Towards Improving Reverse Time Migration Performance by High-speed Lossy Compression

Yafan Huang*, Kai Zhao†, Sheng Di‡, Guanpeng Li*, Maxim Dmitriev§,
Thierry-Laurent D. Tonellot§, and Franck Cappello‡
* University of Iowa, Iowa City, IA, USA
† University of Alabama at Birmingham, Birmingham, AL, USA
‡ Argonne National Laboratory, Lemont, IL, USA
§ EXPEC Advanced Research Center, Saudi Aramco, Saudi Arabia
yafan-huang@uiowa.edu, kzhao@uab.edu, sdi@anl.gov, guanpeng-li@uiowa.edu,
maxim.dmitriev@aramco.com, thierrylaurent.tonellot@aramco.com, cappello@mcs.anl.gov

***Abstract*—Seismic imaging is an exploration method for estimating the seismic characteristics of the earth's sub-surface for geologists and geophysicists. Reverse time migration (RTM) is a critical method in seismic imaging analysis. It can produce huge volumes of data that need to be stored for later use during its execution. The traditional solution transfers the vast amount of data to peripheral devices and loads them back to memory whenever needed, which may cause a substantial burden to I/O and storage space. As such, an efficient data compressor turns out to be a very critical solution. In order to get the best overall RTM analysis performance, we develop a novel hybrid lossy compression method (called HyZ), which is not only fairly fast in both compression and decompression but also has a good compression ratio with satisfactory reconstructed data quality for post hoc analysis. We evaluate several state-of-the-art error-controlled lossy compression algorithms (including HyZ, BR, SZx, SZ, SZ-Interp, ZFP, etc.) in a supercomputer. Experiments show that HyZ not only significantly improves the overall performance for RTM by 6.29∼6.60× but also obtains fairly good qualities for both RTM single snapshots and the final stacking image.**

***Keywords*—Lossy Compression, Performance, Seismic Imaging, Reverse Time Migration**

I. Introduction

Seismic imaging is an exploration method used to estimate the seismic characteristics of the earth's sub-surface by measuring the reflected acoustic energy waves. This technology has been broadly used to explore the sub-surface structure of rock formations for geologists and geophysicists or used to explore mineral, coal, gas, and oil for fuel companies. Reverse time migration (RTM) is a cutting-edge seismic imaging method, which has been widely used in the seismic imaging community.

The parallel RTM code, however, suffers from an extremely large amount of data to process, which turns out to be the major concern for seismic imagining analysis. Specifically, RTM involves two critical stages – a forward propagation of the source wavefield and a backward propagation of the receiver wavefield. During the forward propagation, RTM would generate thousands of 3D snapshots, which occupy extremely large volumes of data and need to be maintained for later access by the RTM. According to [1], for instance, an aperture of 10x10 km and a maximum depth of 8 km may project up to 2,800 terabytes of the data to process for only one shot with 6k snapshots if the maximum frequency is 80 Hz, migration time is set to 6 seconds, and minimum velocity is 1500m/s. During the backward propagation, the 3D snapshots produced by the forward propagation need to be retrieved and processed to generate a stacking image, which is a final analytic result to reveal the structural information of the sub-surface. With the development of parallel programming models (e.g. OpenMP and MPI) and GPU implementation supports (CUDA), the computation costs of RTM are significantly reduced. However, how to efficiently process such a large volume of 3D snapshot data remains a critical challenge.

To address the above-mentioned big data issue, a straightforward solution is transferring the snapshots from memory to peripheral devices temporarily and loading them back to memory upon usage, which however still faces some serious issues or challenges. On the one hand, using the peripheral device to store a vast amount of runtime data may degrade the overall execution performance because of the expensive data transferring cost inevitably. On the other hand, the capacity of the peripheral device may still be not enough, in the consideration of the extremely large amount of data to be produced during the forward propagation. For the CPU environment, for example, the peripheral device could be a parallel file system (PFS), which projects only tens or hundreds of TBs for a regular user on a supercomputer [2].

Another potential solution is compressing the RTM snapshot data to mitigate data transfer costs and storage burden. Specifically, the snapshots produced by the forward propagation are compressed by a data compressor, and the compressed snapshots would be kept in either memory or peripheral devices. Later on, the compressed snapshots will be loaded and decompressed for the backward propagation analysis at

Corresponding author: Sheng Di, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 Cass Avenue, Lemont, IL 60439, USA

runtime. As revealed by many existing studies [3]–[5], lossless compression suffers from very low compression ratios (1.2∼2 in general) for scientific datasets. Thus we focus on error-controlled lossy compression in our work, to achieve both a high compression ratio and user-accepted introduced errors. In order to prevent the RTM execution from being delayed significantly by compression/decompression overhead, the qualified compressor must offer very high compression and decompression throughput. This is because the RTM method generally has quite high performance in its execution [6] due to substantially optimized parallel code. To the best of our knowledge, we are the very first work to integrate lossy compression into the RTM execution framework.

In this paper, we explore the best lossy compression solution that has high speed in both compression and decompression while preserving a good compression quality (high compression ratio and high fidelity of data), so that the overall RTM execution can be improved significantly without any loss of analysis quality. The key contributions in this work are summarized as follows:

- We develop a lossy compression-based RTM framework for seismic imaging analysis by integrating data compression and decompression into forward and backward propagation respectively, to deal with the vast amount of data produced during the RTM execution more efficiently.
- We propose a novel lossy compression method – called *HyZ* which combines two high-speed lossy compression algorithms. The first one is our proposed block-wise regression-based compressor (BR), and the other one is ultra-fast prediction-based compressor SZx [7].
- We integrated different lossy compressors (including HyZ, BR, SZx, SZ, ZFP, SZ-Interp, etc.) into an industrial RTM code and run it with thousands of snapshots on a supercomputer.
- We comprehensively investigate the quality for each of the lossy compressors and their impact to RTM execution results as well as overall performance. Results show that our proposed hybrid compressor HyZ has the best visualization quality in class, also leading to good compression ratios (5+) for users. In overall RTM execution, HyZ outperforms the second-tier lossy compressors (SZ, ZFP, etc.) by up to 2.23× and outperforms execution without lossy compressor by 6.29-6.60×.

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 first analyze an existing ultra-fast prediction-based lossy compression algorithm SZx and explain its limitation in the RTM execution, then we propose the BR compressor along our hybrid solution HyZ. In Section V, we comprehensively evaluate many state-of-the-art compressors by running an industry-level parallel RTM code used in a supercomputer. In Section VI, we conclude the paper with a discussion of future work.

II. Related Work

In this section, we introduce the related works from two perspectives: existing solutions to resolve the big snapshot data issue in RTM and existing lossy compression algorithms that have been developed and used widely.

A. Resolving the Limited Memory Capacity Issue in RTM

The RTM simulation is facing a serious memory burden in practice due to terabytes of snapshot data being generated during forward propagation. To overcome this bottleneck, several solutions from different perspectives have been proposed. Fu et al. [8] adopted an FPGA-based solution to remove memory constraints and provide a high performance. Perrone et al. [9] designed a domain-specific data partition strategy to parallel RTM execution on main memory of different nodes and thus avoid the low I/O bandwidth of disk. AlOnazi et al. [10] addressed this issue by deploying executions on distributed-memory systems equipped with multiple GPUs. Alturkestani et al. [11] leveraged the GPU's High Bandwidth Memory (HBM) as an additional storage media layer to maximize RTM I/O Bandwidth. Although these existing methods achieve promising performance in RTM, the high storage requirement is not mitigated at all, as the size of generated image data remains the same. Several works [8], [9] have explored the possibility of using data compression techniques to trade for the I/O bandwidth and storage cost, while they mainly focus on the lossless compression in their solutions. Many existing studies [12] showed that lossless compressors [13]–[15] suffer from very low compression ratios, especially in a comparison with lossy compressors on scientific datasets.

B. Error-controlled Lossy Compression for Scientific Datasets

Error-controlled lossy compression is a very promising solution to resolve the big data issue in RTM execution. The most important advantage of lossy compression is a significantly higher compression ratio than lossless compression, as demonstrated in prior studies [16], [17]. The existing state-of-the-art lossy compressors include SZ [16], [17], ZFP [18], FPZIP [19], TTHRESH [20], MGARD [21], bit grooming [22], digit rounding [23], and several emerging auto-encoder-based compressors [24]–[26].

In the scientific data lossy compression community, SZ and ZFP are two leading compressors, because of their fairly high compression ratios and high compression speed compared with other state-of-the-art. According to [16], SZ and ZFP have much higher compression ratios than FPZIP because of their innovative algorithms in data prediction and decorrelation. Based on the SZ compression framework, Zhao et al. [5] developed a very effective prediction method that can significantly improve the compression ratio on RTM analysis datasets. However, its speed is comparable or even lower than that of the generic SZ compressor, which does not meet the high-speed requirement in our use case. Although TTHRESH can get much higher compression ratios than SZ and ZFP do, it suffers from very low compression/decompression performance (about one order of magnitude lower) due to its

expensive high-order singular value decomposition (HOSVD). Bitgrooming and digitrounding both suffer from very low compression ratios because they both ignore the data correlations in compression. There have been a few auto-encoder-based lossy compression methods [24]–[26] proposed recently, but none of them are qualified for RTM executions because of very low performance in both compression and decompression. Specifically, Liu et al. [26] proposed an effective method to combine the auto-encoder and SZ compression framework which exhibits the best compression quality from among all auto-encoder-based compressors, but it is still about 2-3× slower than SZ and ZFP.

III. System Design and Problem Formulation

In this section, we describe our designed RTM method integrated with error-controlled lossy compression technologies, and also formulate the research problem.

A. Design of Lossy-compression based RTM

Figure 1 illustrates the workflow of a parallel RTM execution, which corresponds to the practical seismic imaging analysis. Specifically, at the beginning of each run, the RTM execution is triggered based on the input information which contains several key parameters such as problem size, initial background data file, total number of snapshots, and how many time steps a snapshot will be saved for backward propagation analysis. The forward propagation of source wavefield generates a snapshot at each time step; while only a subset of the snapshots (e.g., every K time steps) are selected/kept for the later analysis (see step ❶ in Figure 1) and others are disregarded. In the traditional design, the selected snapshots are kept either in memory (if memory capacity is large enough) or dumped to external devices temporally. After the forward propagation, the user will perform a backward propagation of the receiver wavefield for the analysis of sub-surface structure, which depends on the snapshots generated by the forward propagation. As such, the snapshots saved previously during the forward propagation need to be retrieved for the imaging analysis (see step ❷ in Figure 1). A final stacking image (as the analysis result) would be generated after the backward propagation (see step ❸ in Figure 1).

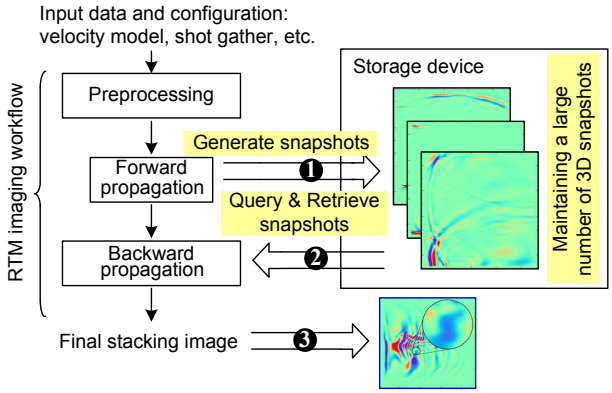


Fig. 1. Illustrating of the big data issue in Reverse Time Migration (RTM)

Through an in-depth investigation of RTM execution workflow [27], [28] and diverse error-bounded lossy compressors [7], [18], [29], we successfully integrated lossy compression techniques in a scalable parallel industrial RTM code. Specifically, unlike the traditional code which keeps the original raw snapshots either in memory or external devices such as parallel file systems (PFS), our design compresses the snapshots by a lossy compression method before saving them. During the backward propagation stage, whenever a snapshot needs to be used, the corresponding compressed data is queried and loaded into the memory, and decompressed for the imaging work.

B. Problem Formulation

Our objective is to optimize the overall end-to-end execution performance of the RTM execution, covering the cost of preprocessing, forward propagation, and backward propagation as well as all possible overheads such as compression time and I/O cost. The fundamental idea is to leverage lossy compression to reduce the volume of forward propagation snapshot data, which thus can significantly reduce I/O cost for the overall execution.

Although lossy compression can obtain significantly higher compression ratios than lossless compression, it may introduce data distortion to the reconstructed data, thus we need to control the compression errors carefully, especially from the perspective of the post hoc seismic analysis. As for the compression quality, we will focus on both the compression ratio and the quality of the reconstructed data. On the one hand, we use commonly-used lossy compression-related metrics such as peak signal-to-noise ratio (PSNR) and structural similarity index measure (SSIM) to check the data distortion from the perspective of lossy compression. On the other hand, with the involvement of seismic researchers from industry, we also evaluate the visual quality of the reconstructed data as Seismic imaging analysts often need to analyze the waves by observing the snapshots and stacking images [27], [28].

IV. HyZ: A High-speed High-fidelity Lossy Compression Method for RTM

In this section, we propose a novel high-speed high-fidelity lossy compression method (called HyZ), which can also maintain a satisfactory compression ratio, for RTM execution and post hoc analysis.

Since RTM method generally has a relatively high parallel performance in seismic imaging analysis, a qualified lossy compressor has to be fast enough and also with good compression ratios. To this end, we first analyze an existing ultra-fast lossy compression algorithm (called SZx [7]) and explain its limitations in RTM execution. We observe that SZx cannot maintain high-fidelity in reconstructing single snapshot images within a satisfactory compression ratio (5+). To solve this issue, we then develop a high-speed regression-based lossy compressor (BR) that can preserve the quality of the snapshot data very well. Finally, to enable error-control in BR compressor, we propose our hybrid compressor design – HyZ,

which integrates SZx to compress blocks that exceed the error bound in BR compressor during RTM execution.

A. Discussing the Limitations of SZx

One ultra-fast lossy compressor that is potentially suitable for RTM execution is SZx, which was proposed by Yu et al. [7]. The design guideline of SZx is making the compressor composed of fairly lightweight operations including only bitwise, addition, and subtraction. By comparison, other state-of-the-art lossy compressors depend on relatively heavier operations such as multiplication (used by SZ, ZFP), variable-length encoding (adopted by SZ, ZFP, MGARD, digit rounding, and bit grooming), and dictionary encoding (adopted by SZ, MGARD, digit rounding, and bit grooming).

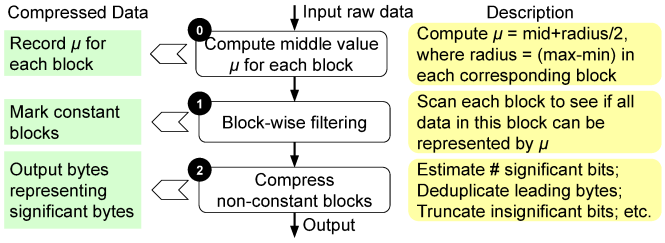


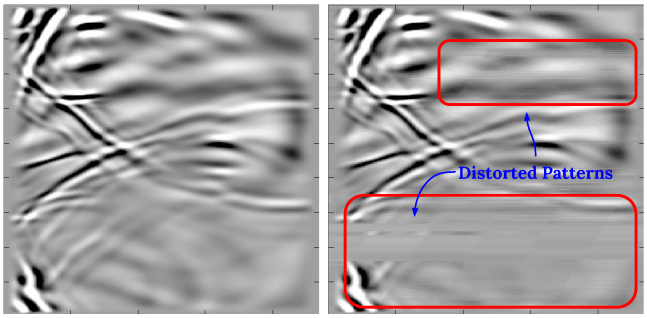
Fig. 2. Compression pipeline (workflow) of SZx

The SZx algorithm splits the whole dataset into many small 1D fixed-size segments (or blocks), and then performs compression on each block separately. It consists of two critical stages to process different data points in the dataset: ❶ block-wise filtering, ❷ compressing non-constant blocks, as illustrated in Figure 2.

Block-wise filtering aims to check each block to see if it can be represented by some constant number μ. Specifically, if the amplitude of the data variation in one block is lower than or equal to twice of the user-specified error bound, the data in this block can be approximated by μ=$(\max - \min)/2$, where $\min$ and $\max$ refer to the minimal value and maximum value in the corresponding data block, respectively. Such blocks are called 'constant blocks'; otherwise, they are 'non-constant blocks'. As such, the block-wise filtering step requires a preprocessing step to compute the middle value μ for each block (shown as step ❶ in Figure 2). After finishing the block-wise filtering step, there will be two outputs committed to the compressed data: the μ array and the block-type array, which record the middle values and block types, respectively.

Such a block-wise filtering method can significantly improve the compression ratio, however, it may reduce the visualization quality for RTM snapshot data. The constant blocks can be reconstructed only based on the μ array and the RTM snapshot data is routinely very smooth in space so that majority of the data belong to the constant blocks. Though such a design can improve compression ratio in RTM data, indiscriminately using the same μ to represent the whole block is likely to cause visible artifacts.

To verify such a situation, Figure 3 visualizes the reconstructed single snapshot (time step=3000) produced by SZx.



(a) Original Data (b) SZx Reconstructed Data

Fig. 3. Visualization of single snapshot image (time step=3000) by SZx with REL=1E-2 error bound (∼5 compression ratio) and 128 block size choice. Human-visible artifacts in (b) are highlighted.

We set the error bound as REL=1E-2 (*c.f.* Section V-A4) to keep the compression ratio at around 5, which is the minimal satisfactory ratio in RTM execution. As shown in Figure 3 (b), a large area of distorted patterns can be observed. Because of the compression algorithm design for 'constant blocks' in SZx, the distorted patterns are also distributed continuously. Also, the PSNR and SSIM are only 54.95dB and 0.7244, respectively. Improving visualization quality requires a smaller error bound such as REL=1E-3 or a smaller block size choice such as 32, which will also cause a lower compression ratio and cannot satisfy the seismic community. In conclusion, the ultra-fast design of SZx is suitable in speed for RTM execution, however, it has some defects with respect to data visualization.

B. Block-wise Regression-based Lossy Compression (BR)

The key idea is splitting the entire dataset into many small blocks (e.g., 4×4×4) and then approximating the data values in each block by a regression hyperplane. This idea is motivated by an important observation that the data points in a very small region in space are likely able to be approximated by a simple hyperplane (e.g., linear regression). We demonstrate three examples in Figure 4: block Ⓐ, block Ⓑ, and block Ⓒ. Because values are continuous between adjacent data points, their colors also exhibit similarly, hence showing their linear hyperplane properties. Such observations are verified on almost all RTM snapshots by checking slopes between adjacent data points.

In our design, to achieve fast compression/decompression speed and high compression ratio, we explore linear regression in our proposed block-wise regression (BR) method, as described below. During the compression phase, each data block is approximated by a linear hyperplane with four coefficients, which can be found as β_0 to β_3 in Formula 1.

$$f(x, y, z) = \beta_1 x + \beta_2 y + \beta_3 z + \beta_0 \quad (1)$$

where x, y, and z denots the relative location (or index) of in the data block. That is, $x = \{0, 1, \cdots, n_1\}$, $y = \{0, 1, \cdots, n_2\}$, $z = \{0, 1, \cdots, n_3\}$, for the data block $n_3 \times n_2 \times n_1$. Thus, only four coefficient values need to be

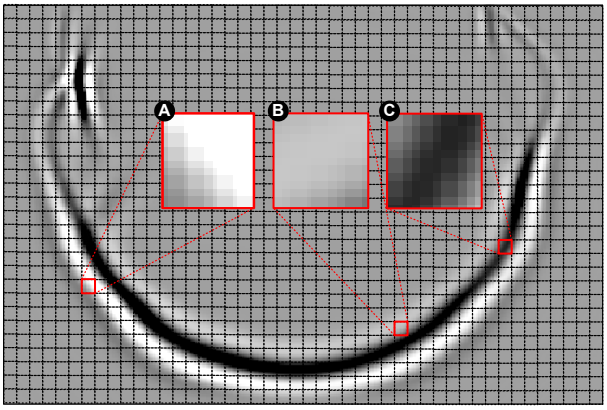


Fig. 4. Illustration of Linearity of Small Data Blocks in An RTM Snapshot

stored to substitute all the data in one block, no matter what size of the block is (e.g. $3 \times 3 \times 3$ or $4 \times 4 \times 4$). During the decompression, each data block would be recovered by the hyperplane reconstructed with the four regression coefficients (i.e. Formula 1). Without loss of generality, again suppose the data block size is $n_3 \times n_2 \times n_1$, then its regression coefficients can be calculated as the following formula (d_{ijk} refers to the data values in the data block at the location $\{i,j,k\}$).

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

where

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

In this compression method, all computations are linear and their required data are independent. Thus, these computations can be highly parallelized, guaranteeing the compression and decompression speeds with multi-thread designs. Because of the nature of linear properties on RTM snapshot data, linear regression also can preserve a high fidelity for reconstructed data, which will be evaluated in Section V. In addition, the compression ratio can be estimated based on the following formula. For instance, when the block size is 4×4×4, the compression ratio is 64/4=16; when the block size is 3×3×3, the compression ratio is 27/4=6.75. Note that though a greater block size leads to a higher compression ratio, it can reduce the reconstructed data fidelity in turn, since it maintains the same number of coefficients on more data points.

$$CR = \frac{n_3 n_2 n_1}{4} \quad (3)$$

1) Discussion 1: High-Order Regressions: Besides our linear regression design, there are also other high-order regression methods. We use quadratic (i.e. 2nd-order) regression here for example, and similar conclusions can be drawn on other high-order regressions (i.e. cubic or 3rd-order regression) as well. Quadratic regression approximates each block via a quadratic hyperplane with 10 coefficients. Compared with linear regression, quadratic regression maintains more coefficients and thus can predict data points more accurately. The formula can be found below.

$$f(x,y,z) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 z + \beta_4 x^2 + \beta_5 xy + \beta_6 xz + \beta_7 y^2 + \beta_8 yz + \beta_9 z^2 \quad (4)$$

Similar to linear regression, the quadratic regression based compression needs to store 10 coefficients in each block, which will be used to reconstruct data during the decompression. Using the least-square method, the 10 regression coefficients can be calculated as the Equation (5).

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

where

$$\begin{bmatrix} 1 & x & y & z & x^2 & xy & xz & y^2 & yz & z^2 \\ x & x^2 & xy & xz & x^3 & x^2y & x^2z & xy^2 & xyz & xz^2 \\ y & xy & y^2 & yz & x^2y & xy^2 & xyz & y^3 & y^2z & yz^2 \\ z & xz & yz & z^2 & x^2z & xyz & xz^2 & y^2z & yz^2 & z^3 \\ x^2 & x^3 & x^2y & x^2z & x^4 & x^3y & x^3z & x^2y^2 & x^2yz & x^2z^2 \\ xy & x^2y & xy^2 & xyz & x^3y & x^2y^2 & x^2yz & xy^3 & xy^2z & xyz^2 \\ xz & x^2z & xyz & xz^2 & x^3z & x^2yz & x^2z^2 & xy^2z & xyz^2 & xz^3 \\ y^2 & xy^2 & y^3 & y^2z & x^2y^2 & xy^3 & xy^2z & y^4 & y^3z & y^2z^2 \\ yz & xyz & y^2z & yz^2 & x^2yz & xy^2z & xyz^2 & y^3z & y^2z^2 & yz^3 \\ z^2 & xz^2 & yz^2 & z^3 & x^2z^2 & xyz^2 & xz^3 & y^2z^2 & yz^3 & z^4 \end{bmatrix}$$

$$\boldsymbol{V} = (V_1, V_2, .., V_9), V_t = \sum_{x=0}^{n_1-1}\sum_{y=0}^{n_2-1}\sum_{z=0}^{n_3-1} g_{xyz}(t) * f_{xyz}, 0 \le t \le 9$$

$g_{x,y,z}(t)$ returns tth element from list $[1, x, y, z, x^2, xy, xz, y^2, yz, z^2]$

In this method, the compression ratio can be calculated as follows, as there are 10 coefficients to store per block.

$$CR = \frac{n_3 n_2 n_1}{10} \quad (6)$$

High-order regression methods may preserve good reconstructed image quality in some circumstances since they store more coefficients within the same block (i.e. the same number of data points). Moreover, more coefficients inside the same block can lead to a lower compression ratio. Given a 3×3×3 block size, for example, linear regression can achieve a $27/4 = 6.75$ compression ratio, while quadratic regression has a compression ratio of $27/10 = 2.7$, which cannot satisfy the RTM execution (5+). In addition, higher-order regression methods require more complex computations while obtaining the coefficients in the compression stage, which significantly reduces the compression speeds in RTM. Therefore, we only consider linear regression in this work.

2) Discussion 2: Other Regression-based Compressors: Besides our BR solution, polynomial regressions (e.g. linear/quadratic regression) have also been adopted in several existing lossy compression techniques [30], [31]. However, our solution substantially differs from the prior works, due to a lightweight design for RTM execution. SZ2.1 [29] leverages linear regression to predict the data values in the high-compression cases [30]. Zhao et al. [31] integrates the quadratic regression into the prediction stage of the SZ compression framework, which can significantly improve the compression quality for some wave-patterned datasets. These related works both treat the regression method as a predictor in the entire compression pipeline. In order to keep a very high compression ratio, they have to compress the coefficients, which may degrade the accuracy of the regression/prediction in turn. As such, they have to operate a few other expensive compression steps (such as quantization, variable-length encoding, and dictionary encoding) to control the compression errors. By comparison, our solution can skip the three expensive compression steps in that we store the regression coefficients as they are in the compressed data. Such a lightweight algorithm has substantially higher compression/decompression speeds, while still preserving the reconstruction quality very well because of the nature of linear properties in RTM snapshot data (to be shown later).

C. HyZ: Combining BR and SZx for RTM Execution

In this subsection, we describe how we construct a hybrid lossy compression framework HyZ by using two compression methods (BR and SZx).

As shown in Section IV-B, BR splits RTM snapshot data into many small blocks and adopts a regression hyperplane to preserve data value inside each block, which preserves the reconstructed RTM data quality and a controllable compression ratio. By only storing the regression coefficients as compressed data, BR is also ultra-fast and hence is suitable for RTM execution. However, due to the lightweight design, BR compressor does not support error-control. For example, we use BR-Linear with 4×4×4 block size to compress a single RTM snapshot data (time step=3000). After the reconstruction, we find there are around less than 1% data points (distributed in 3% blocks) exceeding the REL 1E-2 error bound. Existing error-control designs [29], [31] require time-consuming computations (such as quantization and encoding) that negatively impact the speed constraint in RTM execution. Thus, to further improve the reconstructed data quality for RTM, we integrate SZx into BR compressor and propose HyZ, of which workflow is shown in Figure 5.

The fundamental design of HyZ is to enable fast error-control in BR compressors via another ultra-fast compressor SZx. Specifically, HyZ consists of 3 major steps: ❶ BR compressor, ❷ sampling-based block error checking, and ❸ SZx compressor. Given a user-specified error bound and the input raw data, HyZ first utilizes BR compressor (shown as ❶ in Figure 5) to process data and obtain the corresponding regression coefficients. In order to check if data points are error-bounded inside each block, HyZ then reverses the regression coefficients to the decompressed data. Since data comparison operations are likely to incur extra computation cost, HyZ samples data from eight corners inside a block (i.e. cube) to determine if this block is error-bounded (shown as ❷ in Figure 5). By doing so, more than 97% of outlier blocks can be found across all RTM snapshot data. Though there are still less than 3% blocks that are not error-bounded, we argue that these outlier data points only occupy less than 0.1% of the whole dataset, which in fact has only negligible influence on both statistic image quality (PSNR and SSIM) and visualization quality. Finally, for all the recognized blocks that are not error-bounded, HyZ combines them into an array and adopts SZx to compress this array (shown as ❸ in Figure 5). Note that although constant block compression methodology in SZx may cause visible errors on the RTM dataset, such circumstances only can be observed on consecutive data. However, in this combined array produced by ❷, the data points are actually distributed discretely with respect to their position in the original RTM snapshot, hence not leading to consecutive visible errors. In all, HyZ combines two ultra-fast compressors, hence not only can achieve promising visualization results but also can maintain top-tier compression/decompression speeds.

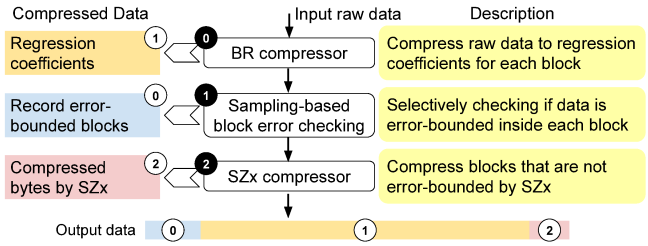


Fig. 5. Compression pipeline (workflow) of HyZ

V. Performance Evaluation

In this section, we present experimental setups and analyze the performance evaluation results.

A. Experimental Settings

We describe the environment, datasets, and compressors as follows.

1) Environment: We perform multi-thread OpenMP experiments using Argonne Bebop supercomputer, in which each node has two Intel Xeon E5-2695 v4 processors and 128 GB of DRAM. Since each Bebop node is equipped with 36 cores, we run our experiments with 36 threads for each compressor.

2) RTM Code and Datasets: We perform the evaluation using an industrial parallel RTM code, which is a production-level RTM implementation. Specifically, we run the code with 3600 time steps at a fraction of Overthrust Belt seismic data of which dimension is 449×449×235. That is, there are 3600 snapshots generated to be compressed during the forward propagation stage and each snapshot contains 449×449×235 single-precision floating-point data points.

3) State-of-the-Art Compressors in Our Evaluation: We evaluate BR, HyZ, and SZx compressors in RTM execution with 4 other state-of-the-art lossy compressors (SZ-Interp [5], SZ-Interp-fast, SZ2.1.12 [29], ZFP0.5.5 [18]). SZ-Interp adopts an interpolation method in the SZ compression framework, which can significantly improve the compression quality in turn. SZ-Interp-fast is a modified version of SZ-Interp, by replacing its expensive steps (Huffman encoding and Zstd) with a lightweight encoder - run-length encoding [32]. SZ2.1 is the latest public version of the SZ compressor, which exhibits fairly high compression quality and performance as validated by many prior studies [17], [30]. ZFP is another outstanding error-bounded lossy compressor, which may outperform SZ2.1 in some cases as verified by existing studies [5]. We have integrated all the lossy compressors into the RTM execution in our experiments.

We also evaluate the compression ratios of 3 state-of-the-art lossless compressors – ZFP, FPZIP, and Zstandard (Zstd) for a comparison. ZFP and FPZIP also support lossless compression in addition to lossy compression as long as all the bit-planes are preserved during the compression. Zstd is an outstanding lossless compressor, which is substantially faster than other lossless compressors according to prior studies [14], [33].

4) Error Bounds for Lossy Compressors: In lossy compressors with error-control, there are two types of error bounds, absolute (ABS) error bound and value-range-based relative (REL) error bound, which are both commonly used in scientific applications [17], [30], [34]. The ABS error bound (denote as δ) is set as a constant. As for REL error bound, it is a linear computation based on the global data value range size, i.e. λr, where $\lambda \in (0,1)$ and r denote relative ratio and data range size respectively. As a result, for a given set of data $D = \{d_1, d_2, ..., d_n\}$, its decompressed data $D' = \{d'_1, d'_2, ..., d'_n\}$ should satisfy error bounds by following equations.

$$\max_{i=1,2,...,n} (d_i - d'_i) \leq \begin{cases} \delta, & \text{ABS error bound } \delta \text{ is used.} \\ \lambda r, & \text{REL error bound } \lambda \text{ is used.} \end{cases} \tag{7}$$

5) Evaluation Metrics:

- Compression Ratio (CR): CR (defined as $\frac{original\ size}{compressed\ size}$) indicates how much the snapshot data can be reduced in memory. The minimum qualified compression ratio for RTM execution is 5 following the suggestions of seismic experts. Besides, our execution node has 128GB memory and the total volume of RTM snapshot data is up to 635GB. And we also avoid the execution crash because of an out-of-memory issue.
- Compression speed and decompression speed: As mentioned, compression/decompression speed is critical to the overall execution performance, as too high overhead may cause significant delays unexpectedly. In general, these two speeds are measured in the form of throughput (i.e., $\frac{original\ size}{compression\ time}$ (GB/s) and $\frac{reconstructed\ size}{decompression\ time}$ (GB/s), respectively).
- PSNR and SSIM: as mentioned in Section III, they are used for quantifying data distortion in RTM execution. Their detailed definitions can be found in [35] and [36].
- Visual quality: We will visualize the data for both snapshots and stacking image to present the impacts of different lossy compressors to the results.
- Overall performance of RTM: Entire execution time of one RTM run that involves four stages: execution kernel (i.e. RTM algorithm computation), I/O write (including data compression in forward propagation), I/O read (including data decompression in backward propagation), and stacking image generation.

B. Evaluation Results and Analysis

We present evaluation results and conclusions as follows.

TABLE I
AVERAGE COMPRESSION RATIOS BASED ON THE SAME ERROR BOUND OVER 3600 SNAPSHOTS IN RTM EXECUTION. (ALL COMPRESSORS ARE OPENMP VERSIONS EXCEPT FOR SZ (SERIAL))

Compressor	Relative Error Bound 1E-3	Relative Error Bound 4E-4	Absolute Error Bound 1E-5	Absolute Error Bound 4E-6	**Average**
SZx	7.58	6.22	9.57	7.72	7.77
SZ-Interp	213.53	116.60	177.92	113.20	155.31
SZ-Interp-fast	17.05	9.09	16.26	11.06	13.37
SZ	26.94	25.51	26.83	24.37	25.91
SZ(Serial)	70.36	50.68	81.47	54.94	64.36
ZFP	26.92	20.65	38.74	32.04	29.59

Compressor	Block Size $3\times3\times3$	Block Size $4\times4\times4$	Block Size $5\times5\times5$	Block Size $6\times6\times6$	**Average**
BR	6.66	15.72	31.11	52.64	26.53

1) Compression Ratio: Table I presents the average compression ratios of different lossy compressors among 3600 snapshots with the same error bounds. For error-bounded compressors (such as SZx, SZ and ZFP), we use two types of error bounds – ABS and REL. Their specific definitions can be found in Section V-A4. For the non-error-bounded compressor BR, we present its compression ratios based on different block sizes. We can clearly observe that all lossy compressors here meet the compression ratio requirement (5+) even when we set relatively low error bounds: e.g., REL=4E-4 and ABS=4E-6. Specifically, SZ-Interp exhibits the highest compression ratios (113.2∼213.53), and SZx has lowest compression ratios (6.22∼9.57). The key reason is that SZ-Interp fully leveraged the data correlation in 3D dimensions by a dynamic interpolation method [5], while SZx is designed based on 1D correlation of the dataset. BR obtains compression ratios from 6.66 to 52.64. SZ-Interp-fast has significantly lower compression ratios than SZ-Interp, in that its run-length encoding is not as effective as Huffman encoding especially when the error bound is relatively low (the pattern with consecutive symbols is very rare in this situation). We notice that SZ has different compression ratios between its OpenMP and Serial (i.e. single thread) versions, and the compression ratio in OpenMP version is degraded by $\sim$2.5$\times$ on average. This is because the OpenMP version

needs to make sure data independence across blocks during compression, which would lose prediction accuracy inevitably.

TABLE II
AVERAGE COMPRESSION RATIOS OF HYZ BASED ON DIFFERENT ERROR BOUNDS AND BLOCK SIZE CHOICES OVER 3600 SNAPSHOTS IN RTM EXECUTION.

HyZ	Relative Error Bound 1E-3	Relative Error Bound 4E-4	Absolute Error Bound 1E-5	Absolute Error Bound 4E-6	**Average**
$3 \times 3 \times 3$	5.01	4.61	5.17	4.99	12.31
$4 \times 4 \times 4$	9.14	8.20	10.29	9.69	
$5 \times 5 \times 5$	13.87	12.24	17.10	15.59	
$6 \times 6 \times 6$	18.50	16.11	24.84	21.52	

Table II presents the compression ratios of HyZ. Since HyZ has error-control by integrating SZx to BR, it has two compression settings, block size and error bound. For these two settings in HyZ, smaller error bounds and block sizes exhibit lower compression ratios. Such results are similar to what we have observed in SZx and BR, respectively. The compression ratio of HyZ is smaller than BR, since it requires extra space for error-control computation. In all, HyZ has an average compression ratio of 12.31, which satisfy the RTM execution.

TABLE III
AVERAGE COMPRESSION RATIOS OF THREE LOSSLESS COMPRESSORS OVER 3600 SNAPSHOTS IN RTM EXECUTION.

Compressor	Zstd	FPZIP	ZFP
Average CR	1.53	2.17	1.85

Table III shows the compression ratios of the lossless compressors. The overall compression ratios of Zstd, FPZIP, and ZFP are in the range of [1.53, 2.17], which is too low to complete the whole execution because of limited memory capacity. Moreover, they are all considerably slower than lossy compressors: the serial versions of FPZIP, ZFP and Zstd are slower than SZ(serial) by ∼3-5×, ∼5-10×, 10+×, respectively, which is far less than the expected compression throughput.

Takeaway 1: HyZ and BR both can meet compression ratio requirement (5+) in RTM execution across 3600 time steps, with 12.31 and 26.53 on average.

TABLE IV
OPENMP COMPRESSION SPEEDS (GB/S) BASED ON THE SAME ERROR BOUND OVER 3600 SNAPSHOTS IN RTM EXECUTION.

Compressor	Relative Error Bound 1E-3	Relative Error Bound 4E-4	Absolute Error Bound 1E-5	Absolute Error Bound 4E-6	**Average**
SZx	12.48	11.08	16.00	8.78	12.09
SZ-Interp	2.87	2.76	2.76	2.47	2.72
SZ-Interp-fast	5.48	5.36	5.53	5.11	5.37
SZ	3.63	3.57	3.61	3.57	3.60
ZFP	2.82	2.56	4.33	4.04	3.44

Compressor	Block Size $3 \times 3 \times 3$	$4 \times 4 \times 4$	$5 \times 5 \times 5$	$6 \times 6 \times 6$	**Average**
BR	5.85	15.64	24.36	32.46	19.58

TABLE V
OPENMP DECOMPRESSION SPEEDS (GB/S) BASED ON THE SAME ERROR BOUND OVER 3600 SNAPSHOTS IN RTM EXECUTION.

Compressor	Relative Error Bound 1E-3	Relative Error Bound 4E-4	Absolute Error Bound 1E-5	Absolute Error Bound 4E-6	**Average**
SZx	19.65	18.13	22.45	15.44	18.92
SZ-Interp	5.26	4.88	4.77	4.48	4.85
SZ-Interp-fast	9.00	9.10	9.61	8.03	8.94
SZ	5.14	4.75	5.01	4.76	4.92
ZFP	0.60	0.54	0.62	0.58	0.59

Compressor	Block Size $3 \times 3 \times 3$	$4 \times 4 \times 4$	$5 \times 5 \times 5$	$6 \times 6 \times 6$	**Average**
BR	10.67	19.36	24.96	26.59	20.40

2) Compression and Decompression Speed: Table IV and Table V present the OpenMP compression and decompression speeds for different lossy compressors over 3600 snapshots in RTM execution, respectively. As shown in the tables, the SZx and our proposed BR significantly outperform other lossy compressors. In absolute terms, the compression speeds of SZx and BR reach 12.09GB/s and 23.67GB/s on one node from Bebop supercomputer, which is about 2∼4× as high as the state-of-the-art lossy compressors SZ and ZFP. Note that all lossy compressors except ZFP have a larger throughput in the decompression stage, and the reason is that decompression requires fewer computations to reconstruct the data. For ZFP's decompression, it does not have OpenMP version, so we can only run the serial version instead, which suffers very low speed as shown in the table.

Table VI presents the OpenMP compression and decompression speeds of HyZ. The average compression and decompression speeds of HyZ are 10.69GB/s and 12.45GB/s, which are 45.40% and 38.97% slower than those of BR, respectively. The key reason is that HyZ introduces sampling-based block error checking and SZx compression stages to enable error-control and thus further improve visualization quality. However, HyZ is still 3∼4× faster than the second tier compressors such as SZ and ZFP. In all, HyZ achieves first-class compression and decompression speeds in RTM execution compared with existing lossy compressors.

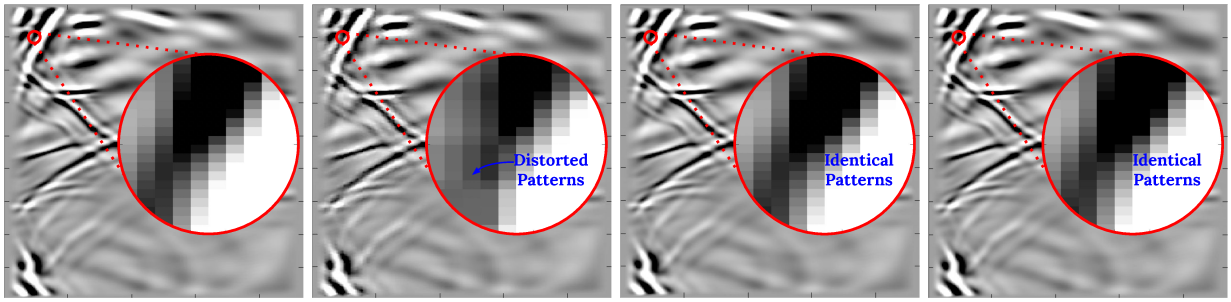
Takeaway 2: HyZ and BR both have top-class compression and decompression speeds. HyZ can achieve 10.69GB/s and 12.45GB/s compression and decompression speeds respectively, while these numbers are 19.58GB/s and 20.40GB/s in our proposed BR compressor.

3) Data Distortion: We evaluate data distortion on both single snapshots and final stacking image in RTM execution. For single snapshot, we use one typical snapshot (time step=3000) for example, and similar results can be obtained in other RTM snapshots as well. Following the suggestions of seismic experts, we choose REL=1E-3 as error bound for examining the visualization errors in HyZ.

Figure 6 visualizes the reconstructed single snapshot produced by BR and HyZ. As seen in Figure 6 (b) and (c), HyZ maintains a better visualization with identical patterns,

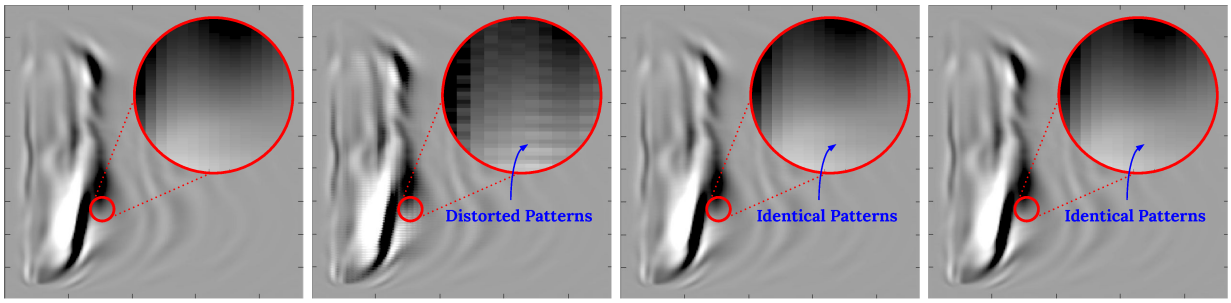
TABLE VI
AVERAGE OPENMP COMPRESSION/DECOMPRESSION SPEEDS (GB/S) OF HYZ BASED ON DIFFERENT ERROR BOUNDS AND BLOCK SIZE CHOICES OVER 3600 SNAPSHOTS IN RTM EXECUTION.

HyZ	Compression Speed					Decompression Speed				
	Relative Error Bound		Absolute Error Bound		Average	Relative Error Bound		Absolute Error Bound		Average
	1E-3	4E-4	1E-5	4E-6		1E-3	4E-4	1E-5	4E-6	
$3 \times 3 \times 3$	4.54	4.28	4.72	4.33	10.69	7.45	7.33	7.65	7.26	12.45
$4 \times 4 \times 4$	8.45	8.37	8.83	8.34		12.71	12.16	12.94	11.83	
$5 \times 5 \times 5$	13.12	11.93	13.42	12.32		14.64	14.43	15.01	13.47	
$6 \times 6 \times 6$	17.64	16.41	17.94	16.40		15.85	15.47	16.01	15.03	



(a) Original Data (b) BR, Block Size=5 × 5 × 5, (PSNR: 50.86dB, SSIM: 0.9546) (c) HyZ, Block Size=5 × 5 × 5, REL=1E-3 (PSNR: 87.91dB, SSIM: 0.9980) (d) HyZ, Block Size=6 × 6 × 6, REL=1E-3 (PSNR: 68.78dB, SSIM: 0.9861)

Fig. 6. Visualization of single snapshot (time step=3000) image by BR and HyZ (REL=1E-3) compressors.



(a) Original Data (b) BR, Block Size=4 × 4 × 4, (PSNR: 74.50dB, SSIM: 0.9713) (c) HyZ, Block Size=4 × 4 × 4, (PSNR: 124.13dB, SSIM: 0.9935) (d) HyZ, Block Size=5 × 5 × 5, (PSNR: 117.35dB, SSIM: 0.9902)

Fig. 7. Visualization of stacking image by BR and HyZ (REL=1E-3) compressors.

while the image produced by BR has a few distorted patterns. Besides, in HyZ reconstructed data, the PSNR and SSIM can reach up to 87.91dB and 0.9546, which is higher than those of BR reconstructed data. When we increase the block size of HyZ from $5\times5\times5$ to $6\times6\times6$ (see Figure 6 (d)), though PSNR and SSIM are slightly reduced from 87.91dB to 68.78dB and from 0.9980 to 0.9861, they are still higher than those snapshot reconstructed by BR compressor. In addition, both $5\times5\times5$ and $6\times6\times6$ in HyZ can preserve identical patterns in visualization. Because of error-control, HyZ is better at preserving single snapshot's data quality.

Figure 7 presents the final stacking image generated by BR and HyZ. As seen in Figure 7 (b) and (c), HyZ maintains a better visualization quality than BR under the same block size choice ($4 \times 4 \times 4$). The corresponding PSNR and SSIM can reach up to 123.13dB and 0.9935, respectively. The same conclusion can also be drawn when we increase the block size of HyZ to $5 \times 5 \times 5$ (see Figure 7 (d)). The key reason HyZ outperforms BR is that the reconstructed snapshots without error-control may introduce extra errors. Such extra errors may only slightly affect the visualization of single snapshot, but they will stack gradually along with the 3600 time steps in RTM execution, leading to visible artifacts at last. In all, HyZ is a better choice for preserving final stacking image.

Takeaway 3: HyZ is superior to BR compressor in preserving the visualization quality for both single snapshots and the final stacking image, with higher PSNR and SSIM.

4) Overall Performance of RTM: Figure 8 presents the overall performance of RTM execution by integrating different lossy compressors. The 'Disk' bar means the baseline RTM execution that offloads uncompressed snapshots to disk. To clearly present the execution time, we breakdown each RTM execution into four major stages: (1) Execution Kernel Time:

Fig. 8. Overall performance of RTM

the core RTM algorithm, (2) I/O Write Time: writing/saving snapshot data during forward propagation including the compression time, (3) I/O Read Time: loading snapshot data in the backward propagation phase including the decompression time, (4) Stacking Image Generating Time: generating the final stacking image after backward propagation. According to the figure, running RTM with HyZ can get a top-tier speedup in class (6.29-6.60×) compared with original execution without compression techniques, because of high performance of both SZx and BR. In particular, SZx and BR obtain a speed-up of 7.49× and 6.93× on average, respectively. All these three compressors lead to top-tier performance. From among them, the HyZ is the best choice because it can preserve high data reconstruction quality for both snapshots and stacking image.

In contrast, the executions with other compressors (SZ, ZFP, SZ-Interp, etc.) would take notably longer time to finish. According to the figure, we can clearly observe that their I/O write times are all considerably higher than that of any top-tier performance compressor (HyZ, SZx, BR). This is due to the fact that they all suffer from substantially higher compression time (see Table IV). It is also observed that the RTM with ZFP has a much higher time cost compared with other lossy compressors. The key reason is that ZFP only supports single-thread in the decompression, which causes significantly longer total I/O read time in turn.

Note that we measure the performance with a single node on Bebop supercomputer. In industrial scenario, the RTM execution can be parallelized with multiple nodes. However, its strategy of integrating lossy compression is the same. In each time step of forward propagation, the host node divides the snapshot data into multiple copies and sends them to other nodes for later execution. These operations will then be conducted reversely in backward propagation. As such, each copy of snapshot data will be compressed/decompressed in its corresponding node concurrently without dependency, and hence our conclusion on performance along with other analyses can remain the same.

Takeaway 4: HyZ and BR both can lead to top-tier performances in RTM execution, increasing the performance on average by 6.46× and 6.93×, respectively.

VI. Conclusion and Future Work

In this paper, we develop a hybrid lossy compression method (HyZ), in order to optimize the overall performance in RTM execution while preserving the data quality very well for users. We perform comprehensive experiments based on a total of 10 state-of-the-art lossless and lossy compressors. The key findings and results are summarized as below:

- Our solution HyZ combining BR and SZx has the top-tier performance in both compression and decompression, with satisfactory compression ratio (5+) and high fidelity reconstructed data in both snapshots and stacking image.
- BR exhibits the best compression and decompression speeds, with 19.58GB/s and 20.40GB/s on average.
- HyZ is a better choice for preserving data fidelity for both single snapshot data and final stacking image.
- HyZ efficiently solves the big data issue in RTM execution and improves its overall performance from 6.29× to 6.60× over the original workflow execution performance.

In the future, we plan to further improve the RTM execution performance by exploring more advanced lossy compression techniques in heterogeneous computation environments.

Acknowledgments

This research was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contract DE-AC02-06CH11357, and supported by the National Science Foundation under Grant OAC-2003709, OAC-2104023, OAC-2211538, and OAC-2211539/2247060. We acknowledge the computing resources provided on Bebop (operated by Laboratory Computing Resource Center at Argonne) and on Theta and JLSE (operated by Argonne Leadership Computing Facility).

References

[1] E. Robein, "Eage e-lecture: Reverse time migration: How does it work, when to use it," https://youtu.be/ywdML8ndYeQ, November 15, 2016.
[2] ORNL, "Summit supercomputer," https://www.olcf.ornl.gov/summit/, 2022.
[3] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Data Compression Conference (DCC'06)*, 2006, pp. 133–142.
[4] K. Zhao, S. Di, D. Perez, X. Liang, Z. Chen, and F. Cappello, "Mdz: An efficient error-bounded lossy compressor for molecular dynamics," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 27–40.

[5] 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 *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 1643–1654.

[6] R. Gomez, "Reducing costs with one-pass reverse time migration," https://developer.nvidia.com/blog/reducing-costs-with-one-pass-reverse-time-migration/, 2021.

[7] X. Yu, S. Di, K. Zhao, jiannan Tian, D. Tao, X. Liang, and F. Cappello, "Szx: an ultra-fast error-bounded lossy compressor for scientific datasets," 2022.

[8] H. Fu and R. G. Clapp, "Eliminating the memory bottleneck: an fpga-based solution for 3d reverse time migration," in *Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays, FPGA 2011, Monterey, California, USA, February 27, March 1, 2011*. ACM, 2011, pp. 65–74.

[9] M. Perrone, L. Liu, L. Lu, K. A. Magerlein, C. Kim, I. Fedulova, and A. Semenikhin, "Reducing data movement costs: Scalable seismic imaging on blue gene," in *26th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2012, Shanghai, China, May 21-25, 2012*. IEEE Computer Society, 2012, pp. 320–329.

[10] A. AlOnazi, H. Ltaief, D. E. Keyes, I. Said, and S. Thibault, "Asynchronous task-based execution of the reverse time migration for the oil and gas industry," in *2019 IEEE International Conference on Cluster Computing, CLUSTER 2019, Albuquerque, NM, USA, September 23-26, 2019*. IEEE, 2019, pp. 1–11.

[11] T. Alturkestani, H. Ltaief, and D. E. Keyes, "Maximizing I/O bandwidth for reverse time migration on heterogeneous large-scale systems," in *Euro-Par 2020: Parallel Processing - 26th International Conference on Parallel and Distributed Computing, Warsaw, Poland, August 24-28, 2020, Proceedings*. Springer, 2020, pp. 263–278.

[12] S. W. Son, Z. Chen, W. Hendrix, A. Agrawal, W.-k. Liao, and A. Choudhary, "Data compression for the exascale computing era-survey," *Supercomputing Frontiers and Innovations*, vol. 1, no. 2, pp. 76–88, 2014.

[13] Blosc compressor, http://blosc.org/, 2018, online.

[14] Zstandard, http://facebook.github.io/zstd/, 2018, online.

[15] M. Burtscher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, Jan 2009.

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

[17] 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 *2017 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2017, pp. 1129–1139.

[18] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

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

[20] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola, "Tthresh: Tensor compression for multidimensional visual data," *IEEE Transaction on Visualization and Computer Graphics*, vol. to appear, 2019, arXiv:1806.05952.

[21] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the univariate case," *Computing and Visualization in Science*, vol. 19, no. 5-6, pp. 65–76, 2018.

[22] C. S. Zender, "Bit Grooming: statistically accurate precision-preserving quantization with compression, evaluated in the netCDF Operators (NCO, v4.4.8+)," *Geoscientific Model Development*, vol. 9, no. 9, pp. 3199–3211, Sep. 2016.

[23] X. Delaunay, A. Courtois, and F. Gouillon, "Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netcdf-4 or hdf5 files," *Geoscientific Model Development*, vol. 12, pp. 4099–4113, 09 2019.

[24] T. Liu, J. Wang, Q. Liu, S. Alibhai, T. Lu, and X. He, "High-ratio lossy compression: Exploring the autoencoder to compress scientific data," *IEEE Transactions on Big Data*, pp. 1–1, 2021.

[25] A. Glaws, R. King, and M. Sprague, "Deep learning for in situ data compression of large turbulent flow simulations," *Phys. Rev. Fluids*, vol. 5, p. 114602, Nov 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevFluids.5.114602

[26] 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 *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 294–306.

[27] A. Bartana, D. Kosloff, B. Warnell, C. Connor, J. Codd, D. Kessler, P. Micikevicius, T. Mckercher, P. Wang, and P. Holzhauer, "Gpu implementation of minimal dispersion recursive operators for reverse time migration," *SEG Technical Program Expanded Abstracts*, vol. 34, pp. 4116–4120, 2015, publisher Copyright: © 2015 SEG.; null ; Conference date: 18-10-2011 Through 23-10-2011.

[28] "Reverse time migration (rtm) technology," http://www.seismiccity.com/RTM.html.

[29] SZ2.1, https://github.com/szcompressor/SZ, 2022.

[30] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," 2018.

[31] 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 *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 89–100. [Online]. Available: https://doi.org/10.1145/3369583.3392688

[32] J. Tian, S. Di, X. Yu, C. Rivera, K. Zhao, S. Jin, Y. Feng, X. Liang, D. Tao, and F. Cappello, "Optimizing error-bounded lossy compression for scientific data on gpus," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 283–293.

[33] J. Liu, S. Li, S. Di, X. Liang, K. Zhao, D. Tao, Z. Chen, and F. Cappello, "Improving lossy compression for sz by exploring the best-fit lossless compression techniques," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 2986–2991.

[34] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu *et al.*, "Understanding and modeling lossy compression schemes on HPC scientific data," in *2018 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2018, pp. 348–357.

[35] D. Tao, S. Di, H. Guo, Z. Chen, and F. Cappello, "Z-checker: A framework for assessing lossy compression of scientific data," *The International Journal of High Performance Computing Applications*, vol. 33, no. 2, pp. 285–303, 2019.

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