# Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets

Xin Liang*, Sheng Di†, Dingwen Tao‡, Sihuan Li*, Shaomeng Li§, Hanqi Guo†, Zizhong Chen*, and Franck Cappello†¶
* University of California, Riverside, CA, USA
†Argonne National Laboratory, Lemont, IL, USA
‡University of Alabama, AL, USA
§National Center for Atmospheric Research
¶ University of Illinois at Urbana-Champaign, IL, USA
xlian007@ucr.edu, sdi1@anl.gov, tao@cs.ua.edu, sli049@ucr.edu,
shaomeng@ncar.edu, hguo@anl.gov, chen@cs.ucr.edu, cappello@mcs.anl.gov

***Abstract*—Today's scientific simulations require a significant reduction of the data size because of extremely large volumes of data they produce and the limitation of storage bandwidth and space. If the compression is set to reach a high compression ratio, however, the reconstructed data are often distorted too much to tolerate. In this paper, we explore a new compression strategy that can effectively control the data distortion when significantly reducing the data size. The contribution is threefold. (1) We propose an adaptive compression framework to select either our improved Lorenzo prediction method or our optimized linear regression method dynamically in different regions of the dataset. (2) We explore how to select them accurately based on the data features in each block to obtain the best compression quality. (3) We analyze the effectiveness of our solution in details using four real-world scientific datasets with 100+ fields. Evaluation results confirm that our new adaptive solution can significantly improve the rate distortion for the lossy compression with fairly high compression ratios. The compression ratio of our compressor is 1.5X∼8X as high as that of two other leading lossy compressors (SZ and ZFP) with the same PSNR, in the high-compression cases. Parallel experiments with 8,192 cores and 24 TB of data shows that our solution obtains 1.5X dumping performance and 1.72X loading performance compared with the second-best lossy compressor, respectively.**

## I. Introduction

An efficient data compressor is increasingly critical to today's scientific research because of the extremely large volume of data produced by high performance computing (HPC) simulations. Such big data are generally stored in a parallel file system (PFS), with limited storage space and limited I/O bandwidth to access. Some climate studies, for example, need to run large ensembles of 1 km×1 km simulations, with each instance simulating 15 years of climate in 24 h of computing time. Every 16 seconds, 260 TB of data will be generated across the ensemble, when estimating even one ensemble member per simulated day [1]. Based on our communication with researchers working on extreme-scale HPC simulations, they expect to see a compression ratio up to dozens of times or even 100:1, meaning that the bit-rate (i.e., the number of bits used to represent one data point on average after compression) should be no greater than 2 and best less than 1.

Although considerably reducing the data size can definitely improve I/O performance significantly as well as the post-analysis efficiency, the decompressed data may suffer from significant distortion compared with the original dataset. If the compression ratio reaches 64:1 (that is, the bit-rate is about 0.5 bit per 32 bit data point), the precision of the decompressed data will degrade significantly in general, even using the best existing lossy compressors such as SZ [2], ZFP [3], and FPZIP [4]. This will cause a huge distortion in the visualization (shown in Section III). The question addressed in this paper is, can we significantly improve the precision of the decompressed data for the lossy compression with a fairly low bit-rate compared to state-of-the-art lossy compressors?

Designing an efficient error-bounded lossy compressor that can significantly reduce the data size with a relatively high resolution of decompressed data is very challenging. We explain this point based on the two most effective lossy compressors [5]: SZ and ZFP. SZ and ZFP adopt largely different compression models: respectively, a data prediction model and an orthogonal transform model. In the data prediction model [2], each data value needs to be predicted by using its adjacent data points in multidimensional space according to the order of scanning data points. Moreover, the data used in the prediction during the compression have to be the decompressed values, in order to guarantee that the error bound is respected during the decompression, which is a strict limitation to the design of SZ [2]. Such a limitation may significantly degrade the prediction accuracy, especially for the lossy compression with a relatively large error bound, leading to limited compression quality. On the other hand, in the orthogonal transform-based compressor such as ZFP [3], the entire dataset has to be split into many small blocks (e.g., 4x4x4 for 3D data), each of which will be transformed to another decorrelated domain individually based on a fixed coefficient matrix. Relatively large error bound setting will introduce significant loss to the coefficients, leading to the over-distortion of decompressed data in turn.

In this paper, we propose an adaptive lossy compression framework in terms of the data prediction compression model that can obtain a stable, high compression quality when the error bound is set to a large value for reaching a high compression ratio. Specifically, our contributions are as follows:

- We propose an adaptive lossy compression framework that is more effective in compressing the scientific

datasets with relatively large error bounds. In particular, we split the whole dataset into multiple non-overlapped blocks, and select the best-fit prediction method based on their data features.
- We develop new prediction methods that are particularly effective for the lossy compression with relatively large error bounds. On the one hand, we develop a hybrid Lorenzo prediction method by combining the classic Lorenzo predictor [6] and the densest mean-value based data approximation method (also called mean-integrated Lorenzo prediction). On the other hand, we develop a linear regression method that can obtain much higher prediction accuracy in this case since the design is beyond the limitation that the decompressed values have to be used in the prediction.
- We explore how to select adaptively and efficiently the best-fit prediction method based on the data features across blocks during the compression. We also propose two optimization strategies that can further improve the prediction accuracy and quantization efficiency.
- We evaluate our proposed compression method compared with six other state-of-the-art lossy compressors, based on four well-known large datasets produced by real-world large-scale HPC simulations with a total of 104 fields. Evaluation results demonstrate that the compression ratio of our compressor is 1.5X∼8X as high as that of the two best lossy compressors (SZ and ZFP) with the same PSNR, respectively, in the high compression ratio cases. Parallel experiments with 8,192 cores and 24 TB of data shows that our solution obtains 1.5X dumping performance and 1.72X loading performance compared with the second-best lossy compressor, respectively.

The rest of the paper is organized as follows. In Section II, we discuss the related work. In Section III, we formulate the research problem. In Section IV, we provide an overview of the design on our new compression framework. In Section V, we present the mean-integrated Lorenzo prediction method. In Section VI, we describe the regression-based prediction method in details. In Section VIII, we present the evaluation results on real-world simulation data with 100+ fields. Finally, we conclude with a vision of future work in Section IX.

## II. Related Work

The I/O bottleneck has become one of the most serious issues for the overall execution performance of today's extreme-scale HPC scientific simulations. The recent I/O performance study [8] shows that the parallel I/O performance may only reach several GB/s on an optimized storage system facilitated with Luster and GPFS. As such, data compression is critical to significantly reduce I/O bottleneck for extreme-scale simulations.

Lossless compressors such as Gzip [9], FPC [10], FPZIP [4], BlosC [11] and other lossless compressors [12] cannot significantly reduce the floating point data size because of the largely random nature of the ending mantissa bits. Specifically, the lossless compression ratios are generally around 2:1 or even lower, according to the recent study [13].

Traditional lossy compressors (such as [14], [15]) do not respect a specific error bound set by users, such that the decompressed data may not be available for scientific analysis.

Error-controlled lossy compression techniques [2], [3], [16] have been considered the best trade-off solution compared with lossless compression, but it still suffers from limited performance gain, especially in situations with terabytes or even petabypes of data to process because of the limited compression ratios. Tao et al. [17] reported that it took about 2,300 seconds to write 1.8 TB of cosmology simulation data on a parallel file system when running 1,024 processes, and the total I/O processing time was up to 500 seconds using the best lossy compressor with a compression ratio of 3.6:1. Their work also concluded that the total I/O performance depends mainly on the compression ratio because of the bounded I/O bandwidth of the PFS. That is, further reducing the data size significantly will definitely improve the total performance.

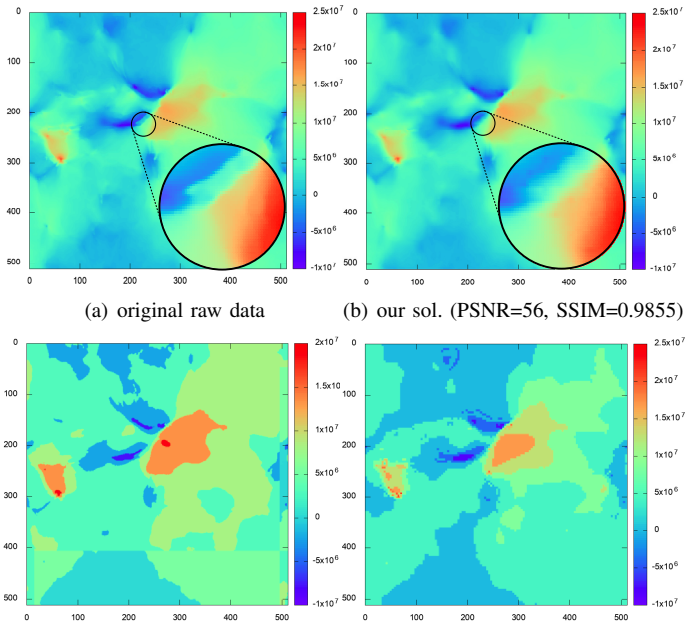
The existing lossy compressors, however, cannot keep high fidelity on the reconstructed data after the significant reduction of data size with a fairly high compression ratio (i.e. 40:1 or higher). The studies [18] of lossy compression of climate simulation data within a large ensemble shows that the compression ratio needs to be around 8:1∼10:1 in order to get an indistinguishable visualization of reconstructed data compared with the original data. Lu et al. [5] demonstrated that in a case study of fusion blob detection, the fusion blobs cannot be detected at all when the compression ratio increases up to 30:1 for ZFP or 100:1 for SZ because of over distorted visualization of the data. More examples of the over distortion of lossy compression data in high-ratio compression cases are presented in Section VIII of this paper. To control the distortion of the data in this situation, we develop an adaptive compression framework and explore more effective data prediction strategies. Evaluation results using four real-world HPC simulations with 100+ fields show that our solution outperforms the second-best lossy compressor significantly, from the perspective of both rate distortion and visualization.

## III. Problem Formulation

In this paper, we target a critical, challenging research issue: how to reduce the distortion of data during an error-bounded lossy compression with a fairly high compression ratio. On the one hand, the decompressed dataset must respect a strict error bound (denoted by $\varepsilon$) on each data point compared with the original dataset. Not only do we need to guarantee the maximum error bounded within an specified error bound, but we also need to maximize the peak single-to-noise ratio (PSNR) for the lossy compression of scientific data. PSNR is a commonly used indicator to assess the distortion of data during the lossy compression. It is defined as follows:

$$psnr = 20 \cdot log_{10}((d_{max} - d_{min})/rmse) \quad (1)$$

where $N$ refers to the number of data points, *rmse* = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - d_i')^2}$; $d_i$ and $d_i'$ refer to the original and



(a) original raw data (b) our sol. (PSNR=56, SSIM=0.9855)

(c) SZ (PSNR=34.7, SSIM=0.7349) (d) ZFP (PSNR=35, SSIM=0.7004)

Fig. 1. Data Distortion of NYX(velocity_x:slice 100) with CR=156:1

decompressed data values, respectively; $d_{max}$ and $d_{min}$ refer to the max and min values, respectively. The larger the PSNR, the lower the mean squared error of the reconstructed data versus original data, meaning higher overall precision of reconstructed data and thus more accurate post-analysis.

Because of the diverse value changes in different regions of a dataset, the reconstructed data would be distorded significantly from the original values after the lossy compression with a high compression ratio such as 100:1. In Fig. 1(a), (c) and (d), we can observe that the visualization using slice 50 of the 3D dataset velocity_x is largely distorted by two existing state-of-the-art lossy compressors, SZ and ZFP, with a high compression ratio up to 156:1. In particular, the decompressed data under SZ has an artifact issue (i.e., the large difference in the bottom), as shown in Fig. 1(c); ZFP simply flushes local regions to a single color, making the reconstructed data unavailable for user's post-analysis. As shown in Fig. 1(b), the decompressed data under our new solution will lead to very high visual quality with the same compression ratio.

In summary, our objective is to maximize the PSNR during error-bounded lossy compression with a high compression ratio such as 100:1. A satisfactory lossy compressor must conform to the following four conditions/criteria.

1) It should respect a specified maximum error: the difference between the reconstructed data and original data must be bounded strictly for each data point from user specified limits.
2) With the maximum error controlled, **PSNR** must also be as high as possible, such that the overall distortion of the data is controlled well for guaranteeing effective post-analysis and high visualization quality.
3) The developed compressor should not degrade the compression quality in the cases demanding high precision (with relatively low compression ratios).
4) The new compression technique must have low computation cost, leading to comparable compression/decompression rate with the existing state-of-the-art compressors such as SZ and ZFP.

## IV. Design Overview

In the following, we first present an overview of our adaptive lossy compression framework and then describe the compression techniques in detail.

The key idea of our adaptive solution is splitting the entire dataset into multiple non-overlapped equal-sized blocks in multidimensional space and selecting the best-fit data prediction method dynamically for each block based on its data feature. Algorithm 1 presents the pseudo-code of the entire design. The basic idea is to select in each block the best-fit prediction method, from among the three data prediction approaches: *classic Lorenzo predictor* [6], *mean-integrated Lorenzo predictor*, and *linear regression-based predictor*. The first one was already adopted by some existing compressors such as SZ and FPZIP, while the other two are proposed as a critical contribution in this paper, because they can improve the lossy compression quality significantly.

**Algorithm 1** Adaptive Error-bounded Compressor

**Input**: user-specified error bound $\varepsilon$
**Output**: compressed data stream in form of bytes

1: Estimate the densest position (denoted as $v_0$) and calculate the frequency (denoted $p_1$) of the densest error-bound-based interval surrounding it;
2: Calculate the densest frequency of classic Lorenzo predictor (denoted $p_2$);
3: **if** ($p_1 > p_2$) **then**
4: $\mu \leftarrow \frac{\sum_{|d_i - v_0| \le \varepsilon} d_i}{\|\{d_i | |d_i - v_0| \le \varepsilon\}\|}$; /*Compute mean value of densest interval*/
5: $\ell$-Predictor $\leftarrow$ mean-integrated Lorenzo predictor;
6: **else**
7: $\ell$-Predictor $\leftarrow$ classic Lorenzo predictor;
8: **end if**
9: **for** (each block in the multi-dimensional space) **do**
10: Calculate regression coefficients; /*4 coefficients/block in 3d dataset*/
11: **end for**
12: Calculate statistics of all coefficients for compression of coefficients later;
13: **for** (each block in the multi-dimensional space) **do**
14: Create a sampling set (denoted $S_M$) with $M$ sampled data points;
15: Compute cost values $E_{reg\text{-}predictor}$ and $E_{\ell\text{-Predictor}}$ based on $S_M$;
16: **if** ($E_{reg\text{-}predictor} < E_{\ell\text{-Predictor}}$) **then**
17: Execute regression-based prediction and quantization;
18: **else**
19: Execute $\ell$-Predictor and quantization;
20: **end if**
21: **end for**
22: Construct Huffman tree according to the quantization array;
23: Encode/compress quantization array by Huffman tree;
24: Compress regression coefficients;

We describe our algorithm in the following text. At the beginning (line 1), the algorithm searches for the densest interval based on the error-bound $\varepsilon$ and calculates its data frequency (denoted by $p_1$), in order to estimate the prediction ability of the *mean-integrated Lorenzo predictor*. This part involves three steps: (1) $\sqrt{N}$ data points will be sampled uniformly in space; (2) the mean value of the sampled data points is calculated and a set of consecutive intervals (each with $2\varepsilon$ in length) will be constructed surrounding the mean value; (3) we then calculate the number of data points in the consecutive intervals and select the one with the highest

frequency of data points as the densest interval, whose center is called the densest position (denoted as $v_0$). Here $\sqrt{N}$ is chosen by heuristics because it already exhibits good accuracy. On line 2, the algorithm checks the prediction ability of the *classic Lorenzo predictor*, by calculating the data frequency (denoted $p_2$) of its error-bound based prediction interval (i.e., [pred_value$-\varepsilon$ , pred_value+$\varepsilon$], where pred_value refers to the predicted value), based on 1% of uniformly sampled data points. The number of sampled data points (i.e., 1%) is a heuristic setting, which is similar to the configuration of SZ. Based on the sampled data points, we select the best-fit Lorenzo predictor (denoted as $\ell$-PREDICTOR) according to our estimated prediction ability of the two predictors (line 3-8). If the best-fit predictor is the mean-integrated Lorenzo predictor, we need to calculate the mean value of the densest interval (line 4), which will be used later.

After determining the best-fit Lorenzo predictor, the algorithm calculates the linear regression coefficients for each block (lines 9-11), as well as the statistics (such as the value range of the coefficients), which will be used in the compression of the coefficients later (Section VI-B).

The most critical stage is scanning the entire dataset and performing the best-fit prediction and linear-scaling quantization in each block (lines 13-21). In each block, $M$ data points (1/8 data points for 2D dataset and 1/9 for 3D dataset) are sampled uniformly in space, in order to select the best-fit prediction method as accurately as possible. The sampling method will be further detailed in Section VI-A. Then, the algorithm determines which prediction method (either regression-based predictor or $\ell$-PREDICTOR) should be used in the current block in terms of their estimated overall prediction errors (denoted by $E_{reg\text{-}predictor}$ and $E_{\ell\text{-PREDICTOR}}$ respectively). How to estimate the prediction errors for the two predictors will be detailed in Section VII-B.

Our algorithm then compresses the quantization array constructed in the prediction stage by Huffman encoding (lines 22-23). It also compresses the regression coefficients for the blocks selecting the regression-based prediction methods, by IEEE 754 binary analysis (detailed in Section VI-B).

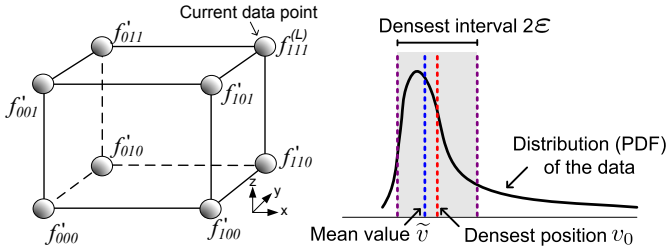
The time complexity of the algorithm is $O(N)$, because the algorithm is composed of three parts, whose time complexities are no greater than $O(N)$. Specifically, the first part estimates densest frequency (lines 1-2) with a time complexity of $O(\sqrt{N})$; the second part calculates regression coefficients (lines 9-11) that costs $O(N)$ in total (for details, see Lemma 1 to be presented later); and the last part performs prediction+quantization (lines 13-21), which also costs $O(N)$ since each data point will be scanned only once.

## V. MEAN-INTEGRATED LORENZO PREDICTOR

In this section, we develop a new, efficient predictor in terms of the classic Lorenzo predictor [6]. Fig. 2(a) illustrates the classic Lorenzo prediction method in a 3D dataset. Specifically, it predicts the current data point based on the following formula for a 3D dataset:

$$f^{(L)}_{111} = f'_{000} + f'_{011} + f'_{101} + f'_{110} - f'_{001} - f'_{010} - f'_{100} \quad (2)$$

where $f^{(L)}$ and $f'$ refer to the predicted value and decompressed value, respectively. $\{111\}$ is the current data point to deal with, and the other seven data points are adjacent to it on a unit cube which have been processed. $f^{(L)}_{111}$ is the predicted value for the data point $\{111\}$ and the decompressed value $f'_{111}$ can be obtained by applying the linear-scaling quantization on the difference between $f^{(L)}_{111}$ and the origin data value at $\{111\}$. The compressor will continue this procedure data point by data point until all the data are processed.




(a) Classic Lorenzo predictor (b) Data approximation by mean value

Fig. 2. Illustration of mean-integrated Lorenzo predictor

The classic Lorenzo predictor has a significant defect: many predicted values would be uniformly skewed from the original values if the error bound is relatively large in the lossy compression, leading to an unexpected artifact issue as illustrated in Fig. 1(c). Our developed mean-integrated Lorenzo predictor can solve this issue well.

The fundamental idea is approximating those data points whose values are clustered intensively by a fixed value, if majority of data values are clustered to a small interval with pretty high density (called *densest interval*). This situation appears in about one-third of the fields of the NYX cosmology simulation [28] and about half the fields of the Hurricane simulation [27]. In the dark matter density field of NYX, for instance, 84+% of the data values are in the range of [0,1], while the remaining data are in the range of [1,1.34$\times 10^4$].

Suppose we are given a dataset $D = \{d_i | i = 1, \ldots, N\}$ such that the interval [$v_0-\varepsilon$,$v_0$+$\varepsilon$] can cover a large percentage of data points, where $\varepsilon$ is the compression error bound and $v_0$ is the densest position (as illustrated in Fig. 2(b)). If this percentage is greater than some threshold, we will select the mean-integrated Lorenzo predicator. In practice, we set the threshold to the sampled prediction accuracy of classic Lorenzo predictor (line 3 in Algorithm 1). However, the classic Lorenzo predictor would be highly over-estimated when error bound is relatively large because the sampling stage does not take into account the impact of decompressed data, leading to a serious artifact issue. In order to mitigate this issue, we let the algorithm opt to select the mean-integrated Lorenzo predictor directly when [$v_0-\varepsilon$,$v_0$+$\varepsilon$] can cover more than half of the data. Now, we need to derive an optimal value to approximate the data in this interval.

*Lemma 1:* The optimal value used to approximate the majority of data points should be the mean value of data in the densest interval [$v_0-\varepsilon$,$v_0$+$\varepsilon$].

*Proof:* If a fixed value $v$ is used to approximate all the data points in this interval, the corresponding MSE can be

represented as follows:

$$MSE = \int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)(x-v)^2 dx$$

where $p_d(x)$ is the probability density function. Then, the problem becomes an optimization problem aiming to minimize MSE. Letting the partial derivative $\frac{\partial MSE}{\partial v} = 0$, we have $-2\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)xdx + 2v\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)dx = 0$. Solving this equation will obtain the optimal value $\widetilde{v}$=$\frac{\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)xdx}{\int_{v_0-\varepsilon}^{v_0+\varepsilon} p_d(x)dx}$. It is the mean value of all data points in this interval (as shown in Fig. 2(b)). This lemma also holds in discrete cases, where the optimal approximation value $\widetilde{v}$ can be calculated by $\frac{\sum_{x\in[v_0-\varepsilon,v_0+\varepsilon]} x}{\|\{x||x-v_0|\leq\varepsilon\}\|}$ in practice, where $\|\{x\,||x-v_0|\leq\varepsilon\}\|$ is the number of data points in the densest interval. ■

## VI. Regression-based Prediction

Although the proposed mean-integrated Lorenzo predictor alleviates the artifact and reduces prediction errors in predicting the intensively-clustered data, it does not work well on the data following a rather uniform distribution. To address this issue, we propose a regression-based prediction model, which can deal with generic datasets. We adopt a linear-regression model instead of a higher-order regression model considering the overhead. Quadratic regression model, for example, requires 2.5X the number of coefficients the linear-regression model needs, with ≥3X the computation workload.

### A. Derivation of Regression Coefficients

In what follows, we derive the regression coefficients from a generic perspective in terms of a dataset with $m$ dimensions ($n_1 \times n_2 \times \cdots \times n_m$), which can be easily extended to block-wise situations. The value at position $\boldsymbol{x} = (i_1, \ldots, i_m)$ is denoted as $f(\boldsymbol{x}) = f_{i_1 \ldots i_m}$, where $i_j$ is its index along each dimension. We also denote $f^{(r)}$ as the linear regression prediction. Then the linear regression model can be built as:

$$f^{(r)}(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{\beta} + \alpha$$

where $\boldsymbol{\beta}$ = ($\beta_1$,…,$\beta_m$) denotes the slope coefficient vector and the constant $\alpha$ is the intercept coefficient. By redefining $\boldsymbol{x'} = (1, i_1, \ldots, i_m)$ and $\boldsymbol{\beta'}$ = ($\beta_0$=$\alpha$,$\beta_1$,…,$\beta_m$), the formula above can be rewritten as:

$$f^{(r)}(\boldsymbol{x'}) = \boldsymbol{x'}^T\boldsymbol{\beta'}$$

Since each data point corresponds to a position $\boldsymbol{x}$ and its value $f(\boldsymbol{x})$, the objective of the regression model is to minimize the squared error (SE) between predicated and original values:

$$SE = \sum_{\boldsymbol{x'}\in\{(1,i_1,\ldots,i_m)|0\leq i_j<n_j\}} (\boldsymbol{x'}^T\boldsymbol{\beta'} - f_{i_1\ldots i_m})^2$$

This is a convex function, and its optimal can be obtained by derivation. The derivation over each element in $\boldsymbol{\beta'}$ will result in a linear system of $m$ unknowns. By solving the linear system, the optimal solution can be achieved as:

$$\boldsymbol{\beta'} = (X^TX)^{-1}X^Ty \tag{3}$$

$X$ is the full permutation of $\{i_1,i_2,\cdots,i_m\}$, where $i_j\in\{0,1,\cdots,n_j-1\}$, and $y$ is the sequence of the corresponding data values ($f_{00\ldots0}$,$f_{00\ldots1}$,$\cdots$,$f_{(n_1-1)(n_2-1)\ldots(n_m-1)}$).

Since $X$ is a $\{\prod_{1\leq i\leq m} n_i\}\times(m+1)$ matrix, computing the closed-form solution (3) is very expensive. However, we can derive the solution to a simple form, significantly reducing the computation cost. Let us take a 3D dataset as an example to describe our method, which can be extended to datasets with higher dimensions easily without loss of generality.

*Lemma 2:* The regression coefficients of a 3D dataset with dimensions $n_1$, $n_2$, $n_3$ can be calculated as:

$$\begin{cases} \beta_1 = \frac{6}{n_1n_2n_3(n_1+1)}(\frac{2V_x}{n_1-1} - V_0) \\ \beta_2 = \frac{6}{n_1n_2n_3(n_2+1)}(\frac{2V_y}{n_2-1} - V_0) \\ \beta_3 = \frac{6}{n_1n_2n_3(n_3+1)}(\frac{2V_z}{n_3-1} - V_0) \\ \beta_0 = \frac{V_0}{n_1n_2n_3} - (\frac{n_1-1}{2}\beta_1 + \frac{n_2-1}{2}\beta_2 + \frac{n_3-1}{2}\beta_3) \end{cases} \tag{4}$$

where $V_0 = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} f_{ijk}$, $\quad V_x = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} i * f_{ijk}$,

$V_y = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} j * f_{ijk}$, $\quad V_z = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1} k * f_{ijk}$.

*Proof:* Substitute the index with the coordinate values. The SE expression will turn out to be:
*SE*=$\sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1}(\beta_0+\beta_1 i+\beta_2 j+\beta_3 k - f_{ijk})^2$
The following linear system can be derived by getting all its partial derivatives over the coefficients and setting them to 0:

$$\sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1}\sum_{k=0}^{n_3-1}\begin{bmatrix}1 & i & j & k\\ i & i^2 & ij & ik\\ j & ij & j^2 & jk\\ k & ik & jk & k^2\end{bmatrix}\begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\\ \beta_3\end{bmatrix} = \begin{bmatrix}V_0\\ V_x\\ V_y\\ V_z\end{bmatrix}$$

Since $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ and $\sum_{i=0}^{n-1} i^2 = \frac{(2n-1)n(n-1)}{6}$, the above linear system can be simplified to:

$$\begin{bmatrix}1 & \frac{(n_1-1)}{2} & \frac{(n_2-1)}{2} & \frac{(n_3-1)}{2}\\ 1 & \frac{(2n_1-1)}{3} & \frac{(n_2-1)}{2} & \frac{(n_3-1)}{2}\\ 1 & \frac{(n_1-1)}{2} & \frac{(2n_2-1)}{3} & \frac{(n_3-1)}{2}\\ 1 & \frac{(n_1-1)}{2} & \frac{(n_2-1)}{2} & \frac{(2n_3-1)}{3}\end{bmatrix}\begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\\ \beta_3\end{bmatrix} = \begin{bmatrix}\frac{V_0}{n_1n_2n_3}\\ \frac{2V_x}{n_1n_2n_3(n_1-1)}\\ \frac{2V_y}{n_1n_2n_3(n_2-1)}\\ \frac{2V_z}{n_1n_2n_3(n_3-1)}\end{bmatrix}$$

After that, Gaussian elimination can be leveraged to transfer the linear system to the following:

$$\begin{bmatrix}1 & \frac{(n_1-1)}{2} & \frac{(n_2-1)}{2} & \frac{(n_3-1)}{2}\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\\ \beta_3\end{bmatrix} = \begin{bmatrix}\frac{V_0}{n_1n_2n_3}\\ \frac{6}{n_1n_2n_3(n_1+1)}(\frac{2V_x}{n_1-1} - V_0)\\ \frac{6}{n_1n_2n_3(n_2+1)}(\frac{2V_y}{n_2-1} - V_0)\\ \frac{6}{n_1n_2n_3(n_3+1)}(\frac{2V_z}{n_3-1} - V_0)\end{bmatrix}$$

Then Equation (4) can be derived accordingly. ■

These coefficients will be used in the regression model for predicting the data accurately. Each data point with index $(i,j,k)$ will be predicted as $f_{ijk}^{(r)} = \beta_0 + i\beta_1 + j\beta_2 + k\beta_3$. Then, we will compute the difference between each predicted value $f_{ijk}^{(r)}$ and its original value $f_{ijk}$, and perform the linear-scaling quantization [2] to convert the floating-point values to integer codes. The data size will be significantly reduced after conducting Huffman encoding on the quantization codes.

### B. Compressing Regression Coefficients

We adopt the block size 6×6×6 for 3D data and 12×12 data for 2D data in our implementation, since such settings already lead to satisfying compression quality. For each block that adopts the linear regression model, four coefficients have to be kept in the compressed bytes together with the encoding of regression-based predicted values in that block. If each

block is a 6×6×6 cube, the overhead of saving the four coefficients (single-precision floating-point values) would be $\frac{4}{6\times6\times6}=\frac{1}{54}$ of the original data size (i.e., bit-rate=0.6). Note that we are targeting high-compression ratios (such as 100:1), so we have to further compress the coefficients significantly by a lossy compression technique with controlled impact of lossy coefficients on the prediction accuracy. We reorganize all the coefficients into four groups and compress each group of coefficients in a similar way by the unpredictable data compression method used in SZ [19].

## VII. Adaptive Selection of Best-fit Predictor

In this section, we analyze the nature of the linear regression-based predictor (proposed in Section VI) and Lorenzo predictor (introduced in Section V). We also propose a cost function (or a metric), based on which we can accurately select the best-fit predictor via a sampling approach.

### A. Analysis of Linear Regression versus Lorenzo Predictor

The two predictors are particularly suitable for various data blocks with different data features.

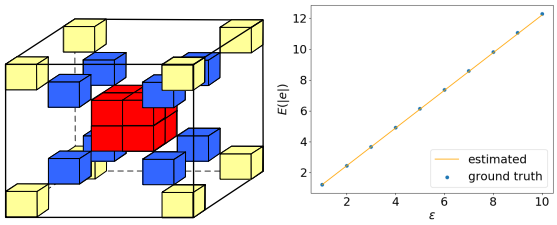
For each data point to predict, the Lorenzo predictor [6] constructs a fixed quadratic hyperplane based on its 7 adjacent data points in a $2\times2\times2$ cube. No coefficients need to be saved for reconstructing the hyperplane during the decompression. However, Lorenzo predictor must conform to a strict condition when being used in lossy compression [2]. The prediction performed during the compression must use the decompressed values instead of original values; otherwise, unexpected data loss would be accumulated during the decompression, introducing significant distortion of data eventually. Using the decompressed values to perform the Lorenzo predictor would degrade the prediction accuracy significantly especially when the error bound is relatively large [2], leading to limited compression quality.

Unlike Lorenzo predictor, our linear regression-based predictor constructs an MSE-minimized linear hyperplane ($f^{(r)}_{111}$=$\beta_0+\beta_1x+\beta_2y+\beta_3z$) with four extra coefficients to store for each data block. Thus the block size cannot be too small. The advantage of this predictor, however, is that it does not depend on the decompressed values during the prediction stage because the hyperplane will be reconstructed by the four coefficients. In contract, linear-regression may not be as accurate as Lorenzo predictor if the data exhibit spiky changes in a pretty small region or the compression error bound is relatively small, because the Lorenzo predictor corresponds to a quadratic hyperplane. This issue inspires us to seek an adaptive solution between the two predictors.

### B. Estimate Prediction Errors and Select Best-fit Predictor

We adopt an effective, lightweight sampling method to select the best of the two predictors. In the following, we use the 3D dataset to explain our idea, cases with other dimensions could be extended similarly.

In a 3D block (6×6×6), we sample 24 points that are distributed along the diagonal lines. These points can also be




(a) Points sampled in a block (b) Decompressed noise estimation

Fig. 3. Sample points and decompressed noise estimation

regarded as on the 8 corners of the innermost 2×2×2 cube, 4×4×4 cube and 6×6×6 cube, respectively, as shown in Fig. 3(a). We denote the set of the 24 sample points by $\{S_{24}\}$. Then, the cost function for both regression model and the Lorenzo predictor is defined as follows:

$$E'_{predictor}=\sum_{(i,j,k)\in\{S_{24}\}}|f^{(p)}_{ijk}-f_{ijk}| \tag{5}$$

where $f^{(p)}_{ijk}$ refers to the predicted values ($f^{(r)}_{ijk}$, $f^{(L)}_{ijk}$). If one predictor is better than the other, it tends to have a lower cost (i.e., summed error). The reason we adopt absolute error instead of squared error in the cost function is that the squared errors may easily over amplify the cost value if there is one outlier, leading to skewed cost values. Formula (5) can be used directly as the cost function (denoted $E_{reg\text{-}predictor}$ in Algorithm 1) for the regression-based predictor.

The Lorenzo predictor, however, would be overestimated because the formula does not take into account the influence of decompressed data, such that we have to further adjust the cost function for the Lorenzo predictor. Without loss of generality, we assume that the compression errors are independent random variables following a uniform distribution in $[-\varepsilon,\varepsilon]$, according to the recent study on the distribution of compression errors [13]. Then, the perturbation in the final prediction will be a random variable $e$ following a shifted and scaled Irwin-Hall distribution with parameter $n=7$, which is a piecewise function. However, the expectation of the absolute value is hard to derive. Fortunately, since the Lorenzo predictor always adopts 7 points to predict the data, the corresponding value can be approximated offline. We take $1M$ samples from this distribution and compute the expectation of their absolute values. The fitting curve is supposed to be a linear curve, because the error bound is only a scale factor for this distribution. Then, we achieved a very good fitting curve ($E(|e|)=1.22\varepsilon$) as illustrated in Fig. 3(b). The cost (i.e., aggregated error) of the classic Lorenzo predictor can be adjusted as follows:

$$\begin{aligned}&E'_{LP}=\textstyle\sum_{(i,j,k)\in\{S_{24}\}}|f^{(L)}_{ijk}-f_{ijk}+e|\\&\leq\textstyle\sum_{(i,j,k)\in\{S_{24}\}}|f^{(L)}_{ijk}-f_{ijk}|+\sum_{(i,j,k)\in\{S_{24}\}}|e|\\&\approx\textstyle\sum_{(i,j,k)\in\{S_{24}\}}|f^{(L)}_{ijk}-f_{ijk}|+24*1.22\varepsilon\end{aligned} \tag{6}$$

We estimate the prediction cost by this inequality because we opt to select the regression model considering the artifact issue that may occur to the Lorenzo predictor in the situation with relatively large error bounds. When the error bound is small, this extra adjustment has little impact on the final selection.

The cost value of the mean-integrated Lorenzo predictor(denoted $E'_{mLP}$) is estimated as the minimum value be-

tween the classic Lorenzo prediction error (i.e., Formula (6)) and mean prediction error, as shown below:

$$E'_{mLP} \approx \sum_{(i,j,k)\in\{S_{24}\}} min(|f^{(L)}_{ijk} - f_{ijk}|+1.22\varepsilon\,,\, |\mu - f_{ijk}|) \quad (7)$$

where $\mu$ is calculated in line 4 of Algorithm 1.

In summary, as for the $\ell$-PREDICTOR proposed in Algorithm 1, we estimate the error cost for the classic Lorenzo predictor and mean-integrated Lorenzo predictor by Formula (6) and Formula (7), respectively. In each data block, we select the final predictor with lowest cost based on their cost functions.

## VIII. PERFORMANCE EVALUATION

In this section, we compare our proposed compression method with three error-bounded compressors (SZ 1.4.13 [2], ZFP 0.5.2 [3], and TTHRESH [20]), which are currently the best existing generic lossy compressors for scientific data compression [5]. We also compare our method with other lossy compressors that are widely used in the scientific simulations, namely VAPOR [21] (wavelet compressor), FPZIP [4] and ISABELA [22]. We also try best to optimize the compression quality of all the six lossy compressors from the perspective of PSNR. For instance, we adopt the absolute error bound mode for both SZ and ZFP in that they lead to better compression quality than other modes in our experiments. For SZ, we adopt one layer in the multi-dimensional prediction and the optimized number of quantization bins based on initial 65,536 bins, which is the best setting as confirmed by the SZ developers. We also asked the developer of VAPOR to optimize the parameters for the datasets used in our experiments.

### A. Experimental Setting

We conduct our experimental evaluations on a supercomputer using 8,192 cores (i.e., 256 nodes, each with two Intel Xeon E5-2695 v4 processors and 128 GB of memory, and each processor with 16 cores). The storage system uses General Parallel File Systems (GPFS). These file systems are located on a raid array and served by multiple file servers. The I/O and storage systems are typical high-end supercomputer facilities. We use the file-per-process mode with POSIX I/O [23] on each process for reading/writing data in parallel [1]. The HPC application data are from multiple domains including CESM-ATM climate simulation [26], Hurricane ISABEL simulation [27], NYX cosmology simulation [28], SCALE-LETKF weather simulation (called *S-L Sim* for short) [29]. Each application involves many simulation snapshots (or time steps). We assess only meaningful fields with relatively large data sizes (other fields have constant data or too small data sizes). Table I presents all the 104 fields across these simulations. The data sizes per snapshot are 1.2 GB, 3.2 GB, 3 GB and 2 GB for the above four applications respectively.

We assess the compression quality based on the four criteria proposed in Section III. Since PSNR is the most critical indicator as discussed in Section III, we mainly adopt this metric to access the distortion of data. In addition, we also evaluate the Pearson correlation and structural similarity (SSIM) index for different compressors because they were mentioned in some literatures [18], [30]. Pearson correlation can be computed by $\rho_{X,Y}=\frac{E(X-\mu_X)E(Y-\mu_Y)}{\sigma_X\sigma_Y}$, where $\mu_c$ and $\sigma_c$ ($c\in\{X,Y\}$) are means and deviations of $X$ and $Y$, respectively. SSIM is considered a critical metric of lossy compression by the climate community [30]. The higher the Pearson correlation or SSIM, the better the compression quality.

[1]POSIX I/O performance is close to other parallel I/O performance such as MPI-IO [24] when thousands of files are written/read simultaneously on GPFS, as indicated by a recent study [25].

TABLE I
SIMULATION FIELDS USED IN THE EVALUATION

| Application | # Fields | Dimensions | Examples |
|---|---|---|---|
| CESM-ATM | 79 | 1800×3600 | CLDHGH, CLDLOW ··· |
| Hurricane | 13 | 100×500×500 | CLOUDf48, Uf48 ··· |
| NYX | 6 | 512×512×512 | dark_matter_density, v_x ··· |
| S-L Sim | 6 | 98×1200×1200 | U, V, W, QC ··· |

### B. Evaluation Results

We first check the maximum compression errors of our compressor using all the 104 fields and confirm that our compressor can respect the error bound (denoted by $\varepsilon$) strictly. We present a few examples in Table II.

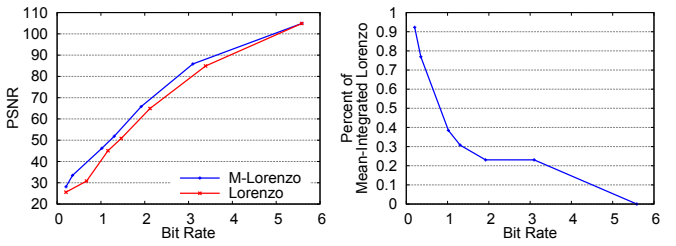
TABLE II
MAXIMUM COMPRESSION ERROR VS. ERROR BOUND

| fields | bound | max err | bound | max err |
|---|---|---|---|---|
| Hurricane-CLOUDf48 | 0.1 | 0.099999955 | 0.01 | 0.00999998 |
| NYX-v_x | 0.1 | 0.0999966 | 0.01 | 0.009999995 |
| CESM-CLDHGH | 0.1 | 0.099949 | 0.01 | 0.0099999 |

In what follows, we first compare the three fundamental predictors (Lorenzo, mean-integrated Lorenzo and linear regression) to showcase the importance of the adaptive design in Fig. 4 and Fig. 5. After that, we compare the overall rate-distortion among all the 6 lossy compressors in Fig. 6, which is the most critical evaluation result in terms of compression quality. We also demonstrate the difference of visual quality across various lossy compressors with the same compression ratios. Finally, we investigate the parallel I/O performance gain with different execution scales using our compressor against two other most efficient state-of-the-art compressors.

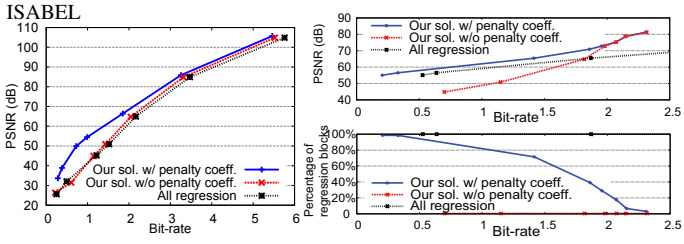
Fig. 4 demonstrates the effectiveness of our mean-integrated Lorenzo predictor (denoted M-Lorenzo) over the original Lorenzo predictor, using Hurricane ISABEL dataset. As shown in Fig. 4 (a), the mean-integrated Lorenzo always leads to a smaller bit-rate (or higher compression ratio) than the original Lorenzo predictor does with the same level of rate distortion (i.e., PSNR). Specifically, the mean-integrated Lorenzo predictor obtains the compression ratio about 3X as high as that of the original Lorenzo predictor, when the PSNR is around 30. Fig. 4 (b) shows the fraction of the fields adopting the mean-integrated Lorenzo predictor in the Hurricane datasets. We observe that the percentage drops as bit-rate increases, due to the fact that higher bit-rate corresponds to higher precision for the original Lorenzo predictor, which leads to lower percentage on choosing the mean-integrated Lorenzo predictor in turn. Similar phenomenons can also be observed in other datasets, which we did not show here because of the page limitation.

In Section VII-B, we derived a *penalty coefficient*, which is critical to determine the best-fit predictor. In Fig. 5, we present




(a) Overall Rate-distortion (b) Percentage of M-Lorenzo Predictor

Fig. 4. Effectiveness of Mean-Integrated Lorenzo Predictor using Hurricane-ISABEL

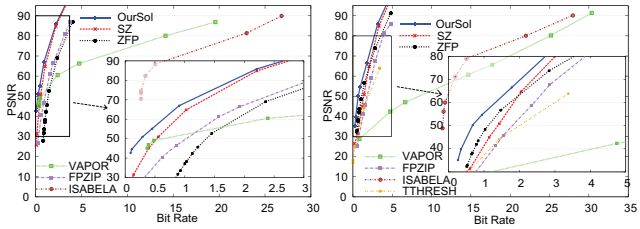



(a) Overall Rate-distortion (b) Detailed analysis using TCf48

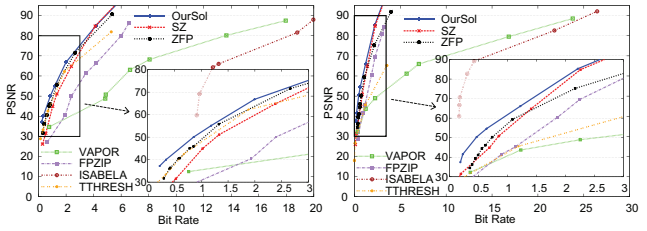
Fig. 5. Significance Analysis for penalty coefficient using Hurricane-ISABEL

the significance of the *penalty coefficient*, by comparing three solutions (the solution with/without penalty coefficient and the solution adopting only regression-based predictor). As shown in Fig. 5(a), the solution with our derived penalty coefficient outperforms the other two significantly. As illustrated in Fig. 5(b), without the penalty coefficient, the compression quality would be degraded significantly, since a large majority (99.6%) of blocks would select the Lorenzo predictor (see red curve in the bottom sub-figure of Fig. 5(b)) because of over-estimation of the Lorenzo prediction ability as discussed in Section VII-B. We also present the percentage of blocks selecting the regression-based predictors, demonstrating that our adaptive solution does select different predictors for different blocks during the compression. The percent of regression blocks drops from 98% to 3% as the bit-rate increases, which is consistent with the compression quality of the two methods.

In Fig. 6, we present the overall rate-distortion (PSNR versus bit-rate (or compression ratio)) calculated using all the fields for each application (Fig. 6 (a) is missing TTHRESH because it cannot work on 2D dataset). It is observed that under the same compression ratio, our solution leads to significantly higher PSNR with than other compressors. Specifically, our solution, SZ and ZFP generally exhibit better rate-distortion than other compressors including FPZIP, VAPOR, and TTHRESH, all of which outperform ISABELA significantly. From among the three best compressors, the PSNR of our compressor is 10%∼100% higher than that of the other two (SZ and ZFP) when the compression ratio is the same. This gap increases with higher compression ratio, as shown in the four sub-figures. In particular, as for CESM-ATM data, when the PSNR is about 45, our compressor leads the compression ratio to 200:1, while SZ and ZFP get the compression ratio of 76:1 and 25:1 respectively. Based on the four applications, we observe that the compression ratio of our compressor is 1.5X∼8X as high as that of SZ and ZFP with the same PSNR, which is a significant improvement.




(a) CESM-ATM (b) Hurricane




(c) NYX (d) S-L Sim

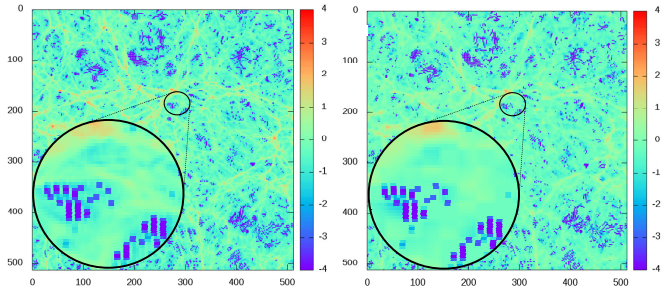
Fig. 6. Rate-distortion (PSNR versus Bit-rate or Compression Ratio)

We select three typical examples from the 104 fields across different applications to demonstrate the visual quality of the decompressed data with different compression ratios compared with the original data (also known as raw data), due to the space limitation of the paper.

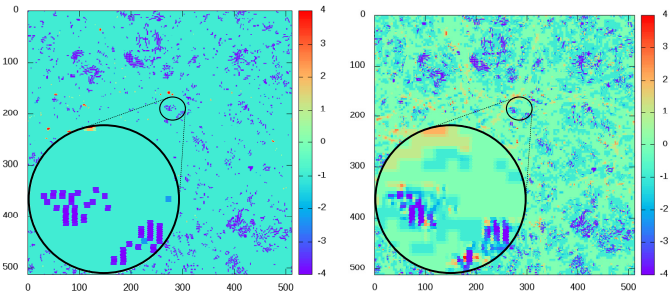
As demonstrated in Section III (Fig. 1)), the decompressed data under our solution has a better visual quality than either SZ or ZFP does, based on the NYX (velocity_x) dataset. Correspondingly, the SSIM index of our solution (0.9855) is higher than that of SZ (0.7349) and that of ZFP (0.7004) by 34% and 40%, respectively.

As mentioned previously, we note that some fields have very large value ranges while most of data are clustered in a small close-to-zero range, so the users compute the logarithm of the data before their analysis or visualization. We compress the log data instead of the original data since this would improve the compression quality for all lossy compressors, as suggested by users. As an example, we present the compression result of dark_matter_density field from NYX simulation in Fig. 7. By zooming in a small region, we can clearly see that our solution has much higher resolution than others. This is consistent with the SSIM measure: the SSIM index of SZ is higher than others by 62.8% and 82.5%, respectively. By observing the decompressed images, SZ suffers from a serious data loss as shown in Fig. 7 (c) because many data points would be flushed to the same values when the error bound is relatively large. ZFP suffers from the unexpected mosaic effect because it splits the entire dataset into pretty small blocks (4×4×4) and the decompressed data will be flushed to the same value in each block when the compression ratio is very high.

Furthermore, we show the visualization result of the CLOUDf field in Hurricane-ISABELA in Fig. 8. Similarly, the decompressed data under our solution has a better visual quality than does SZ or ZFP. Besides some unconspicuous strips (artifacts) in the blue background, SZ also has some distortion in the enlarged region. On the other hand, ZFP has little distortion in the background, but exhibits block-wise effects in the enlarged region. Our solution has little distortion

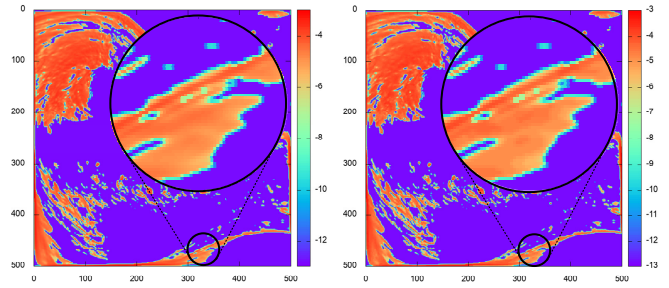


(a) original raw data (b) our sol. (PSNR=29,SSIM=0.6867)

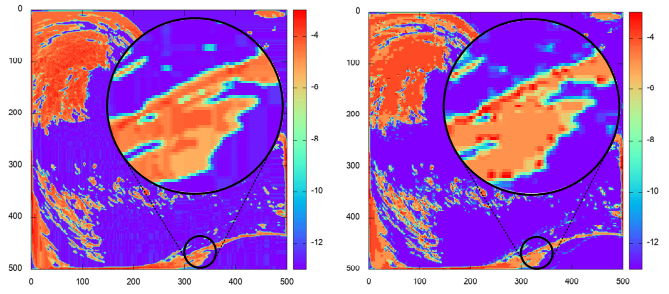


(c) SZ (PSNR=24.5,SSIM=0.4218) (d) ZFP (PSNR=21.3,SSIM=0.3762)

Fig. 7. Data Distortion of NYX(dark_matter:slice 100) with CR=58:1



(a) original raw data (b) our sol. (PSNR=51, SSIM=0.9966)

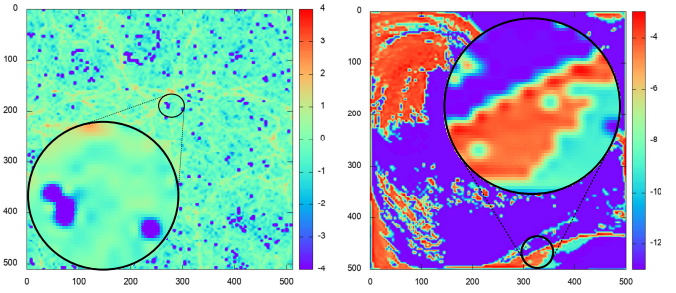


(c) SZ (PSNR=29.9, SSIM=0.6573) (d) ZFP (PSNR=22.5, SSIM=0.8893)

Fig. 8. Data Distortion of Hurricane(CLOUDf:slice 100) with CR=66:1

in both the background and local region, which leads to very high overall visual quality (SSIM 0.9966).

In addition to error-bounded lossy compressors, we also present the visual quality of the down-sampling+interpolation method (widely used in the visualization community) in Fig. 9 for comparison. During the compression, one data point would be sampled uniformly every 4 points for each field in each dimension, leading to the compression ratios of 64:1. During the decompression, tricubic interpolation [31] is used to reconstruct the missing points. We can clearly observe that our solution exhibits much better visual quality (see Fig. 7 (b) vs. Fig. 9 (a); Fig. 8 (b) vs. Fig. 9 (b)), in that the downsampling+interpolation method over-smoothened the



(a) dark_matter_density (CR=64:1, PSNR=18.1,SSIM=0.4345) (b) CLOUDf (CR=64:1,PSNR=17.7, SSIM=0.7681)

Fig. 9. Data distortion of uniform down-sampling and tricubic interpolation of NYX dataset with similar compression ratios

regions with diverse values (Fig. 9 (a)) and over-amplified some boundary data points (Fig. 9 (b)).

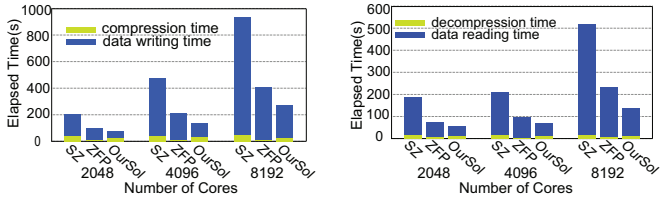
Pearson correlation coefficient has been used to assess the correlation between the original dataset and decompressed dataset [18] by the community. We have validated that the Pearson correlation coefficients under our solution are higher than those of SZ and ZFP in a large majority of cases across all the four applications. We here exemplify the correlation results in Table III using only NYX data due to the space limitation. We run three compressors and tune their compression ratios to be 60:1∼70:1. The reason we cannot fix the compression ratio across fields is that ZFP exhibits piece-wise compression ratios with various error bounds. We did not use fixed-rate mode for ZFP because it is always worse than its fixed-absolute-error mode in our experiments with respect to the rate-distortion.

TABLE III
PEARSON CORRELATION COEFFICIENTS OF 6 FIELDS IN NYX

| fields | our solution | SZ | ZFP | CR |
|---|---|---|---|---|
| dark matter density | 0.959274616 | 0.939890773 | 0.763353467 | 58:1 |
| baryon density | 0.999537914 | 0.986076774 | 0.999529095 | 66:1 |
| temperature | 0.992798653 | 0.870542883 | 0.995984391 | 66:1 |
| velocity_x | 0.999961851 | 0.996793357 | 0.999871972 | 70:1 |
| velocity_y | 0.999944697 | 0.997787266 | 0.999912867 | 70:1 |
| velocity_z | 0.999867946 | 0.992133964 | 0.999826937 | 63:1 |

We evaluate the overall data dumping/loading performance on the NYX simulation using different lossy compressors with the same level of data distortion. Specifically, we set the PSNR for its fields to 60 except for dark matter density (PSNR=30) and baryon density (PSNR=40), because such a setting already reaches a high visual quality (as exemplified in Fig. 1(b) and Fig. 7(b)). The evaluation is weak-scaling. Each rank processes 3 GB data and the total data size increases linearly with the number of cores. We assess the performance by running different scales (2,048 cores $\sim$ 8,192 cores), and each MPI rank needs to process a total of 3 GB data during the execution. The total data size is up to 24 TB when using 8,192 cores, which may take over six hours to dump. We present the breakdown of the data dumping performance (sum of compression time and data writing time) and data loading performance (sum of data reading time and decompression time) in Fig. 10. Since the parallel performance is dominated by the data reading/writing time (to be shown later) and VAPOR, FPZIP and ISABELA have low compression ratios, they exhibit much higher overall data dumping/loading time than the other compressors. Accordingly, we do not present

their results in the figure for the purpose of clearly observing the performance difference among the three best solutions (our solution, SZ and ZFP).




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

Fig. 10. Performance evaluation using NYX

It is observed that the overall data dumping time under our solution takes only 29% and 67% of the time cost by SZ and ZFP when adopting 8,192 cores, which correspond to 3.45X and 1.5X performance, respectively. The key reason is that our compressor leads to significantly higher compression ratios than the other two compressors when the PSNR is in the range of [30,60], as shown in Fig. 6(c). When running the simulation with 8,192 cores, our solution can also obtain 1.72X higher data loading performance (42% lower time cost) than the second best solution (ZFP) does. It is a little higher in comparison with data dumping performance (1.5X), because of the higher decompression rate than the compression rate.

## IX. Conclusion and Future Work

In this paper, we propose an efficient error-bounded lossy compressor, which adaptively selects the best-fit prediction approach from between our improved Lorenzo predictor and an optimized linear-regression based predictor, in terms of the data features in different regions of the dataset. We evaluate our solution using 100+ fields across 4 well-known HPC simulations, by comparing to six existing state-of-the-art lossy compressors (SZ, ZFP, TTHRESH, VAPOR, FPZIP and ISABELA). Experiments demonstrate that our new compressor can achieve up to 8x compression ratio as that of other compressors with the same PSNR. By running the three best lossy compressors (SZ, ZFP and our solution) using up to 8,192 cores, our solution has 1.5x dumping performance and 1.72x loading performance over the second best compressor because of the significant reduction of data size. In the future, we plan to further improve the compression quality in the cases with medium compression ratios (bit-rate).

## References


[1] I. Foster, et al. "Computing Just What You Need: Online Data Analysis and Reduction at Extreme Scales," in *European Conference on Parallel Processing (Euro-Par 2017)*, Jaipur, 2017, pages 3-19.

[2] 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 IPDPS2017*, Orlando, Florida, USA, May 29-June 2, pp. 1129–1139, 2017.

[3] P. Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Trans. on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[4] p. Lindstrom and M. Isenburg, "Fast and Efficient Compression of Floating-Point Data," *IEEE Trans. on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[5] T. Lu, Q. Liu, et al. "Understanding and Modeling Lossy Compression Schemes on HPC Scientific Data," to appear in *32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2018)*, 2018.

[6] L. Ibarria, P. Linderstrom, J. Rossignac, and A. Szymczak, "Outofcore compression and decompression of large ndimensional scalar fields," in *Computer Graphs Forums*, vol. 22, no. 3, pages 343–348, 2003.

[7] Lustre file system. [Online]. Available at: http://lustre.org/.

[8] I/O Performance Evaluation by HPC at Uni.Lu. [Online]. Available at: https://hpc.uni.lu/systems/storage.html.

[9] Gzip compression. [Online]. Available at http://www.gzip.org.

[10] M. Burtscher and P. Ratanaworabhan, "High Throughput Compression of Double-Precision Floating-Point Data," In *Data Compression Conference (DCC'07)*, pp. 293–302, 2007.

[11] BlosC compressor. [Online]. Available at http://blosc.org

[12] Ainsworth, M., Klasky, S., "Compression using lossless decimation: analysis and application," SIAM Journal on Scientific Computing, 39(4), B732-B757, 2017.

[13] P. Lindstrom, "Error Distributions of Lossy Floating-Point Compressors," in *Joint Statistical Meetings*, 2017.

[14] G.K. Wallace, "The JPEG still picture compression standard," in *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pages xviii-xxxiv, 1992.

[15] D. Taubman and M. Marcellin, "JPEG2000 image compression fundamentals, standards and practice: image compression fundamentals, standards and practice," in *Springer Science & Business Media*, vol. 642, 2012.

[16] A.H. Baker, et al. "Evaluating lossy data compression on climate simulation data within a large ensemble," in *Geoscientific Model Development*, vol. 9, pages 4381-4403, 2016.

[17] D. Tao, S. Di, Z. Chen, F. Cappello, "In-Depth Exploration of Single-Snapshot Lossy Compression Techniques for N-Body Simulations," in *IEEE International Conference on Big Data (BigData17)*, 2017.

[18] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, S. A. Mickelson, J. Edwards, M. Vertenstein, A. Wegener, "A methodology for evaluating the impact of data compression on climate simulation data", HPDC'14, pages 203-214, 2014.

[19] S. Di and F. Cappello, "Fast Error-bounded Lossy HPC Data Compression with SZ," In *Proceedings of IEEE 30th International Parallel and Distributed Processing Symposium (IPDPS16)*, pp. 730–739, 2016.

[20] R. Ballester-Ripoll, P. Lindstrom R. Pajarola, "TTHRESH: Tensor Compression for Multidimensional Visual Data," https://arxiv.org/abs/1806.05952, 2018.

[21] Clyne, J., Mininni, P., Norton, A., and Rast, M., "Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation," New Journal of Physics 9 301., 2007.

[22] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N.F. Samatova, "Compressing the Incompressible with ISABELA: In-situ Reduction of Spatio-Temporal Data," in proceedings of European Conference on Parallel and Distributed Computing (Euro-Par), Bordeaux, France, Aug. 2011

[23] B. Welch, "POSIX IO extensions for HPC," Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST05), 2005.

[24] R. Thakur, W. Gropp, E. Lusk, "On implementing MPI-IO portably and with high performance," Proceedings of the sixth workshop on I/O in parallel and distributed systems, pages 23–32, 1999.

[25] A. Turner, "Parallel I/O Performance," [Online]. Available at: https://www.archer.ac.uk/training/virtual/2017-02-08-Parallel-IO/2017_02_ParallelIO_ARCHERWebinar.pdf.

[26] CESM-ATM climate model. [Online]. Available at: http://www.cesm.ucar.edu/models/.

[27] Hurrican ISABEL simulation data. [Online]. Available at http://vis.computer.org/vis2004contest/data.html

[28] NYX simulation. [Online]. Available at: https://amrex-astro.github.io/Nyx/.

[29] SCALE-LETKF weather model. [Online]. Available at: https://github.com/gylien/scale-letkf.

[30] A.H. Baker, H. Xu, D. Hammerling, M. Dorit, S. Li, J.P. Clyne, "Toward a Multi-method Approach: Lossy Data Compression for Climate Simulation Data," International Conference on High Performance Computing, pages 30-42, 2017.

[31] F. Lekien, J. Marsden, "Tricubic interpolation in three dimensions," International Journal for Numerical Methods in Engineering 63.3 (2005): 455-471.