MSz: An Efficient Parallel Algorithm for Correcting Morse-Smale Segmentations in Error-Bounded Lossy Compressors

Yuxiao Li, Xin Liang, Bei Wang, Yongfeng Qiu, Lin Yan, and Hanqi Guo

Abstract—This research explores a novel paradigm for preserving topological segmentations in existing error-bounded lossy compressors. Today's lossy compressors rarely consider preserving topologies such as Morse-Smale complexes, and the discrepancies in topology between original and decompressed datasets could potentially result in erroneous interpretations or even incorrect scientific conclusions. In this paper, we focus on preserving Morse-Smale segmentations in 2D/3D piecewise linear scalar fields, targeting the precise reconstruction of minimum/maximum labels induced by the integral line of each vertex. The key is to derive a series of edits during compression time. These edits are applied to the decompressed data, leading to an accurate reconstruction of segmentations while keeping the error within the prescribed error bound. To this end, we develop a workflow to fix extrema and integral lines alternatively until convergence within finite iterations. We accelerate each workflow component with shared-memory/GPU parallelism to make the performance practical for coupling with compressors. We demonstrate use cases with fluid dynamics, ocean, and cosmology application datasets with a significant acceleration with an NVIDIA A100 GPU.

Index Terms—Lossy compression, feature-preserving compression, Morse-Smale segmentations, shared-memory parallelism.

1 Introduction

The rapid advancement of high-performance computing (HPC) technologies has enabled the generation of vast quantities of scientific data, posing significant challenges to scientists regarding data storage, transmission, and visualization. As such, scientists have recently started to explore compression, especially error-bounded lossy compression, to address the data challenges by ensuring efficient data management and utilization while limiting the amount of distortion introduced by compression. The adoption of error-bounded lossy compression techniques, exemplified by algorithms like SZ [24, 26, 37, 41, 42], ZFP [27], and FPZIP [28], offers a pragmatic solution for reducing scientific data. These methodologies facilitate substantial data reduction while maintaining a predefined accuracy threshold, thus ensuring the utility of compressed datasets for immediate analysis and scientific exploration.

However, an emerging issue with lossy compressors is the inability to preserve topological features such as Morse-Smale (MS) complexes and merge trees in decompressed data, even with bounded error [25, 39]. Topological inconsistencies between the original and decompressed data may result in misinterpretation of the data, even leading to erroneous scientific findings. For example, in molecular dynamics, scientists use MS complexes to segment electron density fields, identifying regions with chemical bonds or weak interactions [33]. Inaccuracies in the segmentations (as exemplified in Figure 1) could lead to wrong interpretations of bonding characteristics and electrostatic interactions. In combustion research, MS complexes help identify reaction, mixing, and quenching zones [6, 7]. Erroneous segmentations could lead to flame structure misinterpretations and subsequently affect the analysis of combustion efficiency and reaction mechanisms. In the visualization of Atmospheric Rivers (ARs) [20], elongated bands of water vapor transport that originate from the tropics to North America and cause flooding, scientists characterize the skeleton of ARs with MS complexes and segmentations to understand the formation and development of ARs better. In case studies later in this paper, we will further exemplify how off-the-shelf lossy compressors distort such essential features.

- *Yuxiao Li, Yongfeng Qiu, and Hanqi Guo are with The Ohio State University. E-mail: {li.14025\|qiu.722\|guo.2154}@osu.edu.*
- *Xin Liang is with the University of Kentucky. E-mail: xliang@uky.edu.*
- *Bei Wang is with the University of Utah. E-mail: beiwang@sci.utah.edu.*
- *Lin Yan is with the Iowa State University. E-mail: linyan@iastate.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

Even the tiniest variations, however small the compression error bound is, could induce significant alternations in the MS segmentation, thereby impacting scientists' understanding of the data. In Figure 1, we compress a molecular dynamics dataset using SZ3 and ZFP with relative error bounds ranging from 10^{-5} to 10^{-2}, which all introduced discrepancies of topological segmentations induced by MS complexes up to 100%; the specific metric measures the discrepancies of segmentations by the percentage of points with a wrong segmentation ID, as explained later in this paper.

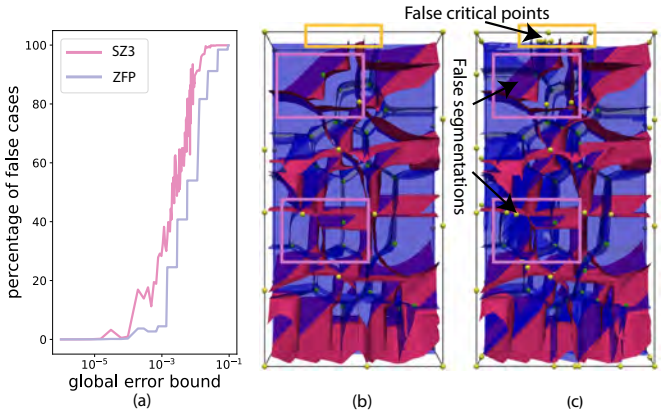

Fig. 1: Impacts of lossy compression (SZ3 and ZFP) on MS segmentations of the Adenine Thymine (AT) dataset: (a) percentages of vertices with wrong segmentation labels w.r.t different error bound; (b) and (c): critical points and separatrices in the original data and SZ3's decompressed data with a relative error bound of 10^{-3}.

This research focuses on preserving Morse-Smale (MS) segmentations [32] in lossy compression workflows; such segmentations provide a preview of the Morse-Smale complex. MS segmentation consists of two significant components of the full MS complex: the extrema designated for each MS complex region and the boundaries separating adjacent MS segmentations. Due to the high computational complexity of the MS complex, MS segmentations are used in diverse applications because they offer a cost-effective way to visualize topological segmentation for visualization and analyses.

To tackle inconsistencies of topological segmentations, we introduce a novel edit-based paradigm for preserving MS segmentations in error-bounded lossy decompressed data in 2D/3D piecewise linear (PL)

scalar fields. Unlike existing efforts that preserve other topological features by tailoring a specific compressor [23, 25, 38, 39], our method directly edits compression outputs, making it theoretically applicable to any error-bounded lossy compressors that applications already trust and engage. These edits also allow us to adjust and verify outputs recurrently without repeatedly calling compressors with different input parameters. Specifically, we identify a subset of data that requires edits, such that the MS segmentation of the decompressed data remains identical to that of the original dataset, while also guaranteeing the global error bound. Since each MS segmentation region consists of a pair of extrema and the points included in the region, our method mainly focuses on preserving these two features by alternatively fixing extrema and integral lines until convergence within finite iterations of the loop: (1) computing the MS segmentation of the current data, (2) identifying the false data points (false critical or regular points), and (3) fixing the false data points until one can reconstruct the exact MS segmentation with the edits during decompression.

We address the performance and scalability of our method with shared-memory/GPU parallelism. While the efficient parallel computation of MS segmentation/complex is already a known challenge [32], the full realization of our vision requires careful parallelization design. Specifically, a key challenge is handling read-write and write-write conflicts when multiple threads attempt to modify the data concurrently. For example, if a vertex is adjacent to two critical points, during the iteration to fix multiple critical points, two parallel threads could simultaneously change the vertex's value; we use atomic intrinsics to avoid such conflicts while maintaining high scalability. As part of our implementation, we also introduce a GPU version of Maack et al. [32] to boost the overall performance of our workflow. We perform a comprehensive performance analysis with CPUs and GPUs on a compute node of Lawrence Berkeley's Perlmutter supercomputer. In summary, the contributions of this paper are multifold:

- A novel edit-based paradigm for preserving MS segmentation within error-bounded lossy decompressed data in 2D/3D piecewise linear scalar fields, theoretically applicable to any existing error-bounded lossy compressors;
- Efficient shared-memory/GPU parallelism that significantly accelerates individual components of our algorithm while addressing read-write/write-write conflicts;
- Comprehensive evaluation of our method across various datasets, two off-the-shelf base compressors (SZ3 and ZFP), and parallel performance on both CPUs and GPUs.

2 Related Work

We review related work on error-bounded lossy compression, topology-preserving compression, and Morse-Smale complexes.

2.1 Error-Bounded Lossy Compression

Error-bounded lossy compressors can significantly reduce data by allowing bounded pointwise error between the decompressed and original data within a user-specified threshold. Note that the number of points/vertices remains constant from the original to the compressed data. However, few existing lossy compressors consider topology features in the decompressed data, which will impact disciplines that require post hoc analysis of the topological structure, thereby introducing deviations in the analytical results.

Error-bounded lossy compressors can be divided into prediction- and transformation-based methods. Examples of prediction-based compressors include the SZ series [8, 9, 24, 26, 37, 41, 42]. For example, SZ1.4 [37] uses the Lorenzo predictor combined with linear-scaling quantization to convert prediction residuals into integers, which are then encoded with customized Huffman coding and lossless compressors like ZSTD [2] and GZIP [1]. QoZ [31] is an optimization of SZ3, focusing on improving the quality of decompressed data under dynamic metrics with parameter auto-tuning. It automatically adjusts the compression based on user-specified quality objectives. FPZIP [28] allows a specified number of bit planes to be ignored, making the data distortion controllable on demand. AE-SZ [30] and SRNN-SZ [29] are examples of prediction-based compressors that use neural networks.

Transformation-based lossy compressors first transform data into an alternative representation, such as wavelet transformation and tensor decomposition, then compress data in the transformed domain. For example, ZFP [27] uses a custom orthogonal block transform to decorrelate data within blocks, transforming original data into sparsely distributed coefficients. These coefficients are then encoded for efficient compression, leveraging the reduced complexity of the transformed data to enhance compression efficiency. TTHRESH [3] is a transform-based lossy compressor that uses bit-plane, run-length, and arithmetic coding to compress the transform coefficients of the higher-order singular value decomposition. SPERR [22] is another transform-based compressor that uses the CDF9/7 discrete wavelet transform.

Metrics commonly used to evaluate the quality of lossy compression include mean square root error (RMSE), peak noise-to-signal ratio (PSNR), and structural similarity (SSIM); see Di et al. [10] for a comprehensive review of this topic. We further describe metrics for evaluating MS segmentations later in this paper.

2.2 Topology Preservation in Lossy Compression

Topology preservation is an emerging topic in the context of error-bounded lossy compression; to our knowledge, our work is the first attempt toward preserving MS complexes by maintaining the consistency of topological segmentations. For scalar fields, previous studies primarily focus on contour/merge tree preservation. Yan et al. [39] introduced TopoSZ to enhance the SZ 1.4 compression algorithm by integrating topological constraints informed by segmentations induced by contour trees. Soler et al. [35] developed a topology-controlled compression scheme to adaptively quantize data in individual topological features to preserve the persistence diagram subject to a persistence simplification threshold. For vector fields, researchers have attempted to retain critical points, yet the preservation of topological segmentations is studied empirically. For example, Liang et al. [23, 25] presented a methodology for preserving critical points in piecewise linear and bilinear vector fields.

2.3 Morse-Smale Complexes and Segmentations

We summarize related work in MS complexes and segmentations and leave a detailed review of key definitions in the next section. MS complexes are a well-studied topological descriptor researched by the topological data analysis (TDA) and visualization communities. Constituents of MS complexes include critical points (maxima, minima, and saddles) and separatrices that connect saddles and extrema. The separatrices also partition the input manifold into regions with monotonous subregions, often referred to as MS segmentations.

In general, two flavors exist for MS complex computation: piecewise linear (PL) Morse theory [4, 12] and discrete Morse theory [14]. We refer readers to Lewiner et al. [21] for a comprehensive review and comparison between the two approaches. Our work primarily relies on the PL-based MS segmentation computation, and we formally review key assumptions and concepts of PL Morse theory in the next section.

With the PL Morse theory, one can further divide algorithms into boundary- and region-growing-based approaches. For boundary-based algorithms, Edelsbrunner et al. [12] first introduced the MS complex for PL 2-manifolds and 3-manifolds [11]. Gyulassy et al. [18] introduced the region-growing algorithm for analyzing MS complexes, which is scalable for 3D or higher dimensions. Banchoff et al. [4] explored the critical points on PL 2-manifolds by analyzing the paths of the steepest ascent and descent.

With Forman's discrete Morse theory, Fugacci et al. [15] proposed an efficient algorithm for large and high-dimensional simplicial complexes. Gyulassy et al. [17] introduced an algorithm that computes MS complexes across different data scales and dimensions, which was later extended to a parallel computing framework [19].

In terms of parallel computation of MS complexes and segmentations, Beucher et al. [5] utilized watershed transformations to analyze and construct MS complexes in the context of image processing and data analysis, which was optimized by Gabrielyan et al. [16] by leveraging GPUs. Yeghiazaryan et al. [40] combined path simplification with watershed transformations for efficiency.

3 Background: Morse-Smale Segmentation in Piecewise Linear Scalar Fields

We review key concepts in piecewise linear MS segmentations (PLMSS) [32]. Formally, for a piecewise linear function (represented with a triangular/tetrahedral mesh) with a distinct value f_i for each vertex i, PLMSS assigns a pair of minimum and maximum labels $\langle m_i, M_i \rangle$ by tracing integral lines along the steepest ascending/descending directions along mesh edges.

Critical points in PL scalar fields reside on the vertices of the underlying triangulation, and they are extracted and classified based on the function values of neighboring vertices. The computation of PLMSS concerns only minimum and maximum. For example, in Figure 2(a), vertices with scalar values 0 and 1 are minimum because their values are the lowest among their neighbors; likewise, vertices with 16 and 20 are maximum.

Integral lines. In PL scalar fields, integral lines of gradient vector fields are constructed by monotonic paths consisting of edges in the triangulation. For example, in Figure 2(b), $5 \to 14 \to 15 \to 20$ is an integral line that extends toward the highest/lowest adjacent vertices until a maximum/minimum is met. We further distinguish backward and forward integral lines based on the ascending/descending direction an integral line is tracing toward.

Ascending/descending segmentations. Once integral curves are traced, one can find the minimum/maximum that a vertex i flows to and further define MS segmentation in PL scalar fields. Starting from i, we denote its converging critical points as the maximum label M_i and minimum label m_i following the ascending and descending integral lines. For example, in Figure 2, nodes in b and c, respectively, are colored by maximum and minimum labels; nodes in d are colored by both maximum and minimum labels.

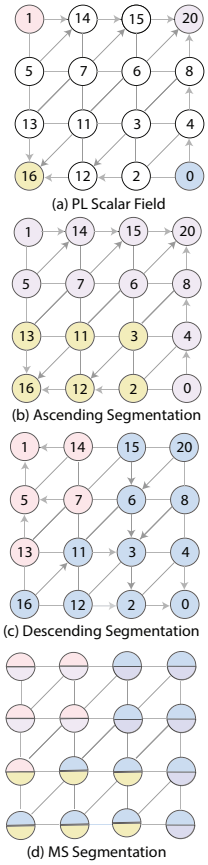

Fig. 2: Constituents of MS segmentations.

Compared with MS complexes, PLMSS is suitable for applications that do not need to compute full MS complexes. The most notable difference is that PLMSS does not concern saddle points, leading to two limitations as reviewed by Maack et al. [32]: (1) PLMSS cannot capture saddle-saddle separatrices, (2) downstream tasks (e.g., persistence simplification) that rely on full MS complexes do not apply to PLMSS. That said, PLMSS offers a fast and practical tool to understand scalar fields.

4 Problem Statement

We formulate the preservation of MS segmentations in 2D and 3D piecewise linear scalar fields. The inputs of our algorithm include both original data f and decompressed data $\hat{f}$ (with exactly the same number of vertices), assuming both data versions are available at the compression time. We assume all scalar field data are Morse; that is, for any two vertices i and j, we have $f_i \neq f_j$; otherwise, we use simulation of simplicity (SoS) [13] to handle non-Morse regions for real-world data. The outputs of our algorithm are a series of edits $\{\delta_i\}$; with the edits, one can derive the final edited value at vertex i as $g_i = \hat{f}_i + \delta_i$. The edited value shall satisfy the following constraints:

Preservation of the global error bound. We must guarantee that the final edited value is subject to the user-prescribed absolute error bound ξ, that is, $|f_i - g_i| \leq \xi$.

Preservation of MS segmentations. Let M and m, respectively, denote the maximum and minimum labels in MS segmentations of the original data, and let $\hat{M}$ and $\hat{m}$ denote the counterparts in the decompressed data. This research aims to precisely align $\hat{M}$ with M and $\hat{m}$ with m. That is, for any vertex v_i, we have $M_i = \hat{M}_i$ and $m_i = \hat{m}_i$.

Preserving PLMSS implies that all extrema are preserved without any false positives or negatives. Under the premise that the location and type of critical points in the decompressed data are identical to those in the original data, to ensure that $\hat{M}$ is precisely equal to M, it is necessary to align the label of each regular point i in the decompressed data with its corresponding label in the original data.

5 Methodology

This section describes the theoretical workflow of computing edits for preserving MS segmentations, as illustrated in Figure 3. The derivation of vertexwise edits is an iterative process involving two distinct loops: the critical point loops (C-loops, Section 5.1) and the regular point loops (R-loops, Section 5.2). The rationale for alternating the two loops is that (1) the correctness of critical points is necessary for fixing regular points, and (2) fixing regular points may introduce new false critical points to be fixed further which are discussed in Section 5.3.

5.1 Critical Point Loops (C-Loops)

We fix four types of false critical points: false positive maxima (FPmax), false positive minima (FPmin), false negative maxima (FNmax), and false negative minima (FNmin). A sublevel loop handles each false type; for example, the FPmax loop executes multiple times until no FPmax exists, followed by the FPmin loop. A C-loop sequentially executes the four subloops in each iteration and exits when no false critical point remains.

Readers may skip the mathematical reasoning below, but the key to (sub)loop convergence is incurring changes that only decrease (or keep) scalar values at each iteration. That is, denoting $g_i^{(k)}$ as the edited value at vertex i during the kth iteration ($k \in \mathbb{I}$), we have

$$\hat{f}_i = g_i^{(0)} \geq \cdots \geq g_i^{(k)} \geq g_i^{(k+1)} \geq \cdots > f_i - \xi, \tag{1}$$

where $\hat{f}_i$ is the initial decompressed value, and $g_i^{(k)}$ progressively approaches to the lower bound $f_i - \xi$. As such, one can make the relationship between scalars on neighboring vertices consistent with that of the original data; formally, for arbitrary two vertices i and j, we have

Lemma 1. *One can find a finite number k of iterations such that $g_i^{(k)} < g_j^{(k)}$, if initially $g_i^{(0)} > g_j^{(0)}$ and $f_i < f_j$.*

This holds because $f_i - \xi = \lim_{k\to\infty} g_i^{(k)} < \lim_{k\to\infty} g_j^{(k)} = f_j - \xi$. Backed by Lemma 1, we design decreasing edits to fix four false cases. Each decreasing edit applies to the vertex with a false critical point (FPmax or FNmin) or a neighbor vertex of the false critical point (FPmin or FNmax), as exemplified below.

5.1.1 False Positive Maximum

Definition 1 (FPmax)**.** *A maximum g_i is false positive if $g_i > g_j$ for all neighboring vertices of i, but one can find at least one neighbor j such that $f_i < f_j$.*

We construct a sequence of iterations for all vertices, following Equation (1), to iteratively eliminate all FPmax:

$$g_i^{(k+1)} := \begin{cases} \left(g_i^{(k)} + f_i - \xi\right)/2, & \text{if } g_i^{(k)} \text{ is FPmax} \\ g_i^{(k)}, & \text{otherwise.} \end{cases} \tag{2}$$

Note that $g_i^{(k)}$ monotonically decreases toward the lower bound $f_i - \xi$ as k increases. We speculate three possible outcomes:

- Case I: $g_i^{(k)}$ remains an FPmax, requiring at least another iteration
- Case II: $g_i^{(k)}$ becomes non-maximum without introducing any new FPmax; for the moment, $g_i^{(k)}$ is fixed;
- Case III: $g_i^{(k)}$ becomes non-maximum but introduces at least one new FPmax $g_j^{(k+1)}$ at a neighboring vertex j;

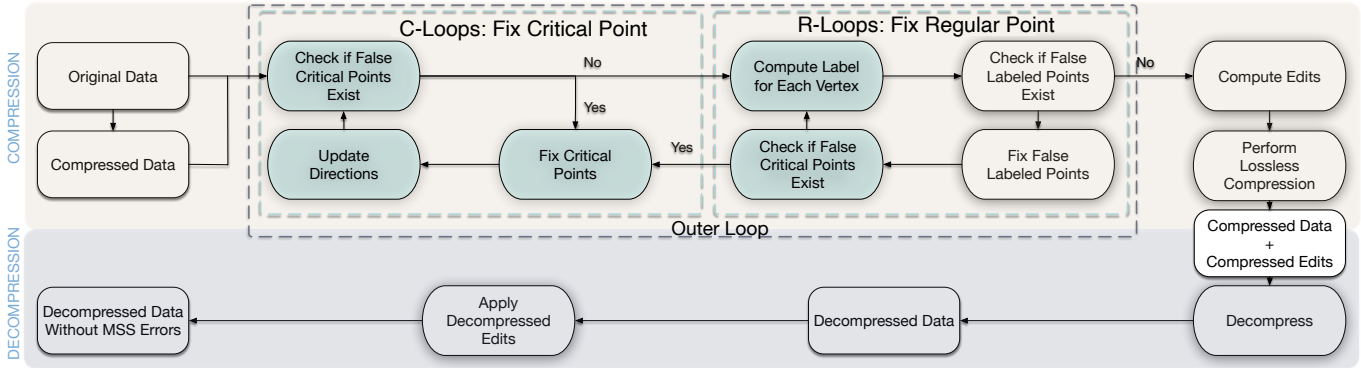

Fig. 3: Compression (top) and decompression (bottom) workflows. Our algorithm derives vertexwise edits with two distinct loops: (1) C-loops that iteratively fix all false critical points and (2) R-loops that fix all false labeled regular points. C- and R-loops execute alternatively until there is no false critical/regular point. The edits, which are losslessly stored, are used to correct MS segmentations in the decompression stage.

Cases I and II are trivial. In Case III, the newly introduced FPmax at j will be eventually fixed in later iterations without making i an FPmax again. Specifically, per Definition 1, one can find a neighboring vertex j that is ascending in the original data ($f_j < f_{j'}$); no matter if j' is i or not, with additional iterations with whichever cases, per Lemma 1, neither i nor j will become FPmax with additional finite iterations.

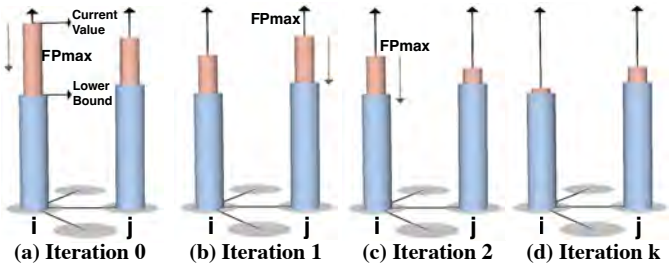

Fig. 4: Fixing an FPmax at vertex i. The height of the blue cylinder above each vertex represents its lower bound ($f-\xi$), and the height of the pink cylinder represents its current value.

We use Figure 4 to help further understand Case III. Initially, vertex i is an FPmax, and the decompressed value is greater than all its neighbors (shown in a). However, in the original data, we have at least one neighbor j such that $f_j > f_i$. To turn i back into a regular point, in the first iteration, we decrease the g_i such that $g_i^{(1)} < g_j^{(1)}$. However, j could become a new FPmax due to the decrement of g_i (shown in b). In the next iteration (c), as we fix the new FPmax j by decreasing g_j, vertex i could turn into FPmax again. The false positives may recur up to a finite number of iterations but will guarantee to vanish because we incrementally drive g_i and g_j closer to their lower bounds. Note that $f_i < f_j$ and the lower bound $f_i - \xi < f_j - \xi$, with a finite number k of iterations, we have $g_i^{(k)} < f_j - \xi < g_j^{(k)}$ (shown in d). At this time, vertex i will no longer become a new false positive maximum.

5.1.2 False Positive Minimum

Definition 2 (FPmin)**.** *A minimum g_i is false positive if $g_i < g_j$ for all neighbors j but one can find at least one neighbor j such that $f_i > f_j$.*

Unlike fixing FPmax, assuming $g_i^{(k)}$ is FPmin, we decrease the value at the *ascending neighbor* j, such that $g_j^{(k)}$ is the maximal among the neighbors of i, and we have

$$g_j^{(k+1)} := \begin{cases} \left(g_j^{(k)} + f_j - \xi\right)/2, & \text{if } g_j^{(k)} \text{ is maximal among } i\text{'s neighbors} \\ g_j^{(k)}, & \text{otherwise.} \end{cases} \tag{3}$$

While one could alternatively fix the FPmin by increasing the value on the same vertex, we impose decreasing edits across all types of false critical points to guarantee convergence across the iterative workflow; otherwise, incompatible strategies may not lead to convergence. Later in this section, we demonstrate how an FPmin is fixed amid a complex process and discuss the convergence.

5.1.3 False Negative Maximum/Minimum

Definition 3 (FNmax/FNmin)**.** *A non-maximum (or non-minimum) g_i is false negative if $f_i > f_j$ (or $f_i < f_j$) for all j in neighbors of i.*

Specifically, for an FNmax g_i, we reduce its ascending neighbor's value:

$$g_j^{(k+1)} := \begin{cases} \left(g_j^{(k)} + f_j - \xi\right)/2, & \text{if } g_j^{(k)} \text{ is maximal among } i\text{'s neighbors} \\ g_j^{(k)}, & \text{otherwise.} \end{cases} \tag{4}$$

For an FNmin g_i, we decrease its own scalar value:

$$g_i^{(k+1)} := \begin{cases} \left(g_i^{(k)} + f_i - \xi\right)/2, & \text{if } g_i^{(k)} \text{ is FNmin} \\ g_i^{(k)}, & \text{otherwise.} \end{cases} \tag{5}$$

Note that both strategies comply with Equation (1) so that the iterations are provably convergent with finite iterations.

5.2 Regular Point Loops (R-Loops)

Once all false critical points are fixed, the next step is to fix all regular points' maximum and minimum labels in the decompressed data. For a falsely labeled regular point, our method involves three steps in each iteration: (1) compute the ascending/descending integral lines in the current edited data $g^{(k)}$, (2) locate the *troublemaker*, which is defined as as the first occurrence of discrepancy along the integral lines (as illustrated in Figure 5(b)), (3) reroute the troublemaker by an edit. Specifically, for a falsely labeled regular point i with a wrong minimum (maximum), let v_t represent the troublemaker's descending (ascending) neighbor, we decrease the value at v_t:

$$g_{v_t}^{(k+1)} := \left(g_{v_t}^{(k)} + f_{v_t} - \xi\right)/2. \tag{6}$$

Note that edits in an R-loop may introduce new false critical points; in this case, we must return to the C-loop to address the new false cases. Similar to C-loops, edits in R-loops always decrease decompressed values such that no new false critical points are introduced after a finite number of iterations.

5.3 Convergence of Alternating C- and R-loops

We alternatively execute C- and R-loops until all false critical/regular points vanish. The workflow converges because all edits progressively

decrease decompressed values towards the lower bound. The convergence is backed by Lemma 1; because the inconsistent ordering of scalar values may cause changes to ascending/descending directions and lead to false cases, the inconsistencies will guarantee to vanish with a finite number of iterations with our decreasing edit strategies.

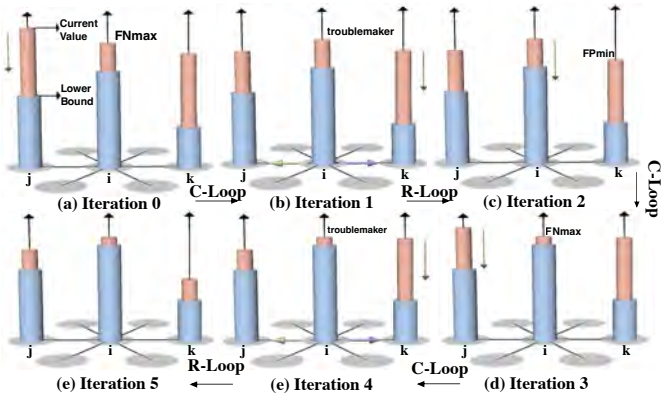

Fig. 5: Fixing an FNmax (and subsequently, a troublemaker, FPmin, FNmax, and another troublemaker) across multiple C- and R-loops. The purple and green arrows in (b) and (d) show the ascending neighbor of vertex i in the original and decompressed data, respectively.

Figure 5 demonstrates a specific case of alternating C- and R-loops to fix an FNmax. We have $f_i > f_j > f_k$ in the original data, but initially $g_j > g_i > g_k$. The FNmax at vertex i is fixed in the first C-loop by decreasing the neighbor j. However, i becomes a troublemaker that causes false regular points. After the troublemaker is fixed by decreasing its neighbor k, k becomes an FPmin, thus triggering another C-loop. The second C-loop fixed the FPmin by decreasing i, which caused a new FNmax case at i. With two additional C- and R-loops for newly introduced false cases, we have no more false cases to address.

6 Parallel Computation and Compression of Edits

This section describes the shared-memory/GPU parallelism that significantly accelerates individual components of our algorithm.

6.1 Parallel C-Loop

We parallelize all subroutines in C-loops, including (1) finding false critical points, (2) fixing false critical points, and (3) updating directions for each vertex, as highlighted in Figure 3.

Finding false critical points checks each vertex in parallel to see if it is a false critical point. We assign each thread with a vertex; if the vertex is false critical, we push the case into a (lock-free) stack for further processing. For both CPU and GPU architectures, we implement the lock-free stack with `atomicAdd` operations with a pre-allocated buffer. For each push, we record the current stack height as h while increasing the height by one; this fetch-and-add operation guarantees to avoid race conditions. We then write the false case (represented by vertex ID and false type) into the hth position of the buffer.

Fixing false critical points derives and applies edits for each false critical point with a thread. Because an edit may change the scalar value of the false critical point or one of its neighbors, multiple threads may edit the same value and cause write-write conflicts. In a conflict, only one edit will be applied when multiple threads preemptively edit the same vertex, but the specific choice made by hardware is random; we call it *preemptive mode*. The preemptive mode would still converge because the missing edits would be applied by a later iteration, albeit of random execution orders. To make executions consistent, we incorporate atomic compare-and-swap (`atomicCAS`) operations in our CPU and GPU implementations. Specifically, the operation compares the incoming edit with the current edit value and arbitrarily applies the most significant edit, making execution orderings deterministic.

Updating directions. We assign each vertex to a thread to update its ascending/descending neighbor for identifying false cases. This step involves a local comparison of scalar values between neighbors.

6.2 Parallel R-Loop

The only component we currently parallelize in the R-loop is the computation of MS segmentation, which dominates the execution time. We implemented path compression [34] used by Maack et al. [32], for the computation of Morse-Smale segmentation. On GPUs, taking ascending segmentation as an example, path compression involves utilizing a (lock-free) list to track regular points that have not yet found their maximum. For these points, the method seeks its largest neighbor, updating each point's value in the list to that of its largest neighbor's largest neighbor. If the value of a point v remains unchanged after the update, meaning the largest neighbor's largest neighbor is itself, it indicates that v has been assigned to a maximum. The iteration concludes once every point has successfully determined its maximum.

6.3 Lossless Compression of Edits

Once all iterations converge and exit with no false critical/regular points, the last step is to store the edits compactly as metadata appended to the lossy compressor's outputs. Each edit δ_i is represented with a key-value pair, the key being the vertex index and the value being the floating-point representation of the edits (See Supplementary Material for a visualization of the spatial distribution of edits). We compress the indices and edit values separately. Regarding the indices, we first sort them in ascending order and compress the differential sequence, because storing the differentials makes it possible to maximize the use of run-length encoding (RLE) and Huffman coding before offloading the edits to a lossless compressor such as ZSTD [2] or GZIP [1].

7 Evaluation of Edited Data

We evaluate our method with datasets from diverse applications, ranging from climate and cosmology to combustion, by measuring the accuracy in MS segmentations based on two state-of-the-art error-bounded lossy compressors, SZ3 and ZFP. Evaluation metrics are as follows; detailed descriptions of datasets and metrics are in Supplementary Material. **Overall Compression Ratio (OCR)** is the compression ratio after the combination of edits, calculated by the combined size of compressed edits and data over the original data size. A higher OCR indicates more effective compression, resulting in a smaller compressed file. **Edit ratio** quantifies the proportion of data points that are edited to fully preserve the MSS in the decompressed data. It is calculated as the number of modified data points divided by the total number of data points in the decompressed data. **Overall bit rate (OBR)** represents the average number of bits required to encode each data point after compression (after the combination of compressed data and edits). It is calculated by dividing the total number of bits used by the total number of data points. Lower bit rates indicate more efficient compression. **Right labeled ratio** is the percentage of points in the data with correct MSS labels, calculated by the number of right labeled points over the number of points in the data. **PSNR distortion** refers to the trade-off between the bitrate and PSNR. It describes how the bitrate affects the PSNR of the decompressed data, where a higher bitrate generally results in a higher PSNR, indicating better quality. This is typically represented as a curve with the x-axis showing the bitrate and the y-axis showing the PSNR of the decompressed data. **MSS distortion** is similar to PSNR distortion, with the y-axis changed to the right labeled ratio. It reflects the trade-off between the bitrate and the degree of preservation of the MSS.

7.1 Features of Interest Preservation

We exemplify features of interest characterized by MS segmentations and how off-the-shelf lossy compressors distort the scientific insights. Examples below are based on SZ3 with a 1% error bound.

Climate. Distortions in **Atmospheric Rivers** (ARs), based on feedback from domain scientists, could significantly impact scientists' understanding of AR formation and development, potentially leading to inaccurate evaluations of ARs' impacts on precipitation and flooding in North America. Scientists characterize AR skeletons by the boundaries of descending segmentations of the Integrated Vapor Transport (IVT)

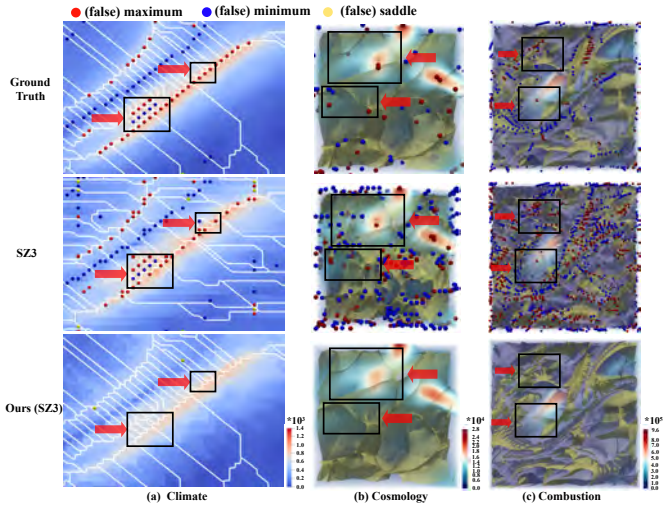

Fig. 6: Feature preservation in SZ3-compressed data for (a) climate, (b) cosmology, and (c) combustion.

field [20]. As shown in Figure 6(a), SZ3 led to distorted segmentation boundaries of the AR and surrounding regions.

Cosmology. Scientists characterize **cosmic walls** in dark matter distributions by ascending segmentation boundaries [36]. Distorted sheet structures of walls, as exemplified in a zoomed-in view of the Nyx data (a 50^3 crop of the entire volume) compressed by SZ3, could lead to incorrect separation between voids and potentially impact the understanding of the cosmic web as shown in Figure 6(b).

Combustion. Scientists use MS segmentations to identify **flame structures** to understand the reaction, mixing, and quenching in the burning dynamics [6,7]. We demonstrate feature distortion in a zoomed-in region (100^3) in Figure 6(c). SZ3 introduced false features and omitted essential separatrices, potentially leading to incorrect identification of burning zones and understanding of fuel consumption areas.

7.2 Fixed-Error-Bound Comparison

We demonstrate that our method can fix MS segmentations from arbitrary outputs with different compressors and error bounds, as shown in Figure 7. We observe that the size of the edits required is roughly proportional to the magnitude of the global error bound. This is reasonable because as the global error bound increases, implying that more errors are introduced into the data, the error rate in the MSS escalates, thereby necessitating a greater number of edits. For more complex datasets, such as CSEM, Nyx, and viscous fingering, the edit ratio is noticeably higher than other datasets. Another observation is that edits for ZFP are generally lower than those for SZ3, while OCR is higher overall than ZFP because ZFP's original compression ratio is lower than SZ3.

7.3 Fixed-Bit-Rate Comparison

We evaluate our method across various datasets to identify the optimal overall bit rate (after the combination of edits) achievable on SZ3 and ZFP, as shown in Figure 8. We run an ensemble of experiments with various error bounds and find relatively the same (overall) bit rate with both SZ3 and ZFP to evaluate the preservation of MSS. With comparable bitrates, the original decompressed data from SZ3/ZFP exhibit varying degrees of MSS distortion across different datasets. Despite a certain bitrate increase due to the introduction of additional edits, as illustrated in the MSS/PSNR distortion plots, our method still ensures the precise preservation of the MSS. From the MSS/PSNR distortion, we can observe that there is a certain balance between OBR and the preservation of MSS. When the original bitrate is lower, indicating a higher data compression ratio, this implies a higher error bound during compression. For a global topological descriptor like the MSS, even minor data alterations can lead to significant distortions in the MSS, potentially necessitating more edits. Conversely, when the original bitrate is higher, the deviation between the decompressed and original data is reduced, lowering the error rate in the MSS and decreasing the necessary edits.

8 Performance Evaluation

We evaluate the scalability and performance of our algorithm with both shared-memory CPU and GPU implementations. We use two datasets, Nyx (512^3) and viscous fingering (128^3), for performance studies because of their larger size and higher topological complexity than others. We use one CPU node (2× AMD EPYC 7763 with 512 GB of DDR4 memory) and one GPU node (single AMD EPYC 7763 CPU with 256 GB of DDR4 DRAM and four NVIDIA A100 GPUs) on the Perlmutter supercomputer at National Energy Research Scientific Computing (NERSC). Our implementation is based on C++, OpenMP, and CUDA; only a single GPU is used for benchmarking.

We measure performance with the end-to-end running time and four mini-benchmarks that evaluate the parallelization of (1) finding false critical points, (2) fixing critical points, (3) updating directions, and (4) MSS computation. For the mini-benchmarks, we artificially reset the data to the initial status the first time the subroutine is called. To ensure robustness and accuracy in our results, each mini-benchmark was executed 1,000 times, and the mean running time was recorded.

8.1 Scalability on CPUs

While we recommend using GPUs for the practical use of our method, we study the scalability of our parallel algorithms on CPUs with different amounts of resources. Figure 9 shows the decreasing end-to-end running time with increasing CPU threads. We also demonstrated the timing breakdown for individual subroutines; for example, updating directions takes up to 80% of the time because all vertices' directions must be updated whenever there is a change in the data.

Figure 10 demonstrates the timings of the four mini-benchmarks with different numbers of threads. Note that the mini-benchmarks do not equally scale to 128 cores because of (1) the number of parallel tasks, (2) load balancing, and (3) locality and contention. First, the number of parallel tasks vary in different components of our algorithm. For example, the number of (false) critical points is usually much smaller than the number of vertices; the former usually leads to insufficient utilization of hardware resources, thereby limiting scalability (Figure 10(b)). Second, in the MS segmentation computation (Figure 10(d)), we follow Maack et al. [32] to terminate an integral line and reuse existing integral lines; this strategy reduces computational cost but leads to imbalanced workloads. Third, our CPU implementation does not consider non-uniform memory access (NUMA) architectures, while our compute node has 8 NUMA domains; meanwhile, concurrent read/write also cause memory contentions even with lock-free data structures. Besides, the underlying operating system may interfere with our parallel execution with many threads. That said, the overall workflow remains scalable as we continue adding threads because the less efficient components (fixing critical points and computing segmentations) only account for up to 23% of the end-to-end time.

8.2 Performance on GPUs

Table 1: Timings of various tasks of our method on the Nyx dataset.

	Serial	OpenMP (Optimal)	CUDA	GPU Accel. cf. OpenMP	GPU Accel. cf. Serial
End-to-end	1192.38s	110s	6.31s	17×	198×
Find false critical points	1.202s	0.26s	0.008s	32×	150×
Update directions	10.46s	0.6s	0.014s	42×	700×
Fix false critical points	0.009s	0.002s	0.00038s	5×	23×
MSS computation	17.87s	2.7s	0.051s	52×	350×

Table 1 shows GPU acceleration of the end-to-end and mini-benchmarks. We achieved a 17× acceleration in the end-to-end performance compared with using all 128 threads on the CPU node; the acceleration is 198× compared with the serial execution. For the mini-benchmarks, compared with serial execution with one single CPU, our GPU implementation achieves up to 700× speed up across all tasks.

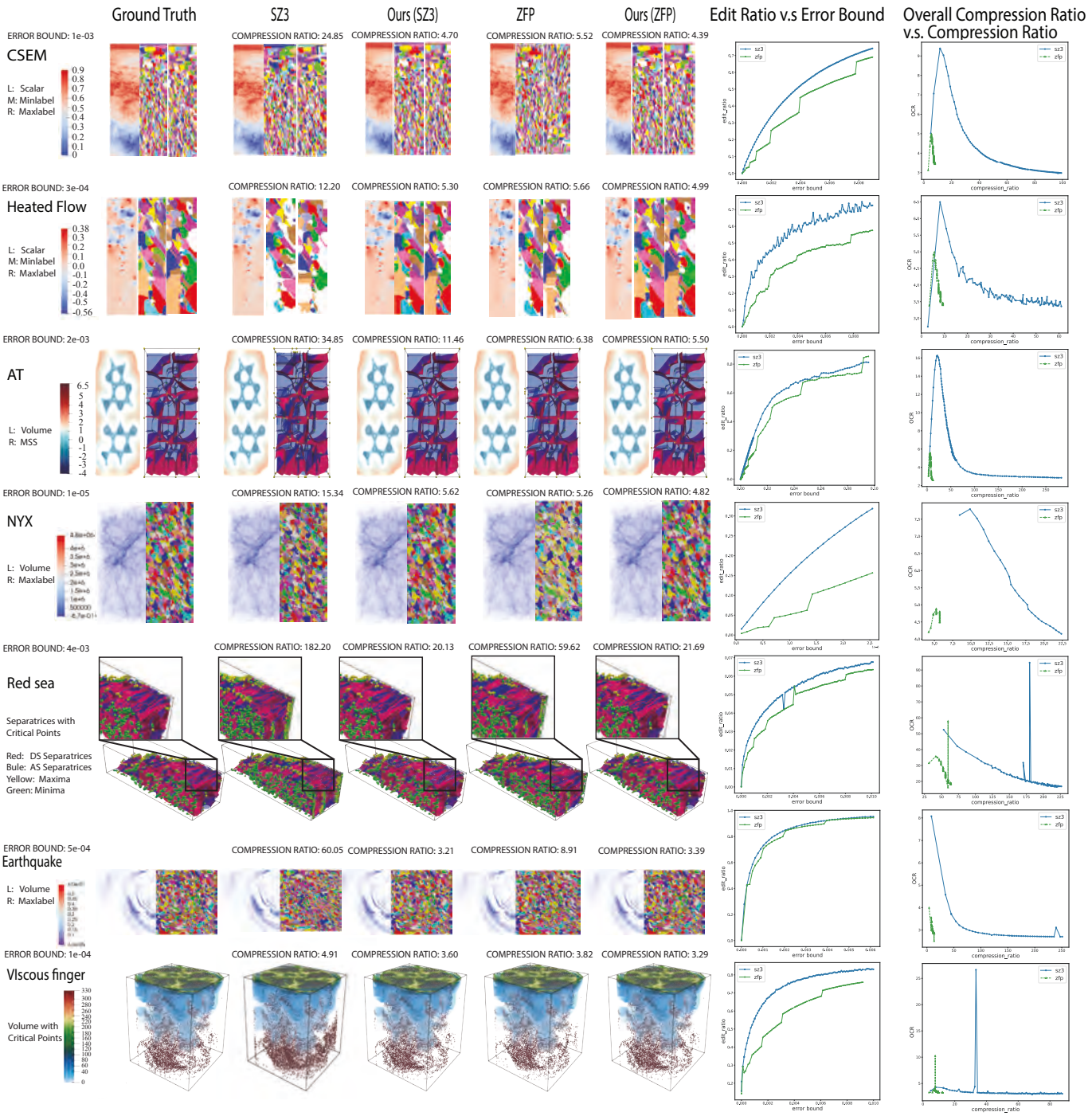

Fig. 7: Fixed-error-bound comparison of lossy compressors on different datasets.

Figure 11 presents an additional performance visualization across multiple C- and R-loops, comparing performance on a CPU (with 32 threads) and a GPU. Across both settings, a notable decrease in time and the number of sub-iterations is observed with increasing iteration. This is because the number of false critical/regular points decreases as the iteration progresses. In each iteration, updating directions for each vertex accounts for a significant portion of the time, approximately 80% for CPU and 50% for GPU. This is followed by the process to find false critical points, around 10% for CPU and 40% for GPU. Notice that measuring the impacts of atomic operations for concurrent read/write on GPU is an open challenge; we leave a formal evaluation of hardware utilization, throughput, and power consumption for the future.

9 Discussion and Limitation

Computation and storage overhead. Preserving MS segmentations comes with a cost, and users may need additional studies to balance their applications' data reduction, performance, and feature preservation needs. Table 2 exemplifies the overhead in timings and compression ratios with multiple datasets. We also compare performance with GZIP and ZSTD, which losslessly compress data and preserve MS segmentations accurately. Our method incurs costs to store the edits as additional metadata attached to the native lossy compression outputs, but still delivers reasonable overall compression ratio with feature preservation. Our overall compression ratios are also better than lossless compressors, which only deliver $\sim 2\times$ ratios for most

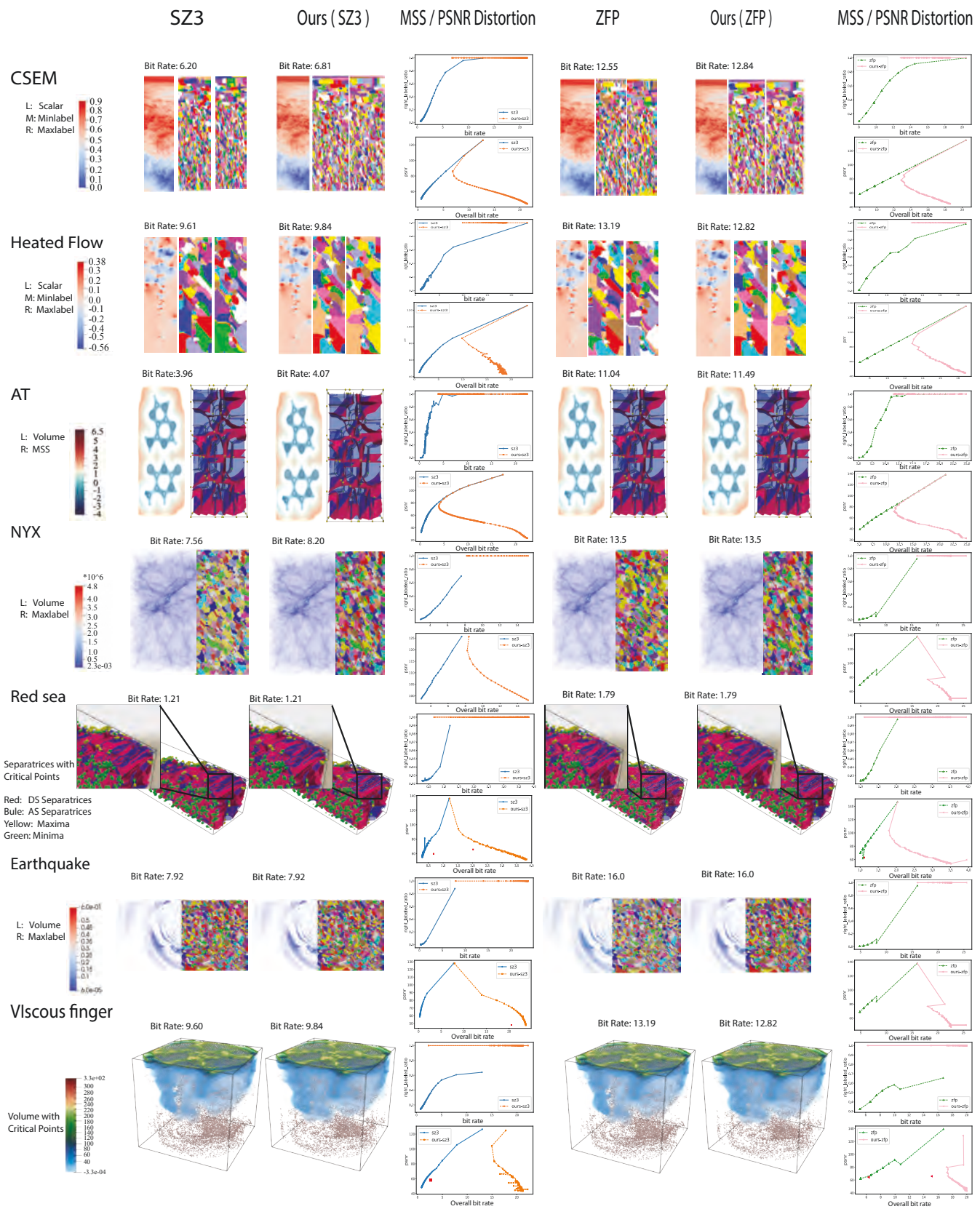

Fig. 8: Fixed-bit-rate comparison of lossy compressors on fixed bit rate on different datasets.

floating-point data [27] (except in cases with empty spaces like the Red Sea data). With an error bound of 10^{-6} based on SZ3, our overall compression ratio for the 512^3 Nyx data is 7.76×, while GZIP is 1.17×. On the computation overhead, our timings are within the same order as

Dataset	Dimensions	Ours-SZ (10^{-6})			Ours-ZFP (10^{-6})			Ours-SZ (5×10^{-6})			Ours-ZFP (5×10^{-6})			GZIP		ZSTD	
		t_{comp}	t_{fix}	OCR	t_{comp}	t_{fix}	OCR	t_{comp}	t_{fix}	OCR	t_{comp}	t_{fix}	OCR	t_{comp}	CR	t_{comp}	CR
AT	$177\times95\times48$	0.19	0.35	3.69	0.12	0.35	3.04	0.17	0.35	5.36	0.13	0.28	5.04	0.24	1.07	1.18	1.17
NYX	512^3	13.8	3.80	7.76	11.8	4.80	4.20	12.98	6.56	7.48	10.75	8.17	4.74	67.8	1.17	9.25	2.03
Heated Flow	150×450	0.08	0.29	2.74	0.03	0.33	3.38	0.08	0.27	3.58	0.04	0.35	3.76	0.64	1.91	1.21	2.00
Red Sea	$500\times500\times50$	0.98	0.51	52.7	0.42	0.51	31.4	0.88	0.55	62.5	0.39	0.46	34.13	4.79	26.4	0.16	31.38
CESM-ATM	1800×3600	0.60	0.61	5.96	0.64	0.42	3.13	0.74	1.41	6.18	0.65	0.98	3.46	3.37	2.08	1.65	2.27

Table 2: Overhead in timings and compression ratios compared with lossy (SZ3 and ZFP) and lossless (GZIP and ZSTD) compressors across different datasets. t_{comp} and t_{fix}, respectively, represents the timings (in seconds) of these compressors and our algorithm.

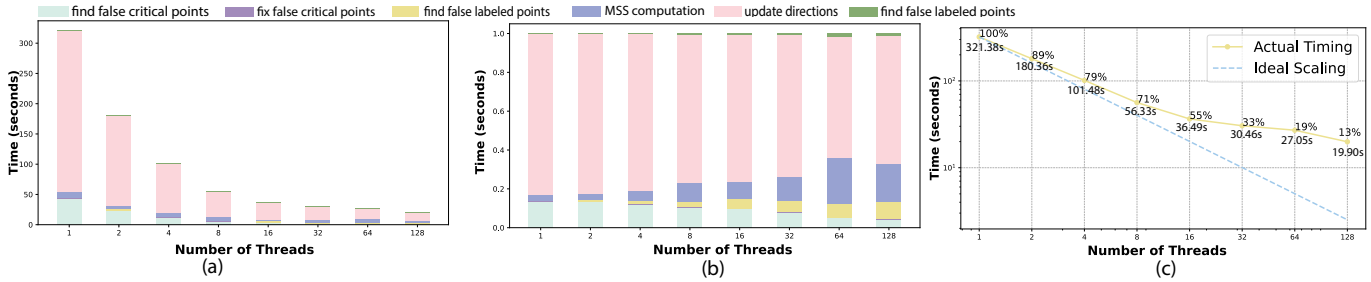

Fig. 9: Strong scalability study for the viscous fingering dataset with OpenMP: (a) performance breakdown and (b) percentage breakdown of computational tasks w.r.t the number of threads; (c) strong scalability study on the end-to-end time; percentages at data points represent the observed efficiency, and numbers show the actual timing.

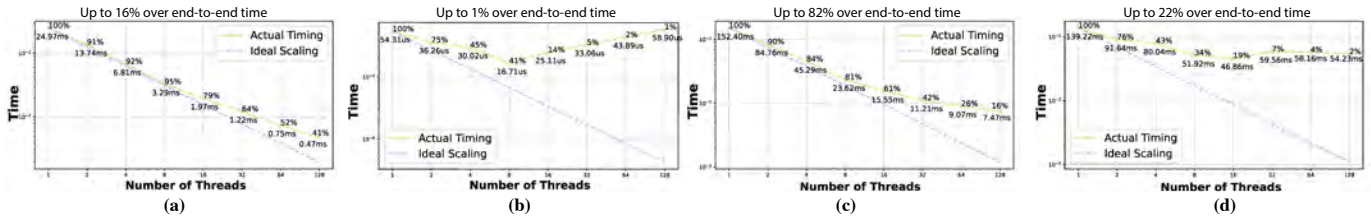

Fig. 10: Strong scalability study of the four tasks w.r.t the number of threads used in our experiment (blue line: actual timing, red line: ideal scaling), percentages at data points represent the observed efficiency, and numbers show the actual timing. (a) find the false critical points; (b) fix false critical points; (c) update directions; (d) MSS computation.

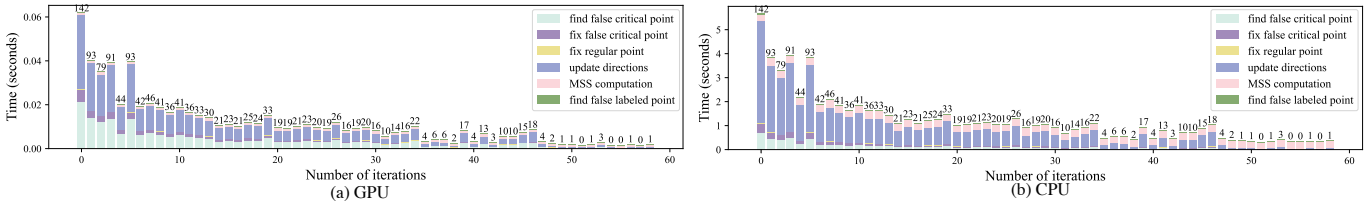

Fig. 11: Timings of outer loop iterations on (a) GPU and (b) CPU with 32 threads; numbers above each bar chart denotes the number of sub-iterations.

the time to compress data with SZ/ZFP. For example, SZ3 takes 13.8 s to compress the Nyx data with 10^{-6} error bound, and our algorithm takes an additional 3.8 s to correct MS segmentations, while GZIP takes 62.8 s. Although the computation and storage overheads vary across different datasets with different dimensionalities and error bounds, we justify the extra time cost considering the feature-preservation capabilities essential for deriving scientific insights.

Preservation of saddles. First, preserving saddles is unnecessary for preserving PLMSS because saddles are not constituents of PLMSS. For example, in the AR application in Figure 6(a), the original and edited data have identical MS segmentations, yet two false saddles exist in the latter. Erroneous saddles do not affect this application because the characterization of ARs relies solely on the segmentation boundaries. Second, as reviewed in Section 3 and by Maack et al. [32], PLMSS cannot capture saddle-saddle pairs and does not support downstream tasks that require the full MS complex; in other words, PLMSS's limitations apply to our method. Third, it is possible to generalize our method in the future to preserve saddles with additional computation and storage costs. Preserving a saddle requires maintaining value ordering among multiple neighboring vertices, making it nontrivial to design an iterative strategy to guarantee convergence.

10 Conclusions and Future Work

We introduce a novel method for preserving PLMSS in lossy decompressed data. Our strategy generates a set of edits at the time of compression. These edits are then applied to the decompressed data, ensuring precise MSS reconstruction while preserving the error bound. Our approach also incorporates a parallel implementation that substantially accelerates our algorithm. We evaluate our methods with datasets from molecular dynamics, climate, combustion, and cosmology applications.

We plan to improve our method in various aspects. First, the compression of edits is achieved through lossless compression, yet this aspect offers substantial room for improvement, potentially enhancing the final compression ratio. Second, our method can be extended to preserve the full MS complex by incorporating the preservation of saddles and saddle-saddle connectors.

Acknowledgments

This research is supported by the U.S. Department of Energy, Office of Advanced Scientific Computing Research (DE-SC0022753) and the National Science Foundation (OAC-2311878, OAC-2313123, OAC-2313124, IIS-1955764, OAC-2330367, OAC-2313122, and OAC-2327266). This research used resources of the National Energy Research Scientific Computing Center (NERSC), a Department of Energy Office of Science User Facility.

References

[1] GZIP. https://www.gzip.org/. Accessed: March 15, 2024. 2, 5

[2] ZSTD. http://www.zstd.net. Accessed: March 15, 2024. 2, 5

[3] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. TTHRESH: Tensor compression for multidimensional visual data. *IEEE Transactions on Visualization and Computer Graphics*, 26(9):2891–2903, 2020. doi: 10.1109/TVCG.2019.2904063 2

[4] T. Banchoff. Critical points and curvature for embedded polyhedra. *Journal of Differential Geometry*, 1(3-4):245–256, 1967. doi: 10.4310/jdg/1214428092 2

[5] S. Beucher and C. Lantuéjoul. Use of watersheds in contour detection. In *Proceedings of Workshop on Image Processing*, pp. 17–21, 1979. 2

[6] P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010. doi: 10.1109/TVCG.2009.69 1, 6

[7] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2011. doi: 10.1109/TVCG.2010.253 1, 6

[8] S. Di and F. Cappello. Fast error-bounded lossy HPC data compression with SZ. In *IPDPS'16: Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium*, pp. 730–739, 2016. doi: 10.1109/IPDPS.2016.11 2

[9] S. Di and F. Cappello. Optimization of error-bounded lossy compression for hard-to-compress HPC data. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):129–143, 2018. doi: 10.1109/TPDS.2017.2749300 2

[10] S. Di, J. Liu, K. Zhao, X. Liang, R. Underwood, Z. Zhang, M. Shah, Y. Huang, J. Huang, X. Yu, C. Ren, H. Guo, G. Wilkins, D. Tao, J. Tian, S. Jin, Z. Jian, D. Wang, M. H. Rahman, B. Zhang, J. C. Calhoun, G. Li, K. Yoshii, K. A. Alharthi, and F. Cappello. A survey on error-bounded lossy compression for scientific datasets, 2024. doi: 10.48550/arXiv.2404.02840 2

[11] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. In *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, pp. 361–370, 2003. doi: 10.1145/777792.777846 2

[12] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse complexes for piecewise linear 2-manifolds. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, pp. 70–79, 2001. doi: 10.1145/378583.378626 2

[13] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990. doi: 10.1145/77635.77639 3

[14] R. Forman. Morse theory for cell complexes. *Advances in Mathematics*, 134(1):90–145, 1998. doi: 10.1006/aima.1997.1650 2

[15] U. Fugacci, F. Iuricich, and L. D. Floriani. Computing discrete Morse complexes from simplicial complexes. *Graphical Models*, 103:101023, 2019. doi: 10.1016/j.gmod.2019.101023 2

[16] Y. Gabrielyan, V. Yeghiazaryan, and I. Voiculescu. Parallel partitioning: Path reducing and union–find based watershed for the GPU. In *ICIP'22: Proceedings of 2022 IEEE International Conference on Image Processing*, pp. 1501–1505, 2022. doi: 10.1109/ICIP46576.2022.9897372 2

[17] A. Gyulassy, P.-T. Bremer, B. Hamann, and V. Pascucci. A practical approach to Morse-Smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1619–1626, 2008. doi: 10.1109/TVCG.2008.110 2

[18] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann. Efficient computation of Morse-Smale complexes for three-dimensional scalar functions. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1440–1447, 2007. doi: 10.1109/TVCG.2007.70552 2

[19] A. Gyulassy, V. Pascucci, T. Peterka, and R. Ross. The parallel computation of Morse-Smale complexes. In *Proceedings of 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pp. 484–495, 2012. doi: 10.1109/IPDPS.2012.52 2

[20] F. Lan, B. Gamelin, L. Yan, J. Wang, B. Wang, and H. Guo. Topological characterization and uncertainty visualization of atmospheric rivers. *Computer Graphics Forum*, 43(3):e15084, 2024. doi: 10.1111/cgf.15084 1, 6

[21] T. Lewiner. Critical sets in discrete Morse theories: Relating forman and piecewise-linear approaches. *Computer Aided Geometric Design*, 30(6):609–621, 2013. doi: 10.1016/j.cagd.2012.03.012 2

[22] S. Li, P. Lindstrom, and J. Clyne. Lossy scientific data compression with SPERR. In *IPDPS'23: Proceedings of 2023 IEEE International Parallel and Distributed Processing Symposium*, pp. 1007–1017, 2023. doi: 10.1109/IPDPS54959.2023.00104 2

[23] X. Liang, S. Di, F. Cappello, M. Raj, C. Liu, K. Ono, Z. Chen, T. Peterka, and H. Guo. Toward feature-preserving vector field compression. *IEEE Transactions on Visualization and Computer Graphics*, 29(12):5434–5450, 2023. doi: 10.1109/TVCG.2022.3214821 2

[24] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *Proceedings of 2018 IEEE International Conference on Big Data*, pp. 438–447, 2018. doi: 10.1109/BigData.2018.8622520 1, 2

[25] X. Liang, H. Guo, S. Di, F. Cappello, M. Raj, C. Liu, K. Ono, Z. Chen, and T. Peterka. Toward feature-preserving 2D and 3D vector field compression. In *Proceedings of 2020 IEEE Pacific Visualization Symposium*, pp. 81–90, 2020. doi: 10.1109/PacificVis48177.2020.6431 1, 2

[26] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao, Z. Chen, and F. Cappello. SZ3: A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE Transactions on Big Data*, 9(2):485–498, 2023. doi: 10.1109/TBDATA.2022.3201176 1, 2

[27] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014. doi: 10.1109/TVCG.2014.2346458 1, 2, 8

[28] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006. doi: 10.1109/TVCG.2006.143 1, 2

[29] J. Liu, S. Di, S. Jin, K. Zhao, X. Liang, Z. Chen, and F. Cappello. SRN-SZ: Deep leaning-based scientific error-bounded lossy compression with super-resolution neural networks, 2023. 2

[30] J. Liu, S. Di, K. Zhao, S. Jin, D. Tao, X. Liang, Z. Chen, and F. Cappello. Exploring autoencoder-based error-bounded compression for scientific data. In *Proceedings of 2021 IEEE International Conference on Cluster Computing*, pp. 294–306, 2021. doi: 10.1109/Cluster48925.2021.00034 2

[31] J. Liu, S. Di, K. Zhao, X. Liang, Z. Chen, and F. Cappello. Dynamic quality metric oriented error bounded lossy compression for scientific datasets. In *SC22: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 62:1–62:15, 2022. doi: 10.1109/SC41404.2022.00067 2

[32] R. G. C. Maack, J. Lukasczyk, J. Tierny, H. Hagen, R. Maciejewski, and C. Garth. Parallel computation of piecewise linear Morse-Smale segmentations. *IEEE Transactions on Visualization and Computer Graphics*, 30(4):1942–1955, 2024. doi: 10.1109/TVCG.2023.3261981 1, 2, 3, 5, 6, 9

[33] A. Otero-de-la Roza. Finding critical points and reconstruction of electron densities on grids. *The Journal of Chemical Physics*, 156(22):224116, 2022. doi: 10.1063/5.0090232 1

[34] R. Seidel and M. Sharir. Top-down analysis of path compression. *SIAM J. Comput.*, 34:515–525, 2005. doi: 10.1137/S0097539703439088 5

[35] M. Soler, M. Plainchault, B. Conche, and J. Tierny. Topologically controlled lossy compression. In *Proceedings of 2018 IEEE Pacific Visualization Symposium*, pp. 46–55, 2018. doi: 10.1109/PacificVis.2018.00015 2

[36] T. Sousbie. The persistent cosmic web and its filamentary structure - I. Theory and implementation. *Monthly Notices of the Royal Astronomical Society*, 414(1):350–383, 2011. doi: 10.1111/j.1365-2966.2011.18394.x 6

[37] 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 *Proceedings of 2017 IEEE International Parallel and Distributed Processing Symposium*, pp. 1129–1139, 2017. doi: 10.1109/IPDPS.2017.115 1, 2

[38] M. Xia, S. Di, F. Cappello, P. Jiao, K. Zhao, J. Liu, X. Wu, X. Liang,

and H. Guo. Preserving topological feature with sign-of-determinant predicates in lossy compression: A case study of vector field critical points. In *ICDE'24: Proceedings of IEEE International Conference on Data Engineering*, 2024 (In Press). 2

[39] L. Yan, X. Liang, H. Guo, and B. Wang. TopoSZ: Preserving topology in error-bounded lossy compression. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):1302–1312, 2024. doi: 10.1109/TVCG.2023.3326920 1, 2

[40] V. Yeghiazaryan and I. Voiculescu. Path reducing watershed for the GPU. In *WACV'18: Proceedings of 2018 IEEE Winter Conference on Applications of Computer Vision*, pp. 577–585, 2018. doi: 10.1109/WACV.2018.00069 2

[41] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cappello. Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation. In *ICDE'21: Proceedings of 2021 IEEE 37th International Conference on Data Engineering*, pp. 1643–1654, 2021. doi: 10.1109/ICDE51399.2021.00145 1, 2

[42] K. Zhao, S. Di, X. Liang, S. Li, D. Tao, Z. Chen, and F. Cappello. Significantly improving lossy compression for HPC datasets with second-order prediction and parameter optimization. In *HPDC'20: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 89–100, 2020. doi: 10.1145/3369583.3392688 1, 2