cuSZ-i: High-Ratio Scientific Lossy Compression on GPUs with Optimized Multi-Level Interpolation

Jinyang Liu [×•], Jiannan Tian [◦•], Shixun Wu [▷•], Sheng Di [◇⋆], Boyuan Zhang [◦], Robert Underwood [◇], Yafan Huang [§], Jiajun Huang [▷], Kai Zhao [¶], Guanpeng Li [§], Dingwen Tao [◦⋆], Zizhong Chen [▷], Franck Cappello [◇]

[×] University of Houston, Houston, TX, USA; jliu217@central.uh.edu
[◦] Indiana University, Bloomington, IN, USA; {jti1,bozhan,ditao}@iu.edu
[▷] University of California, Riverside, CA, USA; {swu264,jhuan380}@ucr.edu, chen@cs.ucr.edu
[◇] Argonne National Laboratory, Lemont, IL, USA; {sdi1,runderwood}@anl.gov, cappello@mcs.anl.gov
[§] University of Iowa, Iowa City, IA, USA; {yafan-huang,guanpeng-li}@uiowa.edu
[¶] Florida State University, Tallahassee, FL, USA; kai.zhao@fsu.edu

***Abstract*—Error-bounded lossy compression is a critical technique for significantly reducing scientific data volumes. Compared to CPU-based compressors, GPU-based compressors exhibit substantially higher throughputs, fitting better for today's HPC applications. However, the critical limitations of existing GPU-based compressors are their low compression ratios and qualities, severely restricting their applicability. To overcome these, we introduce a new GPU-based error-bounded scientific lossy compressor named cuSZ-i, with the following contributions: (1) A novel GPU-optimized interpolation-based prediction method significantly improves the compression ratio and decompression data quality. (2) The Huffman encoding module in cuSZ-i is optimized for better efficiency. (3) cuSZ-i is the first to integrate the NVIDIA Bitcomp-lossless as an additional compression-ratio-enhancing module. Evaluations show that cuSZ-i significantly outperforms other latest GPU-based lossy compressors in compression ratio under the same error bound (hence, the desired quality), showcasing a 476% advantage over the second-best. This leads to cuSZ-i's optimized performance in several real-world use cases.**

I. Introduction

Large-scale scientific applications and advanced experimental instruments produce vast data for post-analysis, creating exascale scientific databases in supercomputing clusters. For instance, Hardware/Hybrid Accelerated Cosmology Code (HACC) [1, 2] may produce petabytes of data over hundreds of snapshots when simulating 1 trillion particles. Those extremely large databases raise tough challenges for the management and utility of data. To this end, data reduction is becoming an effective method to resolve this big data issue. Although traditional methods of lossless data reduction can guarantee zero information loss, they suffer from limited compression ratios. Specifically, lossless compression by roughly 2:1 to 3:1 [3, 4]. Over the past years, error-bounded lossy compressors have strived to address the high compression ratio requirements for scientific data: they get not only very high compression ratios [5, 6, 7, 8] but also perform strict control over the data distortion regarding various modes of user-set error bounds for post-analysis. Notably, CPU-based lossy compressors have assisted scientific simulation to achieve longer simulating time (e.g., CESM-LE [9]), and larger scales (e.g., quantum circuit simulation [10]).

Yet, there are scenarios where even state-of-the-art CPU-based scientific compressors fall short and throttle performance. (1) Current large-scale scientific simulations have offloaded the computationally intensive components to GPU [1, 11, 12, 13] such that GPU, with its limited space, resides large amounts of data temporarily, and moving them to the main memory incurs undesired latency. (2) Advanced instruments have unprecedented peak data acquisition rates, e.g., LCLS-II [14] X-ray imaging can top at 1 TB/s [15] (2023), far beyond what CPU-based compressors can handle. As a reference, QoZ [7] [1] achieves a single-core compressing rate of up to 0.23 GB/s. Thus, high-throughput GPU-based scientific lossy compressors have been developed for in situ data compression, such as cuSZ [16, 17], cuSZx [18], FZ-GPU [19], cuSZp [20], and cuZFP [21], topping at tens to hundreds of gigabytes per second compression throughputs per GPU. Nevertheless, they suffer from a low compression ratio (or quality) at each quality (or ratio) constraint. For instance, although cuSZ has achieved the highest compression ratio among existing GPU-based compressors, it typically only reaches about 10% to 30% (data-dependent) of the CPU-based SZ3 compressor at the same PSNR. Thus, existing works have not fulfilled the requirement of high-ratio-quality error-bounded lossy compression for scientific data on GPU platforms.

In this work, we leverage the strengths of the existing CPU- and GPU-based work to design the new GPU-based error-bounded lossy compressor. Specifically, we expect to achieve sufficient compression throughput, far over typical CPU-based ones, and adequate compression ratios, which are improved over the mentioned GPU-based ones. The challenges are threefold: *first*, many CPU-prototyped predictors carry intrinsic data dependency, inhibiting parallelization; *second*, sophisticated CPU-side tuning techniques that are proven effective can incur high latency even considering customization for GPU; *third*, most high-ratio lossless encoders in existing error-bounded lossy compressors only show poor throughputs on GPU platforms. Responding to these challenges, we introduce a novel, fine-tuned, highly parallelized interpolation-based data predictor. We also introduce a synergetic lossless scheme by coupling the improved Huffman encoding and additional de-redundancy pass and demonstrate it using NVIDIA Bitcomp-lossless. Having overcome the challenges above, we propose a GPU-based

• Jinyang Liu, Jiannan Tian, and Shixun Wu contributed equally to this work.
⋆ Corresponding authors: Sheng Di, Dingwen Tao.

[1] QoZ, as an interpolation-based scientific error-bounded lossy compressor, is the state-of-the-art CPU-SZ variant in terms of rate-distortion.

SC24, November 17–22, 2024, Atlanta, GA, USA
979-8-3503-5291-7/24/$31.00 ©2024 IEEE

error-bounded lossy compressor with both high compression throughput and effectiveness, named cuSZ-i. Overall, cuSZ-i preserves a high compression throughput within the same magnitude as existing GPU-based compressors and profoundly prevails over all others regarding compression ratio and data quality. To our best knowledge, cuSZ-i is the first and the only high-ratio and high-quality GPU-based scientific error-bounded lossy compressor, which fills the major gap between the compression ratio and quality of CPU-based and GPU-based scientific lossy compressors.

We summarize our contributions as follows:

- We develop a GPU-optimized interpolation-based data predictor *G-Interp* with highly parallelized efficient interpolation, which can present excellent data prediction accuracy, thus leading to a high overall compression ratio.
- We design and implement a lightweight interpolation auto-tuning kernel for GPU interpolation to optimize both the performance and compression quality of cuSZ-i.
- We improve the implementation of GPU-based Huffman encoding and import a new lossless module to reduce its encoding redundancy further.
- cuSZ-i substantially improves compression ratio over other state-of-the-art GPU-based scientific lossy compressors up to 476% under the same error bound or PSNR. Meanwhile, it preserves a compression throughput of the same magnitude as other GPU compressors.

The rest of this paper is arranged as follows: § II introduces related works. § III provides the background and motivation for our research. § IV demonstrates the framework of cuSZ-i. The design of cuSZ-i interpolation-based predictor: *G-Interp* is illustrated in detail in § V, and the new design of cuSZ-i lossless encoding modules is proposed in § VI. In § VII, the evaluation results are presented and analyzed. Finally, § VIII concludes our work and discusses future work.

II. Related Work

Scientific data compression has been studied for years to address storage burdens and I/O overheads. The compression techniques are in two classes: lossless and lossy. Compared to lossless compression, lossy compression can provide a much higher compression ratio at the cost of information loss. Moreover, scientific computing practice often requires the error to be quantitatively determined for accurate post-analysis.

Recently, many error-bounded lossy compressors for scientific data have been developed, such as SZ [4, 6, 22], ZFP [23], QoZ [7], SPERR [8], TTHRESH [24], and several AI-assisted works [25, 26, 27]. The usability of these compressors lies in allowing users to strictly control accuracy loss in data reconstruction and post-analysis across various domains. these specialized compressors are also crucial for the treatment of scientific data, one important aspect of which is high-dimensional data interpretation. Previous work [28, 29, 30] emphasize the high-dimensional data continuity, necessitating the preservation of such information. To this end, we are set to design an efficient high-dimensional data compressor.

Moreover, considering the increasing data generation speed of scientific instruments/simulations, and the popularity of heterogenous HPC systems and applications that directly yield data on GPU platforms, GPU-based scientific lossy compressors have been proposed to deliver orders of magnitude higher throughputs than CPU compressors. Specifically, cuZFP [21], the CUDA implementation of transform-based ZFP, leverages discrete orthogonal transform and embedding encoding for compression, allowing the user to specify the desired bit rate. Also, prediction-based GPU compressors have been proposed, including cuSZ, cuSZx [18], FZ-GPU [19], and cuSZp [20]. While sharing the error-boundness in the reconstructed data, these prediction-based compressors can fit into different use scenarios, as detailed below. 1) cuSZ [16, 17] features fully parallelized prediction-quantization with outlier compacted and coarse-grained Huffman coding that encodes multibyte quantization codes using bits inverse to their frequencies; 2) cuSZx [18] features a monolithic design to deliver extremely high throughput at the cost of lower data quality and compression ratio. 3) FZ-GPU [19] alters the compression pipeline by replacing the entire lossless encoding stage with bit-shuffle and dictionary encoding, aiming to deliver higher throughput. 4) cuSZp [20] modifies cuSZ by fusing prediction-quantization and 1D blockwise encoding subroutine into a monolithic GPU kernel, practically achieving high end-to-end throughput. All the high-throughput-oriented GPU-based compressors share the limitation of suboptimal compression ratios. For example, cuSZ, FZ-GPU, and cuSZp all feature Lorenzo prediction, resulting in the ceiled data quality, as described in [4]. Furthermore, note that MGARD-X (CUDA-backend) [31] is another GPU compressor practice; though it achieves comparable rate-distortion to cuSZ, it has been reported to be significantly lower in throughput than other GPU-based compressors [19], making it not fit our use scenarios. Therefore, we exclude it from the discussion.

III. Background and Research Motivation

In this section, we elaborate on our research background. First, we demonstrate cuSZ (the CUDA version of SZ) [16, 17] because its prediction-based nature typically results in a higher compression ratio and makes it the fundamental of cuSZ-i. Next, we specify our research motivation by analyzing the characteristics and limitations of existing GPU-based scientific lossy compressors, then set up our research target.

A. cuSZ *Framework*

cuSZ [16, 17] is a GPU-based scientific error-bounded lossy compression, which shows state-of-the-art compression ratios and distortions among GPU-based error-bounded scientific lossy compressors. Like the CPU SZ framework, cuSZ also has a modular compression framework composed of data prediction, data quantization, and lossless encoding. In these frameworks, predictions are made for each element of the input data, and the predicted values adjusted by quant-codes (denoted by q's) are accurate with respect to the user-specified error bound and replace the original data values. The quant-codes are either encoded when $|q| < R$, an internal parameter, or otherwise

saved as *outliers* when they are too big (i.e., $|q| \geq R$) for efficient encoding. cuSZ features fully parallelized Lorenzo prediction-quantization kernels for compression and decompression. The lossless encoding component features a GPU-based coarse-grained parallelized Huffman encoding, which is the source of the relatively high compression ratios. cuSZ crucially differs from CPU SZ as it does not have a further pass of de-redundancy lossless encoding (e.g., Zstd). This is considered a tradeoff between the throughput and the compression ratio because the sophisticated logic in the redundancy-canceling component can significantly deteriorate the data-processing performance. We refer readers to the cuSZ papers [16, 20] for more details.

B. Research Motivations

In this part, we discuss several key limitations of existing GPU-based scientific lossy compressors such as cuSZ, then propose our design target of cuSZ-i to address those issues.

■ *B.1) Limitation of existing GPU-based lossy compressors in data reconstruction quality* The first limitation of existing GPU-based lossy compressors is that their data reconstruction quality is sub-optimal in various cases. As mentioned before in § II, the Lorenzo data predictor leveraged by cuSZ, cuSZp, and cuSZx have been proven to be inaccurate in many cases [4], so their compression ratios at the desired data qualities are far lower than interpolation-based scientific compressors on CPU such as SZ3 [4, 6]. Therefore, we would like to learn from the advanced predictor design of high-ratio CPU data compressors and propose a GPU-customized high-accuracy data predictor for error-bounded lossy compression on GPUs.

■ *B.2) Limitation of existing GPU-based lossy compressors due to lossless encoding modules* Another critical limitation of existing GPU-based lossy compressors is that their lossless encoding modules produce smaller compression ratios due to the concern of compression throughput. Taking cuSZ as an example, its sole lossless encoding module, Huffman encoding, uses at least 1 bit to map each data element, making the compression bit rate always higher than 1. More importantly, the compressed data of cuSZ still has high redundancy in broad cases. Therefore, we need to address this issue by designing or employing new lossless encoding techniques.

■ *B.3) Design goals of* cuSZ-i Overall, we endeavor to significantly boost the compression quality of existing GPU-based error-bounded lossy compressors by applying a series of optimizations to address the above issues. Our development of cuSZ-i contains and is not limited to (1) leveraging a more effective data prediction scheme for better compression quality, and (2) integrating high-performance lossless modules in the pipeline for better compression ratios. As noted, there are three primary challenges. *First*, the existing CPU-based high-accuracy data predictor, e.g., SZ3 and QoZ interpolators, features many levels of interpolations and will result in heavy data dependency and low throughputs if directly ported to GPUs. *Second*, we need to effectively configure the predictor to boost the compression ratio on GPU. *Third*, we need to efficiently utilize the GPU resource for usable throughputs in real-use scenarios. In the following § IV, we will present how we overcame this challenge by proposing a novel GPU-optimized design of the data predictor to optimize the prediction throughput on GPUs.

IV. cuSZ-i Design Overview

In this section, we present the design of cuSZ-i. First, Fig. 1 presents the pipelines of cuSZ and cuSZ-i. Like cuSZ and many other GPU-based compressors, cuSZ-i is also a prediction-based error-bounded lossy compressor that features data prediction, error quantization, and lossless encoding. In the compression pipeline, cuSZ-i follows the CPU-SZ/cuSZ to approximate input data with a data predictor, and the prediction errors are quantized and recorded. The quantized errors are further encoded and stored to be used in the decompression process. The core innovations in cuSZ-i are as follows.

- We develop a practically GPU-optimized interpolation-based data predictor with efficient parallelization to replace the Lorenzo data predictor [5, 32], leading to much higher data prediction accuracy and better rate-distortion.
- We integrate a lightweight profiling-based auto-tuning mechanism into its kernel to jointly optimize the quality and performance of the interpolation-based data predictor.
- To enhance the throughput when Huffman-encoding quant-codes, the component of building the histogram is optimized.
- To maximally boost the compression ratio with minor speed degradation, we *can* enable another lossless encoding pass (e.g., Bitcomp-lossless) after Huffman encoding in cuSZ-i.

In the remainder of this section, we will detail the innovations we used to design the interpolation-based data predictor practically optimized on GPU.

V. cuSZ-i Interpolation-based Data Predictor

Based on CPU interpolation-based data predictor design [4, 7, 33], cuSZ-i introduces a new interpolation-based prediction scheme, named *G-Interp*, for GPU platforms featuring hardware-software codesign. The new scheme consists of a spline interpolative predictor utilizing anchor points and an auto-tuning strategy. As we will show in most use scenarios, the new interpolative predictor has advantages over the Lorenzo predictor in terms of prediction accuracy and compression ratio.

A. Basic Design Concept of G-Interp

Fig. 2 demonstrates the basic design of *G-Interp* with a 3D example. Though accurate, the existing CPU-based interpolation data predictors [4, 7, 33] are slow (e.g., 0.23 GB/s) and cannot directly be ported to GPU platforms. *G-Interp* becomes a re-design according to the hardware trait to maximize its data-processing parallelism. Compared to the CPU-based interpolation, it exhibits an accuracy-parallelism tradeoff but still preserves substantially higher data prediction accuracy than the baseline Lorenzo. To these ends, we partition the input data into relatively small chunks and eliminate the data dependencies across them. In the illustration, the 3D input data is partitioned into 8^3 chunks (or 16^2 for 2D chunks/512 for 1D chunks). Next, to avoid cross-chunk data dependencies, the interpolation operations need to be confined to limited units of data chunks within a short range. Inspired by QoZ [7], we

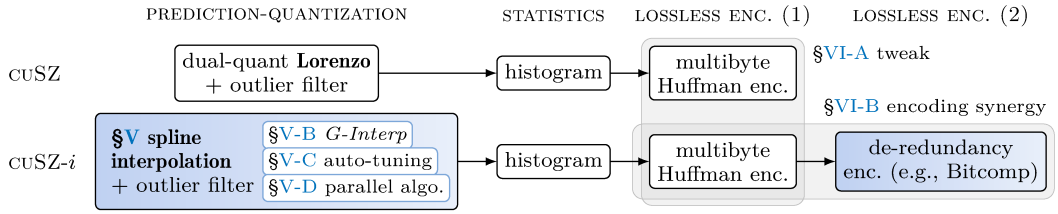


Fig. 1: The compression pipeline of cuSZ-*i* and the comparison between it and cuSZ one. The shading indicates the differentiators of cuSZ-*i* from one design basis, cuSZ.

introduced anchor points in *G-Interp* that are losslessly stored so that all interpolations are bounded in the range of any 2 adjacent anchor points (i.e., within one single data chunk). For example, in the 3D case shown in Fig. 2-1, in each data chunk, one vertex point is assigned as an anchor point (red points in Fig. 2), and 7 other anchor points (on the rest vertices) are borrowed from the surrounding chunks. Afterward, the interpolation within each data chunk is parallelizable and independent of others. In the 3D input data array, approximately 1 of 512 elements becomes anchor points to preserve, whose saving overhead can be further decreased by the additional pass of lossless encoding.

Once the partitioned data chunks and anchor points are set up, as shown in Fig. 2-(3–4), the interpolation-based data predictions in each chunk are performed in parallel. Like in existing works [4, 7], the interpolations are performed level by level, from a large stride to smaller ones (from 4 to 2 and 1 for *G-Interp*), and each chunk is expanded from 1^3 to 2^3, 4^3, lastly 8^3. At each level, interpolations are performed subsequently along each dimension. Moreover, within each chunk, the interpolations on the same level and along the same dimension are also performed in parallel without dependencies. Later, we will feature essential details of this process, including the interpolation splines, error bounds, and the order of interpolation direction.

B. Interpolation Configurations of G-Interp

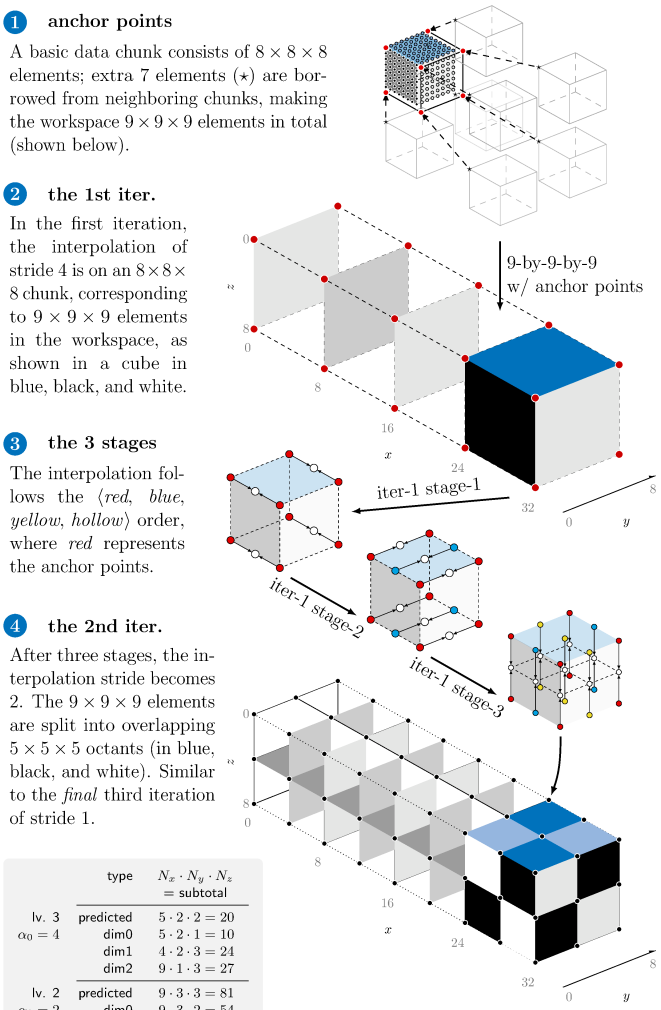
■ *B.1) Interpolation splines* Shown in Fig. 2, for the compression and decompression of a multi-dimensional data grid, *G-Interp* performs the interpolations with 1D splines, executed along different dimensions. For the specific 1D splines, Fig. 3 presents an example with 1D data slices. For each interpolation (to predict x_n with a predicted value p_n), depending on the number of available neighbor points, there are four circumstances:

- 4 neighbors available: The cubic spline interpolation with x_{n-3}, x_{n-1}, x_{n+1}, and x_{n+3} is computed to predict x_n;
- 3 neighbors available: The quadratic spline interpolation with x_{n-3}, x_{n-1}, and x_{n+1} (or x_{n-1}, x_{n+1}, and x_{n+3}) is computed to predict x_n;
- 2 neighbors available: The linear spline interpolation with x_{n-1} and x_{n+1} is computed to predict x_n;
- 1 neighbor available: x_{n-1} serves as a prediction of x_n.

As mentioned, because the predicted values with quant-code adjustment replace the original value, the decompression precisely replays the interpolation-based prediction from the quant-code. Fig. 4 demonstrates the interpolation on a 9×9 2D grid with an anchor stride of 8. The predicted data points with different interpolation splines are marked with their corresponding shapes. We list each spline function in *G-Interp* as follows:

- Linear Spline:

$$p_n = \tfrac{1}{2}x_{n-1} + \tfrac{1}{2}x_{n+1}$$



	type	$N_x \cdot N_y \cdot N_z$ = subtotal
lv. 3	predicted	$5 \cdot 2 \cdot 2 = 20$
$\alpha_0 = 4$	dim0	$5 \cdot 2 \cdot 1 = 10$
	dim1	$4 \cdot 2 \cdot 3 = 24$
	dim2	$9 \cdot 1 \cdot 3 = 27$
lv. 2	predicted	$9 \cdot 3 \cdot 3 = 81$
$\alpha_1 = 2$	dim0	$9 \cdot 3 \cdot 2 = 54$
	dim1	$8 \cdot 3 \cdot 5 = 120$
	dim2	$17 \cdot 2 \cdot 5 = 170$
lv. 1	predicted	$17 \cdot 5 \cdot 5 = 425$
$\alpha_2 = 1$	dim0	$17 \cdot 5 \cdot 4 = 340$
	dim1	$16 \cdot 5 \cdot 9 = 720$
	dim2	$33 \cdot 4 \cdot 9 = 1188$

Fig. 2: *G-Interp*, the parallelized interpolation-based data predictor.

- Quadratic Spline:

$$p_n = -\tfrac{1}{8}x_{n-3} + \tfrac{6}{8}x_{n-1} + \tfrac{3}{8}x_{n+1}$$
$$\text{or } p_n = \tfrac{3}{8}x_{n-1} + \tfrac{6}{8}x_{n+1} - \tfrac{1}{8}x_{n+3}$$

- Cubic Spline (not-a-knot):

$$p_n = -\tfrac{1}{16}x_{n-3} + \tfrac{9}{16}x_{n-1} + \tfrac{9}{16}x_{n+1} - \tfrac{1}{16}x_{n+3}$$

- Cubic Spline (natural):

$$p_n = -\tfrac{3}{40}x_{n-3} + \tfrac{23}{40}x_{n-1} + \tfrac{23}{40}x_{n+1} - \tfrac{3}{40}x_{n+3}$$

We refer the reader to [4] and [33], from which we derived the function from their analyses. It is worth noting that two different

cubic splines serve the same circumstance because each can outperform the others on different datasets. We will use an auto-tuning model of *G-Interp* to select a best-fit cubic spline during each compression task (to be explained in § V-C).

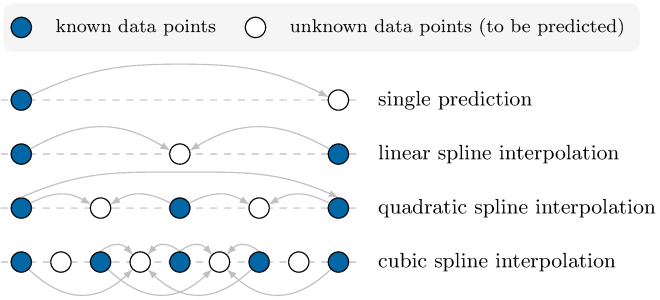


Fig. 3: Interpolation splines.

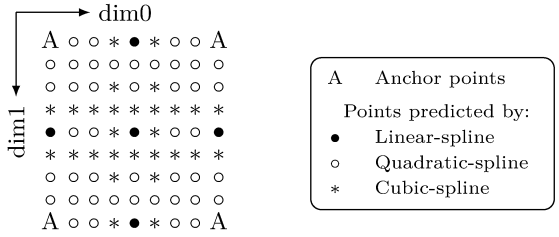


Fig. 4: Illustration of spline interpolations using a 2D slice.

■ *B.2) Level-wise interpolation error bound* The interpolation-based prediction features the synchronous data prediction and error control process. For each input data point, *G-Interp* quantizes the error of its interpolation-based prediction and adjusts the prediction value with an offset of the quantized error. The adjusted prediction will serve as the decompression output and be used to predict subsequent input data points. Therefore, the prediction quality of early interpolations will impact later ones. As verified by [7], for this prediction paradigm, applying lower error bounds on higher interpolation levels (i.e., with larger interpolation strides) can significantly decrease the compression-introduced distortion with little compression ratio degradation. In some cases, it can even improve both compression ratio and distortion. Thus, similarly with [7] and [33], *G-Interp* reduces the error bounds for high-level interpolations according to $\ell = e \cdot (\alpha^{\ell-1})^{-1}$, where $\alpha \geq 1$, e is the global error bound, ℓ is the interpolation level, and e_ℓ is the error bound on level ℓ. α is a parameter of error-bound reduction. In § V-C, we will discuss how the specific α value is determined.

C. Profiling-based Auto-tuning of G-Interp Interpolations

Because the prediction accuracy of interpolation-based data predictors is highly sensitive to their configurations [7], auto-tuning modules are often jointly leveraged with the interpolation-based predictors for preserving the data prediction accuracy [4, 7, 33, 34]. However, existing CPU-based auto-tuning strategies cannot be directly transferred to GPU platforms because they will introduce more computational overhead on GPUs. To this end, we leverage a lightweight profiling-and-auto-tuning kernel for *G-Interp*. This kernel comprises two functionalities: data profiling and interpolation auto-tuning.

■ *C.1) Data-profiling* We profile the input data in 2 steps. First, we compute its value range to acquire both the *absolute* and value-range-based *relative* error bounds (the former divided by the value range equals to the latter). Second, *G-Interp* uniformly samples a small number of data points from the input (e.g., a 4^3 sub-grid for 3D cases). For each sampled point, *G-Interp* performs two instances of cubic spline interpolations along each dimension (e.g., totaling 2×3 tests for 3D cases). The profiling kernel accumulates the interpolation prediction errors separately for each interpolation spline and interpolation dimension.

■ *C.2) Interpolation auto-tuning* With the profiling information, *G-Interp* determines the interpolation configurations by the following strategy. First, *G-Interp* computes the error-bound reduction factor α by a piecewise linear function of the value-range-based relative error bound ϵ:

$$\alpha = A(\epsilon) = \begin{cases} 2 & 10^{-1} \leq \epsilon \\ 1.75 + 0.25 \cdot \frac{\epsilon - 10^{-2}}{10^{-1} - 10^{-2}} & 10^{-2} \leq \epsilon < 10^{-1} \\ 1.5 + 0.25 \cdot \frac{\epsilon - 10^{-3}}{10^{-2} - 10^{-3}} & 10^{-3} \leq \epsilon < 10^{-2} \\ 1.25 + 0.25 \cdot \frac{\epsilon - 10^{-4}}{10^{-3} - 10^{-4}} & 10^{-4} \leq \epsilon < 10^{-3} \\ 1 + 0.25 \cdot \frac{\epsilon - 10^{-5}}{10^{-4} - 10^{-5}} & 10^{-5} \leq \epsilon < 10^{-4} \\ 1 & \epsilon \leq 10^{-5} \end{cases} \quad (1)$$

Dynamically, an optimization of α can be based on both the input data and the value of error bound ϵ. To reduce the computational overhead, we empirically apply this effective calculation instead of a more precise optimization for α. Eq. 1 is established for the reason that from our and [7]'s observations, the optimized value of α is relevant to ϵ and should decrease with the decrease of ϵ.

Lastly, *G-Interp* evaluates the cubic splines and the smoothness of dimensions by profiling interpolation errors. For each dimension, the cubic spline with the lower error will be applied. Moreover, on each level, the interpolation will start from the least smooth dimension (largest in profiled error) and end in the smoothest dimension (smallest in profiled error). This is explained in [4]: the earlier processed dimension will have fewer interpolations performed along it, and vice versa. As a result of performing more interpolations along smoother dimensions, the overall data prediction accuracy can be well optimized.

D. Parallelizing G-Interp on GPU

We practically take advantage of the modern GPU architecture to realize the *G-Interp*'s concepts on the GPU platform. The algorithm-determined data-dependency challenges are as follows: 1) a portion of data loading and storing are non-coalescing, [2] 2) at each interpolation level, dependent traversal stages need to be done, and 3) the finer interpolation depends on the coarser interpolation levels. Our solutions to meet the challenges sufficiently are stated below.

First, we exploit the data-caching capacity paired with the thread numbers for each thread block. A thread block corresponds to four *basic block* of 8^3 elements (featured in § V-A). This builds up a $32_x \times 8_y \times 8_z$ chunk for a coalescing load (Fig. 2-2), minimizing the DRAM transaction. After that,

[2]Memory coalescing is achieved when parallel threads access consecutive global memories to minimize transaction times by loading the same amount of data, allowing for the optimal usage of the global memory bandwidth.

borrowed anchor points are loaded from neighbor blocks before interpolation. Fig. 2-3 shows the first-level interpolation (with a stride of $\alpha_0 = 4$) in the view of a basic block. After the three-stage interpolation in iteration 1 ($\alpha_0 = 4$), the inwardly finer interpolation with stride $\alpha_1 = 2$ continues (Fig. 2-4) and forth for even finer interpolation with $\alpha_2 = 1$.

Next, each GPU thread block interpolates within a $33_x \times 9_y \times 9_z$ data block, which encloses the $32_x \times 8_y \times 8_z$ input data chunk and borrowed anchor points (Fig. 2-2). The interpolations are performed as described in § V-A, but now more neighbor points can be available as 4 basic blocks can share their data. For each interpolation level, Fig. 2-5 enumerates the interpolated points along each dimension (suppose the interpolations are performed from `dim0` with length 9 to `dim2` with length 33). In the level 1 interpolation, the number of interpolated points surpasses the thread number limits of each thread block. Thus, we dynamically assign the number of data points to an active thread.

E. Advantages of G-Interp: A Quantitative Analysis

We present two showcases to clearly show how the cuSZ-i *G-Interp* data predictor advantages over Lorenzo predictor in cuSZ. With better data prediction capability, *G-Interp* generally produces more minor predicting errors. Since those errors will be quantized to a collection of integers, smaller errors will lead to a more concentrated distribution of quantization bins and a higher compression ratio after the Huffman Encoding process.

First, Fig. 5 compares the quantized prediction errors of cuSZ-Lorenzo, *G-Interp*, and CPU-interpolation SZ3 (as a benchmark) when applied to field `pressure` of `Miranda` dataset [35]. In the figure, the nonzero quant-codes (i.e., the prediction error is larger than eb) are colored according to the amplitude of the quant-code. Under the same error bounds, the *G-Interp* design results in much less nonzero quant-codes than cuSZ's Lorenzo predictor and smaller in amplitude, achieving closer outcomes with the CPU-based SZ3. Next, because *G-Interp* exhibits significantly better data prediction accuracy than cuSZ-Lorenzo, its data reconstruction fidelity is also substantially improved over the Lorenzo predictor. In Fig. 6, we propose the comparison between decompression PSNR of *G-Interp* and cuSZ-Lorenzo (under the same error bound, both CPU-based and GPU-based) on 37 snapshots of dataset `RTM` (sample 1 snapshot from every 100 timesteps). We find that *G-Interp* is constantly better than GPU-Lorenzo in terms of PSNR under all error bounds, gaining PSNR improvements of 2.5 to 10 dB. Moreover, attributed to the anchor point design, the PSNR from *G-Interp* even outperforms the CPU version of interpolation (implemented in SZ3 [4]).

VI. Improving cuSZ-i Lossless Modules

In this section, we detail important optimizations on cuSZ-i's lossless encoding module, which crucially boosts the compression ratio of *G-Interp*-predicted quant-codes.

A. Tweaking Existing Huffman Coding

Adapted from the previous practice [16], the procedures of building a histogram of quant-code for the Huffman codebook and conducting Huffman coding based on the codebook remain in the cuSZ-i pipeline. Thanks to the more accurate prediction from *G-Interp*, the histogram contains a more centralized quant-code q distribution. This centralization can be modeled as follows. Recall the outlier-thresholding internal parameter R mentioned in § III-A, for each prediction with respect to the user-specified eb, a much smaller range, denoted by $|q| < r^\bullet < R$, corresponds to an empirical coverage of very few outliers (e.g., $< 1\%$); and at the same eb, *G-Interp* results in a much smaller $r^\bullet$ than Lorenzo. As reported in [36], building a Huffman tree on GPU is worthwhile only when the number of histogram entries (i.e., $r^\bullet$) is large enough. Therefore, we shift the procedure of building the Huffman codebook to CPU, with roughly an end-to-end time of 200 us. This time is *excluded* in the benchmarks in § VII-{C.4, C.5}. Admittedly, this CPU component needs to be further optimized, following approaches such as prebuilding Huffman trees [37]. On the other hand, a much smaller $r_k < r^\bullet$ further signifies this centralization and corresponds to our attempt to allocate small thread-private buffers to cache the count of the center top-k quant-codes. This significantly decreases the transaction between threads' register files and the shared memory buffer. A large k (or r_k) can incur higher register pressure; thus, for graceful degradation, k can fall back to 1, which is still helpful for highly compressible cases.

Last, though highly concentrated, quant-codes in corner cases could still contain modestly deviated numbers. In those cases, we gather them as outliers and losslessly store them with trivial space and time costs using the stream compaction technique.

B. Synergy of Lossless Modules: Huffman + Bitcomp-lossless

Though the Huffman-coding-only practice balances throughput and compression ratio in average cases, its compression is obstructed given that each input quant-code must be represented by at least one bit; At the same time, it can still present redundancy of repeated patterns after encoding (e.g., continuous `0x00` bytes). Nevertheless, sophisticated dictionary-based encoders are either limited in throughput (e.g., GPU-LZ [38]) or compression ratio on GPU. Though previous work proposed several de-redundancy lossless encoding schemes encoding [17, 19, 20] (alternative to Huffman encoding) that can surpass the one-bit-per-element limitation in extremely compressible cases, they still have unsatisfactory compression ratios in many cases, as evidenced in Fig. 7a. Therefore, we propose synergizing Huffman encoding and the subsequent repeated pattern-canceling encoding scheme, which could result in a huge gain in the compression ratio with minimal throughput overhead because this additional scheme only takes inputs of reduced sizes after Huffman encoding. To fit our chart of high ratio and high throughput, we selected Bitcomp-lossless [39] from NVIDIA, a performance-oriented encoder on GPU, after trial and error over a large variety of existing GPU-based bitstream lossless encoders, We are aware of the proprietary nature of NVIDIA's Bitcomp; however, it only serves for demonstration purposes for effectively utilizing the created high compressibility from *G-Interp* on GPU. In § VII, we will present how Bitcomp-lossless helps to greatly remove the redundancy of Huffman-encoded quant-codes generated by cuSZ-i data predictor with negligible computational overhead.

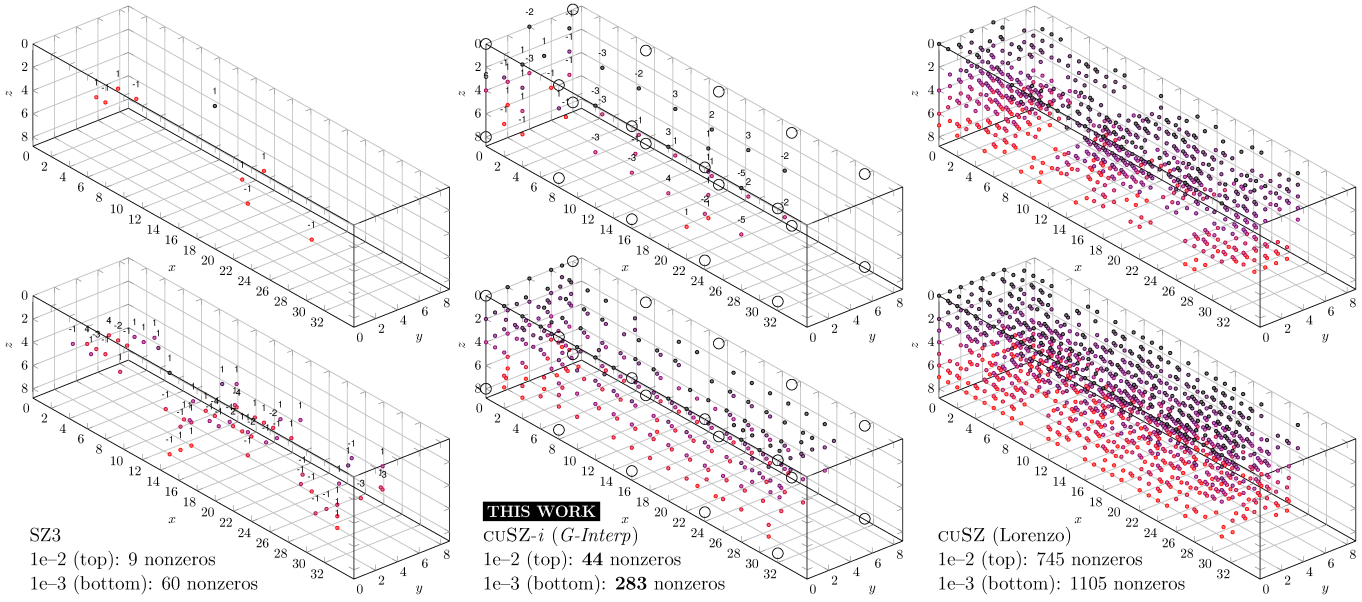


Fig. 5: Showcase: counts of nonzero quant-code among CPU SZ3, GPU *G-Interp*, and GPU Lorenzo. Two relative-to-value-range error bounds are used on `Miranda-Pressure`. Dots in the $33 \times 9 \times 9$ bounding box indicate the nonzeros.

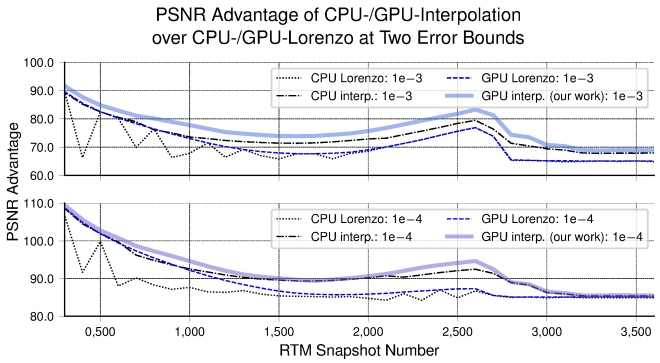


Fig. 6: The PSNR advantage of interpolation over Lorenzo on two error bounds. One snapshot is selected for every 100 among 3700, excluding several ones corresponding to the simulation's initialization phase.

VII. Experimental Evaluation

This section presents our experimental setup and evaluation results. To systematically and convincingly evaluate cuSZ-*i*, experiments with diverse datasets and from various aspects of cuSZ-*i*, together with five other state-of-the-art error-bounded lossy compressors, are presented in this section.

A. Experimental Setup

□ *Platforms.* We evaluate on ❶ NVIDIA A100 GPUs from ALCF-ThetaGPU [40] and Purdue-Anvil [41] for profiling and Globus test and ❷ NVIDIA A40 GPU on ANL-JLSE [42] for profiling. More details are in TABLE I.

□ *Baselines.* We compare our cuSZ-*i* with state-of-the-art GPU-based lossy compressors, namely cuSZ, cuSZp, cuSZx, cuZFP, and FZ-GPU, as baselines.

□ *Test datasets.* The experiments are based on six real-world scientific simulation datasets, detailed in TABLE II. They have been reported as representative of production-level simulations in prior works [4, 16, 19, 33]. Most are open data from Scientific Data Reduction Benchmarks suite [43].

TABLE I: Testbeds for our experiments.

GPU	A100 (40GB)		A40 (48 GB)
testbed	Theta-GPU	Anvil	JLSE
mem. bw.	1555 GB/s		695.8 GB/s
compute (FP32)	19.49 TFLOPS		37.42 TFLOPS
CUDA version	11.4	11.6	11.8
driver version	470.161.03	530.30.02	545.23.06

TABLE II: Information of the datasets in experiments

JHTDB [44]: numerical simulation of turbulence.		
10 files	dim: $512_z \times 512_y \times 512_x$	total: 5 GB
Miranda [35]: hydrodynamics simulation.		
7 files	dim: $256_z \times 384_y \times 384_x$	total: 1 GB
Nyx [11]: cosmological hydrodynamics simulation.		
6 files	dim: $512_z \times 512_y \times 512_x$	total: 3.1 GB
QMCPack [13]: Monte Carlo quantum simulation.		
1 files	dim: $(288 \times 115)_z \times 69_y \times 69_x$	total: 612 MB
RTM [12]: reverse time migration for seismic imaging.		
37 files	dim: $449_z \times 449_y \times 235_x$	total: 6.5 GB
S3D [43]: combustion process simulation.		
11 files	dim: $500_z \times 500_y \times 500_x$	total: 5.1 GB

B. Evaluation Metrics

Our evaluation is based on the following key metrics:

□ *Fixed-error-bound compression ratio (CR).* CR is the original input size divided by the compressed size.

□ *Rate-distortion graph.* This graph shows the compression bit rate and the decompression data PSNR for compressors. The bit rate b is the average of bits in the compressed data for each input element (i.e., $32\times$ the reciprocal of CR).

□ *Fixed-CR visualization.* The visual qualities of the reconstructed data from all the compressors at the same CR.

□ *Throughput.* Compression and decompression throughputs of all the compressors in GB/s.

□ *Data transfer time.* We perform distributed and parallel data transfer tests in the form of lossy compression archives (from all compressors) on multiple supercomputers.

C. Evaluation Results and Analysis

■ *C.1) Compression Ratios.* We compress the datasets under fixed error bounds that are commonly used and list all the results in TABLE III (no results for cuZFP as it does not support absolute error-bounding, and no results for cuSZx on dataset `Nyx` for its runtime errors). TABLE III presents the two-fold achievement of our novel pipeline. (1) In the left half, we present the compression ratios *without* an extra lossless module; even so, cuSZ-*i* outperforms others in 14 of 18 cases, exhibiting 10% to 30% advantages over the second-bests (column 6). (2) In the right half, we examine the full pipeline integrating Bitcomp-lossless. For fairness, we apply Bitcomp-lossless to all other compressors' outputs. *First*, the full pipeline sees a significant gain in compression ratio from the one without Bitcomp-lossless (i.e., comparing column v with 5). *Second*, cuSZ-*i* achieves the highest compression ratios in all 18 cases. cuSZ-*i* expands advantage over the second-best with Bitcomp-lossless enabled (column vi) and tops at 476%. These note that *G-Interp* creates more profound compressibility and is more attuned to the additional pass of lossless encoding than any other compressors. *Last*, it is worth noticing that, for all results in TABLE III, the decompression PSNR of cuSZ-*i* is also much higher than other compressors. If we compare the compression ratio under the same PSNR instead of the same error bound, the improvements from cuSZ-*i* will be amplified (to be detailed in § C.2).

■ *C.2) Compression Rate-Distortion.* Both Interpolation-based data predictor and Bitcomp-lossless contribute to better compression rate-distortion trait. To exhibit the separate contributions, we present the rate-PSNR curves on 6 datasets in 2 parallel series in Fig. 7a: without and with Bitcomp-lossless. When no extra lossless module is appended, cuSZ-*i* has already achieved the best rate-distortion, attributed to the effective *G-Interp* data predictor. On dataset `JHTDB` under a PSNR of 70 dB, or on dataset `QMCPACK` under a PSNR of 80 dB, cuSZ-*i* shows 60% to 80% advantages in compression ratio over the second-best cuZFP/cuSZ. In the with-Bitcomp results, cuSZ-*i* has far better compression ratios than any other, achieving roughly 100% to 500% advantages under the same PSNRs in low-bit-rate cases. In high-bit-rate cases, it also promotes considerable reduction rates. The great extent of compression boost by cuSZ-*i* originates from its high adaptability to Bitcomp-lossless since its *G-Interp* results in highly compressible error quant-codes, which is visualized as the fixed-PSNR bit rate change in auxiliary Fig. 7b. As a reference baseline in Fig. 7a, we also include the rate-distortion of QoZ [7], which reflects the latest interpolation-based art on the CPU platform and one design basis of cuSZ-*i*. We conclude that 1) CPU-based QoZ still features a better compression ratio than cuSZ-*i* due to larger interpolation blocks and more effective lossless modules, but cuSZ-*i* features far better throughputs; 2) nevertheless, cuSZ-*i* (with Bitcomp-lossless) has, for the first time, approximated the best-in-class CPU interpolation-based compressor (i.e., QoZ) in rate-distortion trait.

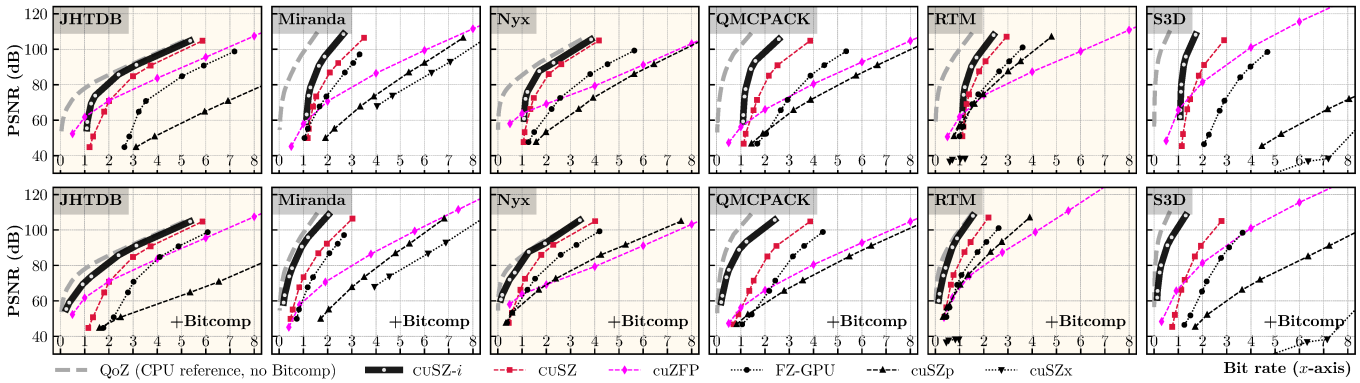
TABLE III: Compression ratios without (columns 1 to 6) and with Bitcomp-lossless (cols. i to vi) at error bounds `1e-2`, `1e-3`, and `1e-4`. The best CRs are boldface, and the second-best are underlined.

Dataset	epsilon	cuSZ	cuSZp	cuSZx	FZ-GPU	cuSZ-*i*	Advant.%	cuSZ	cuSZp	cuSZx	FZ-GPU	cuSZ-*i*	Advant.%
JHTDB	1e−2	26.6	10.3	3.0	12.1	**29.3**	10.2	27.8	19.9	3.1	18.0	**132.0**	374.8
	1e−3	17.7	5.4	2.5	9.9	**25.2**	42.4	17.7	6.0	2.5	11.5	**34.8**	96.6
	1e−4	10.7	3.5	1.8	6.4	**13.3**	24.3	10.7	3.6	1.8	7.8	**13.3**	24.3
Miranda	1e−2	27.1	16.8	7.9	**30.6**	28.5	−6.9	67.4	18.7	8.1	43.9	**174.0**	158.2
	1e−3	22.9	9.6	5.1	19.2	**26.3**	14.8	38.5	10.7	5.2	27.1	**77.2**	100.5
	1e−4	15.3	6.0	3.6	11.8	**19.5**	27.5	19.8	6.7	3.7	15.4	**34.3**	73.2
Nyx	1e−2	**30.2**	20.3	N/A	25.3	29.6	−2.0	71.6	95.9	N/A	84.5	**256.0**	166.9
	1e−3	23.9	9.6	N/A	14.4	**28.0**	17.2	34.4	19.0	N/A	26.2	**66.1**	92.2
	1e−4	15.3	5.7	N/A	8.4	**18.7**	22.2	17.9	7.5	N/A	12.3	**25.1**	40.2
Qmcpack	1e−2	28.6	22.2	3.3	19.0	**29.3**	2.4	46.0	38.7	3.3	30.3	**168.0**	265.2
	1e−3	20.9	10.1	2.5	12.1	**27.7**	32.5	23.7	11.5	2.5	14.7	**78.7**	232.1
	1e−4	14.8	5.6	1.9	8.3	**22.6**	52.7	15.3	5.8	1.9	10.2	**34.6**	126.1
RTM	1e−2	28.7	41.6	**53.7**	32.0	28.8	−46.4	84.1	100.5	70.4	69.7	**234.0**	132.8
	1e−3	24.7	19.8	**30.7**	20.9	27.4	−10.7	50.2	31.1	38.8	35.5	**96.2**	91.6
	1e−4	17.7	10.7	17.4	12.1	**21.5**	21.5	26.6	14.0	21.4	18.4	**45.4**	70.7
S3D	1e−2	28.0	7.2	19.5	15.5	**29.5**	5.4	42.5	18.9	19.9	25.6	**245.0**	476.5
	1e−3	23.3	4.5	9.3	11.8	**28.8**	23.6	28.3	8.8	9.5	16.1	**137.0**	384.1
	1e−4	17.3	3.1	5.0	9.0	**26.0**	50.3	19.0	5.0	5.1	11.6	**58.2**	206.3
		without Bitcomp-lossless						with Bitcomp-lossless					
col. no.		1	2	3	4	5	6	i	ii	iii	iv	v	vi

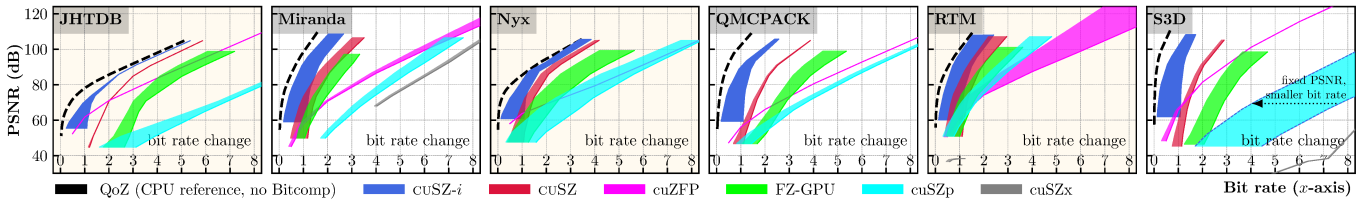
■ *C.3) Case Study of Decompression Visualization.* To further verify the high compression quality of cuSZ-*i*, we visualize certain data decompression snapshots from the datasets. Fig. 8 shows the original and decompression visualization of two data snapshots by six compressors, together with a note of the compression ratios, error bounds, and PSNRs. For each set of visualizations on the same snapshot, we align the compression ratios to the same value (e.g., ∼ 27 for `JHTDB`).

Under a fixed compression ratio, cuSZ-*i* has the best and closest-to-the-original visualization quality from the decompressed data in all cases. In contrast, all other compressors have exhibited severe visualization artifacts. On the visualized decompressions of `JHTDB`, cuSZ-*i* achieved a PSNR of 70.2 dB, which is 8 dB better than the second-best cuZFP. On data snapshot `CO` of `S3D`, under the same compression ratio, around 80 dB. The decompression PSNR of cuSZ-*i* gets a value of 81.3 dB, notably outperforming all other existing compressors (the second-best is only 37.8 dB).

■ *C.4) Compression Throughputs.* After discussing the compression ratio and quality of the compressors, we would like to address the concerns regarding the throughputs. We profiled the compression throughputs of the GPU-based lossy compressors by measuring the kernel execution time with NVIDIA Nsight System on two NVIDIA GPUs: A100 on ThetaGPU and A40 on JLSE. Fig. 9a presents the compression and decompression throughputs of all 6 GPU-based compressors under two different error bounds, `1e-2` and `1e-3`, on the A100 on ThetaGPU. Specifically, cuSZ-*i* indicates the cuSZ-*i* pipeline without Bitcomp-lossless, and cuSZ-*i* w/ Bitcomp indicates the proposed full pipeline of higher compression ratios. We observe that adding



(a) (Bit rate)-PSNR graphs from compression results without (top row) and with Bitcomp-lossless (bottom row) encoding pass.



(b) (Bit rate)-PSNR *leftward change* due to the fixed PSNR and the smaller bit rate from the extra pass of lossless encoding (Bitcomp-lossless).

Fig. 7: Compression (bit rate)-PSNR graphs on six datasets. A curve toward *upperleft* indicates *both* advantageous quality and compression ratios.

Bitcomp-lossless brings negligible overhead to compression throughputs. Also, cuSZ-*i*'s throughput is at the same magnitude as cuSZ and cuSZp. Specifically, it has approximately 60% of cuSZ's compression throughput and 80% to 90% of cuSZ's decompression throughput. Moreover, Fig. 9b presents profiling results on A40 on JLSE. Here, cuSZ-*i* performs closer to cuSZ, reaching 70% to 80% of cuSZ's compression throughput, and nearly the same as cuSZ's decompression throughput.

Due to the intrinsic data dependency and more sophisticated computational operations in spline interpolation kernels, interpolation-based cuSZ-*i* is inevitably slower than Lorenzo-based cuSZ and its derivatives. However, the kernel throughput of cuSZ-*i* is still on the same magnitude as cuSZ, and it can be further improved to better serve as a component for real-time processing. Moreover, although cuSZx, cuZFP, and FZ-GPU have relatively higher throughput than cuSZ-*i*, they present much lower compression ratios than cuSZ-*i*'s, inadequate to many use cases in which data storage and transfer costs need to be minimized. Next, we will see how cuSZ-*i* can be advantageous in real-use tasks that are even sensitive to compression throughputs, and when considering the achieved data decompression quality and the compression ratio simultaneously, cuSZ-*i* have established the Pareto front in scenarios of transferring data over the bandwidth-limited channels.

■ *C.5) Case Study: Distributed Lossy Data Transmission.* In this part, we conduct a practical case study of distributed lossy data transmission as discussed in [45]. Initially, a distributed scientific database is deployed across several supercomputers. Rapid data transfer and access between two distributed machines within this database system are desired. Rather than transfer the original exascale data due to an impractical amount of time, the database system can transfer the significantly smaller compressed data. To this end, an error-bounded lossy compressor operates on the source and destination machines. The overall time required for this process is the sum of local data I/O time, the data compression and decompression time, and the time for transferring the compressed data between the machines. Since the data transfer between remote machines will be performed under relatively low bandwidths, a data compressor with both high ratios and high performance (therefore, the GPU-based is preferred) is required for an optimized data transfer efficiency.

Fig. 10 shows the data-transfer cost in time and decompression data PSNR for this task. Specifically, the source and destination machines are ALCF Theta-GPU and Purdue RCAC Anvil, respectively. Since the local I/O time is irrelevant to this scenario, we compute the overall time only from data compression/decompression time and the data-transfer time of distributed compressed data. Managed by Globus [46], the data-transfer bandwidth between the two machines is approximately 1 GB/s. With the full pipeline (including Bitcomp-lossless) fairly applied to all compressors to further reduce data-transfer costs, cuSZ-*i* has the best-in-class time cost on all high-quality data-transfer cases (PSNR $\geq$ 70 dB), as indicated in Fig. 10. cuSZ-*i* is also very competitive on specific data transfer tasks with low-quality data. For dataset `QMCPACK`, cuSZ-*i* can reduce around 30% of the time cost of the second-best cuSZ at a PSNR of 90 dB. For dataset `S3D`, it can reduce around 40% of the time cost of the second-best cuZFP at a PSNR of 80 dB. Overall, it is evident that the high compression ratio and quality of cuSZ-*i* have gracefully compensated for its speed degradation, implying significance in a great variety of real-world use cases.

VIII. Conclusion and Future Work

In this work, we propose cuSZ-*i*, an efficient GPU-based scientific error-bounded lossy compressor, which exhibits the best compression ratio and quality compared with other related

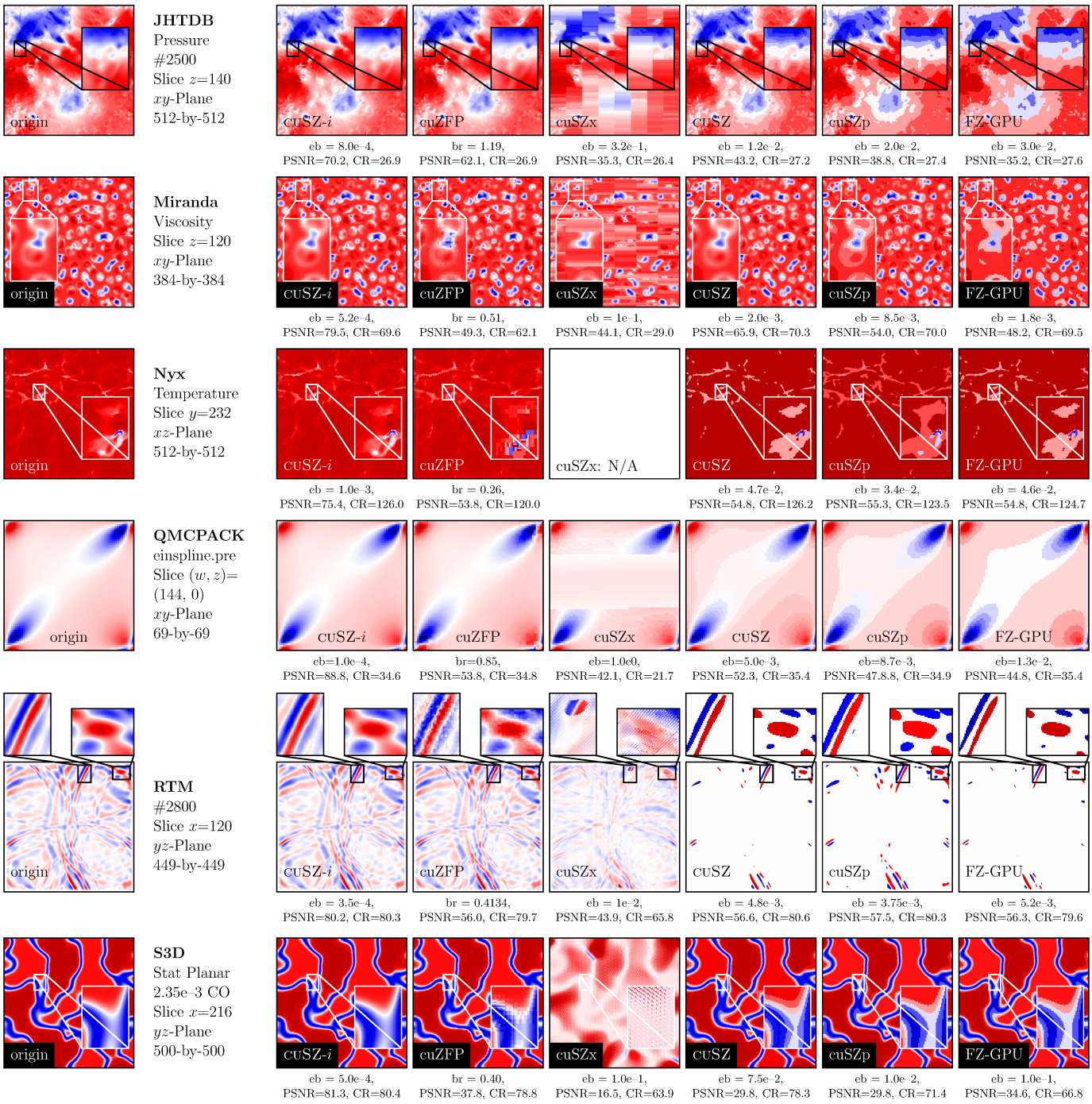
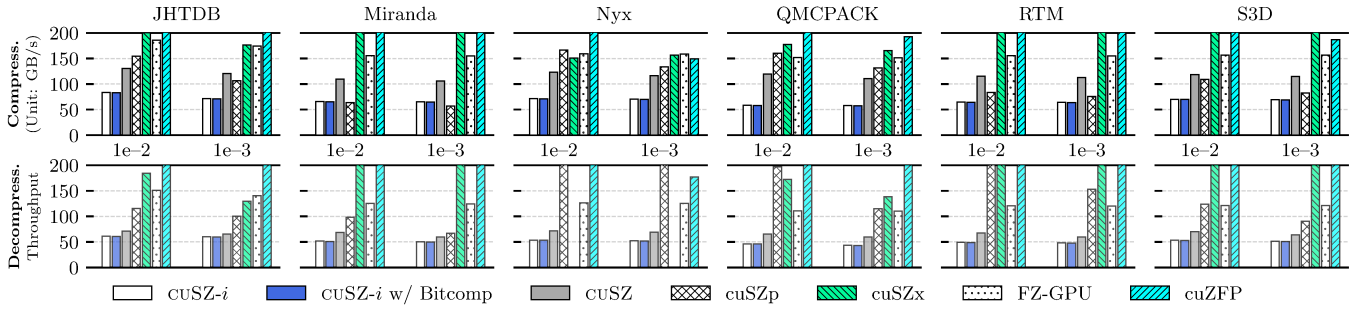
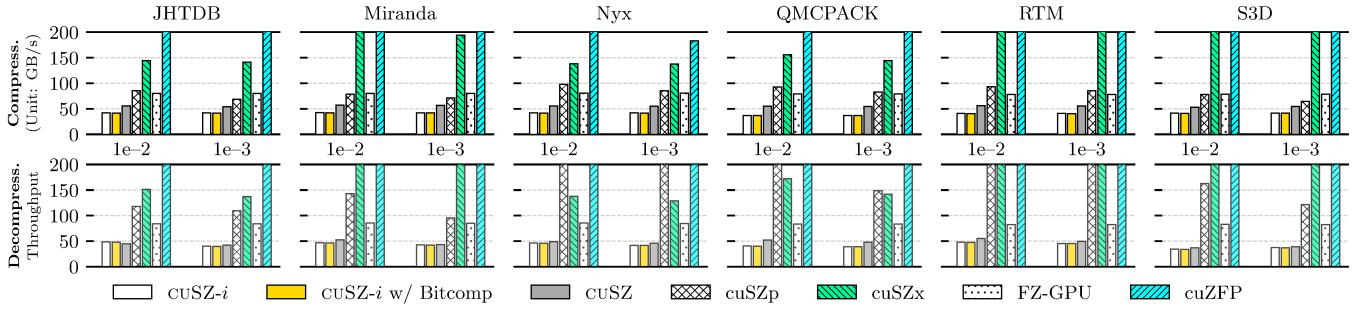


Fig. 8: Showcases of visualized decompressed data, with key subregions zoomed in. For each snapshot, we align the (with-Bitcomp) compression ratio for every compressor with a fixed values. In addition, cuSZ, FZ-GPU, and cuSZp share the same Lorenzo predictor, exhibiting simiar data visualizations. Hence, they are grouped as the last three.



(a) Compression and decompression throughputs on NVIDIA A100.



(b) Compression and decompression throughputs on NVIDIA A40.

Fig. 9: Compression and decompression throughputs on NVIDIA A100 (top row) and A40 (bottom row) for cuSZ-*i*, cuSZ-*i* with Bitcomp-lossless, cuSZ, cuZFP, cuSZp, cuSZx, and FZ-GPU. cuZFP's throughput corresponds to a similar average PSNR with cuSZ-*i*.

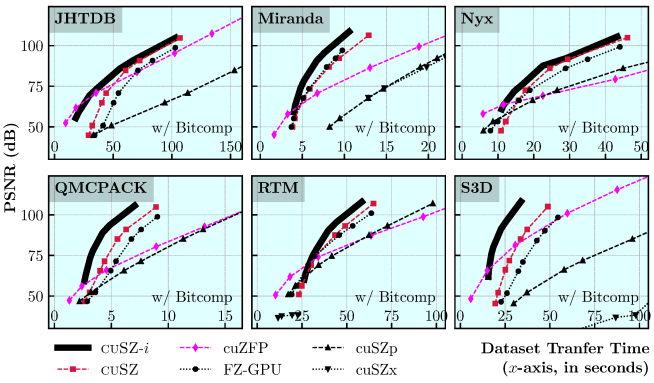


Fig. 10: (Transfer time)-PSNR graphs of compressors on six datasets. A curve toward *upperleft* indicates the advantageous performance in *both* better quality and shorter transfer time.

works. With a highly parallelized interpolation-based data predictor and a lightweight auto-tuning mechanism, cuSZ-*i* achieves state-of-the-art compression quality under the same error bounds or compression ratios rooted in its high data prediction accuracy. With an additional lossless module (NVIDIA Bitcomp), cuSZ-*i* further outperforms other GPU-based scientific lossy compressors in compression ratios by a significant advantage, approaching CPU-based compressors with far higher throughputs than CPU-based ones. In the case studies, cuSZ-*i* presents excellent decompression data visualization that prevails over existing state-of-the-art GPU-based lossy compressors and reduces the distributed lossy data transmission time to a considerable extent for distributed scientific databases.

cuSZ-*i* has a few limitations, e.g., its compression speeds are slower than other GPU-based compressors, its interpolation-based prediction still has lower accuracy than the CPU-based interpolators, and its lossless module involves CPU and is partially dependent on the NVIDIA GPU ecosystem. In the future, we will endeavor to address those limitations, working on improving its speeds, prediction accuracy, and compatibility with more GPU architectures such as AMD and Intel GPUs.

Acknowledgments

This research was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR), under contracts DE-AC02-06CH11357. This work was also supported by the National Science Foundation (Grant Nos. 2003709, 2303064, 2104023, 2247080, 2247060, 2312673, 2311875, and 2311876). We also acknowledge the computing resources provided by Argonne Leadership Computing Facility (ALCF) and Advanced Cyberinfrastructure Coordination Ecosystem—Purdue Anvil through Services & Support (ACCESS).

References

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

[2] S. C. V. Vishwanath and K. Harms, *Parallel I/O on Mira*, https://www.alcf.anl.gov/files/Parallel_IO_on_Mira_0.pdf, Online, 2019.

[3] 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.

[4] 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)*, IEEE, 2021, pp. 1643–1654.

[5] 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 *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA: IEEE, 2018, pp. 438–447.

[6] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao, *et al.*, "SZ3: A modular framework for composing prediction-based error-bounded lossy compressors," *IEEE Transactions on Big Data*, 2022.

[7] 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 *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, IEEE Computer Society, 2022, pp. 892–906.

[8] S. Li, P. Lindstrom, and J. Clyne, "Lossy scientific data compression with sperr," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2023, pp. 1007–1017.

[9] F. Cappello, S. Di, S. Li, X. Liang, A. M. Gok, D. Tao, C. H. Yoon, X.-C. Wu, Y. Alexeev, and F. T. Chong, "Use cases of lossy compression for floating-point data in scientific data sets," *The International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1201–1220, 2019.

[10] X.-C. Wu, S. Di, E. M. Dasgupta, F. Cappello, H. Finkel, Y. Alexeev, and F. T. Chong, "Full-state quantum circuit simulation by using data compression," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, Denver, Colorado: Association for Computing Machinery, 2019, ISBN: 9781450362290. DOI: 10.1145/3295500.3356155.

[11] Nyx simulation, https://amrex-astro.github.io/Nyx/, Online.

[12] S. Kayum *et al.*, "GeoDRIVE – a high performance computing flexible platform for seismic applications," *First Break*, vol. 38, no. 2, pp. 97–100, 2020.

[13] QMCPACK: many-body ab initio Quantum Monte Carlo code, http://vis.computer.org/vis2004contest/data.html, Online, 2019.

[14] https://lcls.slac.stanford.edu/lasers/lcls-ii, Online.

[15] R. Underwood, C. Yoon, A. Gok, S. Di, and F. Cappello, "ROIBIN-SZ: Fast and science-preserving compression for serial crystallography," *Synchrotron Radiation News*, vol. 36, no. 4, pp. 17–22, 2023.

[16] J. Tian, S. Di, K. Zhao, C. Rivera, M. H. Fulp, R. Underwood, S. Jin, X. Liang, J. Calhoun, D. Tao, *et al.*, "CUSZ: An efficient gpu-based error-bounded lossy compression framework for scientific data," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 3–15.

[17] 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)*, Los Alamitos, CA, USA: IEEE Computer Society, Sep. 2021, pp. 283–293. DOI: 10.1109/Cluster48925.2021.00047.

[18] X. Yu, S. Di, K. Zhao, D. Tao, X. Liang, F. Cappello, *et al.*, "SZx: An ultra-fast error-bounded lossy compressor for scientific datasets," *arXiv preprint arXiv:2201.13020*, 2022.

[19] B. Zhang, J. Tian, S. Di, X. Yu, Y. Feng, X. Liang, D. Tao, and F. Cappello, "FZ-GPU: A fast and high-ratio lossy compressor for scientific computing applications on gpus," *arXiv preprint arXiv:2304.12557*, 2023.

[20] Y. Huang, S. Di, X. Yu, G. Li, and F. Cappello, "cuSZp: An ultra-fast gpu error-bounded lossy compression framework with optimized end-to-end performance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–13.

[21] cuZFP, https://github.com/LLNL/zfp/tree/develop/src/cuda_zfp, Online, 2019.

[22] X. Liang, S. Di, S. Li, D. Tao, B. Nicolae, Z. Chen, and F. Cappello, "Significantly improving lossy compression quality based on an optimized hybrid prediction model," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19, Denver, Colorado: Association for Computing Machinery, 2019, ISBN: 9781450362290. DOI: 10.1145/3295500.3356193. [Online]. Available: https://doi.org/10.1145/3295500.3356193.

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

[24] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola, "TTHRESH: Tensor compression for multidimensional visual data," *IEEE transactions on visualization and computer graphics*, vol. 26, no. 9, pp. 2891–2903, 2019.

[25] 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)*, IEEE, 2021, pp. 294–306.

[26] J. Liu, S. Di, S. Jin, K. Zhao, X. Liang, Z. Chen, and F. Cappello, "Scientific error-bounded lossy compression with super-resolution neural networks," in *2023 IEEE International Conference on Big Data (BigData)*, Los Alamitos, CA, USA: IEEE Computer Society, 2023, pp. 229–236. DOI: 10.1109/BigData59044.2023.10386682. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/BigData59044.2023.10386682.

[27] J. Han and C. Wang, "Coordnet: Data generation and visualization generation for time-varying volumes via a coordinate-based neural network," *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[28] D. Wang, J. Pulido, P. Grosset, S. Jin, J. Tian, J. Ahrens, and D. Tao, "TAC: Optimizing error-bounded lossy compression for three-dimensional adaptive mesh refinement simulations," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '22, Minneapolis, MN, USA: Association for Computing Machinery, 2022, 135–147, ISBN: 9781450391993. DOI: 10.1145/3502181.3531458. [Online]. Available: https://doi.org/10.1145/3502181.3531458.

[29] D. Wang, J. Pulido, P. Grosset, J. Tian, S. Jin, H. Tang, J. Sexton, S. Di, K. Zhao, B. Fang, Z. Lukić, F. Cappello, J. Ahrens, and D. Tao, "AMRIC: A novel in situ lossy compression framework for efficient i/o in adaptive mesh refinement applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC

'23, Denver, CO, USA: Association for Computing Machinery, 2023, ISBN: 9798400701092. DOI: 10.1145/3581784.3613212. [Online]. Available: https://doi.org/10.1145/3581784.3613212.

[30] D. Wang, J. Pulido, P. Grosset, S. Jin, J. Tian, K. Zhao, J. Ahrens, and D. Tao, "TAC+: Optimizing error-bounded lossy compression for 3d amr simulations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 3, pp. 421–438, 2024. DOI: 10.1109/TPDS.2023.3339474.

[31] *MGARD-X: A portable implementation of the MGARD lossy compressor supporting various types of GPUs and CPUs*, https://github.com/CODARcode/MGARD/blob/master/doc/MGARD-X.md.

[32] 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*, Orlando, FL, USA: IEEE, 2017, pp. 1129–1139.

[33] J. Liu, S. Di, K. Zhao, X. Liang, S. Jin, Z. Jian, J. Huang, S. Wu, Z. Chen, and F. Cappello, "High-performance effective scientific error-bounded lossy compression with auto-tuned multi-component interpolation," *arXiv preprint arXiv:2311.12133*, 2023.

[34] J. Liu, S. Di, K. Zhao, X. Liang, Z. Chen, and F. Cappello, "FAZ: A flexible auto-tuned modular error-bounded compression framework for scientific data," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 1–13.

[35] Miranda Radiation Hydrodynamics Data, https://wci.llnl.gov/simulation/computer-codes/miranda, Online, 2019.

[36] J. Tian, C. Rivera, S. Di, J. Chen, X. Liang, D. Tao, and F. Cappello, "Revisiting Huffman Coding: Toward extreme performance on modern gpu architectures," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, May 17-21, 2021*, IEEE, 2021, pp. 881–891.

[37] M. Shah, X. Yu, S. Di, M. Becchi, and F. Cappello, "Lightweight huffman coding for efficient gpu compression," in *Proceedings of the 37th International Conference on Supercomputing*, ser. ICS '23, Orlando, FL, USA: Association for Computing Machinery, 2023, 99–110, ISBN: 9798400700569. DOI: 10.1145/3577193.3593736.

[38] B. Zhang, J. Tian, S. Di, X. Yu, M. Swany, D. Tao, and F. Cappello, "GPULZ: Optimizing lzss lossless compression for multi-byte data on modern gpus," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 348–359.

[39] NVIDIA, https://developer.nvidia.com/nvcomp, Online.

[40] *Theta/ThetaGPU - Argonne Leadership Computing Facility*, https://www.alcf.anl.gov/alcf-resources/theta, 2021.

[41] *Anvil - Purdue RCAC*, https://www.rcac.purdue.edu/anvil.

[42] *Joint laboratory for system evaluation – evaluating future high-performance computing platforms*, https://www.jlse.anl.gov/, (Accessed on 03/15/2021).

[43] Scientific Data Reduction Benchmarks, https://sdrbench.github.io/, Online, 2019.

[44] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, "A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence," *Journal of Turbulence*, no. 9, N31, 2008.

[45] Y. Liu, S. Di, K. Chard, I. Foster, and F. Cappello, *Optimizing scientific data transfer on globus with error-bounded lossy compression*, 2023. arXiv: 2307.05416 [cs.DC].

[46] R. Ananthakrishnan, K. Chard, I. Foster, and S. Tuecke, "Globus platform-as-a-service for collaborative science applications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 290–305, 2015.

Appendix: Artifact Description/Artifact Evaluation

Artifact Description/Evaluation (AD/AE)

We considered the integrated reading experience for the readers and merged the description and evaluation sections of the research paper artifacts.

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 We develop a GPU-optimized interpolation-based data predictor *G-Interp* with highly parallelized efficient interpolation, which can present excellent data prediction accuracy.

C_2 We design a lightweight interpolation auto-tuning kernel for GPU interpolation to optimize both the performance and compression quality of cuSZ-i.

C_3 We improve the implementation of GPU-based Huffman encoding and import a new lossless module to reduce its encoding redundancy further.

C_4 cuSZ-i improves compression ratio over other state-of-the-art GPU-based scientific lossy compressors by up to 476% under the same error bound or PSNR. Meanwhile, it preserves a compression throughput of the same magnitude as other GPU compressors.

B. Computational Artifacts

A_1 The NVIDIA-Bitcomp-integrated cuSZ-i code repository. DOI: 10.5281/zenodo.13334684.

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1	§4, §5
A_1	C_2	§5
A_1	C_3	§6
A_1	C_4	§7

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

This is our core artifact and is related to all contributions described in §I.1.

Expected Results

With the provided setup, the artifacts reproduce the experimental results reported in the paper, verifying cuSZ-i's high compression ratio and quality and moderate throughput.

Expected Reproduction Time (in Minutes)

Normally,

- The setup can be completed in 10 minutes.
- The executions should take 1 hour.
- The analysis can take a few minutes.

For compatibility, we *also* provide the `spack` installation instruction to replicate our tested environment:

- Please refer to Installation and Deployment on 2.
- The *alternative* Spack installation/deployment significantly increases the setup time to one hour due to building everything from the source code.
- The time for executions and the analysis remains unchanged.

Artifact Setup (incl. Inputs)

Hardware

We require NVIDIA A100 GPU (40-GB, i.e., the common variant) to cover the essential functionality and, optionally, NVIDIA A40 GPU to cover the throughput scalability.

Software

- We require an up-to-date mainstream Linux distro as the base environment.
  - e.g., CentOS 7 onward, Ubuntu 22.04.
- We require CUDA SDK of version 11.4 onward but lower than 12.5 (i.e., 11.4 to 12.4, inclusively).
  - corresponding to CUDA driver of version 470 onward.
- We require C++17-compliant host compiler.
  - e.g., GCC 9.3 onward.
- We require a modern cmake build system.
  - e.g., 3.18 onward.
- In addition to the basic setup above, to maximize compatibility, we recommend GCC 9.3 and CUDA 12.4.1.

Datasets / Inputs

The data setup will be done in setting up the workplace. The details are listed as follows.

- `JHTDB`
  - Though hosted on `https://turbulence.pha.jhu.edu/` as open data, it requires a token to access the data, which prohibits us from automating the data preprocessing. Thus, we don't include JHTDB datafields for the artifacts.
- `Miranda`, `Nyx`, `QMCPack`, `S3D`
  - hosted on `https://sdrbench.github.io`
  - Note that `Miranda` and `S3D` are originally double-precision and conversion to single-precision is conducted when setting up.
- `RTM` data are from proprietary simulations
  - which are not open to the public.
  - We exclude the use of `RTM` in this artifact.

Setup: Compilers (Normal)

To use `module-load` to set up the toolchain:

```
## Please change the version accordingly.
module load cuda/12.4.1
module load gcc/9.3.0
```

Setup: Compilers (Alternative)

(Skip this if the normal way works.) To use Spack to set up the toolchain:

```
cd $HOME
git clone -c feature.manyFiles=true \
    https://github.com/spack/spack.git
```

```
    ## Now, initialize Spack on terminal start
    ## It is recommended to add the next line to
    ## "$HOME/.bashrc" or "$HOME/.zshrc"
. $HOME/spack/share/spack/setup-env.sh
    ## For other shells, please refer to the
    ## instruction by typing (quotes not included)
    ## "$HOME/spack/bin/spack load"
spack compiler find
spack install gcc@9.3.0
spack install cuda@12.4.1%gcc@9.3.0
```

Setup: Workspace

To set up the compressors and analyzer,

```
## ❶ get the artifacts repo
cd $HOME    ## It can be anywhere.
git clone --recursive \
    https://github.com/jtian0/24_SC_artifacts.git \
    sc24cuszi
cd sc24cuszi

## ❷ setup
## If you use CUDA 11
# source setup-all.sh 11 <WHERE_TO_PUT_DATA_DIRS>
## If you use CUDA 12
source setup-all.sh 12 <WHERE_TO_PUT_DATA_DIRS>

## ‼ reset the workspace without removing data
bash setup-all.sh purge

## ❸ prepare the data
python setup-data-v2.py

## ❹ Python dependencies
## Anaconda is recommended. Please refer to
## https://docs.anaconda.com/anaconda/install/linux/
conda install -y pandas
conda install -y scipy
```

Artifact Execution

Navigate back to the workplace using `cd $WORKSPACE`. Then, run for each dataset. By default, the following commands trigger the execution of all compressors; if a specific compressor is the focus, e.g., cuSZ-i, `--cmp cuSZi` is to be appended to the end of each command. The compressor options are `cuSZ`, `cuSZi`, `FZGPU`, `cuSZp`, `cuzfp`, and `cuSZx`.

```
## ${DATAPATH} is set in setup-all.sh

## Nyx
THIS_DATADIR=SDRBENCH-EXASKY-NYX-512x512x512
python script_data_collection.py  \
    --input ${DATAPATH}/${THIS_DATADIR} \
    --output ${DATAPATH}/${THIS_DATAIDR}_log \
    --dims 512 512 512

## Miranda
THIS_DATADIR=SDRBENCH-Miranda-256x384x384
python script_data_collection.py  \
    --input ${DATAPATH}/${THIS_DATADIR} \
    --output ${DATAPATH}/${THIS_DATAIDR}_log \
    --dims 384 384 256

## QMC
THIS_DATADIR=SDRBENCH-SDRBENCH-QMCPack
python script_data_collection.py  \
    --input ${DATAPATH}/${THIS_DATADIR} \
    --output ${DATAPATH}/${THIS_DATAIDR}_log \
    --dims 69 69 33120

## S3D
THIS_DATRADIR=SDRBENCH-S3D
python script_data_collection.py  \
    --input ${DATAPATH}/${THIS_DATADIR} \
    --output ${DATAPATH}/${THIS_DATAIDR}_log \
    --dims 500 500 500
```

Artifact Analysis (incl. Outputs)

The analysis results are in tabular form (`.csv` files) in `${DATAPATH}/<dataset name>_csv`, which include per-dataset compression ratios/bitrates, compression and decompression throughputs (with or without Bitcomp-Lossless), PSNR, etc. Testers can check the results and compare them with the ones reported in the paper. The used commands are as follows. Also, a single compressor can be specified by appending `--cmp [compressor name]` to the end of a command.

```
## ${DATAPATH} is set in setup-all.sh

## Nyx
THIS_DATADIR=SDRBENCH-EXASKY-NYX-512x512x512
python script_data_analysis.py  \
    --input ${DATAPATH}/${THIS_DATADIR}_log \
    --output ${DATAPATH}/${THIS_DATADIR}_csv \
    --dims 512 512 512

## Miranda
THIS_DATADIR=SDRBENCH-Miranda-256x384x384
python script_data_analysis.py  \
    --input ${DATAPATH}/${THIS_DATADIR}_log \
    --output ${DATAPATH}/${THIS_DATADIR}_csv \
    --dims 384 384 256

## QMC
THIS_DATADIR=SDRBENCH-SDRBENCH-QMCPack
python script_data_analysis.py  \
    --input ${DATAPATH}/${THIS_DATADIR}_log \
    --output ${DATAPATH}/${THIS_DATADIR}_csv \
    --dims 69 69 33120

## S3D
THIS_DATRADIR=SDRBENCH-S3D
python script_data_analysis.py  \
    --input ${DATAPATH}/${THIS_DATADIR}_log \
    --output ${DATAPATH}/${THIS_DATADIR}_csv \
    --dims 500 500 500
```