Significantly Improving Lossy Compression for HPC Datasets with Second-Order Prediction and Parameter Optimization

Kai Zhao
University of California, Riverside
Riverside, CA
kzhao016@ucr.edu

Sheng Di*
Argonne National Laboratory
Lemont, IL
sdi1@anl.gov

Xin Liang
University of California, Riverside
Riverside, CA
xlian007@ucr.edu

Sihuan Li
University of California, Riverside
Riverside, CA
sli049@ucr.edu

Dingwen Tao
Washington State University
Pullman, WA
dingwen.tao@wsu.edu

Zizhong Chen
University of California, Riverside
Riverside, CA
chen@cs.ucr.edu

Franck Cappello
Argonne National Laboratory
Lemont, IL
cappello@mcs.anl.gov

ABSTRACT

Today's extreme-scale high-performance computing (HPC) applications are producing volumes of data too large to save or transfer because of limited storage space and I/O bandwidth. Error-bounded lossy compression has been commonly known as one of the best solutions to the big science data issue, because it can significantly reduce the data volume with strictly controlled data distortion based on user requirements. In this work, we develop an adaptive parameter optimization algorithm integrated with a series of optimization strategies for SZ, a state-of-the-art prediction-based compression model. Our contribution is threefold. (1) We exploit effective strategies by using 2nd-order regression and 2nd-order Lorenzo predictors to improve the prediction accuracy significantly for SZ, thus substantially improving the overall compression quality. (2) We design an efficient approach selecting the best-fit parameter setting, by conducting a comprehensive priori compression quality analysis and exploiting an efficient online controlling mechanism. (3) We evaluate the compression quality and performance on a supercomputer with 4,096 cores, as compared with other state-of-the-art error-bounded lossy compressors. Experiments with multiple real-world HPC simulations datasets show that our solution can improve the compression ratio up to 46% compared with the second-best compressor. Moreover, the parallel I/O performance is improved by up to 40% thanks to the significant reduction of data size.

*Corresponding author: Sheng Di, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439

HPDC '20, June 23–26, 2020, Stockholm, Sweden

ACM ISBN 978-1-4503-7052-3/20/06.
https://doi.org/10.1145/3369583.3392688

KEYWORDS

Lossy Compression; Science Data; Parameter Optimization; Rate Distortion; High-Performance Computing

ACM Reference Format:
Kai Zhao, Sheng Di, Xin Liang, Sihuan Li, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2020. Significantly Improving Lossy Compression for HPC Datasets with Second-Order Prediction and Parameter Optimization. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20), June 23–26, 2020, Stockholm, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3369583.3392688

1 INTRODUCTION

Extremely large amounts of data are being produced by today's high-performance computing (HPC) applications. Serious conflicts between the vast volume of data produced and the limited resources (such as limited storage space, I/O bandwidth, and memory capacity) significantly hinders today's HPC applications from scaling up in a parallel environment. According to cosmologists, HACC cosmology simulations [16] may produce 20+ petebytes of data during one run when simulating 1 trillion particles for hundreds of timesteps (or snapshots), while the most powerful supercomputer—the Summit supercomputer at Oak Ridge National Laboratory (ORNL) [36]—can provide only hundreds of terabytes of storage for ordinary users or at most several petabytes for specific users. Quantum computing simulation [20] may produce up to 32 exabytes of data, which need to be compressed and decompressed during the simulation because of inadequate memory space (e.g., Summit has only 2.8 PB of memory capacity in total).

Compression techniques designed particularly for big science data have been studied for years. Lossless compressors are not suitable for science data in that the science data are composed mainly of floating-point values that involve disordered ending mantissa bits in their binary representations, such that few repeated patterns could be found in the data streams. Error-bounded lossy compression has been considered a promising solution because not only can it significantly reduce the data size (by 10× or even 100×) but it

can also strictly control the data distortion based on user-specified error bounds. In fact, error-bounded lossy compressors have been broadly verified as helpful in saving storage space and improving I/O performance for many production-level applications across different domains, such as cosmology [19, 30], molecular dynamics [31], climate [18, 44], and quantum computing [20].

Error-bounded lossy compression can be categorized into two models: prediction-based or transform-based. In general, the former performs data prediction for each value in the dataset and then converts the floating-point values to integer quantization codes, followed by an entropy encoding [17] and dictionary coding [48]. SZ [12, 24, 37], FPZIP [27], and ISABELA [21] are three typical examples adopting the prediction-based model. The transform-based compression model performs orthogonal data transforms to convert the original dataset to another data domain and then removes insignificant values [33] or adopts embedded coding [25] to shrink the size. Typical examples are ZFP [25] and wavelet-based compression [33]. Much prior work [4, 12, 26] has demonstrated that SZ and ZFP are the two top error-bounded lossy compressors in most cases; however, none of them can always exhibit the best compression quality on all datasets.

Our research objective is to significantly improve the compression quality of the SZ compression model [12, 37] for most of the datasets across from different domains. Such a research goal is challenging. On the one hand, SZ has been developed for many years, so its design and implementation have been tuned to a fairly optimized level, making further improvement to the compression quality difficult. On the other, many parameter settings (such as block size, dimension order, and regression order) are involved in the prediction-based compression model, making it nontrivial to select the best-fit combination to get the optimal compression quality, especially because of fairly diverse data characteristics in the datasets.

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

- We develop optimization strategies utilizing 2nd-order Lorenzo and 2nd-order regression prediction to improve the prediction accuracy significantly for SZ, such that the overall compression quality can be improved prominently in many cases.
- We design an efficient approach selecting the best-fit parameter settings during the compression. Specifically, we perform a comprehensive priori compression quality analysis to filter out the inferior settings based on error bounds and data characteristics, and we then exploit an efficient online controlling mechanism to determine the best-fit setting at runtime.
- We evaluate the compression quality and performance by running our new compression solution on a supercomputer with 4,096 cores, as compared with other state-of-the-art error-bounded lossy compressors.

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we formulate the research problem. In Section 4, we provide an overview of our design and implementation. Section 5 and Section 6 describe our major solution (2nd-order prediction and parameter optimization) in detail. In Section 7, we present the evaluation results from using multiple real-world simulation datasets on a supercomputer. In Section 8, we present our conclusions and discuss our future work.

2 RELATED WORK

To mitigate the storage burden and I/O bottleneck presented by huge volumes of data, researchers have developed many data compressors. Lossless compressors [2, 5, 9, 11, 17, 47, 48] can guarantee that reconstructed data suffer from no data distortion; however, they cannot significantly reduce the scientific data size because of the random ending mantissa bits in the floating-point values. Their compression ratios are usually around 2 [26, 32, 34], far from the desired level for large-scale scientific simulations running on modern HPC systems [6, 13].

In contrast, error-bounded lossy compressors have been effective in significantly reducing the science data volume for extreme-scale simulations while being able to strictly control the data distortion based on user requirements on pointwise compression errors. Two state-of-the-art models exist for error-bounded lossy compression: prediction-based [7, 12, 14, 21, 23, 24, 27, 37] and transform-based [10, 25, 33, 39, 43]. SZ [12, 24, 37], ISABELA [21], FPZIP [27], and NUMARCK [7] are typical prediction-based compressors. Prior work [24] shows that SZ leads the compression quality among all the prediction-based compressors. SZ includes four key steps: data prediction, linear-scaling quantization, customized variable-length encoding, and dictionary encoding such as gzip [11] or zstd [48]. Vapor [10] and ZFP [25] are typical transform-based compressors. They use different data transformation methods (wavelet transform and a customized (non)orthogonal transform, respectively) and different encoding algorithms. Recent research [25, 37] indicates that ZFP is one of the best error-controlled lossy compressors for scientific simulation datasets. ZFP compresses the dataset block by block (blocksize: 4×4 for 2D data and 4×4×4 for 3D data). Each block involves four steps: exponent alignment, fixed-point alignment, (non)orthogonal block transform to decorrelate the values, and embedded encoding of the ordered coefficients one "bit plane" at a time.

No existing error-bounded lossy compressor can always exhibit the best compression quality (or rate distortion) over all other compressors in most cases. Prior experiments [38], for example, show that neither SZ nor ZFP consistently provides the best compression results on the 13 fields of the Hurricane ISABEL dataset or on the 100+ fields of the CESM-ATM climate simulation dataset.

To address this issue, some researchers studied how to improve the compression quality by combining the two compression models intuitively. Lu et al. [28] concluded that SZ and ZFP were the two best error-bounded lossy compressors. The authors also proposed a solution to estimate the compression ratios for SZ and ZFP, respectively. In [38], they explored an online approach that can select the better strategy between SZ and ZFP in terms of peak signal-to-noise ratio (PSNR). This solution, however, [24] is subject to the existing compression quality and performance of SZ and ZFP. Moreover, the two related works both used the outdated version of

SZ (SZ1.4), which exhibits much worse compression quality than does the latest SZ version (SZ2.0) [24]. Liang et al. [22] proposed a compression method that treats ZFP's data transform as one predictor (called a transform-based predictor) in the SZ compression model and selects the better one between SZ's built-in predictor and the transform-based predictor, which can prominently improve the compression quality beyond SZ and ZFP. Compared with all these works, we develop an efficient approach that can further improve the compression quality of SZ. Specifically, experiments with multiple real-world HPC simulation datasets show that our approach can improve the compression quality by 10%~46% over the second-best approach in most cases.

3 PROBLEM FORMULATION

Our objective in this work is to significantly improve the compression quality for error-bounded lossy compression. Similar to related work [22, 24, 37], we focus on structured datasets (i.e., 1D, 2D, or 3D structured mesh), because unstructured datasets (unable to be represented by a regular mesh grid) either need particular compression strategies [1] or are treated as 1D datasets for simplicity [8].

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

In the error-bounded lossy compression community, three ways have been formulated to assess compression quality in general.

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

(2) Using rate distortion, a common indicator in the visualization community. Rate distortion involves two metrics: bit rate and data distortion. Bit-rate distortion is the average number of bits used to represent one data point after compression. The smaller the bit rate, the higher the compression ratio. Distortion is usually evaluated by using the peak signal-to-noise ratio, which is defined in Formula (1). In general, the higher the PSNR, the better the compression result.

$$PSNR = 20 \cdot \log_{10}(\max(d_i) - \min(d_i)) - 10\log_{10}(MSE(D, D')) \quad (1)$$

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

(3) Checking the visual quality of the reconstructed data compared with the original raw data, by aligning the compression ratios to the same level for different compressors. This method is also widely used by existing error-bounded lossy compression developers [10, 12, 21, 25, 27, 33, 37] and HPC application users [16, 20, 30, 44].

We use all three assessment metrics for comparing our solution with other state-of-the-art lossy compressors such as SZ [12, 37] and ZFP [25]. We will also evaluate the I/O performance of these compressors on a supercomputer.

4 DESIGN OVERVIEW

We adopt the SZ compression model because it exhibits the best compression quality (rate distortion) from among the different compressors in literature and as confirmed by our experiments. SZ adopts four stages in the compression: data prediction, linear-scale quantization, Huffman encoding, and dictionary encoding (such as Zstd [48]). Here we focus mainly on how to improve the data prediction accuracy with as little overhead as possible and how to determine the best parameter settings for the overall compression. Our work involves only the prediction and quantization steps because the other steps involve lossless compression that already has optimized settings.

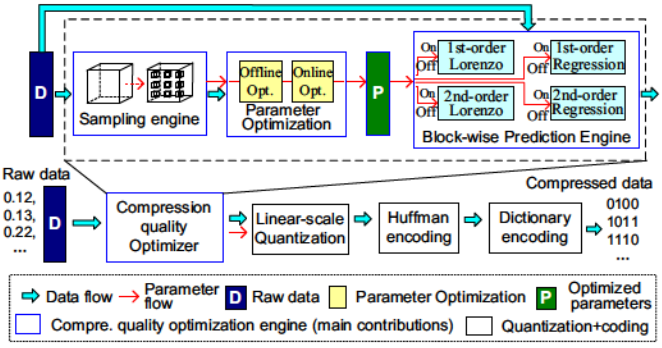


Figure 1: Design Overview

We present the design overview of our method in Figure 1, in which we highlight the main contributions by purple rectangles. Specifically, we develop a compression quality optimizer that includes three key engines working systematically: sampling engine, parameter optimization engine, and blockwise prediction engine.

- *Sampling Engine.* The sampling engine is designed for significantly reducing the overall overhead of our compression quality optimization solution. At the compression runtime, our approach selects a small portion of the whole dataset by a uniform sampling method, and the subsequent steps (i.e., parameter optimization and blockwise selection) are performed on top of the sampled dataset.
- *Parameter Optimization Engine.* The parameter optimization engine addresses two critical issues: (1) how to estimate the overall compression ratio as accurately as possible based on the sampled dataset and (2) how to select the best-fit parameters as efficiently as possible. As for the first issue, simply assembling a new dataset with the uniformly sampled data blocks and performing lossy compression on top of it would cause a large deviation of the estimation (demonstrated later). Accordingly, we develop an effective method that can estimate the compression ratios accurately for various parameter settings. As for the second issue, we design a two-stage (offline and online) optimization strategy that can find the best-fit parameter setting with a fairly low time complexity at runtime. Details are given in Section 6.
- *Blockwise Prediction Engine.* Blockwise prediction is the most important step in our design. In addition to the traditional prediction method [12, 24, 37] (either 1st-order Lorenzo or

1st-order regression), we introduce two new prediction methods, 2nd-order Lorenzo and 2nd-order regression, which can improve the overall compression quality significantly. Based on the optimized parameter settings selected by the parameter optimization engine, the blockwise prediction engine checks the compression quality for each data block and selects the best choice from among the four prediction methods for each block. The 2nd-order prediction methods is detailed in Section 5, and how to select the best-fit prediction method is described in Section 6.

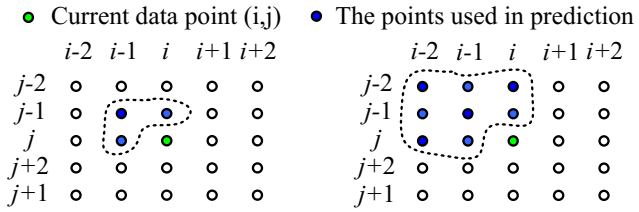
5 SECOND-ORDER DATA PREDICTION

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

5.1 Second-Order Lorenzo Prediction

Second-order Lorenzo prediction was proposed by other researchers conceptually in the literature. For instance, it was called *two-layer prediction* in [37]. However, no compressors are using this idea in practice because of its limitations (detailed later). For instance, the authors in [37] reported that they did not achieve higher prediction accuracy in their experiments with 2nd-order Lorenzo prediction. In our work, we combine 2nd-order Lorenzo prediction with other prediction methods to make it work effectively. In what follows, we review the 1st-order and 2nd-order Lorenzo predictor and then discuss the pros and cons of the two predictors and in what situations 2nd-order Lorenzo is better than 1st-order Lorenzo prediction.

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

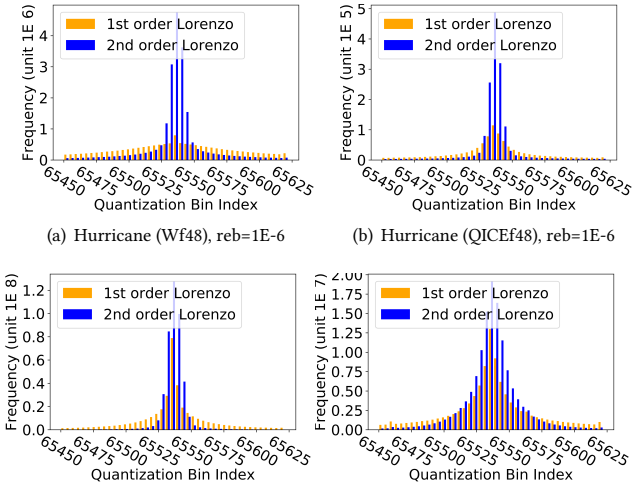


(a) 1st-order Lorenzo (b) 2nd-order Lorenzo

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

In general, the more data points used, the higher the prediction accuracy will be. For example, the average prediction accuracy on the QMCPack dataset [20] is about 0.00197 and 0.00062 when using 1st-order and 2nd-order Lorenzo predictor, respectively. On the other hand, we note that SZ needs to use the decompressed data with biased values to do the prediction instead of the original data, in order to fully respect the preset error bound during the decompression. In this sense, the more data points involved, the more the compression errors impact the prediction accuracy, causing a lower prediction accuracy. Tao et al. [37] demonstrated that 2nd-order Lorenzo prediction does not work as well as the 1st-order Lorenzo, so they adopted only the 1st-order Lorenzo in the released SZ compressor.

We note, however, that 2nd-order Lorenzo prediction may significantly improve the compression ratio, especially when the error bound is required to be relatively low. Figure 3 demonstrates the frequency distribution of quantization bins generated by the 1st-order and 2nd-order Lorenzo predictors with the same compression error bound for four example datasets. In principle, the sharper the distribution is, the higher the compression ratio will be. We observe that when the relative error bound[1] is set to the order of 1E-6~1E-8, the 2nd-order Lorenzo predictor turns out to be better than the 1st-order Lorenzo. The key reason is discussed as follows. SZ has to perform the data prediction using decompressed data each with certain errors, which may impact the prediction accuracy in turn. If the error bound is small enough, the impact of decompressed data to the prediction accuracy will be very small. This result is also verified by our evaluation of the percentage breakdown of different predictors used in compression (discussed in Section 7).



(a) Hurricane (Wf48), reb=1E-6 (b) Hurricane (QICEf48), reb=1E-6

(c) QMCPack dataset 1, reb=1E-7 (d) Scale-LETKF (Pres), reb=1E-8

Figure 3: Frequency Distribution of Quantization Bins between 1st- and 2nd-order Lorenzo Prediction

5.2 Second-Order Regression-Based Prediction

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

The generic formula can be derived based on a m-dimensional dataset ($n_1 \times n_2 \times \cdots \times n_m$). The independent variable vector of the m-dimensional dataset is denoted as $\boldsymbol{x} = (x_1, x_2, .., x_m)$. Its corresponding dependent variable vector is $f_{\boldsymbol{x}}$. We use $f^{2r}(\boldsymbol{x})$ to denote the prediction value of $\boldsymbol{x}$ by 2nd-order regression:

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

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

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, ..\beta_{m_2})$ represents the coefficient vector in which β_0 is the intercept coefficient. We denote the total number of coefficients by $m_2 = m \times (m+1)/2 + m + 1$.

The 2nd-order regression uses $f^{2r}(\boldsymbol{x})$ to estimate the dependent variable $f_{\boldsymbol{x}}$. The objective is to minimize the mean squared error between $f^{2r}(\boldsymbol{x})$ and $f_{\boldsymbol{x}}$, as shown in Equation (3).

$$f_{obj} = \arg\min_{\beta} \sum_{\forall \boldsymbol{x}} (t(\boldsymbol{x}^T)\boldsymbol{\beta} - f(\boldsymbol{x}))^2 \quad (3)$$

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

For a 3D dataset, (x_1, x_2, x_3) in Formula (2) can be replaced by coordinates (i, j, k), where $0 \le i < n_1, 0 \le j < n_2, 0 \le k < n_3$. Then, the objective function can be simplified to

$$f_{obj} = \arg\min_{\beta} \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \sum_{k=0}^{n_3-1} (\beta_0 + \beta_1 i + \beta_2 j + \beta_3 k + \beta_4 i^2 + \beta_5 ij + \beta_6 ik + \beta_7 j^2 + \beta_8 jk + \beta_9 k^2 - f_{ijk})^2. \quad (4)$$

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

$$\sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} \sum_{k=0}^{n_3-1} A\boldsymbol{\beta}^T = \boldsymbol{V}^T \quad (5)$$

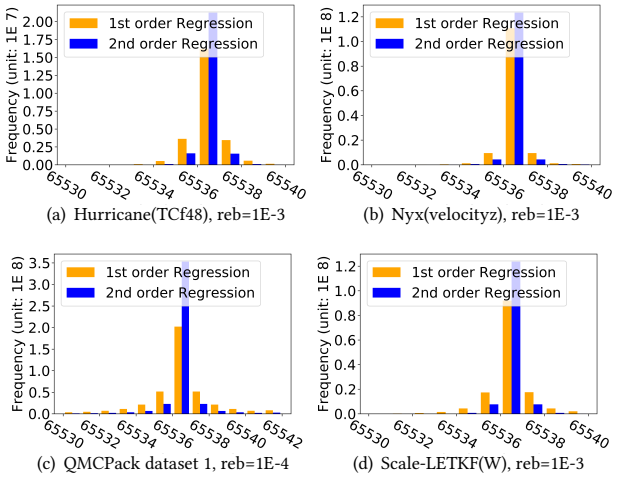
$$A = \begin{bmatrix} 1 & i & j & k & i^2 & ij & ik & j^2 & jk & k^2 \\ i & i^2 & ij & ik & i^3 & i^2j & i^2k & ij^2 & ijk & ik^2 \\ j & ij & j^2 & jk & i^2j & ij^2 & ijk & j^3 & j^2k & jk^2 \\ k & ik & jk & k^2 & i^2k & ijk & ik^2 & j^2k & jk^2 & k^3 \\ i^2 & i^3 & i^2j & i^2k & i^4 & i^3j & i^3k &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

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

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

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

Figure 4 demonstrates the frequency distribution of quantization bins generated after the 1st-order regression predictor versus the 2nd-order regression predictor with the same compression error bound for four example datasets. The 2nd-order regression exhibits sharper distribution than does the 1st-order regression, which means that the 2nd-order regression will likely obtain higher compression ratios in these cases.



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

(c) QMCPack dataset 1, reb=1E-4 (d) Scale-LETKF(W), reb=1E-3

Figure 4: Frequency Distribution of Quantization Bins between 1st- and 2nd-Order regression Prediction

Next we compress the 10 coefficients ($\beta_0 \sim \beta_1 0$) used to construct the hyperplane. Specifically, we compress them using the SZ compressor, because this can lead to outstanding compression ratios that other compressors such as FPZIP [27], ZFP [25], or bit-truncation methods [15] cannot achieve.

6 PARAMETER OPTIMIZATION

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

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

Based on our new compression design supporting 2nd-order prediction, we summarize a total of 12 critical parameters for the whole compression. Five of them are from SZ (version 2.0), as shown in Table 1, and the other 7 parameters are based on the 2nd-order prediction we designed, as shown in Table 2.

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

Table 1: SZ Parameters

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

Table 2: Extended Parameters in Our Solution

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

The 12 parameters are determined by our in-depth analysis of their impact on the compression ratios based on experiments using 5 real-world simulation datasets each with multiple time steps, involving about 100 fields and thousands of data files in total. Different settings of these parameters may lead to largely different compression ratios. We demonstrate three examples in Figure 5, Figure 6, and Figure 7. For instance, based on Figure 5(a), SZ's compression ratio is 180:1 and 100:1 on Hurricane(TCf48) with the error bound 5E-3, when its block size is set to 5 and 11, respectively. In Figure 6, none of $pred\_dim = 2$ or $pred\_dim = 3$ can always exhibit the best compression ratio when the error bound is between 1E-3 and 1E-4. In Figure 7, 8192 is the best setting for quan_bin to compress the Hurricane dataset with a 1E-5 error bound. However, in order to compress the same dataset with 1E-7 error bound, the best setting for *quan_bin* is 1024.

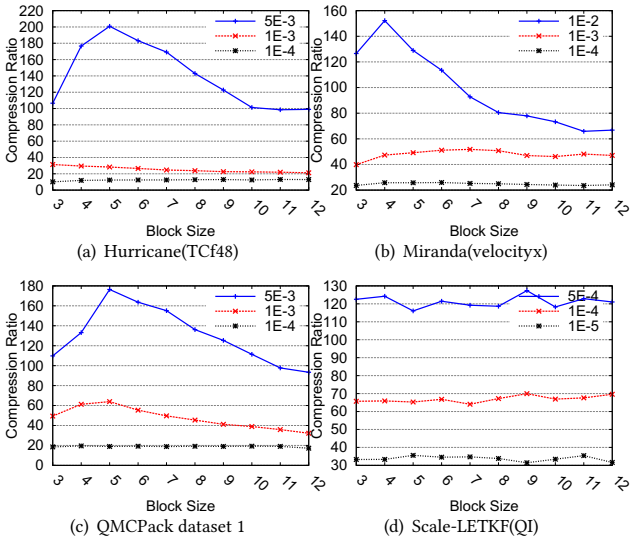


Figure 5: Change of Compression Ratios with Block Sizes

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

6.1 Optimizing Compression Quality Estimation over Sampled Dataset

Accurately estimating the compression quality based on the sampled dataset is critical to selecting the best-fit parameter settings

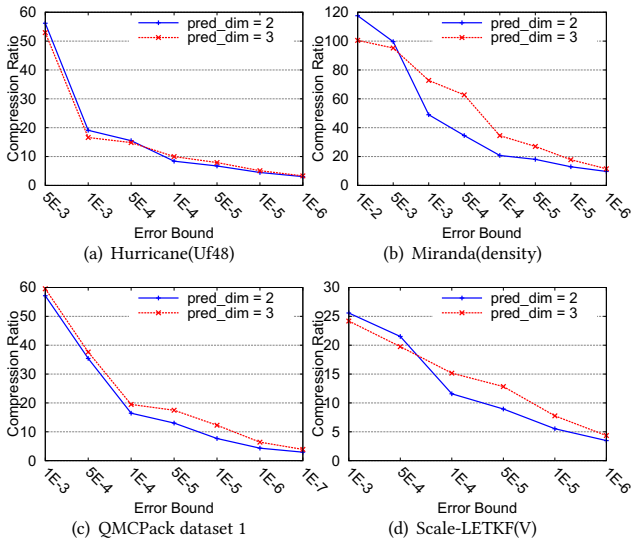


Figure 6: Change of Compression Ratios with Various Prediction Dimensions

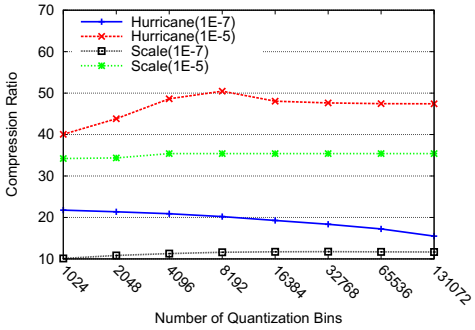


Figure 7: Change of Compression Ratios with Numbers of Quantization Bins (Hurricane(QCLOUDf48) and Scale(QI))

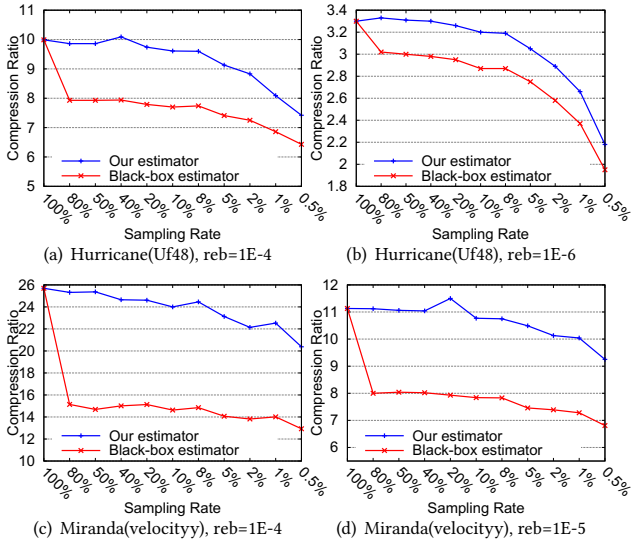
and predictors at runtime. To this end, we design an approach that takes into account how the data will be predicted and quantized for each block, in that the existing compression quality estimation methods are not suitable for our case. Lu et al. [28], for exampe, proposed a sampling-based estimation method based on the distribution of quantization bins, which can estimate the compression ratios of SZ to a certain extent. Since this method can support only Huffman encoding but not dictionary encoding (zstd), it cannot be applied to our estimation. Moreover, since it is designed based on SZ 1.4, which has no regression predictor, it cannot estimate the compression ratios accurately for SZ 2.0.

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

cause biased estimation of compression ratios because it totally ignores the compression principle.

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

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



(a) Hurricane(Uf48), reb=1E-4 (b) Hurricane(Uf48), reb=1E-6

(c) Miranda(velocityy), reb=1E-4 (d) Miranda(velocityy), reb=1E-5

Figure 8: Comparison of Estimation Accuracy (sampling rate refers to the fraction of sampled data to the full data; sampling rate = 100% refers to the full dataset)

6.2 Offline Parameter Optimization

Table 3: Range of Parameters

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

[1] s means use the estimation value provided by SZ

[2] b means use block_size as reg_coef_linear

Bold values in column 3 are used in the first step of manual parameter search.

Finding the best parameter combination for a given dataset is a multivariable optimization problem. The objective is to find the maximum compression ratio using the same compression function and original data. Gradient-based algorithms such as gradient descent are difficult to apply for this problem since it is unclear whether the compression function itself is a differentiable function; also, the derivative is hard to obtain even if this function is differentiable. Derivative-free methods such as coordinate descent and (meta)heuristic methods such as simulated annealing, genetic algorithm, and ant colony optimization could be used to find the approximate global optimization in a large search space if the derivative is unknown or nonexistent. However, those (meta)heuristic algorithms are all time-consuming (generally requiring 20+ iterations to converge a near-global optimum solution [42]). By contrast, we need to control the number of iterations to under 20 such that the analysis overhead can be limited within 100% when the sampling rate is set to 8%. To this end, we propose an offline + online parameter optimization method.

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

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

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

Algorithm 1 Manual Parameter Search

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

```
compressMode ← no_sampling
for (enable_lorenzo, enable_2ndlorenzo, enable_2ndregression, pred_dim, block_size) in (values from Table 3) do
    for (reg_coef_intercept, reg_coef_linear, 2ndreg_coef_intercept, 2ndreg_coef_linear, 2ndreg_coef_poly) in (bold values from Table 3) do
        Do compression, Record parameter settings and compression ratio
    end for
end for
for (enable_regression, block_size, reg_coef_intercept, reg_coef_linear) in (values from Table 3) do
    Do compression, Record parameter settings and compression ratio
end for
for (enable_2ndlorenzo, block_size, 2ndreg_coef_intercept, 2ndreg_coef_linear, 2ndreg_coef_poly) in (values from Table 3) do
    Do compression, Record parameter settings and compression ratio
end for
for quan_bin in (Values from Table 3) do
    Do compression, Record parameter settings and compression ratio
end for
```

outstanding candidates. The final results are shown in the last column of Table 3. By this selection method, we considerably reduce the number of parameter values to be focused on during online parameter optimization.

6.3 Online Parameter Optimization

Our solution searches the best parameters based on the outstanding candidates generated by the offline optimization. This auto parameter search process is an online process, which means it will be executed every time when we run the compressor. The subsets generated by sampling are used to find the best parameters. After that, the original datasets will be compressed by our compressor using the best parameters. The overhead of the online parameter optimization process is around 100% based on the runtime of SZ and the overhead of second-order predictors is 20%∼50% based on SZ. Thus, the total runtime overhead of our solution is around 120%∼150% based on SZ.

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

Although the online auto parameter search performs on top of the outstanding candidates generated by an offline parameter optimization, this solution is also efficient on new datasets, as we verify in Section 7.4.

7 PERFORMANCE EVALUATION

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

7.1 Experimental Settings

Table 4 describes the five applications, which all require compression techniques to store big science data [16, 20, 30, 44]. In particular, QMCPack here involves three datasets that are stored in three scales—288×115×69×69 (1 field), 816×115×69×69 (2 fields), and 6192×115×69×69 (1 field)—corresponding to 0.6 GB, 3.4 GB, and 13 GB, respectively. We call them QMCPack dataset 1, QMCPack dataset 2, and QMCPack dataset 3, respectively. Since our experiments involve parallel processes each with several gigabytes, the de facto total data size is up to 10+ terabytes for one application in our experiments, when the execution scale is 4,096 cores.

Table 4: Applications Used in Our Experiments

Name	Domain	# Fields	Size Per Snapshot
Hurricane [18]	Weather	13	1.3 GB (= 13× 96MB)
Miranda [29]	Hydrodynamics	7	1 GB (= 7 × 144MB))
QMCPack [20]	Atom/Molecules	4	∼17 GB (=0.6 + 3.4 +13) GB
Scale-LETKF [44]	Weather	12	6.4 GB (=12×539MB)
NYX [30]	Cosmology	6	3.1 GB (=6×512MB)

We conducted our experiments on the Bebop supercomputer [35] at Argonne National Laboratory using up to 4,096 cores. Specifically, the experiments involve 64∼128 nodes, and each node is equipped with 128 GB memory and two Intel Xeon E5-2695 v4 processors (each with 16 cores). Its storage system adopts a General Parallel File System (GPFS) equipped with 2 I/O nodes, and the I/O system is a typical high-end supercomputer facility. We perform data writing/reading by a file-per-process method with POSIX I/O [45] in parallel.[1]

We compare our solution with three state-of-the-art lossy compression methods: SZ2.1.8 [24], ZFP0.5.5 [25], and a hybrid model [22], which have been confirmed as the best in class [8, 22, 28]. The hybrid model merges the SZ2.0 and ZFP0.3.1 to get the best compression quality, while suffering from 200% time overhead [22].

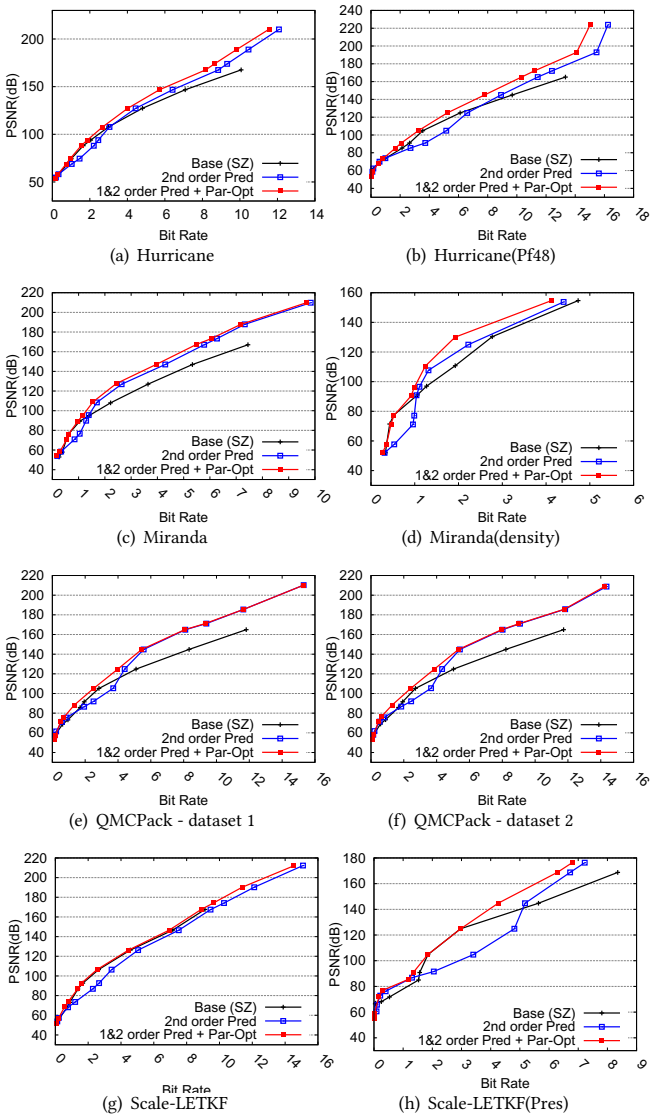
In what follows, we first present the compression quality results based on second-order prediction and parameter optimization and then present the overall compression quality in terms of the indicators defined in our problem formulation (Section 3). We also evaluate the I/O performance gain by running a series of parallel experiments on a supercomputer with up to 4,096 cores, and compare the results with those of other existing state-of-the-art compressors.

7.2 Assessment of Second-Order Prediction

In Figure 9, we present the rate distortion improvement obtained with the second-order prediction (shown as blue curves in the figure) over the original design in SZ 2.0 (called Base(SZ) and shown as black curves in the figure) that uses the 1st-order prediction. As mentioned in Section 3, the higher the PSNR, the better the

[1] Another researcher [41] verified that POSIX I/O has comparable performance with parallel I/O, such as MPI-IO [40] when reading/writing thousands of files simultaneously on GPFS. We also further verified that the read/write performance difference of POSIX IO and MPI-IO is within ±10% on this supercomputer, when the execution scales between 2k cores and 8k cores.

compression quality; and the lower the bit rate, the higher the compression ratio. We can clearly see that using 2nd-order prediction (see Section 5) can significantly improve the compression quality over the original SZ with 1st-order prediction in many cases, especially with relatively high bit rates or relatively high precision. For instance, the compression ratios can be improved by about 50% when the PSNR is greater than 120 dB for the Miranda and QMCPack simulation. The main reason is the high-order nature of the datasets. However, we can also see that at some bit rates, the original SZ with 1st-order prediction outperforms the one with 2nd-order prediction. As shown in Figure 9, for instance, 1st-order prediction is much better than 2nd-order prediction when the bit rate is in the range of [1.5,5] for the Scale-LETKF(Pres) field. This result provides motivation for adopting both 1st- and 2nd-order predictions in the compression.



(a) Hurricane (b) Hurricane(Pf48)
(c) Miranda (d) Miranda(density)
(e) QMCPack - dataset 1 (f) QMCPack - dataset 2
(g) Scale-LETKF (h) Scale-LETKF(Pres)

Figure 9: Breakdown Compression Quality Analysis

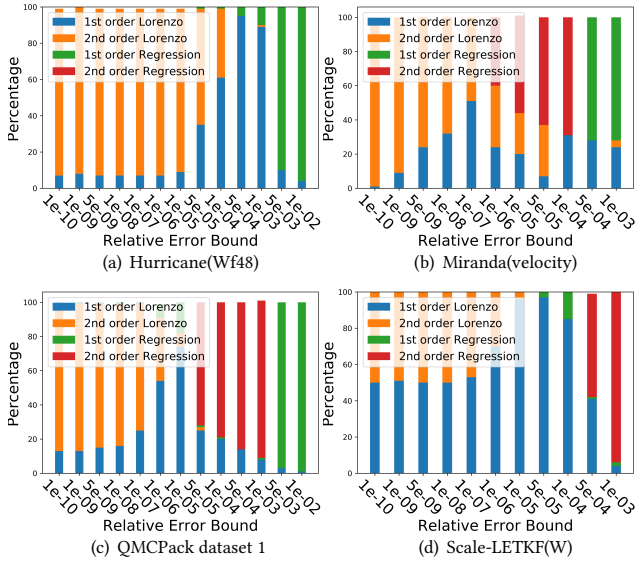
7.3 Assessment of Parameter Optimization

In Figure 9, we also demonstrate the further compression quality improvement (see the red curves versus the blue curves) when using parameter optimization strategies on top of 2nd-order prediction. In absolute terms, the rate distortion can be improved by 4%~50% in most cases, depending on the bit rates. Such a significant improvement is attributed to our design of integrating both 1st-order prediction and 2nd-order prediction (four predictors in total) and an efficient online parameter optimization strategy selecting the best-fit parameter setting at runtime in fine granularity (as per block) (for details, see Section 6.1 and Section 6). The variation in the rate distortion improvement shows that the default parameter settings of the original SZ is nearly optimal for some datasets while it is far from the optimal level for some other datasets. This confirms the significance of our parameter optimization in order to achieve the optimal results for all datasets.

To demonstrate the effect of our parameter optimization engine, in Figure 10 we illustrate the percentage breakdown of the four different prediction methods used in the compression of different applications or fields. We clearly observe an interesting distribution pattern of the four prediction methods in terms of different error bounds. Specifically, when the error bound is relatively large (such as 1E-2), the regression-based predictor would take a major role, since the Lorenzo predictor may suffer from huge prediction errors in this situation because of the impact of decompressed data (keep in mind that Lorenzo prediction has to be performed by using decompressed data during the compression stage). When the error bound is relatively small, the Lorenzo prediction would outperform the regression-based prediction. In particular, when the error bound is extremely small, our optimization engine selects the 2nd-order Lorenzo predictor in most blocks. This action is consistent with our analysis in Section 5.1: many of the application datasets actually exhibit high-order smoothness, such that the 2nd-order Lorenzo predictor is more accurate for data prediction, especially with small compression error bounds.

7.4 Overall Compression Quality

In Figure 11 we present the overall compression quality (rate distortion) based on five real-world scientific simulation datasets, and we demonstrate the result of one example field for Hufficane ISABEL, Miranda, and Scale-LETKF, respectively. The blue curve (called optimum) refers to the ideal level obtained by our offline parameter searching (MS) for optimal parameters. As highlighted in the figures, our compression solution can improve the compression ratios over SZ (see red curve versus black curve) by 20+% for Hurricane, by ~40+% for Miranda, and by ~30+% for QMCPack, respectively, with the same PSNR. Our solution also exhibits the best compression quality from among all existing compressors on the three applications. Specifically, with the same PSNR, its overall compression ratio is higher than that of the second best compressor generally by 20~25% and by 5~10% and 20~30% for the three applications, respectively. For some specific fields, the improvement can be up to 46%, as shown in Figure 11(b). As for the simulation Scale-LETKF and NYX, our solution still leads to the best compression quality from among all the compressors, although it has no prominent improvement over the second-best compressor, probably because



(a) Hurricane(Wf48) (b) Miranda(velocity)

(c) QMCPack dataset 1 (d) Scale-LETKF(W)

Figure 10: Percentage Breakdown of Four Predictors Used in the Blockwise Compression

the default parameter setting of the original SZ is also (or nearly) the best choice in those cases.

Figure 12 presents the compression quality of the QMCPack dataset 2 and dataset 3 compared with the QMCPack dataset 1 shown in Figure 11(g). Note that our offline parameter optimization was performed not based on these two QMCPack datasets, which are largely different from the QMCPack dataset 1 in scale. Based on the figure, we clearly see that for both datasets our solution can still get much better compression quality than the others can. This means that our optimization method can also be applied effectively on new simulation datasets that were not included in our offline optimization analysis.

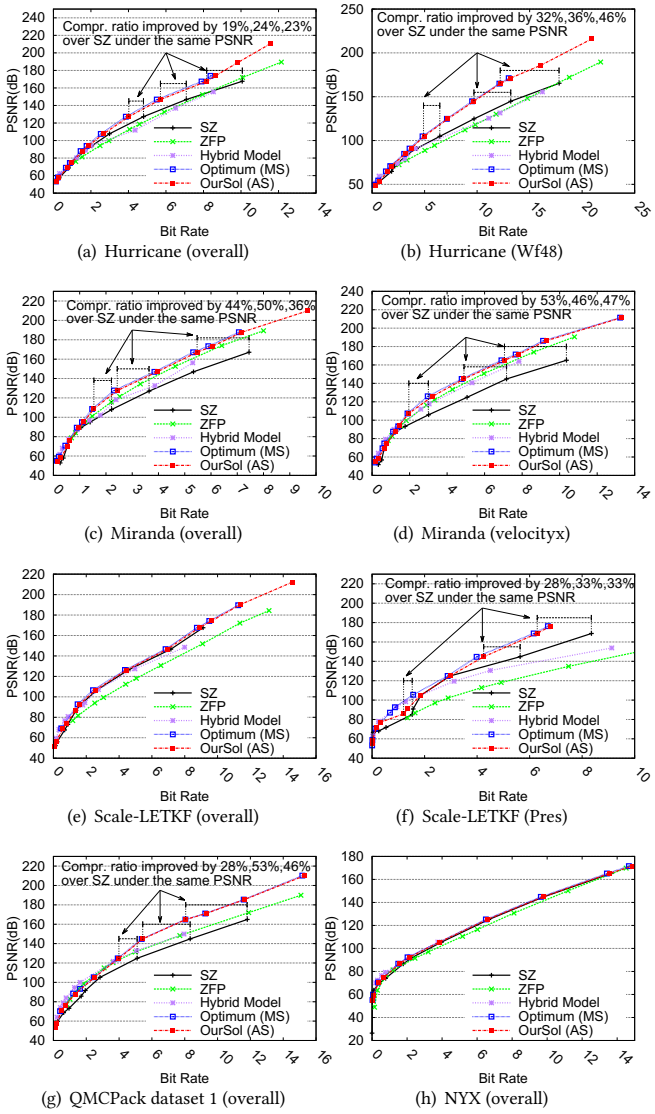
We also evaluate the autocorrelation metric of the compression errors (as shown in Table 5), in order to check the randomness of the compression errors. The users generally expect to see close-to-zero autocorrelation results, because this introduces less bias to their post-analysis. Table 5 shows that our solution achieves comparable autocorrelation values of compression errors compared with SZ, indicating the same randomness of compression errors.

Table 5: Lag One Autocorrelation of Compression Error

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

7.5 I/O Performance Evaluation

In this subsection, we present the parallel I/O evaluation results based on two scientific simulations (Hurricane and Miranda) on the Bebop supercomputer [35]. The value-range-based relative error



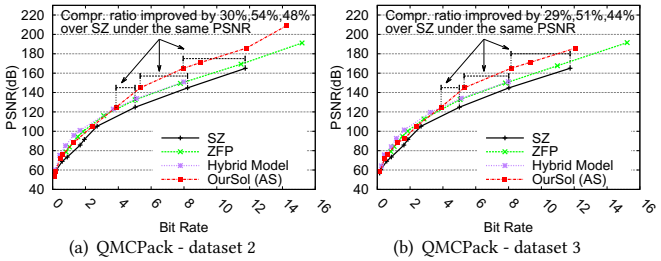
(a) Hurricane (overall) (b) Hurricane (Wf48)

(c) Miranda (overall) (d) Miranda (velocityx)

(e) Scale-LETKF (overall) (f) Scale-LETKF (Pres)

(g) QMCPack dataset 1 (overall) (h) NYX (overall)

Figure 11: Overall Evaluation



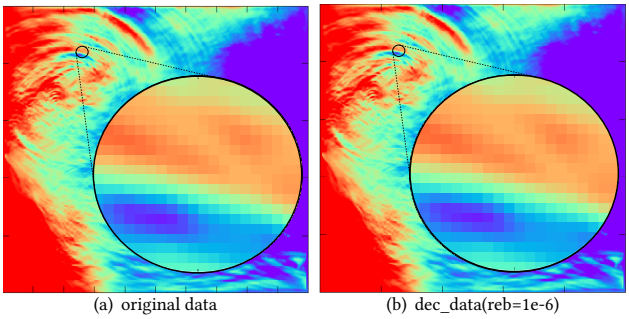
(a) QMCPack - dataset 2 (b) QMCPack - dataset 3

Figure 12: Evaluation on Multiple QMCPack Dataset

bounds are set to 1E-6 and 1E-5 respectively. We first show that the lossy compression with these two error bounds leads to fairly high precision of the reconstructed data compared with the original

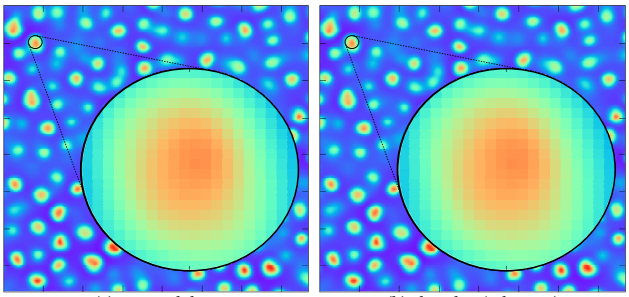
raw data. We then show the parallel I/O performance when using different compressors.

The reconstructed data under lossy compression with these error bounds are of fairly high precision. On the one hand, some domain scientists [3] recommend keeping the structural similarity index measure (SSIM) [46] no less than 0.99995, based on their postanalysis using existing lossy compressors. The reconstructed data in our experiments here can get an overall SSIM up to 0.99999+, so the data are supposed to be acceptable to users w.r.t. SSIM. On the other hand, to confirm that the error bounds in our evaluation lead to high precision of the reconstructed data, we demonstrate the visual quality of the reconstructed data for the two applications in Figure 13 and Figure 14. We zoom in on a small region to 625× for each image.



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

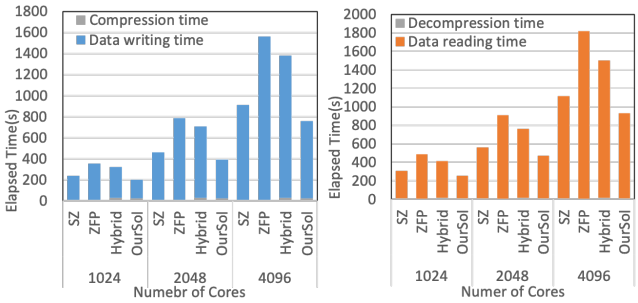
Figure 13: Visualization of Hurricane(Uf48)



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

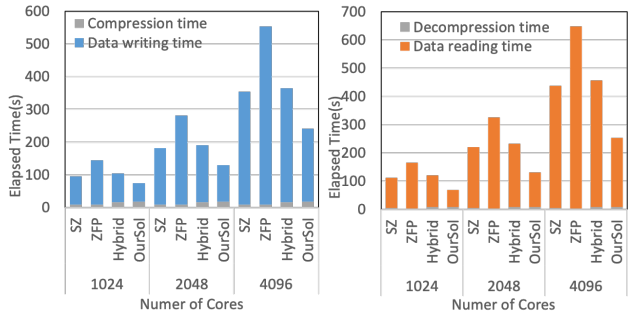
Figure 14: Visualization of Miranda (velocityz)

We present the parallel I/O performance evaluation results in Figure 15 and Figure 16. Without any compression techniques, it took 6,141 s and 4,881 s to store the original data and 7,274 s and 5,891 s to read the original data (using 4,096 processes) because of limited I/O bandwidth. The figures clearly show that the parallel I/O performance with compression techniques is always less than 1,800 seconds. In particular, our solution has the least overall elapsed times, which are 20%~40% less than the times when using the second-best lossy compressor (SZ). This is due to the significantly reduced data sizes achieved by our compressor. Such a performance gain can benefit the applications significantly. On the one hand, for the applications suffering a bottleneck in I/O cost, the overall runtime can be reduced significantly. On the other hand, the storage requirement would be decreased for each application, enabling more applications to run on supercomputers.



(a) Data dumping performance (b) Data loading performance

Figure 15: Parallel Performance on Hurricane



(a) Data dumping performance (b) Data loading performance

Figure 16: Parallel Performance on Miranda

8 CONCLUSIONS AND FUTURE WORK

In this paper, we present an efficient solution to significantly improve the compression quality for the datasets produced by parallel scientific simulations. In our solution, we develop more efficient methods (2nd-order prediction) based on both Lorenzo prediction and regression prediction. We develop an efficient algorithm that can select the best-fit predictors and optimized parameter settings at runtime. We thoroughly evaluate the compression quality and performance on a supercomputer with 5 real-world scientific simulations. The key findings are summarized below.

- The 2nd-order prediction can improve the compression ratio by 50+% when the PSNS is around 120 dB for the Miranda and QMCPack simulations.
- Our parameter optimization can further improve the compression by 4%~50% in most cases.
- When using lossy compression techniques, the overall I/O times are reduced to several hundreds of seconds from the original several hours on the supercomputer.
- Our solution has the least overall elapsed I/O times, which are 20%~40% less than the times when using the second-best lossy compressor.

As future work, we plan to explore more effective prediction models and coding algorithms.

9 ACKNOWLEDGMENTS

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. This work was also supported by the National Science Foundation under Grants CCF-1513201, CCF-1619253, OAC-1948447, and OAC-2034169. We acknowledge the computing resources provided on Bebop, which is operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

REFERENCES

[1] Mark Ainsworth, Ozan Tugluk, Ben Whitney, and Scott Klasky. 2018. Multilevel techniques for compression and reduction of scientific data—the univariate case. *Computing and Visualization in Science* 19, 5-6 (2018), 65–76.

[2] Francesc Alted. 2017. Blosc, an extremely fast, multi-threaded, meta-compressor library.

[3] Allison H Baker, Dorit M Hammerling, Sheri A Mickelson, Haiying Xu, Martin B Stolpe, Phillippe Naveau, Ben Sanderson, Imme Ebert-Uphoff, Savini Samarasinghe, Francesco De Simone, et al. 2016. Evaluating lossy data compression on climate simulation data within a large ensemble. *Geoscientific Model Development* 9, 12 (2016), 4381–4403. https://doi.org/10.5194/gmd-9-4381-2016

[4] Allison H. Baker, Haiying Xu, Dorit M. Hammerling, Shaomeng Li, and John P. Clyne. 2017. Toward a Multi-method Approach: Lossy Data Compression for Climate Simulation Data. In *High Performance Computing*. Springer International Publishing, Cham, 30–42.

[5] Martin Burtscher and Paruj Ratanaworabhan. 2009. FPC: A high-speed compressor for double-precision floating-point data. *IEEE Trans. Comput.* 58, 1 (Jan 2009), 18–31.

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

[7] Zhengzhang Chen, Seung Woo Son, William Hendrix, Ankit Agrawal, Wei-keng Liao, and Alok Choudhary. 2014. NUMARCK: machine learning algorithm for resiliency and checkpointing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, IEEE, New York, NY, USA, 733–744.

[8] Jong Youl Choi and et al. 2018. Coupling exascale multiphysics applications: Methods and lessons learned. In *Proceedings of IEEE International Conference on eScience*. IEEE, IEEE, New York, NY, USA, 442–452.

[9] S. Claggett, S. Azimi, and M. Burtscher. 2018. SPDP: An Automatically Synthesized Lossless Compression Algorithm for Floating-Point Data. In *2018 Data Compression Conference*. IEEE, New York, NY, USA, 335–344. https://doi.org/10.1109/DCC.2018.00042

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

[11] L Peter Deutsch. 1996. GZIP file format specification version 4.3.

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

[13] Ian T. Foster et al. 2017. Computing just what you need: Online data analysis and reduction at extreme scales. In *European Conference on Parallel Processing*. Springer, Springer International Publishing, Cham, 3–19.

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

[15] L. A. B. Gomez and F. Cappello. 2013. Improving floating point compression through binary masks. In *2013 IEEE International Conference on Big Data*. IEEE, New York, NY, USA, 326–331.

[16] Salman Habib, Vitali A. Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, Katrin Heitmann, Kalyan Kumaran, Venkatram Vishwanath, Tom Peterka, Joseph A. Insley, David Daniel, Patricia K. Fasel, and Zarija Lukic. 2016. HACC: extreme scaling and performance across diverse architectures. *Commun. ACM* 60, 1 (2016), 97–104.

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

[18] Hurricane ISABEL simulation data. 2019. http://vis.computer.org/vis2004contest/data.html. Online.

[19] Sian Jin, Pascal Grosset, Christopher M Biwer, Jesus Pulido, Jiannan Tian, Dingwen Tao, and James Ahrens. 2020. Understanding GPU-Based Lossy Compression for Extreme-Scale Cosmological Simulations. https://arxiv.org/abs/2004.00224. Online.

[20] Jeongnim Kim and et al. 2018. QMCPACK: an open source ab initio quantum Monte Carlo package for the electronic structure of atoms, molecules and solids. *Journal of Physics: Condensed Matter* 30, 19 (2018), 195901.

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

[22] Xin Liang, Sheng Di, Sihuan Li, Dingwen Tao, Bogdan Nicolae, Zizhong Chen, and Franck Cappello. 2019. 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*. 1–26.

[23] Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2018. An Efficient Transformation Scheme for Lossy Data Compression with Point-wise Relative Error Bound. In *IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, New York, NY, USA, 179–189.

[24] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. 2018. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data*. IEEE, IEEE, New York, NY, USA.

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

[26] Peter Lindstrom. 2017. Error Distributions of Lossy Floating-Point Compressors. *Joint Statistical Meetings* 1, 1 (2017), 2574–2589.

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

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

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

[30] NYX simulation. 2019. https://amrex-astro.github.io/Nyx. Online.

[31] EXAALT project. 2019. https://www.exascaleproject.org/project/exaalt-molecular-dynamics-at-the-exascale-materials-science/. Online.

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

[33] Naoto Sasaki, Kento Sato, Toshio Endo, and Satoshi Matsuoka. 2015. Exploration of lossy compression for application-level checkpoint/restart. In *2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, New York, NY, USA, 914–922.

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

[35] Bebop supercomputer. 2019. Available at https://www.lcrc.anl.gov/systems/resources/bebop. Online.

[36] ORNL Summit supercomputer. 2019. https://www.olcf.ornl.gov/summit/. Online.

[37] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. 2017. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, New York, NY, USA, 1129–1139.

[38] Dingwen Tao, Sheng Di, Xin Liang, Zizhong Chen, and Franck Cappello. 2019. Optimizing Lossy Compression Rate-Distortion from Automatic Online Selection between SZ and ZFP. *IEEE Trans. Parallel Distrib. Syst.* 30, 8 (2019), 1857–1871.

[39] David Taubman and Michael Marcellin. 2013. *JPEG2000 Image Compression Fundamentals, Standards and Practice*. Springer Publishing Company, Incorporated, New York, NY, USA.

[40] Rajeev Thakur, William Gropp, and Ewing Lusk. 1999. On Implementing MPI-IO portably and with high performance. In *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems (IOPADS '99)*. ACM, New York, NY, USA, 23–32.

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

[42] Robert Underwood, Sheng Di, Jon C. Calhoun, and Franck Cappello. 2020. FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data. https://arxiv.org/abs/2001.06139. Online.

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

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

[45] Brent Welch. 2005. POSIX IO extensions for HPC. In *4th USENIX Conference on File and Storage Technologies (FAST05)*. USENIX Association, USA, 1.

[46] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612.

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

[48] zstd. 2019. https://github.com/facebook/zstd/releases. Online.