[107] to obtain quantitative actionable information about material states and guidance for future experiments. As the detector constantly captures images at high frequency, a large amount of raw images needs to be transferred to the supercomputer. Our compression techniques can significantly improve the transfer efficiency in this application. In this section, we will evaluate the data quality awareness of our compression techniques with both visualization and the results of downstream analysis.

The raw data for each HEDM scan consist of 1440 frames, each frame of 2048x2048

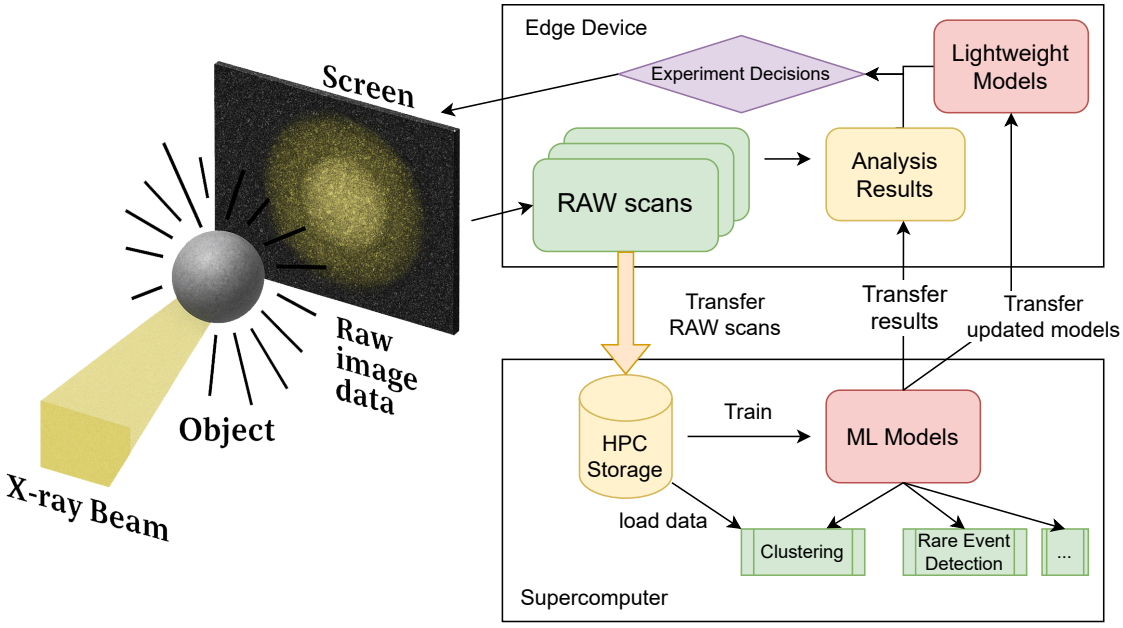

Figure 4.26: The illustration of a typical HEDM workflow without compression. We hope to add our compression into the workflow to improve the efficiency of transferring raw scans to the supercomputers.

pixels. Each raw data file is roughly 12.5GB, with a constant-sized header. The pixels are represented by 16-bit unsigned integers ranging from 1,000 to 17,000. The prediction-based lossy compression can easily reduce the data size for this type of data because the values are quite continuous. Figure 4.27 shows the result of applying an error bound 30 to compress the test scan, the compression ratio reaches 119, while the visualization of the decompressed data looks almost identical to the original data. If the error bound is set too large (for instance 300), the compression ratio will be extremely high, but we can see noticeable artifacts and distortions in the visualization results.

Next, we evaluate how much distortion lossy compression brings to downstream analysis. We used the raw file of the base scan to build an embedding model and a KMeans clustering model and then sent the original and decompressed test scan files to the models to obtain a REI score. Note that the REI score is a predicted value by the ML model, but we can view it

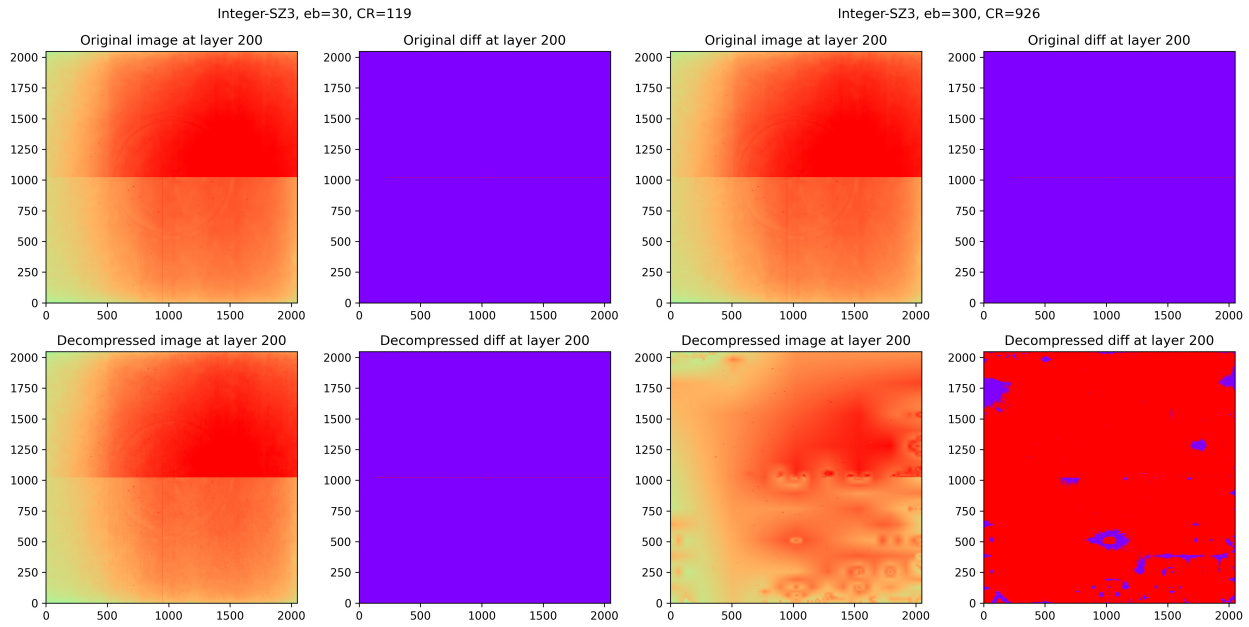

Figure 4.27: Visualization results of the original and decompressed data of the layer 200 out of the 1440 frames. Note that we pick a random number 200 for illustration, and the visualization of other layers look very similar.

as ground truth because we only modify the test data by compression without changing the base scan data or the ML models. As shown in Figure 4.28, when the error bound increases, the REI score increases because the distortion caused by compression creates unwanted patterns/artifacts that influence the model's decision and make it think there are more rare events in the image. When the error bound is set too large (100 in this case), the compression creates too much distortion that is very harmful to the REI score calculation. According to Zheng et al.[107], the REI scores of most materials remain minimum until the mechanical load increases to a point where the material "cracks" and the REI score suddenly increases from 0.3 to above 0.8. Therefore, if compression only causes a small REI deviation without blurring the boundary for the material to crack, the error is in an acceptable range.

We further evaluated compression performance on logarithmic transformed data by applying the natural logarithm to each data point prior to compression, followed by lossy compression of the transformed data. During decompression, an exponential function is

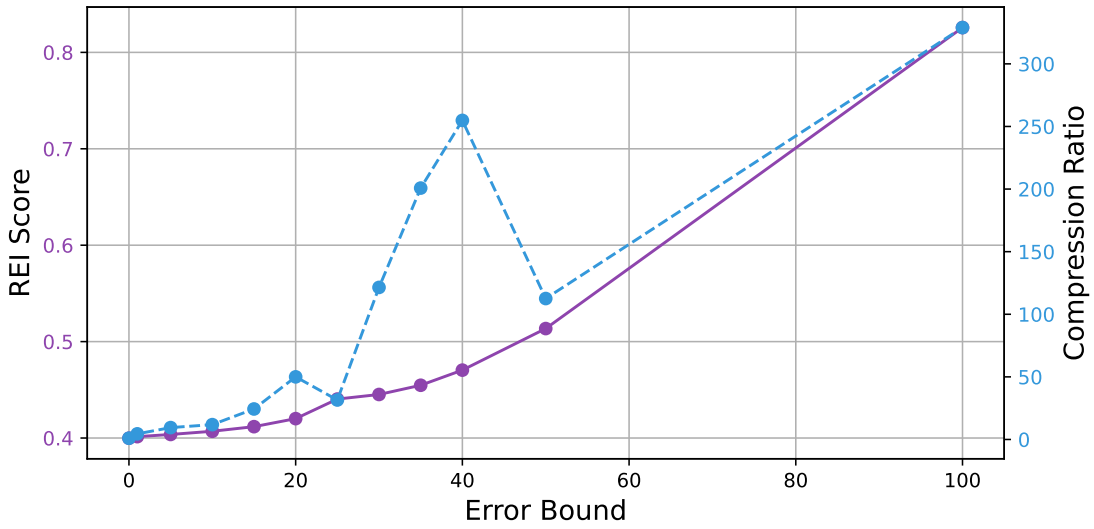

Figure 4.28: The REI scores and compression ratio with respect to the error bound settings. When the error bound is 0, it indicates the original file's REI score and compression ratio is 1 as it is the original file without compression. For the downstream experiments, the REI score should be as close to the original file as possible for less distortion.

applied to restore the data to its original scale. The results are presented in Figure 4.29. Using logarithmic transformed data yields a smoother compression curve, as prediction-based compression techniques are more effective on floating-point data. We also observe that the compression ratio can get much higher while keeping the REI score very low and close to the original data: when the error bound is set to 0.01, the REI score stays low enough while the compression ratio is close to 600. Moreover, this approach aligns the error bound values with those commonly used in existing literature, avoiding the need for unintuitive large integers that are often difficult for practitioners to interpret or configure.

Since compression introduces computational overhead, we aim to investigate under which network conditions it can actually improve overall transfer efficiency. Let the compression/decompression overhead time be $T_{overhead}$, the total file size be N, the compression ratio be CR, the network speed be S. To ensure the total transfer time is less than direct transfer, we need

$$T_{overhead} + \frac{N}{CR \cdot S} < \frac{N}{S}$$

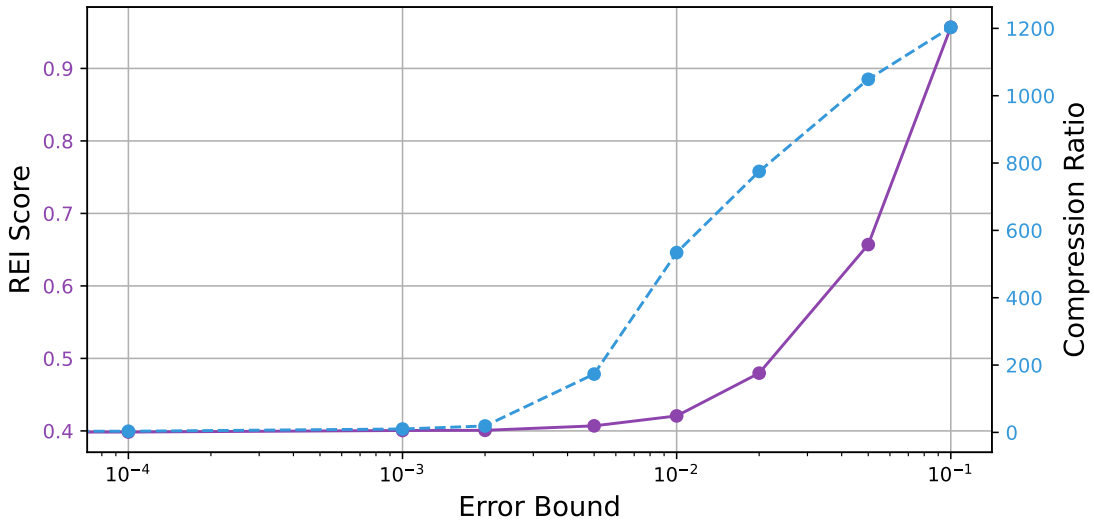

Figure 4.29: Floating-point data compression on the log-transformed data with a smoother error bound to compression ratio curve.

Therefore, the network speed cannot exceed a certain value given the compression throughput:

$$S < (1 - \frac{1}{CR})\frac{N}{T_{overhead}}$$

Figure 4.30 presents the maximum allowed network speed for the given compression setting to be useful in transfer optimization. We note that when the compression ratio reaches 10, parallel compression with 16 threads can already optimize the transfer performance for 2000Mbps networks. This means that we can set a very small error bound, keep the data distortion minimum, and still expect to have a performance gain for data transfer. This insight is very useful for users who want to preserve a very high level of data fidelity while involving lossy compression algorithms in the workflow.

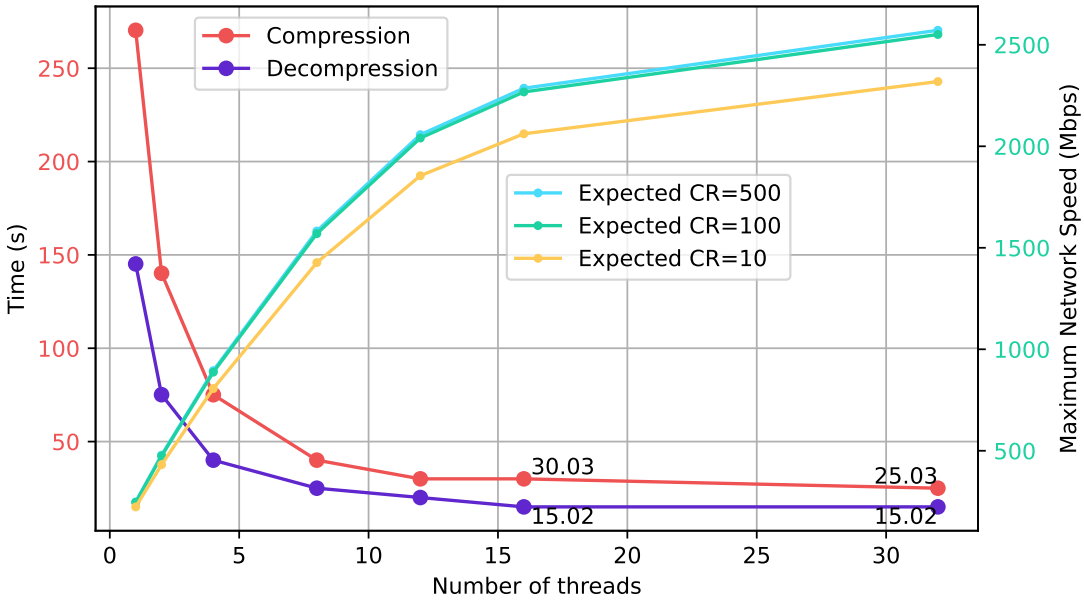

Figure 4.30: The efficiency of compression based on different network conditions. We use our proposed layer-by-layer logscale parallel compression method with layer depth 64, error bound 0.01, and compression ratio 500. The error bound setting almost does not affect the compression time so we also plotted two other compression ratio setting with lower error bound in the same figure.

CHAPTER 5

FUTURE WORK

In this chapter, we discuss future work as well as some open-ended ideas that emerged from designing, developing, and testing the Globazip framework. In general, we want to expand the current compression/transfer framework to accommodate more data types and application scenarios.

5.1 KV Cache Compression

The Globazip framework aims to enhance data transfer efficiency through lossy compression, especially for data types that existing methods struggle to compress effectively. One such emerging data type is the KV cache in large language models (LLMs). To perform complex tasks, users often prepend LLM inputs with lengthy contexts containing thousands of tokens. For example, a user might provide domain-specific text to enable the model to generate content based on specialized knowledge not embedded in the model itself. However, processing long contexts introduces a delay in response generation, as the LLM cannot begin producing a response until the entire context is loaded and processed. The computational cost of processing a long context increases super-linearly with its length. To mitigate this, many systems reduce context-processing delays by storing and reusing the KV cache, which allows them to skip redundant computations when the same context is used again.

Nonetheless, when the next input token arrives, the KV cache of a reused context may not be in the local GPU memory; instead, the KV cache needs to be retrived from another machine, causing extra network delays. Some systems assume the KV cache of a context is always kept in the same GPU memory between requests[31], and some others assume the KV cache is small enough to be sent quickly by a fast interconnection[75, 108]. There are also a few existing works that tried to mitigate this network delay problem by compressing the KV

Cache to bitstreams[68, 16, 97]. It is interesting to explore how existing methods handle the optimization of compression, transfer, and decompression to reduce the latency for generating the first token in LLMs. Developing more efficient and specialized algorithms for compressing KV cache data presents a promising direction for further improvement. Integrating the Globazip framework into the LLM context could be a valuable future endeavor, expanding the range of data types that can be effectively compressed and transferred.

5.2 Automated Data Quality Check

Globazip offers users a manual approach to preview data and test multiple error bounds interactively. However, many users may primarily want to transfer data as quickly as possible while ensuring that the data quality remains high and suitable for future use. They may not be concerned with the specific error bound chosen, but they need confidence that any loss of data quality will not compromise their downstream analysis pipeline. To address this, we should integrate a mechanism into applications like the APS workflow[26]. This mechanism would involve running a small analysis model on the compressed data and comparing the results with those from the original data. The entire process should be automated, allowing the framework to determine an appropriate error bound based on the comparison results.

The process can become more complex, as evaluating the entire dataset may require significant amount of time and computational resources. To address this challenge, we will explore various methods for strategically sampling a representative subset of the data. By doing so, we aim to enable the downstream model to achieve a reasonably accurate conclusion while minimizing the computational cost and processing time.

5.3 Automated Compressor Selection

Multiple compression algorithms (including ZFP[59], MGARD[4], SZ[53], SZ3[57]) are available in the GlobaZip app but they can be suitable for different user requirements and datasets. One way is to use the aforementioned compression performance prediction method to estimate the compression ratio and quality, but it only works for prediction-based compression methods and when the prediction algorithm or quantization algorithm change, the estimation can be inaccurate. A more reliable way is to sample certain parts of the data and run the actual compression/decompression of each compressor and compare the results.

There are a few challenges in comparing the compressors for the optimization of transfer:

1. **The sampled data do not necessarily describe the whole dataset**. For example, if we use the first a few layers as samples, these layers may contain trivial data. Especially for time-series simulation, the beginning stage usually contains data that are far simpler than the average data in the middle. If we sample data by jumping points (e.g. select one point for every ten points), the prediction-stage results usually will be different for whole dataset compression because the prediction is based on different data points. The same problem applies to transform-based compressors as the data blocks consist of different data points and would likely generate inconsistent results.

2. **The selection process needs to happen very fast**. There is always an option to directly transfer the data without compression, especially when the network speed is fast. If the selection and comparison cost too much time, there is no point to compress the data at all for data transfer purpose, unless users only want to store the data in a compressed format to save storage spaces.

3. **Quantitative metrics may not demonstrate the artifacts of compressors well**. Quantitative metrics like PSNR and SSNI are just one number to describe the whole dataset. They cannot capture important lossy compression artifacts that happen in

local regions. A more intuitive and comprehensive description of the decompressed data is needed. A proper visualization mapping can aid this situation.

It is valuable to explore automated methods that can identify the most suitable compressor for a given application, as this can significantly enhance both efficiency and performance. Traditional approaches to compressor selection often rely on manual evaluation and empirical testing, which can be time-consuming and prone to human error. By leveraging machine learning and optimization techniques, it is possible to develop automated systems that analyze key factors such as input data characteristics, target performance metrics, and resource constraints to recommend the most effective compressor. Future work in this area could focus on building adaptive models that continuously learn and improve based on feedback from real-world performance data. Additionally, exploring reinforcement learning methods could enable the system to dynamically adjust its recommendations as new data becomes available or as the operating environment changes. Another promising direction is the integration of neural architecture search (NAS) techniques to automate the design of custom compressor configurations tailored to specific workloads. Furthermore, research into multi-objective optimization could help balance trade-offs between compression speed, accuracy, and computational cost, ensuring that the selected compressor aligns with the broader system requirements. These advancements would not only streamline the selection process but also lead to more efficient and scalable data processing pipelines.

CHAPTER 6

CONCLUSION

Inspired by the widespread demand for storing and transferring large amounts of scientific data in a daily routine, I explored lossy compression methods to optimize the overall transfer performance. In this thesis, I packaged my findings and developed a user-facing app Globazip that integrates several state-of-the-art compression algorithms and allows users to orchestrate compression/transfer tasks on remote computing clusters. The findings are summarized according to the five research questions I proposed at the beginning of this thesis.

In section 3.1, I explore five different data constraints that the compressors need to handle. I design a multi-interval and a multi-region compression algorithm that can significantly improve the visual quality in the critical value intervals and regions with the same or even higher compression ratios. In the Nyx cosmology simulation, the multi-interval error-bounded lossy compression can preserve the halo cells perfectly with a high compression ratio up to 78, while the uniform error-bounded compression suffers significant distortion of cells. In the Hurricane Katrina simulation, multi-interval error-bounded compression can improve the compression ratio from 37 (based on SZ) to 80 (improved by 116%), even with higher data fidelity in maintaining the shape of hurricane. The evaluation for the bitmap-based solution shows that the cost to satisfying a customized complex region requirement is acceptable and my solution can possibly be generalized to suit all kinds of fine-grained error bound settings.

In section 3.2, I explore using features extracted from config, data, compressors to predict the compression time and data quality. I also provide a user-friendly interface for users to preview the data on remote computing clusters without transferring the data itself. The decision tree model can predict the wanted performance metrics rather precisely with acceptable computation overhead.

In section 3.3, I propose and implement a novel reference-based genome sequence com-

pression algorithm. I improve the sequence alignment procedure, propose a dominant bitmap method for quality score compression, and design a compressed file storage architecture for better parallel read capabilities. My proposed method achieves better compression ratios than state-of-the-art algorithms, by employing a better alignment algorithm and an optimized quality score compression method, and by not keeping string identifiers and read orders.

In section 3.4, I propose a simple yet effective layer-by-layer method to parallelize the compression of floating-point tensors. The proposed method uses a series of 2D or relatively thin 3D layers with prediction-based compression. It avoids the memory limit constraint and allows parallel compression for extremely large files. Because of the independence of each layer, the method also allows users to compress only part of the data to visualize the decompressed data without spending too much computation resources.

In section 3.5, I introduce my design of the compression/transfer orchestration app that offers long-term endpoints deployed on multiple computing clusters. Users interact with the app through a Qt5-based universal user interface that runs on their personal computer. The app provides a uniform authentication procedure for users to run (de)compression and transfer tasks on different clusters and help users avoid unreliable long-term download connection problems through their laptops.

In section 3.6, I explore ways to further improve the data transfer rates according to network patterns. I find file grouping can significantly improve the transfer speed when users need to transfer a large amount of small files. Parallel compression is also very beneficial for compressing large amounts of independent small files. Since (de)compression requires application of the compute nodes, the waiting time for the compute nodes to be assigned can be taken into consideration for further transfer time optimization.

Finally, in chapter 4, all the proposed methods are evaluated thoroughly with several real-world datasets.

I envision that the work presented in this thesis, when combined with additional work in related areas, will lead to the contruction of cutting-edge data transfer platforms that involves lossless and lossy compression. As shown in this thesis, the overall transfer time is greatly reduced when applying my proposed compression algorithms especially for transfer tasks to cloud servers where computers are connected by relatively slower networks. I also make efforts to ensure the data quality can be previewed and respected for downstream analysis tasks even after lossy compression. While my work in building Globazip have shown significant progress in many of the areas to be explored, there is still much work to do as some of the future directions are described in chapter 5.

REFERENCES

[1] BGISEQ500 PCRfree NA12878 CL100076243 L01. https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/NA12878/BGISEQ500/.

[2] DNBSEQ-T7 WES PE150 . https://db.cngb.org/search/experiment/CNX054776 4/.

[3] Katrina simulation. https://adcirc.org/home/documentation/example-problems/katrina-run-2015-nws-20-example/. Online.

[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] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Isahagian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. *Serverless Computing: Current Trends and Open Problems*, pages 1–20. 12 2017.

[6] Matěj Bartík, Sven Ubik, and Pavel Kubalik. Lz4 compression algorithm on fpga. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 179–182, 2015.

[7] Gaëtan Benoit, Claire Lemaitre, Dominique Lavenier, Erwan Drezen, Thibault Dayris, Raluca Uricaru, and Guillaume Rizk. Reference-free compression of high throughput sequencing data with a probabilistic de bruijn graph. *BMC bioinformatics*, 16(1):1–14, 2015.

[8] Blosc compressor. http://blosc.org/, 2018. Online.

[9] Abraham Bookstein, Vladimir Kulyukin, and Timo Raita. Generalized hamming distance. *Information Retrieval*, 5, 10 2002.

[10] John Bresnahan, Mike Link, Rajkumar Kettimuthu, Dan Fraser, and Ian T Foster. Gridftp pipelining. In *TERAGRID 2007 CONFERENCE*, 2007.

[11] Martin Burtscher and Paruj Ratanaworabhan. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58(1):18–31, 2008.

[12] S Chandak, K Tatwawadi, I Ochoa, M Hernaez, and T Weissman. Spring: a next-generation compressor for fastq data. *Bioinformatics*, Aug 2019.

[13] Kyle Chard, Steven Tuecke, and Ian Foster. Globus: Recent enhancements and future plans. In *XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, XSEDE16, New York, NY, USA, 2016. Association for Computing Machinery.

[14] Ryan Chard, Yadu Babuji, Zhuozhao Li, Tyler Skluzacek, Anna Woodard, Ben Blaiszik, Ian Foster, and Kyle Chard. funcX: A federated function serving fabric for science. In *29th Intl Symp on High-Performance Parallel and Distributed Computing*, page 65–76, 2020.

[15] Jian-Wen Chen, Chao-Yang Kao, and Youn-Long Lin. Introduction to h.264 advanced video coding. volume 2006, pages 6 pp.–, 02 2006.

[16] Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. Do large language models need a content delivery network? *arXiv preprint arXiv:2409.13761*, 2024.

[17] Bobbie Chern, Idoia Ochoa, Alexandros Manolakos, Albert No, Kartik Venkat, and Tsachy Weissman. Reference based genome compression. *IEEE Inf Theory Workshop, ITW*, 04 2012.

[18] Peter J. A. Cock, Christopher J. Fields, Naohisa Goto, Michael L. Heuer, and Peter M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767–1771, 12 2009.

[19] Marc Davis, George Efstathiou, Carlos S Frenk, and Simon DM White. The evolution of large-scale structure in a universe dominated by cold dark matter. *The Astrophysical Journal*, 292:371–394, 1985.

[20] X. Delaunay, A. Courtois, and F. Gouillon. Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netcdf-4 or hdf5 files. *Geoscientific Model Development*, 12(9):4099–4113, 2019.

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

[22] Sheng Di, Jinyang Liu, Kai Zhao, Xin Liang, Robert Underwood, Zhaorui Zhang, Milan Shah, Yafan Huang, Jiajun Huang, Xiaodong Yu, Congrong Ren, Hanqi Guo, Grant Wilkins, Dingwen Tao, Jiannan Tian, Sian Jin, Zizhe Jian, Daoce Wang, MD Hasanur Rahman, Boyuan Zhang, Shihui Song, Jon C. Calhoun, Guanpeng Li, Kazutomo Yoshii, Khalid Ayed Alharthi, and Franck Cappello. A survey on error-bounded lossy compression for scientific datasets, 2025.

[23] Sheng Di, Dingwen Tao, Xin Liang, and Franck Cappello. Sz tutorial hands-on guide. `https://www.mcs.anl.gov/~shdi/download/sz-hands-on.pdf`, 2018. Online.

[24] James Diffenderfer, Alyson Fox, Jeffrey Hittinger, Geoffrey Sanders, and Peter Lindstrom. Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing*, 02 2019.

[25] Shen R et al. High-throughput snp genotyping on universal bead arrays. *Mutat Res.*, pages 70–82, 06 2005.

[26] Thomas E. Fornek. Advanced photon source upgrade project preliminary design report, 9 2017.

[27] Brian Friesen, Ann Almgren, Zarija Lukić, Gunther Weber, Dmitriy Morozov, Vincent Beckner, and Marcus Day. In situ and in-transit analysis of cosmological simulations. *Computational Astrophysics and Cosmology*, 3(1):1–18, 2016.

[28] Arkaprabha Ganguli, Robert Underwood, Julie Bessac, David Krasowska, Jon C. Calhoun, Sheng Di, and Franck Cappello. A lightweight, effective compressibility estimation method for error-bounded lossy compression. In *2023 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 247–258, 2023.

[29] Umesh Ghoshdastider and Banani Saha. Genomecompress: A novel algorithm for dna compression, 2007.

[30] Jeff Gilchrist. Parallel data compression with bzip2, 01 2004.

[31] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine learning and systems*, 2024.

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

[33] Gzip. `https://www.gzip.org/`. Online.

[34] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, Katrin Heitmann, Kalyan Kumaran, Venkatram Vishwanath, Tom Peterka, Joe Insley, et al. HACC: extreme scaling and performance across diverse architectures. *Communications of the ACM*, 60(1):97–104, 2016.

[35] T.J. Hacker, B.D. Athey, and B. Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 10 pp–, 2002.

[36] Mikel Hernaez, Dmitri Pavlichin, Tsachy Weissman, and Idoia Ochoa. Genomic data compression. *Annual Review of Biomedical Data Science*, 2:19–37, 2019.

[37] Xu Hongxin. DNBSEQT7 WES-PE150 demo data. `https://db.cngb.org/search/project/CNP0003660/`, 10 2022.

[38] Xu Hongxin. MGISEQ-200 WES PE100 demo data. `https://db.cngb.org/search/project/CNP0003664/`, 11 2022.

[39] Zhi-An Huang, Zhenkun Wen, Qingjin Deng, Ying Chu, Yiwen Sun, and Zexuan Zhu. Lw-fqzip 2: a parallelized reference-based compression of fastq files. *BMC bioinformatics*, 18:1–8, 2017.

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

[41] Hurricane ISABELA Simulation Datasets. `http://vis.computer.org/vis2004contest/data.html`.

[42] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[43] 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 *28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, page 159–170, New York, NY, USA, 2019. Association for Computing Machinery.

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

[45] Daniel C Jones, Walter L Ruzzo, Xinxia Peng, and Michael G Katze. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic acids research*, 40(22):e171–e171, 2012.

[46] JE Kay, C Deser, A Phillips, A Mai, C Hannay, G Strand, JM Arblaster, SC Bates, G Danabasoglu, J Edwards, et al. 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.

[47] Carl Kingsford and Rob Patro. Reference-based compression of short-read sequences using path encoding. *Bioinformatics*, 31(12):1920–1928, 02 2015.

[48] Divon Lan, Ray Tobler, Yassine Souilmi, and Bastien Llamas. Genozip: a universal extensible genomic data compressor. *Bioinformatics*, 37(16):2225–2230, 02 2021.

[49] Myoungkyu Lee and Robert D. Moser. Direct numerical simulation of turbulent channel flow up to $Re_\tau \approx 5200$. *Journal of Fluid Mechanics*, 774:395–415, jul 2015.

[50] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Process-

ing Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 06 2009.

[51] Sihuan Li, Sheng Di, Kai Zhao, Xin Liang, Zizhong Chen, and Franck Cappello. Resilient error-bounded lossy compressor for data transfer. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[52] Xin Liang, Sheng Di, Sihuan Li, Dingwen Tao, Bogdan Nicolae, Zizhong Chen, 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*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.

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

[54] 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*. IEEE, 2018.

[55] 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 *IEEE International Conference on Cluster Computing*, pages 1–11, 2019.

[56] Xin Liang, Ben Whitney, Jieyang Chen, Lipeng Wan, Qing Liu, Dingwen Tao, James

Kress, Dave Pugmire, Matthew Wolf, Norbert Podhorszki, and Scott Klasky. Mgard+: Optimizing multilevel methods for error-bounded scientific data reduction, 2020.

[57] 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. *IEEE Transactions on Big Data*, pages 1–14, 2022.

[58] Linac Coherent Light Source (LCLS-II). `https://lcls.slac.stanford.edu/`, 2017. Online.

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

[60] 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.

[61] 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 *IEEE International Conference on Cluster Computing*, pages 294–306, 2021.

[62] Jinyang Liu, Sheng Di, Kai Zhao, Xin Liang, Zizhong Chen, and Franck Cappello. Dynamic quality metric oriented error bounded lossy compression for scientific datasets. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22. IEEE Press, 2022.

[63] Yuanjian Liu, Sheng Di, Kyle Chard, Ian Foster, and Franck Cappello. Optimizing Scientific Data Transfer on Globus with Error-Bounded Lossy Compression . In *2023*

IEEE 43rd International Conference on Distributed Computing Systems (ICDCS), pages 703–713, Los Alamitos, CA, USA, July 2023. IEEE Computer Society.

[64] Yuanjian Liu, Sheng Di, Kai Zhao, Sian Jin, Cheng Wang, Kyle Chard, Dingwen Tao, Ian Foster, and Franck Cappello. Understanding Effectiveness of Multi-Error-Bounded Lossy Compression for Preserving Ranges of Interest in Scientific Analysis . In *2021 7th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-7)*, pages 40–46, Los Alamitos, CA, USA, November 2021. IEEE Computer Society.

[65] Yuanjian Liu, Sheng Di, Kai Zhao, Sian Jin, Cheng Wang, Kyle Chard, Dingwen Tao, Ian Foster, and Franck Cappello. Optimizing multi-range based error-bounded lossy compression for scientific datasets. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 394–399, 2021.

[66] Yuanjian Liu, Sheng Di, Kai Zhao, Sian Jin, Cheng Wang, Kyle Chard, Dingwen Tao, Ian Foster, and Franck Cappello. Optimizing error-bounded lossy compression for scientific data with diverse constraints. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4440–4457, 2022.

[67] Yuanlai Liu, Zhengchun Liu, Rajkumar Kettimuthu, Nageswara Rao, Zizhong Chen, and Ian Foster. Data transfer between scientific facilities – bottleneck analysis, insights and optimizations. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 122–131, 2019.

[68] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 38–56, 2024.

[69] Matthew V. Mahoney. The zpaq compression algorithm, 2015. https://api.semanticscholar.org/CorpusID:13248511.

[70] Santiago Marco-Sola, Jordan M Eizenga, Andrea Guarracino, Benedict Paten, Erik Garrison, and Miquel Moreto. Optimal gap-affine alignment in O(s) space. *Bioinformatics*, 39(2):btad074, 02 2023.

[71] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, 37(4):456–463, 09 2020.

[72] Reverse Time Migration. Online.

[73] Miranda. https://wci.llnl.gov/simulation/computer-codes/miranda.

[74] NYX simulation. https://amrex-astro.github.io/Nyx. Online.

[75] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132, 2024.

[76] QMCPack. https://qmcpack.org/. Online.

[77] Md Hasanur Rahman, Sheng Di, Kai Zhao, Robert Underwood, Guanpeng Li, and Franck Cappello. A feature-driven fixed-ratio lossy compression framework for real-world scientific datasets. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1461–1474, 2023.

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

[79] A.H. Robinson and C. Cherry. Results of a prototype television bandwidth compression scheme. *Proceedings of the IEEE*, 55(3):356–364, 1967.

[80] Muhammad Sardaraz and Muhammad Tahir. Sca-ngs: Secure compression algorithm for next generation sequencing data using genetic operators and block sorting. *Science Progress*, 104(2):00368504211023276, 2021. PMID: 34143692.

[81] Muhammad Sardaraz, Muhammad Tahir, Ataul Aziz Ikram, and Hassan Bajwa. Seq-compress: An algorithm for biological sequence compression. *Genomics*, 104(4):225–228, 2014.

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

[83] 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.

[84] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[85] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello. Fixed-PSNR lossy compression for scientific data. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 314–318, 2018.

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

[87] 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 *IEEE International Parallel and Distributed Processing Symposium (IEEE IPDPS)*, pages 1129–1139. IEEE, 2017.

[88] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. Fixed-PSNR Lossy Compression for Scientific Data . In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 314–318, Los Alamitos, CA, USA, September 2018. IEEE Computer Society.

[89] 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.

[90] Robert Underwood, Julie Bessac, David Krasowska, Jon C Calhoun, Sheng Di, and Franck Cappello. Black-box statistical prediction of lossy compression ratios for scientific data. *The International Journal of High Performance Computing Applications*, 37(3-4):412–433, 2023.

[91] Robert Underwood, Chunhong Yoon, Ali Gok, Sheng Di, and Franck Cappello and. Roibin-sz: Fast and science-preserving compression for serial crystallography. *Synchrotron Radiation News*, 36(4):17–22, 2023.

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

[93] Xin-Chuan Wu, Sheng Di, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. Amplitude-aware lossy compression for quantum circuit simulation, 2018.

[94] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. Full-state quantum circuit simulation by us-

ing data compression. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[95] Yuting Xing, Gen Li, Zhenguo Wang, Bolun Feng, Zhuo Song, and Chengkun Wu. Gtz: a fast compression and cloud transmission tool optimized for fastq files. *BMC bioinformatics*, 18(16):233–242, 2017.

[96] Guillermo Dufort y Álvarez, Gadiel Seroussi, Pablo Smircich, José Sotelo-Silveira, Idoia Ochoa, and Álvaro Martín. Renano: a reference-based compressor for nanopore fastq files. *bioRxiv*, 2021.

[97] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*, 2024.

[98] Esma Yildirim, JangYoung Kim, and Tevfik Kosar. How GridFTP pipelining, parallelism and concurrency work: A guide for optimizing large dataset transfers. In *SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 506–515, 2012.

[99] Xiaodong Yu, Sheng Di, Kai Zhao, Jiannan Tian, Dingwen Tao, Xin Liang, and Franck Cappello. Ultrafast error-bounded lossy compression for scientific datasets. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '22, page 159–171, New York, NY, USA, 2022. Association for Computing Machinery.

[100] C. S. Zender. Bit grooming: statistically accurate precision-preserving quantization with compression, evaluated in the netCDF operators (NCO, v4.4.8+). *Geoscientific Model Development*, 9(9):3199–3211, 2016.

[101] Jialing Zhang, Xiaoyan Zhuo, Aekyeung Moon, Hang Liu, and Seung Woo Son. Efficient encoding and reconstruction of HPC datasets for checkpoint/restart. In *Proceedings of the 35th International Conference on Massive Storage Systems and Technology (IEEE MSST19)*, 2019.

[102] Yongpeng Zhang, Linsen Li, Yanli Yang, Xiao Yang, Shan He, and Zexuan Zhu. Lightweight reference-based compression of fastq data. *BMC bioinformatics*, 16:1–8, 2015.

[103] Chunchun Zhao. String correction using the damerau-levenshtein distance. *BMC Bioinformatics*, 06 2019.

[104] 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 *IEEE 37th International Conference on Data Engineering*, pages 1643–1654, 2021.

[105] 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.

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

[107] Weijian Zheng, Jun-Sang Park, Peter Kenesei, Ahsan Ali, Zhengchun Liu, Ian Foster, Nicholas Schwarz, Rajkumar Kettimuthu, Antonino Miceli, and Hemant Sharma. Rapid detection of rare events from *in situ* X-ray diffraction data using machine learning. *Journal of Applied Crystallography*, 57(4):1158–1170, Aug 2024.

[108] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving, 2024.

[109] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[110] Zstd. `https://github.com/facebook/zstd/releases`. Online.