FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data

Robert Underwood*†, Sheng Di†, Jon C. Calhoun‡, Franck Cappello†
*School of Computing Clemson University, Clemson, SC 29634
†Argonne National Laboratory, Lemont, IL 60439
‡Holcombe Department of Electrical and Computing Engineering
Clemson University, Clemson, SC 29634

***Abstract*—With ever-increasing volumes of scientific floating-point data being produced by high-performance computing applications, significantly reducing scientific floating-point data size is critical, and error-controlled lossy compressors have been developed for years. None of the existing scientific floating-point lossy data compressors, however, support effective fixed-ratio lossy compression. Yet fixed-ratio lossy compression for scientific floating-point data not only compresses to the requested ratio but also respects a user-specified error bound with higher fidelity. In this paper, we present FRaZ: a generic fixed-ratio lossy compression framework respecting user-specified error constraints. The contribution is twofold. (1) We develop an efficient iterative approach to accurately determine the appropriate error settings for different lossy compressors based on target compression ratios. (2) We perform a thorough performance and accuracy evaluation for our proposed fixed-ratio compression framework with multiple state-of-the-art error-controlled lossy compressors, using several real-world scientific floating-point datasets from different domains. Experiments show that FRaZ effectively identifies the optimum error setting in the entire error setting space of any given lossy compressor. While fixed-ratio lossy compression is slower than fixed-error compression, it provides an important new lossy compression technique for users of very large scientific floating-point datasets.**

I. Introduction

Today's scientific research applications produce volumes of data too large to be stored, transferred, and analyzed efficiently because of limited storage space and potential bottlenecks in I/O systems. Cosmological simulations [1], [2], for example, may generate more than 20 PB of data when simulating 1 trillion particles over hundreds of snapshots per run. Climate simulations, such as the Community Earth Simulation Model (CESM) [3], may produce hundreds of terabytes of data [4] for each run.

Effective data compression methods have been studied extensively. Since the major scientific floating-point datasets are composed of floating-point values, however, lossless compressors [5]–[7]cannot effectively compress such datasets because of high entropy of the mantissa bits. Therefore, error-bounded lossy compressors have been widely studied because they not only significantly reduce the data size but also prevent data distortion according to a user's specified error bound. Most existing lossy compressors consider the error bound preeminent and endeavor to improve the compression ratio and performance as much as possible subject to the error bound.

However, many scientific application users have requirements for the compression ratio. These requirements are determined by multiple factors such as the capacity of the assigned storage space, I/O bandwidth, or desired I/O performance. Hence, these users desire to perform fixed-ratio lossy compression—that is, compressing data based on the required compression ratio instead of only strictly respecting user's error bound. In this case, the lossy compressor needs to adjust the error bound to respect the target user-specified compression ratio, while minimizing the data distortion. The user can also provide additional constraints regarding the data distortion (such as maximum error bound) to guarantee the validity of the results from reconstructed data. While fixed-ratio compression can be obtained by simply truncating the mantissa of the floating-point numbers, this approach may not respect the user's diverse error constraints. With such additional constraints, the lossy compressor should make the compression ratio approach the expected level as closely as possible, while strictly respecting the data distortion constraints.

In this paper, we propose a generic, efficient fixed-ratio lossy compression framework, FRaZ, that is used to determine the error settings accurately for various error-controlled lossy compressors, given the particular target compression ratio with a specific scientific floating-point dataset. Our design involves two critical optimization strategies. First, we develop a global optimum searching method by leveraging Davis King's global minimum finding algorithm [8] to determine the most appropriate error setting based on the given compression ratio and dataset in parallel. Second, our parallel algorithm optimizes the parameter searching performance by splitting the search range into distinct regions, parallelizing on file, and—in the offline case—by time-step.

Constructing a generic, high-fidelity framework for fixed-ratio lossy compression poses many research challenges. First, as we elaborate in Section V, the relationship between error bounds and compression ratios is not always monotonic because of the use of dictionary encoder phases in some

compressors such as SZ. Second, since we aim to create a generic framework such that more compressors can be included in the future, we cannot utilize properties of the specific compressors we use to optimize performance as has been done in prior work [9], [10] and must instead treat the compression algorithm as a black box. This means that our algorithm cannot take advantage of properties induced by block size or expected behavior induced for a particular data distribution. Third, since we treat the compressors as a black box, we must carefully study how to modify existing algorithms to minimize the calls to the underlying compressors and orchestrate the search in parallel, in order to have a tool that is useful to users while working around the limitations of the various current and potential future compressors.

We perform the evaluation for our framework based on the latest versions of state-of-the-art lossy compressors (including SZ [11]–[13], ZFP [14], and MGARD [15]), using well-known real-world scientific floating-point datasets from the public Scientific Data Reduction Benchmark (SDRBench) [16]. We perform the parallel performance evaluation on Argonne's Bebop supercomputer [17] with up to 416 cores. Experiments show that our framework can determine the error setting accurately within the user-tolerable errors based on the target compression ratios, with very limited time overhead in real-world cases.

The remainder of the paper is organized as follows. In Section II, we introduce the background of this research regarding various state-of-the-art lossy compressors, and we present several examples about user's requirement on specific compression ratios. In Section III, we compare our work with the related work from fixed-rate compression, image processing, and signal processing. In Section IV, we present a formal problem formulation to clarify our research objective. In Section V, we describe our design and our performance optimization strategies. In Section VI, we present the evaluation results. In Section VII, we present or conclusions and end with a vision of future work.

II. Background

In this section, we describe the research background, including the existing state-of-the-art error-controlled lossy compressors and fixed-ratio use cases.

A. Error-Bounded Lossy Compression

1) SZ: SZ has been widely evaluated in the scientific floating-point data compression community [18]–[21] , showing that it is one of the best compressors in its class.

SZ is designed based on a blockwise prediction-based compression model. It splits each dataset into many consecutive non-overlapped blocks (such as $6\times6\times6$ for a 3D dataset) and performs compression in each block. It includes four key steps:

- *Step 1: data prediction.* SZ adopts a hybrid data prediction method (either a 1-layer Lorenzo predictor [22] or linear regression method) to predict each data point by its neighboring values in the multidimensional space.
- *Step 2: linear-scaling quantization.* Each floating-point data value is converted to an integer number in terms of the formula $quantization\_code = \frac{predicted\_value - true\_value}{2\epsilon}$, where ϵ refers to the user-specified error bound (i.e., linear-scaling quantization).
- *Step 3: entropy encoding.* A Huffman encoding algorithm customized for integer code numbers is then applied to the quantization codes generated by Step two.
- *Step 4: dictionary encoder.* A dictionary encoder such as Gzip [7] or Zstd [6] is used to significantly reduce the Huffman-encoded bytes generated from Step three.

SZ allows one to set an absolute error bound to control the data distortion in the compression.

2) ZFP: ZFP [14] is another outstanding error-controlled lossy compressor and is also broadly assessed and used in the scientific floating-point data compression community. ZFP transforms floating-point data to fixed-point values block by block (block size is $4\times4\times4$ for 3D datasets) and adopts an embedded coding to encode the generated coefficients.

ZFP also provides an absolute error bound to control the data distortion. Although ZFP provides another fixed-rate compression mode, allowing users to do the data compression based on a given compression ratio, the compression quality is significantly worse than the absolute error-bound mode. We demonstrate this in Section VI-B4. Thus, how to efficiently fix compression ratio based on absolute error-bound mode is critical to ZFP.

3) MGARD: MultiGrid Adaptive Reduction of Data (MGARD) [15] is an error-controlled lossy compressor supporting multilevel lossy reduction of scientific floating-point data. MultiGrid is designed based on the theory of multigrid methods [23]. An important feature of MGARD is providing guaranteed, computable bounds on the loss incurred by the data reduction. MGARD provides different types of norms, such as infinity norm and L2 norm, to control the data distortion. The infinity norm is equivalent to the absolute error bound, and the L2 norm mode can be used to control the mean squared error (MSE) during the lossy compression.

Although SZ, ZFP, and MGARD provide advanced features to control the distortion of lossy compression, none of them provide high-fidelity fixed-ratio compression.

Accuracy of EBLC is dictated at compression time by selection of an error bound and error bounding type — e.g., absolute, relative, number of bits and are selected to minimize impact on quantities of interest in scientific simulations. For use cases that preform data analytics on lossy compressed data, trial-and-error is often used to identify acceptable compression tolerances [24]. The trial-and-error is often done offline to ensure that the selected error bound is robust for multiple time-steps and does diminish the quality of the analysis. However, if the lossy compressed data is used to advance the simulation the simulation trial-and-error is possible [25], but recent works have explored the relation of compression error to numerical errors present in the simulation and provide strategies on error tolerance selection [21], [26], [27].

B. Fixed-Ratio Use Cases

In this subsection, we describe several fixed-ratio use cases to demonstrate the practical demands on the fixed-ratio compression by real-world application users.

The first use case is significant reduction of storage footprint. On the ORNL Summit system, for example, the capacity of the storage space is limited to 50 TB for each project by default. Many scientific floating-point simulations (such as the CESM climate simulation and HACC cosmological simulation) may produce hundreds of terabytes of data in each run (or even over 1 PB of data), such that the compression ratio has to be 10:1 or higher to avoid execution crash due to no space being left on storage. Even if a larger storage allocation is awarded or purchased at considerable financial cost, projects generating extreme volumes often face the need to reduce their storage footprint in order to make room for their next executions. Fixed-rate compression provides the ability to store multiple simulations given a fixed amount of storage but suffers from large inaccuracies in the data which high compression ratios are required (see Figure 10).

The second practical use case explores best-fit lossy compression solutions based on the user's post-analysis requirement (such as visual quality or specific analysis property) by at fixed compressed sizes. None of the existing error-controlled lossy compressors provide the fixed-ratio compression mode, however, and therefore users have to seek the best-fit choice by conducting inefficient trial-and-error strategies with different error settings for each compressor to achieve a target compression ratio. Furthermore, there is no universal model that accurately predicts compression ratio based on compressor configuration for a variety of input data.

The third practical use case involves the matching of I/O bandwidth constraints and accelerating the I/O performance. Advanced light-source instruments, such as the Advanced Photon Source and Linac Coherent Light Source (LCLS-II), may generate image data at an extremely high acquisition rate, such that the raw data cannot be stored efficiently for post-analysis because of limited I/O bandwidth. Specifically, LCLS-II is producing instrument data with up to 250 GB/s while the corresponding storage bandwidth is only 25 GB/s. Thus the designers of LCLS-II expect to reduce the data size with a compression ratio of 10 or higher [28]. Spring8 [29] researchers also indicate that their data could be generated with 2 TB/s, which is expected to be reduced to 200 GB/s after data compression.

We note that users often require random-access decompression across time steps, which means that they prefer to be able to decompress the data individually at each time-step because decompressing the whole dataset with all time-steps requires a significant amount of time or is impossible because of memory allocation limits.

III. Related Work

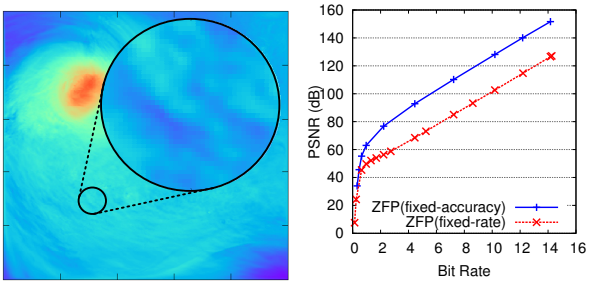
In the compression community, the similar type of compression is called "fixed-rate compression," where the *rate* here refers to the bit rate, which is defined as the number of bits used to represent one symbol (or data point) on average after compression. The lower the bit rate, the higher the compression ratio. Hence, fixing the bit rate means fixing the compression ratio. In the remainder of this section, we compare our work with prior work in these areas.

In addition to the fixed-accuracy modes (i.e., accuracy and precision error-bounding modes), ZFP offers a fixed-rate mode [14]. The fixed-rate mode of ZFP offers precise control over the number of bits per symbol in the input data. It operates by transforming the data into a highly compressible domain and truncating each symbol to reach the appropriate rate. However, the fixed rate mode of ZFP is not error bounded, and it suffers from significantly lower compression quality than does the fixed-accuracy mode of ZFP. Figure 1 (b) demonstrates the compression quality of ZFP using fixed-accuracy mode and fixed-rate mode, respectively. One can clearly see that the latter exhibits much worse rate distortion than does the former (up to 30 dB difference with the same bit-rate in most cases). Rate distortion is an important indicator to assess the compression quality. Its detailed definition can be found in Section VI-B (4). Figures 1 (c) and (d) clearly show that the fixed-rate mode results in much lower visual quality (e.g., more data loss) than the fixed-accuracy mode with the same compression ratio, 50:1 in this example. In absolute terms, the fixed-accuracy mode leads to much higher peak signal-to-noise ratios (PSNR) and lower auto-correlation of compression errors (ACF(error)), which means better compression quality than the fixed-rate mode. In ZFP's website and user guide, the developer of ZFP also points out that the fixed rate mode is not recommended unless one needs "to bound the compressed size or need random access to blocks" [14].

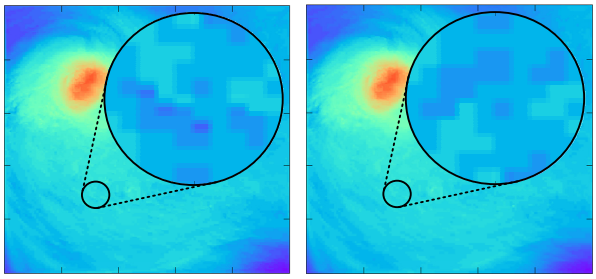
In contrast, not only does our framework fix the compression ratio but it also achieves higher compression quality for different compressors (such as SZ, ZFP, and MGARD) based on their error-bounding mode. Additionally, it can provide random access to the same level as can ZFP's fixed rate mode when supported by the underlying compressor. Since our framework utilizes a control loop to bound the compression ratio, it may suffer a lower bandwidth than ZFP's fixed-rate mode to a certain extent. The tradeoff for this lower bandwidth is compressed data of far higher quality for the same compression ratio, which we demonstrate in detail in the evaluation section.

The literature also includes studies investigating the use of fixed-ratio compressors for images. One such work [30] is JPEG-LS, a fixed-ratio compressor for images. This work adopts a combination of a prediction system for data values and two runs of Golumb-Rice encoding to encode RGB values for images. The first run of the Golumb-Rice encoding is used to estimate the quantization level used in the the second run of the encoder. Golumb-Rice assumes integer inputs, whereas our work is applicable on all numeric inputs.

Some work also has been done on fixed-ratio compression in digital signal processing [31] In this domain, adaptive sampling techniques are used to maintain a budget for how many points to transmit. When a point is determined to provide



(a) original raw data (b) Rate Distortion



(c) ZFP(fixed-accuracy) (d) ZFP(fixed-rate)

Fig. 1. Data distortion of ZFP in fixed-accuracy mode and fixed-rate mode (Hurricane TCf field) with CR=50:1; (ZFP.fixed-accuracy: PSNR=55.3, max error=4.2, SSIM=0.94, ACF(error)=0.67; ZFP.fixed-rate: PSNR=45.4, max error=33.7, SSIM=0.94, ACF(error)=0.72)

new information (using a predictor, interpolation scheme, or some other method), it is transmitted and the budget is expended as long as there is remaining budget. If the budget is spent, then no points are transmitted until the budget is refilled. Over time, the budget is increased to keep the rate constant. In contrast, our work does not rely on a control loop to maintain the error budget, so can look at the data holistically to decide where to place the loss in our signal, allowing for more accurate reconstructions.

IV. Problem Formulation

In this section, we formulate the research problem, by clarifying the inputs, constraints, and the target of our research.

Before describing the problem formulation, however, we introduce some related notations as follows. Given a specific field f at time-step t of an application, we denote the dataset by $D_{f,t} = \{d_1, d_2, \ldots, d_n\}$, where d_i refers to the original value of data point i in the dataset and n is the number of elements. We denote its corresponding decompressed dataset by $D'_{f,t}= \{d'_1, d'_2, \cdots, d'_n\}$, where d'_i refers to the reconstructed value after the decompression. We denote the original data size and compressed data size by $s(D_{f,t})$ and $s(D'_{f,t})$, respectively. The compression ratio (denoted by ρ) then can be written as $\rho(D_{f,t}) = \frac{s(D_{f,t})}{s(D'_{f,t})}$. Moreover, we denote the target compression ratio specified by the users as $\rho_t(D_{f,t})$, and the real compression ratio after the compression as $\rho_r(D_{f,t}, e)$.

The fixed-ratio lossy compression problem is formulated as follows, based on whether it is subject to an error-control constraint or not.

- *Nonconstrained fixed-ratio compression*: The objective of the nonconstrained fixed-ratio lossy compression is to confine the real compression ratio to be around the target compression ratio within a user-specified tolerable error (denoted by ϵ), as shown below.
$$\rho_t(D_{f,t}) - \epsilon \leq \rho_r(D_{f,t}) \leq \rho_t(D_{f,t}) + \epsilon \quad (1)$$
- *Error-control-based fixed-ratio compression*: The objective of the error-control-based fixed-ratio compression is to tune the compression ratio to be within the acceptable range $[\rho_t(D)-\epsilon, \rho_t(D)+\epsilon]$, while respecting the user-specified error bound (denoted by e), as shown below.
$$\begin{aligned} &\rho_t(D_{f,t}) - \epsilon \leq \rho_r(D_{f,t}, e) \leq \rho_t(D_{f,t}) + \epsilon \\ &s.t.\ \Pi(D_{f,t}, D'_{f,t}) \leq e, \end{aligned} \quad (2)$$
where $\Pi(D_{f,t}, D'_{f,t})$ is a function of error control. For instance, $\Pi(D_{f,t}, D'_{f,t})=\max_i |d_i - d'_i|$ for the absolute error bound, and $\Pi(D_{f,t}, D'_{f,t})=\sum_{d_i \in D_{f,t}, d'_i \in D'_{f,t}} (d_i - d'_i)^2$ for the mean squared error bound.

We summarize the key notation in Table I.

TABLE I
Table of Key Notation

Notation	Description
D	original data set for all time-steps and fields
D_f	original data set for all time-steps of a particular field
$D_{f,t}$	original data set for a particular field and time-step
$D'_{f,t}$	decompressed data set for a particular field and time-step
ρ_t	target compression ratio
ρ_r	real compression ratio
ϵ	acceptable error for ρ_t
e	error bound for compression
γ	maximum value of the loss function
θ	fixed parameters of the compressor
N	the number of dimensions
n	the total number of data points
T	the number of time-steps
α	the degree overlap between error-bound search ranges
U	maximum allowed compression error

V. Design and Optimization

In this section, we present the design of our fixed-ratio lossy compression framework and optimization strategies.

A. Design Overview

Figure 2 shows the design overview with highlighted boxes indicating our major contributions in this paper and the relationship among different modules in the framework. As shown in the figure, our FraZ framework is composed of five modules, and the optimizing autotuner and parallel orchestrator are the core modules. They are, respectively, in charge of (1) searching for the optimal error setting based on the target compression ratio with few iterations and (2) parallelizing the overall tuning job involving different searching spaces for each field and different time-steps and across various fields. We develop an easy-to-use library (called Libpressio [32]) to build a middle layer for abstracting the discrepancies of the APIs of different compressors.

We list our major contributions as follows:

1) Formulated fixed-ratio compression as an optimization problem in a way that converges quickly without resorting to multiobjective optimization

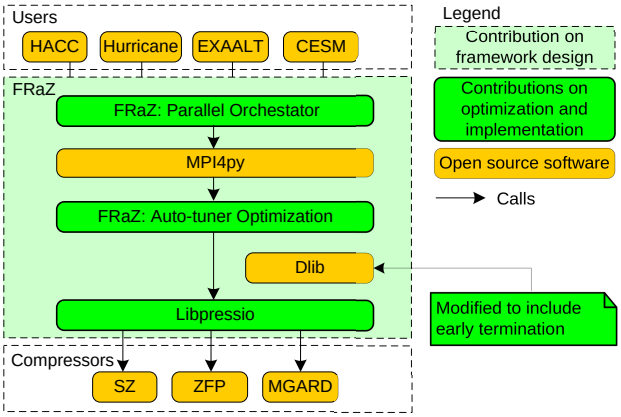


Fig. 2. Design overview and summary of our contributions

2) Evaluated several different optimization algorithms to find one that works on all of our test cases, and then modified it to improve performance for our FRaZ
3) Implemented and ran parallel search to improve the throughput of the technique

B. Autotuning Optimization

In this subsection, we describe our autotuning solution in detail, which includes three critical parts: (1) exploration of the initial optimization methods, (2) construction of a loss function, and (3) improvements to the optimization algorithm that involves how to deal with infeasible target compression ratio requirement and determines the exact error-bound setting.

1) Exploration of Initial Optimization Methods: In this subsection we describe how we choose which optimization method to use as a starting point for later refinement.

Before detailing FRaZ's optimizing autotuning method, we first analyze why the straightforward binary search is not suitable for our case. On the one hand, the application datasets may exhibit a non-monotonic compression ratio increase with error bounds. We present a typical example in Figure 3, which uses SZ to compress the QCLOUDf field of the hurricane simulation dataset. We can clearly see that the compression ratios may decrease significantly with larger error bounds in some cases. We also observe the spiky changes in the compression ratios with increasing error bounds on other datasets (not presented here due to space limit). The reason is that SZ needs to use decompressed data to do the prediction during the compression, which may cause unstable prediction accuracy. Moreover, SZ's fourth stage (dictionary encoder) may find various repeated occurrences of bytes based on output of the third stage, because a tiny change to the error bound may largely affect the Huffman tree constructed in the third phase of SZ. By comparison, our autotuning search algorithm is a general-purpose optimizer and takes into account the irregular relationship between compression ratios and error bounds. On the other hand, even on the datasets where monotonicity holds, binary search may still be slower than FRaZ's optimizing autotuner. For example, when searching for the target compression ratio 8:1 at the 48^{th} time-step on the Hurriane-CLOUD field, our method requires only 6 iterations to converge to an acceptable solution, whereas binary search needs 39 iterations. The reason is that binary search may spend substantial time searching small error bounds, which would not result in an acceptable solution because it climbs from the minimum possible error bound to the user-specified upper limit.

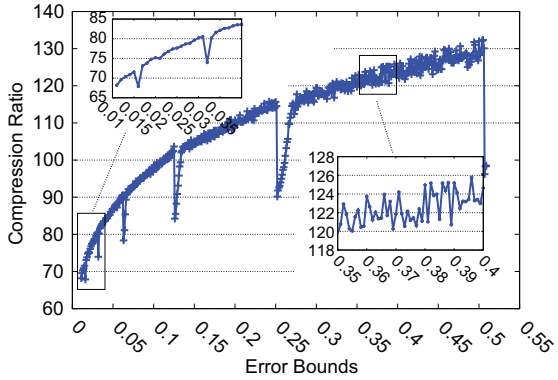


Fig. 3. Example based on the hurricane simulation dataset (field: QCLOUDf.log10) showing that the relationship between error bounds and compression ratios is not always monotonic

When developing FRaZ's optimizing autotuner, we considered a number of different techniques to perform the tuning. Since we are developing a generic method, we cannot construct a general derivative function that relates the change in error bound to the change in compression ratio. Therefore, we need to decide between methods that use numerical derivatives and derivative-free optimization because the derivative of the compression ratio with respect to the error bound is unknown. The methods using numerical derivatives approximate the slope of the objective function by sampling nearby points. Some methods that fall in this category are gradient descent (i.e., Newton-like methods such as [33] and ADAM [34]). However, when evaluating an error bound to determine the compression ratio, we must run the compressor since we are using the compressors as black boxes, which may take a substantial amount of compared with the optimization problem. In this sense, numerical derivative-based methods are too slow.

We therefore turned our consideration to derivative-free optimization. We considered methods such as BOBYQA [35], but they do not handle a large number of local optimums. This ability is essential for developing a robust tuning framework for lossy compression because many of the functions that relate error bounds to compression ratios look like the plot on the left of Figure 4: a step-like function with perhaps a slight upward slope on each step. In practice, we noted that it is easily able to escape the local optima in these functions.

We also took into account a variety of implementations of these algorithms, such as the ones in [8], [36], [37]. We decided between these libraries using three criteria: (1) correctness of the result, (2) time to solution, and (3) modifiability and readability of code. Ultimately we started with a black-box optimization function called `find_global_min` from the commonly used Dlib library from which we make

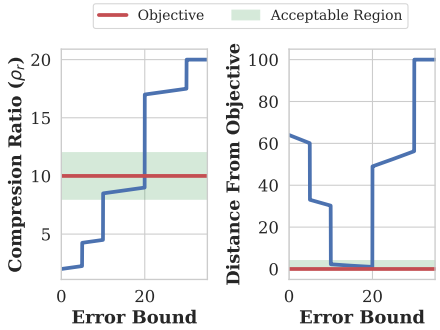


Fig. 4. Illustration of autotuning optimization function: On the left is a hypothetical relationship between an error-bound level and compression ratio for some compressor and dataset. The target compression ratio is marked as a red line, and the acceptable region is colored green. On the right is the corresponding loss function using our method. The green area above the target compression ratio refers to the acceptable region. In this case, where there are blue points in the acceptable region, we call the result feasible. If the acceptable region was below the blue points, we would call it infeasible.

our modifications [8]. The global-minimum-finding algorithm designed by Davis King that combines the works of [38] and [39]. It requires a deterministic function that maps from a vector to a scalar, a vector of lower bounds, and a vector of upper bounds as inputs. At a high level the algorithm works as follows. It begins with a randomly chosen point between the upper and lower bounds. Then, it alternates between a point chosen by the model in [38], which approximates the function by using a series of piecewise linear functions and chooses the global minimum of this function, and the model in [39], which does a quadratic refinement of the lowest valley in the model. According to [8], this method performs well on functions with a large number of local optimums, and this performance was confirmed by our experience.

2) Construction of Loss Function: Now that we have an optimizer framework, we need to construct a loss function. First, we created a closure for each compressor, $\rho_r(D_{f,t}, e)$ that transformed its interface including a dataset D and parameters θ in a function accepting only the error bound e. To create the closure, we developed libpressio [32]—a generic interface for lossy compressors that abstracts between their differences so that we could write one implementation of the framework for SZ, ZFP, and MGRAD.

To convert this to a loss function, we chose the distance between the measured compression ratio and target compression ratio $\rho_r(D_{f,t}, e) - \rho_t(D_{f,t})$. Now, the function that relates an error bound to a compression ratio is an arbitrary function that may or may not have a global or local optimum. Therefore, we transformed the function by applying a clamped square function (i.e. $\min\left(x^2, \gamma\right)$, where γ is equal to 80% of the maximum representable double using IEEE 754 floating-point notation). This maps the possible range of the input function from the range $(-\infty, \infty)$ to the range $[0, \gamma]$. The benefits of this are twofold. First, the function now has a lowest possible global minimum we can optimize for. Second, the function now has a highest possible value that avoids a bug in the Dlib `find_global_min` function that causes a segmentation fault. We also considered the function $\min\left(|x|, \gamma\right)$, but found that the quadratic version converged faster. This leaves us with the final optimization function $l(e) = \min\left(\left(\rho_r(D_{f,t}, e) - \rho_t(D_{f,t})\right)\right)^2, \gamma\Big)$.

3) Development of Worker Task Algorithm: Our next insight was that often the exact match of the compression ratio is not always feasible and is neither desired nor required. It may not always be feasible because for some compressors, for example ZFP's accuracy mode, the function that maps from the error bound to the compression ratio is a step function, such that not all compression ratios are feasible. In addition, it may not always be desired or required because the user might accept a range of compression ratios and prefer finding a match quickly rather than waiting for a more precise match.

Looking again at Figure 4, we see a typical relationship between an error bound and the compression ratio. If the user asks for a compression ratio of 15, no error bound would satisfy that request using this compressor. In contrast, FRaZ will return the closest point that it observes to the target; in the case of Figure 4 it would report an error bound that results in a compression ratio near 17.5. Depending on the user's global error tolerance, this value near 17.5 may or may not be within the user's acceptable region, meaning it may or may not be a feasible solution.

Another case that the solution may be infeasible is when needed error bound required to meet the objective is above the user's specified upper error bound, U. In this case, FRaZ will report the error bound that resulted in the closest that it observed to the target compression ratio, and the user can run FRaZ again with the default upper bound, which is equal to the maximum allowed level of an error bound by the compressor. If FRaZ identifies a solution in this case, the user can evaluate whether to relax the perhaps overly strict error tolerance to meet the objective or decide that the fidelity of the results is more important and that the bound cannot be relaxed. Alternatively, the user can try a different compressor backend that implements the same error bound.

In fact, determining the exact error bound that produces a specified compression ratio may not be desired or required. The reason is that a large number of iterations may be needed in order to converge to an error bound, and the user would rather trade time for accuracy. Therefore, we implemented a version of Dlib's `find_global_min` that implements a global cutoff parameter $\epsilon \in [0, 1]$. Specifically, we allow the algorithm to terminate if the result of the optimization function results in a value in the range: $[0, \epsilon^2 \rho_t(D_{f,t})^2]$. This has a substantial impact on the performance on the typical case.

We combine these insights into our worker task algorithm, as shown in Algorithm 1.

C. Parallelism Scheme

After optimizing the serial performance via the design of the optimization algorithm, we develop a parallel optimization method using Dlib's built-in multithreaded optimization mode. Some compressors (such as SZ and MGARD) do not support being run with different settings in a multithreaded context because of the use of global variables. In this situation, we

Algorithm 1 WORKER TASK

Input: target ratio ($\rho_t(D_{f,t})$), acceptable error ϵ, dataset $D_{f,t}$, prediction p, region's lower bound l, region's upper bound u
Output: real compression ratio $\rho_r(D_{f,t}, e)$, recommended error bound setting e

1: **if** $p \neq 0$ **then**
2: $\quad \rho_r(D_{f,t}, e) \leftarrow compress(D_{f,t}, p)$ /*If a prediction was provided, try it first.*/
3: **end if**
4: **if** $(1-\epsilon) * \rho_t(D_{f,t}) \leq \rho_r(D_{f,t}, e)) \leq (1+\epsilon) * \rho_t(D_{f,t})$ **then**
5: $\quad$ **return** $\rho_r(D_{f,t}), p$ /*terminate if $\rho_r(D_{f,t}, e)$ meets requirement.*/
6: **end if**
7: $\rho_r(D_{f,t}, e), e \leftarrow train\_with\_cutoff(D_{f,t}, l, u, \rho_t(D_{f,t}), \epsilon)$
8: **return** $\rho_r(D_{f,t}, e), e$

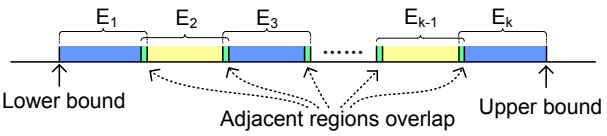


Fig. 5. Illustration of error bound ranges. We divide the range from lower bound to upper bound into K slightly overlapping regions ($E_1, E_2, \ldots E_K$). The overlap is a small fixed percentage of the width of the regions (i.e., 10%). Each region (E_1, E_2, ..., E_K) is then passed from the parallel orchestrator to the autotuning optimizer. Note that the ends E_1 and E_K are slightly smaller to preserve the error bound range.

can only treat each compression as a non-multithreaded task because we are developing a generic framework.

We use multiple processes based on MPI to parallelize the search by error bound range. Figure 5 provides an overview of our method. Rather than a serial search over the entire lower to upper bound range, we divide the range into k overlapping regions. We then give each of the k regions to separate MPI processes, and use Algorithm 2 to process them. As the processes complete, we test whether we have satisfied our objective subject to our global threshold ϵ (line 7–9). If so, we terminate all tasks that have not yet begun to execute (line 10–14). If a particular task finishes and we have not satisfied our objective, we do nothing. If all the tasks finish and we still have not met our objective, we conclude that the requested compression ratio is infeasible (line 18–25).

So why do we overlap the error bound regions? Overlapping the regions avoids extremely-long worst-case search time in the optimization algorithm. Since we terminate early once a solution is found, FRaZs runtime depends on the region containing the target. Without small overlapping, if the target error-bound coincides a border, its MPI_rank iterates longer lacking stationary-points for quadratic refinement.

To limit the effects of waiting on wrong guesses, we constrain the number of iterations to a maximum value. We considered limiting by time instead, but we were unable to find a heuristic that worked well across multiple datasets, fields, and time-steps. This is because the compression time is a function of the dataset size, the entropy of the data contained within, and properties of each compressor.

Limiting the amount of wasted computational resources is desirable. Since we are dividing on error-bound range, a small number of the searches (typically one) are expected to return successfully if the requested ratio is feasible. Additionally, there seems to be a floor for how many iterations are required

Algorithm 2 TRAINING

Input: target compression ratio $\rho_t(D_{f,t})$, acceptable error ϵ, dataset D_t, max allowed compression error U
Output: real compression ratio $\rho_r(D_{f,t}, e)$, recommended error bound setting e

1: $tasks[N]$
2: $done \leftarrow false$
3: **for** $(i, (l, u)) \in make\_error\_bounds(U)$ **do**
4: $\quad tasks[i] \leftarrow launch\_task(D_t, l, u, \rho_t(D_{f,t}), \epsilon, h)$
5: **end for**
6: **while** $notdone$ **do**
7: $\quad last\_task \leftarrow next\_completed(tasks)$
8: $\quad candiate \leftarrow compression\_ratio(last\_task)$
9: $\quad$ **if** $\rho_t(D_{f,t})(1-\epsilon) \leq candidate \leq \rho_t(D_{f,t})(1+\epsilon)$ **then**
10: $\quad\quad done \leftarrow true$
11: $\quad\quad$ **for** $task \in tasks$ **do**
12: $\quad\quad\quad cancel\_if\_not\_finished(task)$
13: $\quad\quad$ **end for**
14: $\quad$ **end if**
15: $\quad done \leftarrow has\_next(completed)$
16: **end while**
17: $\rho_r(D_{f,t}, e) = \infty$
18: **for** $task \in tasks$ **do**
19: $\quad$ **if** finished(task) **then**
20: $\quad\quad \rho \leftarrow compression\_ratio(task)$
21: $\quad\quad$ **if** $(\rho_r - \rho)^2 < (\rho_t - \rho)^2$ **then**
22: $\quad\quad\quad \rho_r = \rho$
23: $\quad\quad$ **end if**
24: $\quad$ **end if**
25: **end for**
26: **return** $\rho_r(D_{f,t}, e), error\_bound(task)$

to converge for a particular mode of a compressors. Hence, there is limited benefit to splitting into more than a few ranges, and cores could perhaps be more efficiently used for other fields. Preliminary experiments found that 12 tasks per a particular field and time-step dataset offered an ideal tradeoff between efficiency and runtime, and we set it as the default. The user can choose to use more tasks, however.

One can also perform additional optimization of multiple time-step data. Often, subsequent iterations in a large simulation do not differ substantially and have similar compression properties. Therefore we ran the first time-step as before, but then we assumed that the error bound found by the previous iteration was correct for the next full dataset. If our assumption proved correct, we continued on and skipped training. Otherwise, we reran the training and adopted the new trained solution for the next step. We then repeated this process over the remaining datasets. In practice, we retrained only a small percentage of the time. On the hurricane dataset, for example, we retrained only 4 times on the CLOUD field.

We also take advantage of the embarrassingly parallel nature of parallelizing by fields, as shown in Algorithm 3. The results show some additional speedup.

VI. PERFORMANCE AND QUALITY EVALUATION

In this section, we first describe our experimental setup, including hardware, software, and datasets. We then describe our evaluation metrics and results using five real-world scientific floating-point datasets on Argonne's Bebop supercomputer [17].

Algorithm 3 PARALLEL BY FIELD

Input: target ratio $\rho_t(D_{f,t})$, ϵ, dataset D, max allowed compression error U
Output: real compression ratio $\rho_r(D_{f,t}, e)$, recommended error bound setting e

```
for D_f ∈ D /*in parallel*/ do
    p ← 0
    for D_{f,t} ∈ D_f do
        ρ_r(D_{f,t}, e), e ← parallel_error_bound(D_t, ϵ, U)
        if (1 − ϵ) * ρ_t(D_{f,t}) ≤ ρ_r(D_{f,t}, e) ≤ (1 + ϵ) * ρ_t(D_{f,t}) then
            p ← e
        end if
    end for
end for
```

A. Experimental Setup

1) Hardware and Software Used for Evaluation: The hardware and software versions we used on the Bebop supercomputer [17] are given in Table II.

TABLE II
HARDWARE AND SOFTWARE VERSIONS USED

Hardware	Description
CPU	36 Core Intel Xeon E5-2695v4
MEM	128GB DDR4 Ram
NIC	Intel Omni-Path HFI Silicon 100 Series

Software	Description	Software	Description
OS	CentOS 7	SZ	2.1.7
CC/CXX	gcc/g++ 8.3.1	ZFP	0.5.5
MPI	OpenMPI 2.1.1	MGARD	0.0.0.2
Dlib	2.28	Singularity	3.0.2

We have packaged our software as a Singularity container for reproducibility.

2) Datasets used for Experiments: In our experiments, we evaluated our designed fixed-ratio lossy compression framework based on all three state-of-the-art compressors described in Section II, using five real-world scientific simulation datasets downloaded from scientific data reduction benchmark [16]. The raw data are all stored in the form of single-precision data type (32-bit floating point). We describe the five application datasets in Table III.

TABLE III
DATASET DESCRIPTIONS

Name	Domain	# Time-steps	Dim.	# Fields	Total size
Hurricane	Meteorology	48	3	13	59 GB
HACC	Cosmology	101	1	6	11 GB
CESM	Climate	62	2	6*	48 GB
Exaalt	Moledular Dyn.	82	1	3	1.1 GB
NYX	Cosmology	8	3	5	35 GB

* A limited number of fields had multi-time step data. Only fields for which multiple time step data were included.

We chose these datasets for a few reasons: First, they offer results over multiple time-steps, which matches well user's practical post-analysis with a certain simulation period. Second, the datasets use floating-point data which are often not served well by traditional lossless compressors. Third, the datasets are commensurate with the use cases of fixed-ratio compression described in Section II.

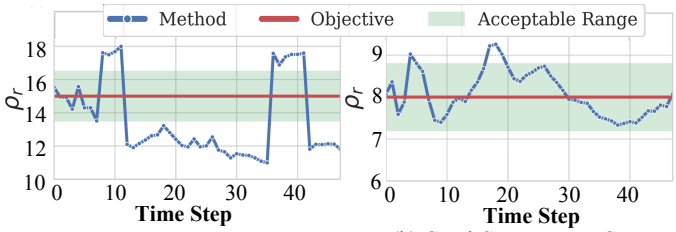
In some cases, we are not able to use all the datasets with all compressors. We run all the experiments for all datasets and compressors when possible. MGARD supports only 2d and 3d data so it is not tested on the HACC and Exaalt datasets. We adopt 6 typical fields for CESM application because other fields exhibit similar results with one of them (CLDHGH CLDLOW, CLOUD, FLDSC, FREQSH, PHIS). We generally noted similar results for each dataset and compressor.

B. Experimental Results

Over the course of our experiments, we evaluated four properties of FRaZ using the datasets from SDRBench [16]:

1) How close do we get to the target compression ratio when it is feasible?
2) How long does it take to find the target compression ratio or determine that it is infeasible?
3) How does the runtime of the algorithm scale as the number of cores increase?
4) How does FraZ compare with existing fixed-rate methods in terms of rate distortion and visual quality?

1) How close do we get to the target compression ratio?: How close we get to the target compression ratio depends heavily on whether the requested compression ratio is feasible for the underlying compressor used. Figure 6 (a) and Figure 6 (b) show a bad case and a good case, respectively.



(a) Bad Convergence Case (b) Good Convergence Case

Fig. 6. Demonstration of two types of convergence cases (Hurricane-CLOUD)

In Figure 6 (a), we see an example of where $\rho_t(D_{f,t})$ is infeasible for most time-steps for the CLOUD field. The early time-steps compress within the acceptable range, but by time-step ten the $\rho_t(D_{f,t}) = 15$ is no longer feasible. The reason is that as the time-steps progress, the properties of the dataset change, affecting the ability of compressor to compresses it at this level. As a result, we oscillate between a compression ratio that is larger and a compression ratio that is smaller. However, a larger tolerance (i.e., $\epsilon = .2$) would have allowed even this case to converge for all time-steps.

In Figure 6 (b), we see an example of where the algorithm converges on over 90% of the time-steps. In this case, we quickly converge to the acceptable range and are able to often reuse the previous time steps error bound for future iterations. In this particular case, we have to retrain only four times over the course of the simulation on iterations: 0, 8, 15, 29. Thus, the algorithm can quickly process many time-steps.

2) How long does it take to reach the target compression ratio?: When evaluating the algorithm, we wanted to consider how long the algorithm takes to find the target compression ratio. This again depends greatly on whether $\rho_t(D_{f,t})$ is feasible or not. Therefore, we considered a large number of possible $\rho_t(D_{f,t})$'s for different datasets. The results of this search are shown in Figure 7. We can see that some compression ratios require far longer total times. Figures 6 (a) and (b) show a zoomed in view of $\rho_t(D_{f,t}) = 8$ and $\rho_t(D_{f,t}) = 15$. The difference in runtime is explained by the difference in the

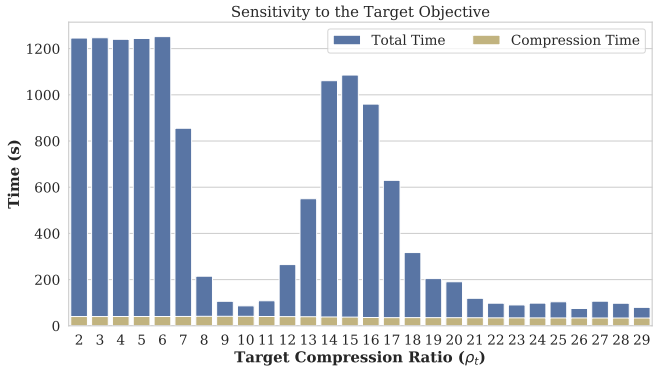


Fig. 7. Sensitivity of FRaZ to the choice of $\rho_t(D_{f,t})$: This is because not all values of $\rho_t(D_{f,t})$ are elements of the co-domain of the function that relates ρ and e.

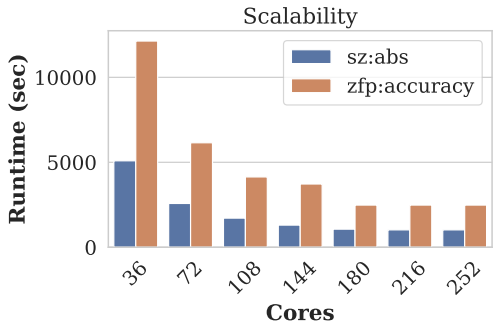


Fig. 8. Scalability: our solution reaches the optimal performance at 180–216 cores, in that the total time is equal to the longest task's wallclock time at this scale.

number of time-steps that converge. In the case shown in Figure 6 (a) relatively few time-steps converged because the objective was infeasible with the specified compressor; in the case shown in Figure 6 (b) almost all time-steps converged because the objective was feasible. This resulted in about a 10x difference in performance between the two cases.

Why do low target compression ratios have long runtimes? Many of the lossy compressors have an effective lower bound for the compression ratio. In Figure 7, it is about 7.5. This effective lower bound on the compression ratio, means that FRaZ will never meet its objective and spends the remainder of the time searching until it hits its timeout.

How does this change across datasets? In general, the more feasible compression ratios near the target, the better FRaZ preformed. Each dataset had a compressor which was able to more accurately compress and decompress the data.

How does this change between compressors? Generally SZ took less time than ZFP or MGARD even though ZFP may take less time for each compression. This is because ZFP typically had fewer viable compression ratios than SZ due to limitations of ZFP's transform based approach. As a result, FRaZ took more time-steps which took the maximum number of iterations lengthening the total runtime. The difference in runtime between SZ and ZFP for a representative dataset can be seen in Figure 8 below.

3) How does the algorithm scale?: To evaluate how well the algorithm scales, we considered the runtime of the algorithm as it scales over multiple cores on ANL Bebop [17].

Figure 8 shows the strong scalability of the algorithm. We see that the algorithm scales by time-step and field levels for the first 180–216 cores with steep decreases in runtime due to parallelism at early levels and then less additional parallelism after that. This is because the runtime of the algorithm is lower bounded by the longest running worker task. All of the datasets we tested has at least one fields that takes substantially longer to compress than others. And the scalability of this algorithm is limited by the longest of these. In the case of the Hurricane dataset using the error-bounded compressor SZ, the QCLOUD field took 1022 seconds to compress while the 75 percentile is less than 500 and the 50 percentile is less than 325.

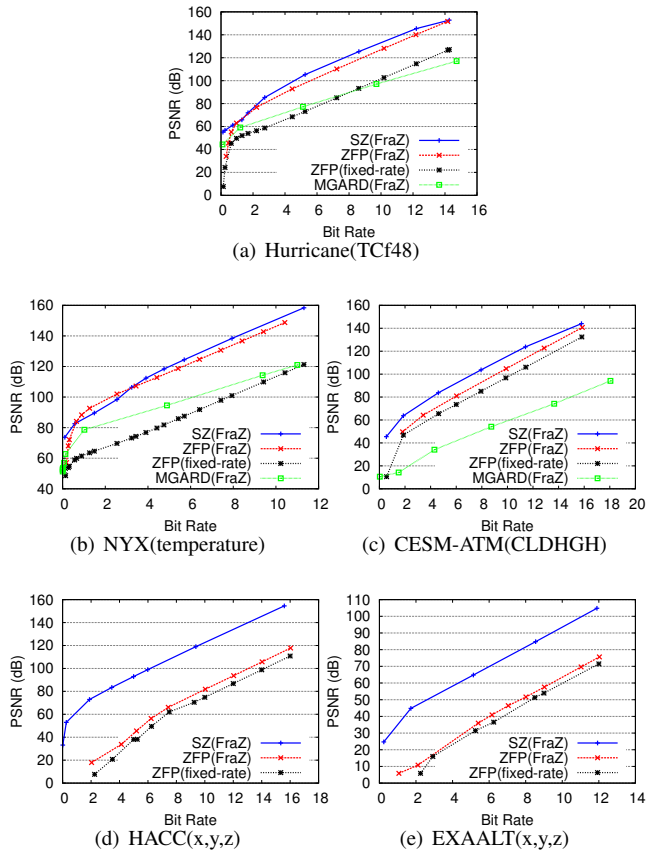
What accounts for the substantial difference between the scalability of FRaZ using ZFP and SZ? Those familiar with ZFP likely know that it is typically faster than SZ, but this seems to contradict the result in Figure 8. This result is explained by considering the individual fields rather than the overall scalability. For the cases in which ZFP finds an error bound that satisfies the target compression ratio, it is much faster. However ZFP often expresses fewer compression ratios for the same error bound range, resulting in more infeasible compression ratios and thus increasing the runtime. ZFP expresses few compression ratios because it uses a flooring function in the minimum exponent calculation used in fixed-accuracy mode.

Fields may take longer for a variety of reasons: (1) the $\rho_t(D_{f,t})$ may not be feasible for one or more of the time-steps, (2) the dataset may have higher entropy resulting in a longer encoding stage for algorithms such as SZ, or (3) the fields may be of different sizes, and larger fields take longer.

4) How does FRaZ compare with the existing fixed-rate compression methods in terms of rate distortion and visual quality?: We present the rate distortion in Figure 9, which shows the bit rate (the number of bits used per data point after the compression) versus the data distortion. Peak signal-to-noise ratio (PSNR) is a common indicator to assess the data distortion in the community. PSNR is defined as $20 \cdot log_{10}(\frac{d_{max}-d_{min}}{rmse})$ where $rmse = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - d_i')^2}$, and d_{max} and d_{min} refer to the max and min value, respectively. In general, the higher the PSNR, the higher the quality of decompressed data.

In this figure, one can clearly see that ZFP (FRaZ) provides consistently better rate distortion than does ZFP (fixed-rate) across bit-rates (i.e., across compression ratios). Moreover, SZ (FRaZ) exhibits the best rate distortion in most cases, which is consistent with the high compression quality of SZ as presented in our prior work [13]. That being said, FRaZ can maintain the high fidelity of the data very well during the compression, by leveraging the error-bounded lossy compression mode for different compressors.

In addition to the rate distortion, we present in Figure 10 visualization images based on the same target compression ratio, to show fixed-ratio compression approach preserves visual quality. We wanted to set compression ratio of 100:1,



(a) Hurricane(TCf48)

(b) NYX(temperature)

(c) CESM-ATM(CLDHGH)

(d) HACC(x,y,z)

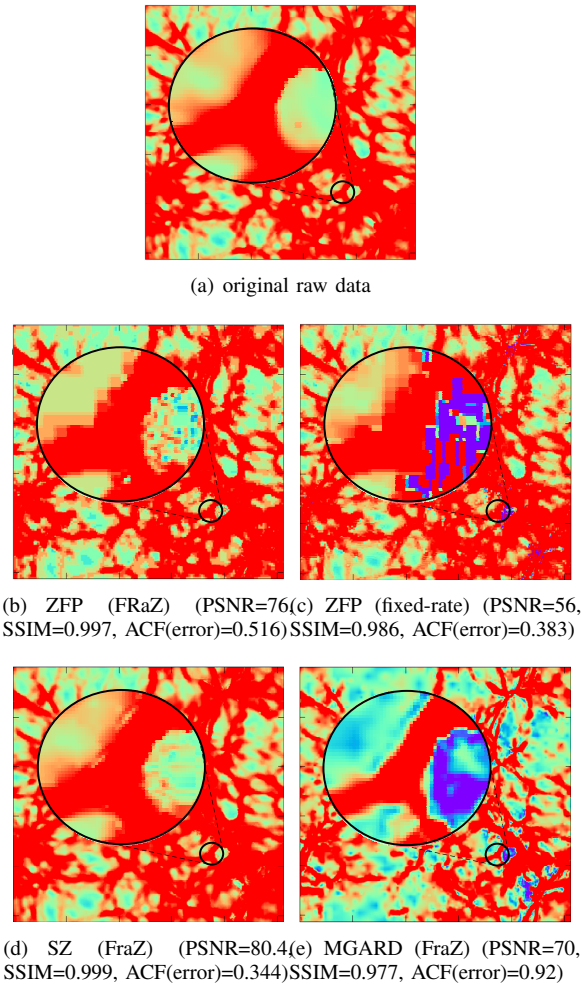
(e) EXAALT(x,y,z)

Fig. 9. Rate distortion of lossy compression (MGARD is missing in (d) and (e) because it does not support 1D dataset)

but the closest fesible compression ratio for ZFP is ∼85:1 (see Section VI-B). Hence, we set the target compression ratio to be 85:1 for all compressors. Because of the space limit, we present the results only for NYX-temperature field; other fields/applications exhibit similar results. All the results are generated by FRaZ except for the ZFP(fixed-rate). ZFP(FRaZ) exhibits a much higher visual quality than does ZFP(fixed-rate) (see Figure 10 (b) vs. Figure 10 (c)), because FRaZ tunes the error bound based on fixed-accuracy mode, which has a higher compression quality than ZFP's built-in fixed-rate mode. ZFP(FraZ) exhibits higher PSNR than does ZFP(fixed-rate), which means higher visual quality. We also present the structural similarity index (SSIM) [40] for the slice images shown in the figure. SSIM indicates similarity in luminance, contrast, and structure between two images; the higher SSIM, the better. Our evaluation shows that ZFP(fixed-rate) has lower SSIM than ZFP(FRaZ) – i.e. better quality. From among all the compressors here, MGARD(FRaZ) leads to the lowest visual quality (as well as lowest PSNR and SSIM), because of inferior compression quality of MGARD on this dataset.

VII. Conclusions and Future Work

We have presented a functional, parallel, black-box autotuning framework that can produce fixed-ratio error-controlled lossy compression for scientific floating-point HPC datasets. Our work offers improvements over existing fixed-rate methods by better preserving the data quality for equivalent compression ratios. We showed that FRaZ works well for a variety of datasets and compressors. We discovered that FRaZ generally has lower runtime for dataset and compressor combinations that produce large numbers of feasible compression ratios.



(a) original raw data

(b) ZFP (FRaZ) (PSNR=76, SSIM=0.997, ACF(error)=0.516)

(c) ZFP (fixed-rate) (PSNR=56, SSIM=0.986, ACF(error)=0.383)

(d) SZ (FraZ) (PSNR=80.4, SSIM=0.999, ACF(error)=0.344)

(e) MGARD (FraZ) (PSNR=70, SSIM=0.977, ACF(error)=0.92)

Fig. 10. Visualization of NYX (temperature:slice 256) with CR ≈ 85:1 (CR of ZFP(FRaZ) = 84, CR of ZFP(fixed-rate) = 85.3, CR of SZ(FRaZ) = 86, CR of MGARD(FRaZ) = 85.7)

A number of areas for potential improvement exist. First, we would like to consider arbitrary user error bounds. By user error bounds, we mean error bounds that correspond with the quality of a scientist's analysis result relative to that on noncompressed data, such as [24] which identifies a particular SSIM in lossy compressed data required for valid results in their field. Second, we would like to develop an online version of this algorithm to provide in situ fixed-ratio compression for simulation and instrument data. Third, we would like to further improve the convergence rate of our algorithm to make it applicable for more use cases.

Acknowledgment

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations - the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem,

including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative.

The material was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357, and supported by the National Science Foundation under Grant No. 1619253 and 1910197.

We acknowledge the computing resources provided on Bebop, which is operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

This material is also based upon work supported by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists, Office of Science Graduate Student Research (SCGSR) program. The SCGSR program is administered by the Oak Ridge Institute for Science and Education (ORISE) for the DOE. ORISE is managed by ORAU under contract number DE-SC0014664. All opinions expressed in this paper are the authors and do not necessarily reflect the policies and views of DOE, ORAU, or ORISE.

REFERENCES

[1] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, V. Vishwanath, T. Peterka, J. Insley *et al.*, "HACC: extreme scaling and performance across diverse architectures," *Communications of the ACM*, vol. 60, no. 1, pp. 97–104, 2016.

[2] NYX simulation, https://amrex-astro.github.io/Nyx, online.

[3] J. Kay, C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. Arblaster, S. 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*, vol. 96, no. 8, pp. 1333–1349, 2015.

[4] I. Foster, M. Ainsworth, B. Allen, J. Bessac, F. Cappello, J. Y. Choi, E. Constantinescu, P. E. Davis, S. Di, W. Di *et al.*, "Computing just what you need: online data analysis and reduction at extreme scales," in *European Conference on Parallel Processing*, 2017, pp. 3–19.

[5] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[6] Zstd, https://github.com/facebook/zstd/releases, online.

[7] Gzip, https://www.gzip.org/, online.

[8] D. King. Dlib C++ Library - Optimization. [Online]. Available: http://dlib.net/optimization.html#global_function_search

[9] X. Liang, S. Di, D. Tao, Z. Chen, and F. Cappello, "An efficient transformation scheme for lossy data compression with point-wise relative error bound," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept. 2018, pp. 179–189.

[10] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1857–1871, 2019.

[11] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *IEEE International Parallel and Distributed Processing Symposium (IEEE IPDPS)*. IEEE, 2016, pp. 730–739.

[12] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *IEEE International Parallel and Distributed Processing Symposium (IEEE IPDPS)*. IEEE, 2017, pp. 1129–1139.

[13] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," *2018 IEEE International Conference on Big Data (Big Data)*, pp. 438–447, 2018.

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

[15] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the univariate case," *Computing and Visualization in Science*, vol. 19, no. 5, pp. 65–76, Dec 2018.

[16] Scientific Data Reduction Benchmark, https://sdrbench.github.io/, online.

[17] Bebop, https://www.lcrc.anl.gov/systems/resources/bebop, online.

[18] A. H. Baker, H. Xu, D. M. Hammerling, S. Li, and J. P. Clyne, "Toward a multi-method approach: Lossy data compression for climate simulation data," in *High Performance Computing*. Cham: Springer International Publishing, 2017, pp. 30–42.

[19] S. Li, S. Di, X. Liang, Z. Chen, and F. Cappello, "Optimizing lossy compression with adjacent snapshots for n-body simulation data," in *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018, pp. 428–437.

[20] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, Y. Alexeev, H. Finkel, and F. T. Chong, "Full state quantum circuit simulation by using data compression," in *IEEE/ACM 30th The International Conference for High Performance computing, Networking, Storage and Analysis (IEEE/ACM SC2019)*, 2019, pp. 1–12.

[21] P. Lindstrom, "Error distributions of lossy floating-point compressors," in *Joint Statistical Meetings*, 2017, pp. 2574–2589.

[22] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large n-dimensional scalar fields," in *Computer Graphics Forum*, vol. 22, no. 3. Wiley Online Library, 2003, pp. 343–348.

[23] U. Trottenberg and A. Schuller, *Multigrid*. Orlando, FL: Academic Press, Inc., 2001.

[24] A. H. Baker, D. M. Hammerling, and T. L. Turton, "Evaluating image quality measures to assess the impact of lossy data compression applied to climate simulation data," *Computer Graphics Forum*, vol. 38, no. 3, pp. 517–528, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13707

[25] N. Sasaki, K. Sato, T. Endo, and S. Matsuoka, "Exploration of lossy compression for application-level checkpoint/restart," in *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 914–922. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2015.67

[26] J. Calhoun, F. Cappello, L. N. Olson, M. Snir, and W. D. Gropp, "Exploring the feasibility of lossy compression for pde simulations," *The International Journal of High Performance Computing Applications*, vol. 33, no. 2, pp. 397–410, 2019.

[27] P. Triantafyllides, T. Reza, and J. C. Calhoun, "Analyzing the impact of lossy compressor variability on checkpointing scientific simulations," in *In the Proceedings of the 2019 IEEE International Conference on Cluster Computing*, ser. Cluster '19. Washington, DC, USA: IEEE Computer Society, 2019.

[28] F. Cappello, S. Di, and et al., "Use cases of lossy compression for floating-point data in scientific data sets," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1201–1220, 2019.

[29] Spring8, http://www.spring8.or.jp/en/, 2019, online.

[30] S. Kim, M. Kim, J. Kim, and H. Lee, "Fixed-Ratio Compression of an RGBW Image and Its Hardware Implementation," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 4, pp. 484–496, 2016.

[31] C. A. Andrews, J. M. Davies, and G. R. Schwarz, "Adaptive data compression," *Proceedings of the IEEE*, vol. 55, no. 3, pp. 267–277, 1967.

[32] "Libpressio," Codesign Center for Online Data Analysis and Reduction. [Online]. Available: https://github.com/CODARcode/libpressio

[33] A. Fischer, "A special Newton-type optimization method," *Optimization*, vol. 24, no. 3-4, pp. 269–284, 1992.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] M. J. Powell, "The BOBYQA algorithm for bound constrained optimization without derivatives," *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pp. 26–46, 2009.

[36] M. J. Saltzman, "COIN-OR: an open-source library for optimization," in *Programming languages and systems in computational economics and finance*. Springer, 2002, pp. 3–32.

[37] G. Gabriele and K. Ragsdell, "OPTLIB: An optimization program library," *Mechanical Engineering Modern Design Series*, vol. 4, 1977.

[38] C. Malherbe and N. Vayatis, "Global optimization of Lipschitz functions." [Online]. Available: http://arxiv.org/abs/1703.02628

[39] M. J. D. Powell, "The NEWUOA software for unconstrained optimization without derivatives," in *Large-Scale Nonlinear Optimization*, G. Di Pillo and M. Roma, Eds. Springer US, 2006, vol. 83, pp. 255–297. [Online]. Available: http://link.springer.com/10.1007/0-387-30065-1_16

[40] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.