# Structural-Similarity-Preserving Lossy Data Compression for CPUs and GPUs

Alex Fallin and Martin Burtscher
Texas State University, USA
Emails: {waf13, burtscher}@txstate.edu

***Abstract*—Effective lossy compression is important for large scientific datasets. Existing approaches control the point-wise error but not the visual quality, which is often measured using the Structural Similarity Index (SSIM). This paper introduces the minSSIM metric and SBLC, the first SSIM-bounded compressor. It is fully CPU-GPU compatible and guarantees the user-provided minSSIM bound, which it enforces by strategically tightening the error bound on specific values in the data. It does this entirely during compression, ensuring that decompression is very fast. Using 6 input suites from the SDRBench repository, an RTX 4090 GPU, and a NOA error bound of 0.001, SBLC decompresses at a throughput of 493 GB/s on average. Additionally, at tighter error bounds, it produces higher compression ratios than all but one of the evaluated GPU compressors while yielding the highest average SSIM. SBLC is the only tested compressor that does not produce any regions with low SSIM values, all while also guaranteeing point-wise error bounds.**

***Index Terms*—lossy data compression, guaranteed error bounds, structural similarity index, CPU/GPU parallel**

## I. Introduction

Many important scientific instruments and simulations generate more data than can be feasibly managed in terms of storage capacity and throughput [9, 25]. Data compression is an obvious choice to address this issue. There are two commonly used types of compression: lossless and lossy. Lossless compression compresses the data without introducing any differences between the original data and the decompressed data. Unfortunately, lossless compression is generally not able to yield sufficiently high compression ratios for large-scale scientific workflows. For example, ZFP [14] can losslessly compress the NYX file baryon_density [31] by only a factor of 1.34. In contrast, lossy compression can yield much higher compression ratios, depending on the selected error bound. Using the same NYX input with an absolute error bound of 0.01, ZFP obtains a compression ratio of 4.92. However, this increase comes at a price. As the name suggests, lossy compressors "lose" some information and are unable to perfectly reconstruct the data. To lose this data in a controlled fashion, many lossy compressors employ point-wise error bounding. In the example above, an absolute error bound of 0.01 means that each value cannot be off by more than $\pm 0.01$.

In addition to speed, compression ratio, and error bounding requirements, end users that analyze the decompressed data are often interested in preserving specific properties that might be lost in the compression process. For this reason, many state-of-the-art lossy compressors support preservation of quantities of interest [9]. A key quantity of interest is the Structural Similarity Index Measure (SSIM) [39], which focuses on visual quality. SSIM is an important metric for any compressor used to handle data that is to be visualized, but no existing compressor offers SSIM guarantees on reconstructed data.

Current lossy compressors often produce reasonable overall SSIM values [15, 28]. But because SSIM is a high-level average, there are typically regions in the decompressed output that have much worse SSIM values than the overall average as we show in Section VI. This can be an issue if these regions happen to be in an area that is of interest for visualization.

As a remedy, we introduce the SSIM-Bounded Lossy Compressor (SBLC), which offers strict control of both point-wise error and a new metric we call minSSIM. By controlling minSSIM, we not only guarantee that the high-level average SSIM is above the requested bound but also that there are no local SSIM values below the requested bound. Additionally, SBLC supports CPUs and GPUs while maintaining bit-for-bit parity of the compressed and decompressed data. On the SDRBench inputs [41], it achieves comparable compression ratios to other CPU/GPU compatible compressors at the same error bound while yielding a higher SSIM and similar decompression throughputs. Additionally, by binding the minSSIM, SBLC ensures that all regions in the reconstructed data meet the requested quality.

This paper makes the following main contributions.

- **A new SSIM metric**: We describe minSSIM, a metric to analyze the worst-case SSIM in reconstructed data.
- **A new lossy compressor**: We present a new lossy base compressor that quantizes values into floats rather than integer bins, thus obviating the need for dequantization while still providing high compression ratios on GPUs.
- **An SSIM-preserving lossy compressor**: We introduce SBLC, an algorithm that guarantees the minSSIM to not drop below the requested bound anywhere, which also guarantees the overall SSIM bound.
- **Implementation and optimization**: We provide a detailed description of the implementation, optimization, and parallelization strategies employed to make SBLC efficient on both CPUs and GPUs.
- **Comprehensive evaluation**: We compare SBLC to six state-of-the-art error-bounded compressors in terms of compression/decompression throughput, compression ratio, and reconstructed data quality.

Our C++/OpenMP and CUDA implementations of SBLC are freely available through GitHub [16].

The rest of this paper is organized as follows. Section II provides background. Section III summarizes related work. Section IV explains the SBLC algorithm. Section V describes the evaluation methodology. Section VI presents and discusses the results. Section VII concludes the paper with a summary.

## II. Background

This section describes the point-wise error control that SBLC employs as well as the SSIM metric and its variants.

### A. Point-Wise Absolute Error (ABS)

The point-wise absolute error of a data value is the *difference* between the original value of the data point and its reconstructed value [32]. The absolute error of a value $x$ is defined as $e_{abs} = |x_{original} - x_{reconstructed}|$. Therefore, to guarantee an absolute error bound of $\varepsilon$, each value in the reconstructed file must satisfy $e_{abs} \leq \varepsilon$. In other words, each reconstructed value must be in the following range: $x_{original} - \varepsilon \leq x_{reconstructed} \leq x_{original} + \varepsilon$.

ABS error bounds are useful when the data is homogeneous in magnitude or when the user is not interested in areas where values may be small relative to the error bound, that is, when the user cares mainly about the "big picture".

### B. Point-Wise Normalized Absolute Error (NOA)

The point-wise normalized absolute error is the ABS error normalized by the value range $R = x_{max} - x_{min}$, that is, the range between the largest and the smallest value in the input. The normalized absolute error of a value $x$ is defined as $e_{noa} = |\frac{e_{abs}}{R}|$. To guarantee an error bound of $\varepsilon$, each data value in the reconstructed file must satisfy $e_{noa} \leq \varepsilon$. Therefore, each reconstructed value must be in the following range: $x_{original} - \varepsilon R \leq x_{reconstructed} \leq x_{original} + \varepsilon R$.

NOA error bounds are convenient when the user has multiple datasets at different scales but only wants to specify one absolute error bound for all of them. SBLC supports both ABS and NOA point-wise error bounds.

### C. Quantization

Most of the lossy compressors described below in Section III employ a form of quantization. For example, an ABS quantizer typically uses the supplied error bound $\epsilon$ to quantize the floating-point values into bins. It does this by multiplying the original value by the inverse of two times the error bound, (i.e., $\frac{0.5}{\epsilon}$) and rounding the result to the nearest integer, which yields a bin number. All values within $\pm\varepsilon$ of the center of a bin are thus mapped to the same bin and will be reconstructed to the center value. The resulting sequence of bin numbers is generally much more compressible than the original floats.

### D. SSIM

SSIM is a metric that measures visual quality by comparing a reconstructed image to its original [39]. Its focus on structure and local neighborhood is in contrast to other commonly used lossy metrics like the peak signal-to-noise ratio (PSNR) and the mean squared error (MSE), which are focused mainly on the error itself. SSIM is the average of many local SSIM windows. Our implementation uses the same constants, window size (i.e., 7), and stride (i.e., 2) as the Quick Compression Analysis Toolkit (QCAT) [1], which is widely used to evaluate scientific lossy floating-point compressors.

Each local SSIM is calculated over the original values $x$ and the reconstructed values $y$ using the equation $SSIM = luminance(l) \times contrast(c) \times structure(s)$, where $l(x,y) = \frac{2\mu_x\mu_y+c_1}{\mu_x^2+\mu_y^2+c_1}$, $c(x,y) = \frac{2\sigma_x\sigma_y+c_2}{\sigma_x^2+\sigma_y^2+c_2}$, and $s(x,y) = \frac{\sigma_{xy}+c_3}{\sigma_x\sigma_y+c_3}$. Here, $\mu$ is the average over the values in the window, $\sigma$ is the standard deviation, and $\sigma_{xy}$ is the covariance. The $c$ values are small positive constants. The final SSIM is calculated by iterating across all possible windows given the window size and stride and averaging the resulting local SSIM values. This global SSIM value is always between -1 and 1. Values near 1 indicate strong similarity to the original data. As an example, adding a small gaussian blur would lower the SSIM value, but not by much. Values near 0 indicate no structural commonality. For example, comparing random noise to an original input yields SSIMs near 0. SSIM values below zero are uncommon in the image domain but do occur when working directly with float values. A SSIM below zero indicates a negative correlation between the original and the reconstructed data.

There are several SSIM variants of note. Multi-scale SSIM (MS-SSIM) [38] is similar to traditional SSIM but performs multiple sub-sampling steps for each window to evaluate the SSIM at different scales. This yields a more robust metric but is significantly more expensive to compute. Data SSIM (DSSIM) [6] was developed for direct application to floating-point data. It operates on a normalized version of the floating-point data rather than on the raw data. This helps with the constant selection and eliminates the possibility for negative structure components. DSSIM quantizes the data into 256 bins to align with a linear color mapping. These changes yield DSSIM values that are slightly different from traditional SSIM values but are faster to compute due to not having to first generate an image. Similarly and like QCAT, our SSIM computation operates directly on the floating-point data without generating an image and does not perform any quantization or normalization. In other words, SBLC and minSSIM use the traditional version of SSIM as a metric.

## III. Related Work

SSIM is a common metric for comparing lossy compressors [9], but no published codes exist that directly bound SSIM or minSSIM like SBLC does. Hence, this section describes works that target SSIM or visual quality in compression. It also discusses the state-of-the-art error-bounded lossy floating-point compressors with which we compare SBLC.

### A. Machine-Learning-Optimized SSIM

There are many machine-learning (ML) approaches for lossy compression, but they mainly focus on images rather than scientific floating-point data. We briefly discuss these ML approaches below. They do not directly bound the SSIM/MS-SSIM but instead include it in their loss functions, optimizing

for visual quality without offering a strict bound like SBLC. However, due to the focus on image data and the generally slow speed of ML-based compressors for both encoding and decoding, we do not compare to such approaches.

Ballé, Laparra, and Simoncelli [7] introduce learned compression, which approaches the problem as a rate-distortion optimization problem and uses an autoencoder along with an entropy model to compress images without using traditional compression transformations. They report MS-SSIM rates that are consistently higher than those of both JPEG [37] and JPEG2000 [33]. Johnston et al. [24] augment recurrent convolutional neural networks with hidden-state priming, spatially adaptive bit rates, and perceptually-weighted training loss to increase both SSIM and MS-SSIM. Ballé et al. [8] improve on their prior work via the introduction of a hyperprior. This helps improve the accuracy and bitrate by providing additional information to help with variance in the hyperparameters. Minnen, Ballé, and Toderici [30] further expand on existing approaches by combining a hyperprior approach with an autoregressive context model to capture both local and global context in the network architecture. The autoregressor, however, is inherently serial and thus slower than previous approaches.

Cheng et al. [10] improve on entropy modeling with the addition of gaussian mixture likelihoods, which allow for better representations of complex latent distributions. Attention is also added to both the encoder and decoder to focus the bitrate on areas of spatial interest.

Cui et al. [12] demonstrate a model that allows for the latent representation to be scaled via parameters during inference. This allows for the same kind of bitrate control that is seen in non-ML-based lossy compressors such as ZFP without the need for training multiple models for each desired bitrate. He et al. [18] further improve on autogregressive entropy models using unevenly grouped space-channel contexts. This allows for more parallelism than was previously possible for autoregressor models, yielding the highest compression speeds at a given quality of 28 MB/s encoding and 24 MB/s decoding.

Most ML based compressors in this area optimize for (but do not guarantee) SSIM via their loss function during training. They perform well when compared to other compression tools like JPEG, but their inference time leaves much to be desired when compared with a traditional compressor like SBLC.

### *B. Error-bounded Lossy Compressors*

Supporting error bounds is important to preserve data integrity for downstream analysis. The following compressors offer error bounding as a feature of their compression algorithms. Many of these works use SSIM as a metric for comparison but none support SSIM or minSSIM bounding.

Jiao et al. [23] explore the mitigation of compression artifacts that arise from quantization. Their work alleviates such artifacts using linear interpolation at the border between quantization bins. To do this, they first identify the quantization boundaries, determine the distance to those boundaries for each data point, and then interpolate each point relative to the boundaries to compute an offset to be added to smoothen the decompressed data. These offsets are still error controlled, so the interpolation does not introduce error-bound violations. This interpolation, which focuses on visual quality, improves SSIM by up to 108%. The corresponding code is not yet available, so we cannot compare to it. Our approach also targets visual quality, but SBLC focuses on adaptively preserving SSIM rather than smoothening only a specific type of artifact.

There are four main lossy compressors in the SZ family. They all use prediction in their compression pipeline. SZ2 [26] employs Lorenzo prediction [21] and linear regression followed by quantization. SZ3 [28, 40] is an improvement that generally produces higher compression ratios with similar throughput. Both SZ2 and SZ3 utilize entropy coding plus lossless compression after the lossy stage (e.g., Huffman [20] followed by GZIP [13] or ZSTD [11]). SZ2 and SZ3 are CPU-only compressors. cuSZ [34, 35] is a CUDA implementation that is based on a different, more GPU-friendly algorithm. It performs Lorenzo prediction and quantization followed by multi-byte Huffman coding. cuSZp [19] yields high compression ratios and higher decompression throughputs than cuSZ. It splits the data into blocks and then quantizes and predicts the values in all nonzero blocks, which are ultimately compressed by a fixed-length encoder that employs bit-shuffle operations. The lossy stage in SBLC differs from the traditional quantization methods used by the SZ compressors in that it outputs actual floating-point values rather than bin numbers, meaning the SBLC's dequantizer is a no-op and takes no time to execute. We compare to SZ3 and cuSZp in Section VI.

We also compare to ZFP [14, 29], a widely used tranform-based compressor. It is specifically designed for in-memory array compression and supports on-the-fly random-access decompression. ZFP splits the input into blocks, converts each value into an integer, decorrelates the integers, reorders the data, and converts the values from two's-complement to negabinary representation. Then, it groups the bits from most to least significant. Finally, the shuffled bits are losslessly compressed. Our approach has a few commonalities with ZFP (e.g., converting to negabinary format).

MGARD [27] supports compression and decompression across CPUs and GPUs. It uses multigrid hierarchical data refactoring to decompose and recompose the data to a specified accuracy via selective loading based on the hierarchy. Additionally, while MGARD supports the $L_\infty$ norm like many other lossy compressors, it is optimized towards preserving quantities of interest via the $L_2$ norm. For this reason, the performance when using the $L_\infty$ norm is diminished both in throughput and compression ratio, but we must use it to directly compare to the other error-bounded lossy compressors.

PFPL [15] is a high-throughput CPU and GPU compatible lossy compressor. It first quantizes the input and converts the bin values to magnitude-sign representation. For performance, it stores outliers inline rather than separately. It then splits the input into 16kB chunks to allow for fast parallel compression and decompression. Each chunk is compressed using a pipeline of lossless transformations. First, the data is delta encoded [22] and converted to negabinary. Second, it is bit shuffled, similar

to ZFP. Last, the data is compressed using repeated zero-byte elimination. SBLC uses a similar lossless compression pipeline. However, the lossy step is entirely different, which is where the SSIM is guaranteed.

SLEEK [2] is an ultra-fast compressor for GPUs designed to operate at the speed of memory copies. It guarantees the error bound and is CPU/GPU compatible. SLEEK's lossy step is based on a quantization method that guarantees that all values can be quantized within the error bound without the need for storing outliers. The lossless portion of SLEEK is simple by design to provide high speed. It groups the quantized values into 512-byte chunks and analyzes them to determine how many leading zeros can maximally be removed from all words in the chunk without losing data. This works well due to the leading zeros introduced by the quantizer.

Overall, SBLC shares some similarities with the state of the art in its lossless compression pipeline and with the error-bound guarantees it supplies. More recent work explicitly targets visual quality but does so using smoothing at reconstruction time rather than controlling the loss in the compression stage. SSIM, while a common metric of comparison, is not guaranteed by any prior approach. In contrast, SBLC not only guarantees the user-specified SSIM but also offers an even stronger minSSIM guarantee. Moreover, it ensures these guarantees entirely in the encoder and, therefore, incurs no overhead in the decoder.

## IV. Approach

In this section, we describe the minSSIM metric, the SBLC algorithm, and its parallelization and optimizations.

### A. minSSIM

The minSSIM metric is described in Algorithm 1. It is a simple modification of the global SSIM calculation described in Section II. Instead of averaging all of the individual window SSIMs, we record the lowest value. The main motivation behind this modification is to keep track of the worst case.

**Algorithm 1** minSSIM Calculation

1: $minSSIM \leftarrow 1$
2: **for** each window $w$ **do**
3: $\quad win_{ssim} \leftarrow$ localSSIM($w$)
4: $\quad minSSIM \leftarrow \min(minSSIM, win_{ssim})$

### B. The SBLC Algorithm

The overall SBLC algorithm is shown in Figure 1. The lossy quantization of SBLC is described at a high level in Algorithm 2. SBLC takes in both a minSSIM threshold and a traditional error bound. The first step on Line 1 is where the lossy portion of SBLC is performed with the given traditional error bound. As mentioned in Section III, SBLC does not perform binning but instead directly outputs a rounded float value that is within the error bound. To do this, SBLC calculates a threshold value based on the error bound. This threshold is used to determine the number of least significant bits (LSBs) from the mantissa that can be ignored without resulting in an error-bound violation. This original error bound is saved on Line 3. It retains as much compressibility as possible. Additionally, all values are immediately encoded so they can be analyzed in the following main computation loop.

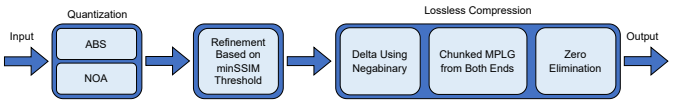



Fig. 1: The high-level SBLC algorithm

**Algorithm 2** The SBLC Lossy Compression Algorithm

1: **for** each point $p$ **do** ▷ parallel loop
2: $\quad p_{round} \leftarrow$ ABS or NOA rounded version of $p_{val}$
3: $\quad p_{orig\_eb} \leftarrow$ error bound corresponding to $p_{round}$
4: $\quad p_{new\_eb} \leftarrow$ error bound corresponding to $p_{round}$
5: $done \leftarrow$ false
6: **while** not $done$ **do**
7: $\quad done \leftarrow$ true
8: $\quad$ **for** each set of non-overlapping windows $wins$ **do**
9: $\quad\quad$ **for** each window $w$ in $wins$ **do** ▷ parallel loop
10: $\quad\quad\quad count\_low \leftarrow 0$;
11: $\quad\quad\quad$ **while** localSSIM($w$) $< minSSIM\_thr$ **do**
12: $\quad\quad\quad\quad done \leftarrow$ false
13: $\quad\quad\quad\quad count\_low \leftarrow count\_low + 1$
14: $\quad\quad\quad\quad$ **for** each point $p$ in $w$ **do**
15: $\quad\quad\quad\quad\quad local\_eb \leftarrow p_{orig\_eb}/2^{count\_low}$
16: $\quad\quad\quad\quad\quad p_{new\_eb} \leftarrow \min(p_{new\_eb}, local\_eb)$
17: $\quad\quad\quad\quad\quad p_{round} \leftarrow$ rounded using $p_{new\_eb}$

The main loop starts on Line 5 of Algorithm 2. This is where the error bound of each value is refined, if necessary, to also preserve the requested minSSIM bound. The loop iterates over all local SSIM windows in the input file. For each of these windows, the local SSIM is calculated. If it is below the minSSIM threshold, every value within that window is refined. This is done by halving the error bound. Instead of simply reducing the error bound for each window element by half every time the minSSIM is violated, we calculate a new floor error bound using the original bound and a counter for that window and apply this floor to the existing error bound. This is beneficial due to the overlapping nature of the local SSIM windows. If we were to simply halve the error bounds of all points in the window, overlapping windows might result in tighter error bounds than necessary. Conversely, if we were to use only the counter and the original error bound instead of the new floor, then a window may overwrite a lower error bound that a preceding window determined was required. Note that this iterative refinement is not limited to minSSIM. It also works for other local quality metrics (e.g., Local Normalized Cross-Correlation and Gradient Magnitude Similarity Deviation) by changing the logic that determines if a window must be refined.

After these new error bounds have been established, the values in the window are encoded again, and the local SSIM is recalculated and compared with the minSSIM threshold. This refinement is guaranteed to iterate at most $m$ times, where $m$ is the number of mantissa bits (e.g., 23 for IEEE single-precision floating-point values), for each window after which every value in the window would be losslessly preserved.

Because the lossy values are output directly as the final rounded float, the encoded values do not need to be decoded during decompression, that is, Algorithm 2 is only necessary during the compression step, not the decompression step.

### C. Compression

The above transformation and refinement is to lossily encode the data such that it is both below the error bound and above the minSSIM bound, but it does not shrink the data, it only makes it more compressible. To perform the actual compression, SBLC employs a lossless 3-stage pipeline that comprises one data transformation and two compression steps. This pipeline was designed using the LC framework [4] to be effective and fast on both CPUs and GPUs. It works as follows.

The first lossless stage, as outlined in Figure 2, computes the difference sequence [22] of the output of the lossy stage after statically casting the floating-point values to integers. If the values produced by SBLC's lossy stage are close to each other, which they are in many cases as the input data tends to be smooth, this transformation yields residuals that cluster around zero. Importantly, this stage stores the residuals in negabinary format. Negabinary is a representation of values in base -2 [36]. Unlike in two's-complement representation, small positive and small negative negabinary values both have many leading zero bits, as shown in the third row of Figure 2. This is exploited in the later compression stages.

Of course, the difference sequence may also contain some large residuals. The second lossless stage is designed to handle such cases. It is an extension of MPLG [3] and subdivides the 16 kB chunks into 512-byte subchunks. For each subchunk, it finds the largest number of leading zero bits that can be truncated from all values without losing any data. The subchunking is crucial because any large residuals that limit the number of bits to be truncated only affect their subchunk. Unlike prior implementations of MPLG, and unique to our approach, we check both leading and trailing zeros. This is key because the difference sequence tends to produce leading zeros, while the rounded mantissas tend to produce trailing zeros. This enhancement allows the second lossless stage to eliminate both leading and trailing zeros, which yields higher compression ratios than eliminating just one or the other.

While the two previous stages operate at word granularity, the final stage operates at byte granularity. As illustrated in Figure 3, it generates a bitmap in which each bit corresponds to a byte of the input. A cleared bit indicates that the corresponding byte is zero. Otherwise, the bit is set. All zero bytes are then removed from the input. Hence, the compressed output consists of the bitmap and the non-zero bytes from the input. The size of the bitmap is always the size of the input divided by 8, but the number of non-zero bytes depends on the data. Since the bitmap represents considerable overhead, we compress it using a similar algorithm that creates a second, smaller bitmap in which a cleared bit means the byte repeats and a set bit means it does not. Only the non-repeating bytes of the first bitmap are emitted along with the second 8-times-smaller bitmap. This process is iteratively applied 4 times, generating a shorter bitmap in each iteration (plus the non-repeating bytes), until the bitmap is only a few bytes long [5].

### D. Parallelization and Optimization

It is important to note that, given the window size and stride we use (7 and 2, respectively), there is overlap between adjacent windows. This not only affects the algorithm as described in Section IV-B but also how it can be parallelized. Due to the importance of CPU-GPU parity and deterministic results, the main computation loop in Algorithm 2 is split into chunks of non-overlapping windows, which are then processed in parallel with barriers between chunks instead of running the entire input in parallel, which would cause data races on overlapping windows. The latter solution can be made correct with atomic operations but would produce non-deterministic results depending on the interleaving of the threads.

On the CPU, we flatten the nested loops (2 for 2D and 3 for 3D inputs) with the "collapse" OpenMP keyword and use a dynamic schedule to assign the work. On the GPU, we use a thread-based approach and manually flatten the nested loops to improve performance. Within the main computation, accesses to the data and error bound arrays are all independent due to the chunked layout we used to ensure determinism.

While the above parallelization strategy works, it is insufficient to yield good performance without further optimization. The most computationally expensive portion of the algorithm is the local SSIM calculations. Luckily, it is unnecessary for a large portion of the windows after the first iteration. For this reason, we implemented a "checklist". Similar to a worklist, the checklist ensures that we only do work when needed, but it is not entirely data driven as it maintains the original deterministic topology-driven order. The first iteration of SBLC is completely topology driven. All windows are visited and have their local SSIMs compared to the minSSIM threshold. When a window is found to be in violation, it is iteratively fixed (see Algorithm 2). The violating window and all of its overlapping neighbors are set to "modified" on the checklist. In the following iteration, the windows are visited in the same order, but no work is done for windows that were not marked as modified in the prior iteration. We chose this strategy because we noticed that, in the later iterations, the number of minSSIM violating windows decreases rapidly.

Additionally, while we keep the window size and stride consistent with QCAT in our evaluation, both parameters can be changed. Changing either of them may affect performance. Increasing the window size or lowering the stride while holding the other parameter constant causes additional overlap between windows. This, in turn, causes additional work because any time a window is changed all overlapping

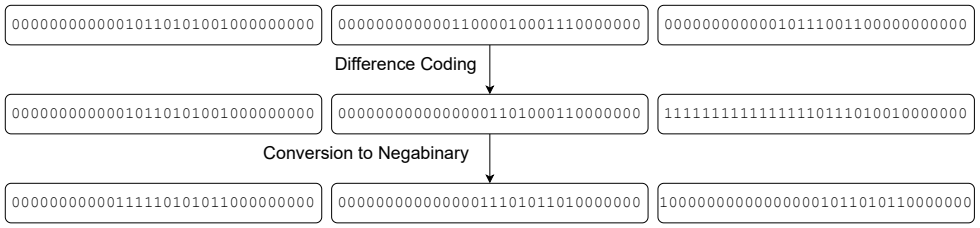



Fig. 2: Difference coding and negabinary conversion in the first lossless stage

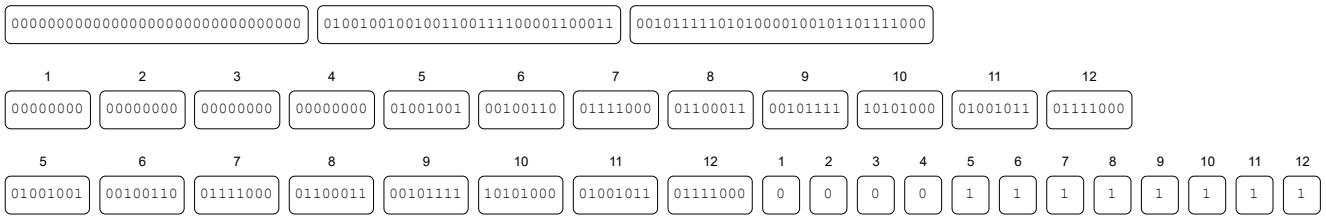



Fig. 3: Zero-byte elimination in the final lossless stage; further compression of the bitmap is not shown

windows must go back on the worklist. Conversely, decreasing the window size or increasing the stride boosts performance.

## V. Experimental Methodology

We compare SBLC to 6 state-of-the-art lossy compressors, described in Section III, on two CPU systems and two GPUs.

The first system has 2 sockets with Intel Xeon Gold 6226R CPUs, each with 16 physical cores, a base clock of 2.9 GHz, and a boost clock of 3.9 GHz. The second system has a single AMD Ryzen Threadripper 3970X CPU with 32 cores, a base clock speed of 3.7 GHz, and a boost clock of 4.5 GHz.

The first of our two GPUs is an NVIDIA A100 (Ampere architecture) with 6,912 processing elements distributed over 108 multiprocessors. It has 40 GB of global memory. The second GPU is an NVIDIA RTX 4090 (Ada Lovelace architecture) with 16,384 processing elements distributed over 128 multiprocessors. Its global memory has a capacity of 24 GB.

We compiled the CPU codes using the build processes supplied by their respective authors. When not specified, we used the "-O3 -march=native" flags. Unless automatically determined, the thread count was set to the number of CPU cores as hyperthreading usually does not help. We compiled the GPU codes using the "-O3 -arch=sm_89" flags for the RTX 4090 and the "-O3 -arch=sm_80" flags for the A100.

TABLE I: Information about the used FP32 input suites

| Name | Description | Files | Dimensions | Size (MB) |
|---|---|---|---|---|
| CESM-ATM | Climate | 33 | 26 × 1800 × 3600 | 674 |
| EXAALT Copper | Molecular Dyn. | 6 | Various 2D | 68 to 358 |
| Hurricane Isabel | Weather Sim. | 13 | 100 × 500 × 500 | 100 |
| NYX | Cosmology | 6 | 512 × 512 × 512 | 537 |
| SCALE | Climate | 12 | 98 × 1200 × 1200 | 564 |
| QMCPACK | Quantum MC | 2 | 33,120 × 69 × 69 | 631 |

We used the 6 single-precision suites shown in Table I as inputs for the compressors, a total of 72 files. They are from the SDRBench repository [31, 41], which hosts real-world scientific datasets for compression evaluation. The table lists the suite's name, short description, number of files, input dimensions, and the size per file in megabytes.

To keep the number of inputs reasonable and make the comparison as fair as possible, we excluded some SDRBench datasets. We only use the 3D CESM-ATM inputs as they are similar to the other CESM-ATM inputs and 3D is a commonly used dimension. We use only the EXAALT Copper suite as it is in the middle in terms of size for the EXAALT sets. We use the raw (i.e., not cleared) data from the Hurricane ISABEL set. Additionally, we exclude SDRBench datasets that are incompatible with at least one of the tested compressors because they are either too large or in a proprietary format.

For all compressors, we measured the execution time of the compression and decompression functions, excluding reading the input file, verifying the results, and transferring data to and from the device. We ran each experiment 9 times and collected the compression ratio, median compression throughput, and median decompression throughput. The plots report the geometric mean of the geometric mean of each suite so as not to overemphasize suites with more files. Additionally, the use of the geometric rather than the arithmetic mean helps dampen any inputs that significantly outperform the general case [17].

We present the results in x/y-scatter plots. The two dimensions are compression ratio and either compression or decompression throughput. One or both axes are logarithmic to capture the range of compression ratios and/or throughputs.

For all compressors, the triangular data point denotes a NOA error bound of 1E-2, the square 1E-3, and the pentagon an error bound of 1E-4. We chose the NOA error bound due to its adaptability to different data ranges. Since not all of the third-party code directly support NOA but all support ABS, we use a precomputed ABS error bound that corresponds to the NOA error bound, even on codes that directly support the NOA error bound. For the third-party compressors that support serial and parallel execution or CPU and GPU execution, we only show the fastest version if the compression ratio is the

same between the versions. Otherwise, we show all versions. For ZFP, which supports serial and parallel compression but only serial decompression, we show serial results only. We always show all versions of SBLC.

For SZ3, we show results for two versions: $SZ3_{Serial}$ and $SZ3_{OMP}$. The OpenMP version of SZ3 produces different compression ratios, and therefore different files, than the serial version, but both versions can be used to compress and decompress interchangeably.

Since not all compressors perform a warm-up before timing, when present, we disable any warm-up code. The use of the median of 9 runs should compensate for warm-up behavior.

The presented compression ratios are the uncompressed file size divided by the compressed file size. Rather than listing the measured runtimes, we show throughputs (i.e., the uncompressed file size divided by the runtime) because throughput, like compression ratio, is a higher-is-better metric.

## VI. Performance Evaluation

In this section, we evaluate 6 state-of-the-art lossy compressors from the literature as well as SBLC. We first compare the local SSIM performance of SBLC and its base compressor to SZ3, the leading lossy compressor for CPUs, on the ISABEL dataset. Next, we evaluate SSIM across all input suites for all tested compressors. Last, we discuss the throughput and compression ratios of the lossy compressors on the 3 different NOA error bounds described in Section V. Unless otherwise specified, we set the minSSIM bound to 0.85. For reference, we include SBLC's lossy base compressor in some of the results. This base compressor is identical to SBLC but the minSSIM guarantee has been removed.

### A. Local SSIM Analysis

Figure 4 shows an original, uncompressed slice of the Hurricane Isabel QSNOW dataset. Figure 5 presents the corresponding local SSIM values from the SBLC base compressor, SBLC, and SZ3 on the same slice.

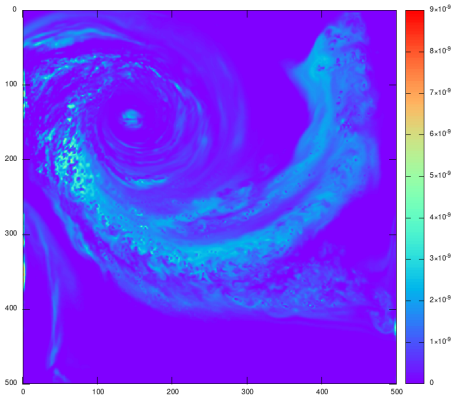



Fig. 4: Hurricane Isabel, QSNOW field, slice 12

These figures demonstrate why preserving local SSIM is important. When looking at the original QSNOW image, it is clear that the region of interest is probably near the center of the hurricane structure, but the SZ3 local SSIMs show that that region only has a small number of windows with high SSIM. Moreover, a large portion of the hurricane structure is at a local SSIM value of 0 indicating no similarity, with a few small areas in the negative range, which indicates inverse structure. Additionally, the SSIM is not smoothly distributed in the SZ3 figure; rectangular blocks of low SSIM values are directly next to areas with high SSIM values, which may introduce the appearance of structure where there is none. The base compressor of SBLC does not fare any better. Most of the region where visualization may be important has a SSIM near 0. What may be insidious is that, because SSIM is an average metric, the high SSIM areas at the top of both SZ3's and the base compressor's output skew the average SSIM to be high. In contrast, SBLC, which guarantees a minSSIM bound, shows the entire data slice at high local SSIM values.

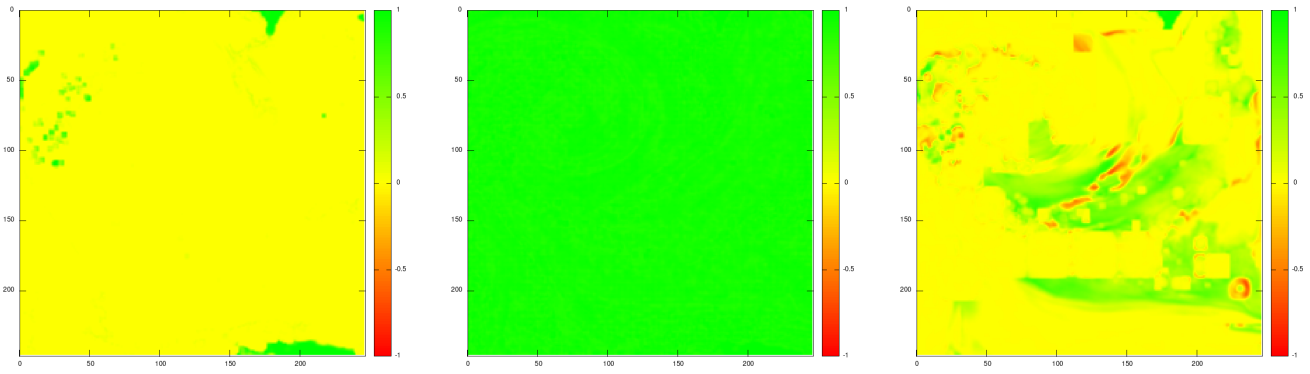
Figure 6 shows a histogram, i.e., the frequency distribution, of local SSIM values across the entire hurricane Isabel input suite for SZ3, SBLC, and SBLC's base compressor. We sorted the local SSIM values into 41 equal-sized bins (with the exception of the first and last bins, which are half-size), which are represented on the x-axis. The leftmost bin corresponds to a local SSIM of -1 and the rightmost bin to a local SSIM of 1. Note the logarithmic y-axis. On these inputs, SZ3 has 4% of its local SSIM values below 0, 21% are in the bin that includes 0, and the remaining 75% are above 0 with 60% in the bin corresponding to a local SSIM of 1. SBLC's base compressor similarly has a spike at 0 with 38% of the local SSIM values, and 57% of the values are in the highest bin. SBLC, with its minSSIM set to 0.85, has no local SSIM values below this threshold. Note that it has not only a larger count in the 0.85 bin but also in the higher bins relative to the base compressor. This is because adjusting the error bounds in one window also boosts the SSIM of overlapping neighboring windows, even if they already meet the minSSIM threshold.

The existing lossy compressors, including SBLC's base compressor, are SSIM agnostic and have the potential to generate misleading visualizations. In contrast, these results show that SBLC's control of minSSIM guarantees the entire reconstructed dataset to be at or above the requested SSIM.

### B. Quality of Reconstructed Data

Table II shows the average SSIM and the minSSIM for the tested codes across the 3 NOA error bounds. For all metrics, higher is better. The highest entries for each metric and error bound are in bold print. In this section we present results for SBLC using minSSIM thresholds of 0.50, 0.70, 0.85, and 0.95.

The table highlights the large difference between SBLC and the other tested compressors. SBLC delivers the highest SSIM and minSSIM values in all cases, even with a minSSIM bound of 0.5. This is a direct result of its refinement step as the base compressor performs similarly to all other tested compressors in terms of minSSIM and SSIM. The results shows that the refinement not only ensures that the minSSIM is at or above the requested bound but also greatly boosts the



(a) Base compressor (b) SBLC (c) SZ3

Fig. 5: Local SSIM visualization on ISABEL QSNOW slice 12 using the base compressor, SBLC, and SZ3 with a NOA error bound of 1E-3. Colors closer to green are near the best SSIM of 1. Colors closer to red are near the worst SSIM of -1.

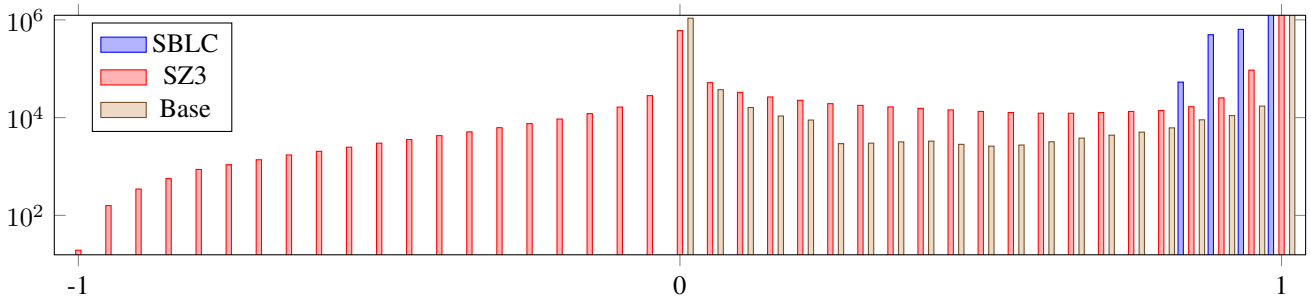



Fig. 6: Distribution of local SSIM values across all ISABEL inputs for the base compressor, SBLC, and SZ3

TABLE II: Average SSIM and minSSIM for the tested codes on the 3 NOA error bounds

| | 0.01 | | 0.001 | | 0.0001 | |
|---|---|---|---|---|---|---|
| | SSIM | minSSIM | SSIM | minSSIM | SSIM | minSSIM |
| ZFP | 0.82 | -0.03 | 0.88 | 0.24 | 0.92 | 0.35 |
| SZ3 | 0.61 | -0.69 | 0.75 | -0.35 | 0.81 | 0.01 |
| SZ3 OMP | 0.61 | -0.69 | 0.76 | -0.34 | 0.82 | 0.00 |
| MGARD | 0.82 | 0.07 | 0.87 | 0.27 | 0.91 | 0.52 |
| cuSZp | 0.46 | -0.39 | 0.71 | -0.15 | 0.81 | 0.21 |
| PFPL | 0.53 | -0.26 | 0.76 | 0.01 | 0.84 | 0.34 |
| SLEEK | 0.58 | -0.23 | 0.77 | 0.03 | 0.85 | 0.35 |
| Base | 0.58 | -0.23 | 0.77 | 0.04 | 0.85 | 0.34 |
| SBLC 0.50 | 0.85 | 0.54 | 0.92 | 0.63 | 0.95 | 0.72 |
| SBLC 0.70 | 0.91 | 0.72 | 0.95 | 0.77 | 0.97 | 0.83 |
| SBLC 0.85 | 0.95 | 0.86 | 0.97 | 0.88 | 0.99 | 0.91 |
| SBLC 0.95 | **0.98** | **0.95** | **0.99** | **0.96** | **0.99** | **0.97** |

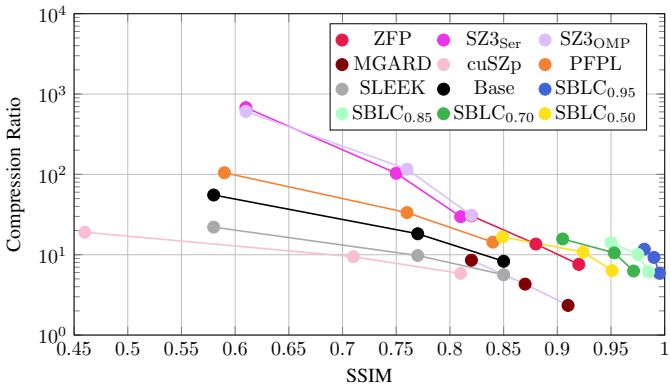



Fig. 7: Compression ratio vs. SSIM for 3 NOA error bounds

overall SSIM. Surprisingly, for almost all other compressors, the minSSIM is a negative value, which indicates the potential for misinterpreting results when they are visualized.

Figure 7 plots the average SSIM versus the compression ratio for all tested compressors on the three NOA error bounds. Compression ratio is plotted on a logarithmic and SSIM on a linear scale. Both are higher-is-better metrics, so points closer to the top right are better.

These results show that SBLC is customizable using the minSSIM bound. While all other compressors, including the SBLC base compressor, produce a wide range of SSIM values, the four thresholds of SBLC produce consistiently good SSIM results. The 0.5 threshold expectedly yields the lowest SSIM values of the SBLC results, but it is still better than the other the tested compressors. It also delivers the highest compression ratios of the SBLC thresholds. This contrasts with the 0.95-threshold SBLC results, which produce marginally lower compression ratios but substantially higher SSIM values. Additionally, for all minSSIM thresholds on the tighter two error bounds, SBLC compresses more than all but one of the other tested GPU compressors. The figure visually reinforces

that, if SSIM is a priority metric, using the minSSIM guarantee that SBLC provides yields tunable results that are beyond what other compressors from the literature can reach. Overall, Table II and Figure 7 show that SBLC provides a substantial SSIM benefit by guaranteeing the minSSIM.

### C. NOA Error Bounds

**Compression:** Figures 8, 9, 10, and 11 show scatter plots of the compression ratio versus the compression throughput on our A100 GPU, 4090 GPU, Intel CPU, and AMD CPU, respectively, for three NOA error bounds. For reference, SBLC's base compressor is included in the 4090 results as $\text{Base}_{\text{CUDA}}$.

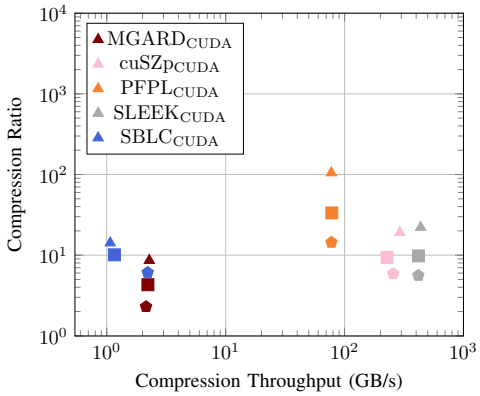



Fig. 8: A100 GPU geometric-mean compression ratio and compression throughput for the 3 NOA error bounds

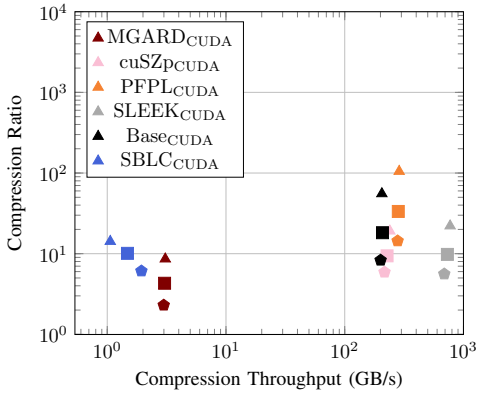



Fig. 9: 4090 GPU geometric-mean compression ratio and compression throughput for the 3 NOA error bounds

Since SBLC produces bit-for-bit the same compressed and decompressed data on the CPU and the GPU, its compression ratios are the same on all tested devices. It compresses well on the GPU relative to the other lossy codes. PFPL yields the highest compression ratios at all 3 tested error bounds, but for the two tighter and more realistic error bounds, SBLC yields higher compression ratios than the remaining 3 GPU codes. As mentioned, its compression ratio can be tuned by changing the minSSIM bound. Using a lower bound, SBLC compresses more as the base results show. This is most evident at the coarsest error bound, where the base SBLC compressor yields

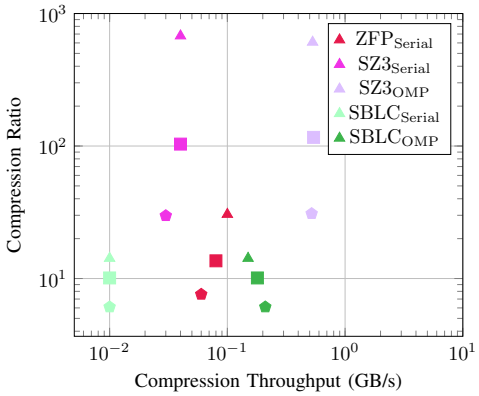



Fig. 10: Intel CPU geometric-mean compression ratio and compression throughput for the 3 NOA error bounds

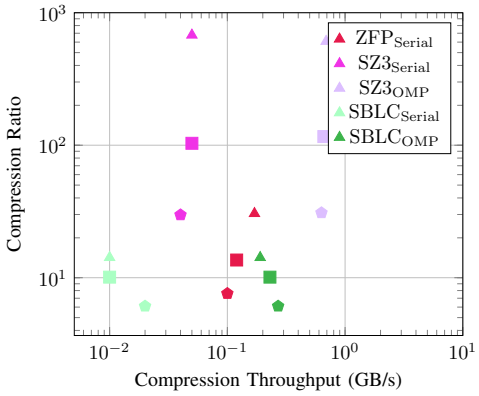



Fig. 11: AMD CPU geometric-mean compression ratio and compression throughput for the 3 NOA error bounds

a compression ratio of 55.4, on average, which is significantly higher than all tested compressors except PFPL.

On the CPU, both versions of SZ3 compress very well. This is due to their CPU-only nature, which allows for data transformations that are generally unfriendly for implementation on GPUs. ZFP yields compression ratios much closer to SBLC while still being higher in all cases. Comparing SBLC to ZFP illustrates the cost of bounding minSSIM. At the coarsest tested error bound, ZFP compresses, on average, 2.2 times more than SBLC and 1.2 times more at the tightest error bound. However, when compared to the SBLC's base compressor, ZFP compresses 40% less at the coarsest tested error bound and 10% less at the tightest.

On the GPU, SLEEK delivers the highest compression throughput. This is expected because SLEEK was designed to be close in speed to a device-to-device memcpy. SBLC compresses the slowest, with one exception on the A100. The low compression throughput of SBLC is not surprising, however, due to the iterative refinement step to guarantee the minSSIM. Note that, while the other tested GPU compressors generally produce similar throughput on all tested error bounds, SBLC is 2× faster on the tightest error bound than the coarsest. This is because coarser error bounds require more iterations to fix the minSSIM-violating local windows. Moreover, on the 4090,

the base compressor without any minSSIM guarantee produces compression throughputs close to both PFPL and cuSZp at an average throughput of 204.7 GB/s across the 3 error bounds.

On the CPU, SBLC shows the same increase in performance as the error bound tightens and less work must be done to guarantee the minSSIM. The serial version of SBLC is significantly slower than all other tested CPU compressors, but the parallel OpenMP version is faster than all except the parallel version of SZ3, showing that SBLC is still practical on the CPU with 267 MB/s of throughput at the tightest error bound. Overall, when given a relatively strict minSSIM, SBLC tends to compress more slowly than the other tested lossy compressors, at least on the used minSSIM of 0.85.

**Decompression:** Figures 12, 13, 14, and 15 show scatter plots of the compression ratio versus the decompression throughput on our A100 GPU, 4090 GPU, Intel CPU, and AMD CPU, respectively, using 3 NOA error bounds. The base compressor is again included in the 4090 results for reference. Since the compression ratios are the same as they are for compression, we focus our discussion on the throughput.

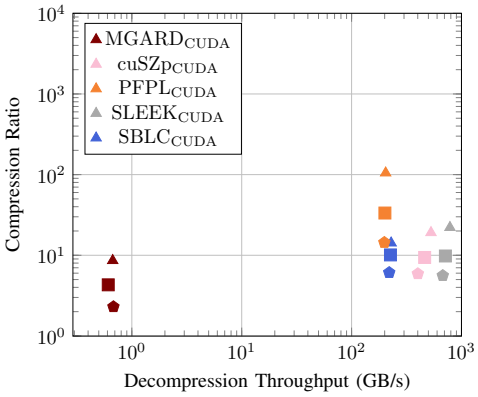



Fig. 12: A100 GPU geometric-mean compression ratio and decompression throughput for the 3 NOA error bounds

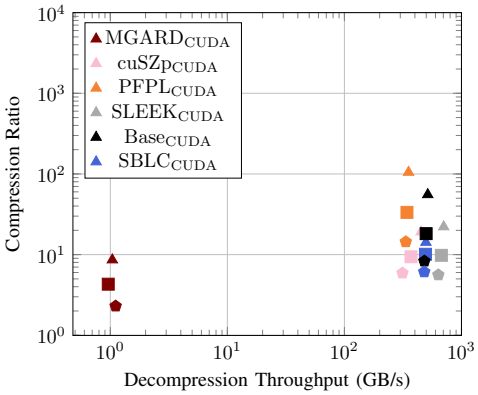



Fig. 13: 4090 GPU geometric-mean compression ratio and decompression throughput for the 3 NOA error bounds

As described in Section IV, SBLC performs all of the work required to bound the minSSIM during compression. For this

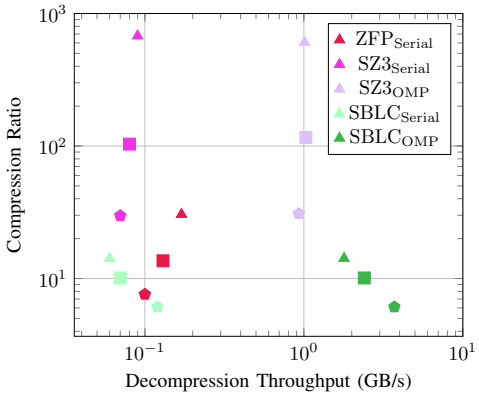



Fig. 14: Intel CPU geometric-mean compression ratio and decompression throughput for the 3 NOA error bounds

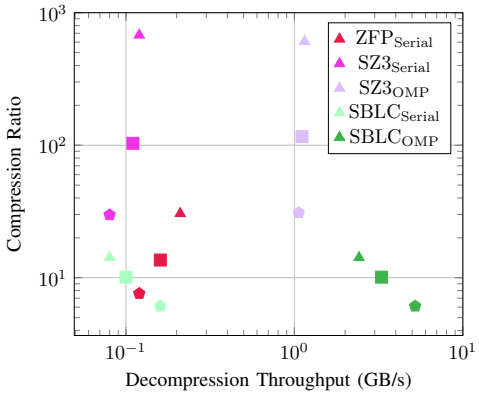



Fig. 15: AMD CPU geometric-mean compression ratio and decompression throughput for the 3 NOA error bounds

reason, SBLC decompresses much faster than it compresses. It performs on par with the other tested GPU compressors with the exception of SLEEK, which is faster in all cases. On the A100, which has a high memory bandwidth relative to its compute performance, cuSZp decompresses a little faster than SBLC. Conversely, on the 4090, where the ratio of compute to memory performance is more balanced, SBLC is faster than both PFPL and cuSZp with an average throughput of 490.1 GB/s across all tested error bounds. Additionally, the base compressor performs identically to SBLC in decompression speed as there is no minSSIM computation to be performed.

On the CPU, SBLC has a similar throughput as the other tested lossy compressors serially. When it comes to the parallel OpenMP codes, SBLC is always faster than the other tested CPU codes, including the parallel version of SZ3.

Overall, SBLC compresses well relative to the related work at tighter error bounds while being the only compressor that preserves the minSSIM. The compression speed is dependent on the requested minSSIM bound, but the base compressor is capable of achieving similar compression speeds to the other tested lossy compressors, meaning the performance can be tuned to fit an application. SBLC's decompression throughput is unaffected by the minSSIM bound and is on-par with or better than the other tested compressors on the CPU and the

GPU, all while maintaining the requested minSSIM bound.

## VII. Summary and Conclusions

This paper introduces the minSSIM metric to aid users in ensuring visual quality for scientific floating-point data by guaranteeing that every local SSIM window is at or above the user-specified minSSIM bound. It further introduces the first SSIM-Bounded Lossy Compressor (SBLC), which combines minSSIM guarantees with a new, fast, and effective lossy base compressor. SBLC accomplishes this by selectively tightening the error bound on certain values in the dataset. Importantly, it does this without explicitly recording the used error bound. We evaluated 6 leading lossy compressors on 2 CPUs and 2 GPUs using 6 single-precision floating-point suites from the SDRBench repository. SBLC delivers the highest SSIM and minSSIM across all tested inputs. Moreover, it provides tunable compression performance, both in terms of compression ratio and compression speed, based on the requested minSSIM and NOA error bounds. Further, SBLC yields decompression throughputs on par with the state of the art on GPUs and decompresses faster on CPUs than all tested lossy compressors. Additionally, SBLC supplies guaranteed error bounding and full CPU-GPU compatibility, which only few of the tested compressors do. We hope that SBLC will assist scientists with lossy compression when visualization is important by respecting not only the point-wise error bound but also the visual fidelity of the data for downstream analysis.

## VIII. Acknowledgments

This work has been supported by the U.S. National Science Foundation (NSF) under Award CCF-2403380, by the Department of Energy (DOE), Office of Science, Advanced Scientific Computing Research (ASCR) under Award DE-SC0022223, and by an equipment donation from NVIDIA Corporation.

## References

[1] Quick Compression Analysis Toolkit (QCAT). https://github.com/szcompressor/qcat. Accessed: 2026-03-17.

[2] Anju Mongandampulath Akathoott, Andrew Rodriguez, and Martin Burtscher. SLEEK: Compressing Memory Copies for Floating-Point Data on GPUs. In *2026 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 874–887, May 2026.

[3] Noushin Azami and Martin Burtscher. Compressed In-memory Graphs for Accelerating GPU-based Analytics . In *2022 IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 32–40, Los Alamitos, CA, USA, Nov 2022. IEEE Computer Society. https://doi.org/10.1109/IA356718.2022.00011.

[4] Noushin Azami, Alex Fallin, Brandon Burtchell, Andrew Rodriguez, Benila Jerald, Yiqian Liu, and Martin Burtscher. LC Git Repository. https://github.com/burtscher/LC-framework, 2025. Accessed: 2025-01-07.

[5] Noushin Azami, Alex Fallin, and Martin Burtscher. Efficient Lossless Compression of Scientific Floating-Point Data on CPUs and GPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, pages 395–409, New York, NY, USA, 2025. Association for Computing Machinery. https://doi.org/10.1145/3669940.3707280.

[6] Allison H. Baker, Alexander Pinard, and Dorit M. Hammerling. On a Structural Similarity Index Approach for Floating-Point Data. *IEEE Transactions on Visualization and Computer Graphics*, 30(9):6261–6274, September 2024. https://doi.org/10.1109/TVCG.2023.3332843.

[7] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end Optimized Image Compression. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. https://doi.org/10.48550/arXiv.1611.01704.

[8] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. https://doi.org/10.48550/arXiv.1802.01436.

[9] Franck Cappello, Robert Underwood, Yuri Alexeev, Alison Baker, Ebru Bozdağ, Martin Burtscher, Kyle Chard, Sheng Di, Kyle Gerard Felker, Paul Christopher O'Grady, Hanqi Guo, Yafan Huang, Peng Jiang, Sian Jin, Petter Johansson, Shaomeng Li, Xin Liang, Erik Lindahl, Peter Lindstrom, Zarija Lukić, Magnus Lundborg, Danylo Lykov, Masaru Nagaso, Kento Sato, Amarjit Singh, Seung Woo Son, Shihui Song, William Tang, Dingwen Tao, Jiannan Tian, Kazutomo Yoshii, and Kai Zhao. What to Support When You're Compressing: The State of Practice Gaps and Opportunities for Scientific Data Compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, page 1966–1979, New York, NY, USA, 2025. Association for Computing Machinery. https://doi.org/10.1145/3712285.3759856.

[10] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned Image Compression with Discretized Gaussian Mixture Likelihoods and Attention Modules. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. https://doi.org/10.48550/arXiv.2001.01568.

[11] Yann Collet and Murray Kucherawy. Zstandard Compression and the 'application/zstd' Media Type. RFC 8878, feb 2021. https://doi.org/10.17487/RFC8878.

[12] Ze Cui, Jing Wang, Shangyin Gao, Bo Bai, Tiansheng Guo, and Yihui Feng. Asymmetric Gained Deep Image Compression With Continuous Rate Adaptation, 2022. https://doi.org/10.48550/arXiv.2003.02012.

[13] L. Peter Deutsch. GZIP file format specification version 4.3. RFC 1952, may 1996. https://doi.org/10.17487/RFC1952.

[14] James Diffenderfer, Alyson L. Fox, Jeffrey A. Hittinger,

Geoffrey Sanders, and Peter G. Lindstrom. Error Analysis of ZFP Compression for Floating-Point Data. *SIAM Journal on Scientific Computing*, 41(3):A1867–A1898, 2019. https://doi.org/10.1137/18M1168832.

[15] Alex Fallin, Noushin Azami, Sheng Di, Franck Cappello, and Martin Burtscher. Fast and Effective Lossy Compression on GPUs and CPUs with Guaranteed Error Bounds. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 874–887, June 2025. https://doi.org/10.1109/IPDPS64566.2025.00083.

[16] Alex Fallin and Martin Burtscher. SBLC Git Repository. https://github.com/burtscher/minSSIM, 2026. Accessed: 2026-09-04.

[17] Philip J. Fleming and John J. Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Commun. ACM*, 29(3):218–221, mar 1986. https://doi.org/10.1145/5666.5673.

[18] Dailan He, Ziming Yang, Weikun Peng, Rui Ma, Hongwei Qin, and Yan Wang. ELIC: Efficient Learned Image Compression with Unevenly Grouped Space-Channel Contextual Adaptive Coding, 2022. https://doi.org/10.48550/arXiv.2203.10886.

[19] Yafan Huang, Sheng Di, Xiaodong Yu, Guanpeng Li, and Franck 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*, SC'23, Denver, CO, USA, 2023. Association for Computing Machinery. https://doi.org/10.1145/3581784.3607048.

[20] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. https://doi.org/10.1109/JRPROC.1952.273898.

[21] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. Out-of-core Compression and Decompression of Large n-dimensional Scalar Fields. *Comput. Graph. Forum*, 22:343–348, 09 2003. https://doi.org/10.1111/1467-8659.00681.

[22] F. Jager. Delta Modulation — A Method of PCM Transmission Using the One Unit Code. *Philips Res. Repts.*, 7, 01 1952. https://doi.org/10.1002/j.1538-7305.1969.tb01118.x.

[23] Pu Jiao, Sheng Di, Jiannan Tian, Mingze Xia, Xuan Wu, Yang Zhang, Xin Liang, and Franck Cappello. Mitigating Artifacts in Pre-quantization Based Scientific Data Compressors with Quantization-aware Interpolation, 2026. https://doi.org/10.48550/arXiv.2602.20097.

[24] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy T. Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. Improved Lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks. *CoRR*, abs/1703.10114, 2017. https://doi.org/10.48550/arXiv.1703.10114.

[25] J. E. Kay, C. Deser, A. Phillips, A. Mai, C. Hannay, G. Strand, J. M. Arblaster, S. C. Bates, G. Danabasoglu, J. Edwards, M. Holland, P. Kushner, J.-F. Lamarque, D. Lawrence, K. Lindsay, A. Middleton, E. Munoz, R. Neale, K. Oleson, L. Polvani, and M. Vertenstein. "The Community Earth System Model (CESM) Large Ensemble Project: A Community Resource for Studying Climate Change in the Presence of Internal Climate Variability". *Bulletin of the American Meteorological Society*, 96(8):1333–1349, 2015. https://doi.org/10.1175/BAMS-D-13-00255.1.

[26] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-Controlled Lossy Compression Optimized for High Compression Ratios of Scientific Datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447, 2018. https://doi.org/10.1109/BigData.2018.8622520.

[27] Xin Liang, Ben Whitney, Jieyang Chen, Lipeng Wan, Qing Liu, Dingwen Tao, James Kress, David Pugmire, Matthew Wolf, Norbert Podhorszki, and Scott Klasky. MGARD+: Optimizing Multilevel Methods for Error-Bounded Scientific Data Reduction. *IEEE Transactions on Computers*, 71(7):1522–1536, 2022. https://doi.org/10.1109/TC.2021.3092201.

[28] Xin Liang, Kai Zhao, Sheng Di, Sihuan Li, Robert Underwood, Ali M. Gok, Jiannan Tian, Junjing Deng, Jon C. Calhoun, Dingwen Tao, Zizhong Chen, and Franck Cappello. SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors. *IEEE Transactions on Big Data*, 9(2):485–498, 2023. https://doi.org/10.1109/TBDATA.2022.3201176.

[29] Peter Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014. https://doi.org/10.1109/TVCG.2014.2346458.

[30] David Minnen, Johannes Ballé, and George Toderici. Joint Autoregressive and Hierarchical Priors for Learned Image Compression. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10794–10803, 2018. https://doi.org/10.48550/arXiv.1809.02736.

[31] SDRBench Inputs, https://sdrbench.github.io/, 2023. https://sdrbench.github.io/.

[32] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck 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 (IPDPS)*, pages 1129–1139, 2017. https://doi.org/10.1109/IPDPS.2017.115.

[33] D.S. Taubman and M.W. Marcellin. JPEG2000: standard for interactive imaging. *Proceedings of the IEEE*, 90(8):1336–1357, 2002.

https://doi.org/10.1109/JPROC.2002.800725.

[34] Jiannan Tian, Sheng Di, Xiaodong Yu, Cody Rivera, Kai Zhao, Sian Jin, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. Optimizing Error-Bounded Lossy Compression for Scientific Data on GPUs. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 283–293, Los Alamitos, CA, USA, September 2021. IEEE Computer Society. https://doi.org/10.1109/Cluster48925.2021.00047.

[35] Jiannan Tian, Sheng Di, Kai Zhao, Cody Rivera, Megan Hickman Fulp, Robert Underwood, Sian Jin, Xin Liang, Jon Calhoun, Dingwen Tao, and Franck Cappello. 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*, PACT '20, page 3–15, New York, NY, USA, 2020. Association for Computing Machinery. https://doi.org/10.1145/3410463.3414624.

[36] Louis B. Wadel. Negative Base Number Systems. *IRE Transactions on Electronic Computers*, EC-6(2):123–123, 1957. https://doi.org/10.1109/TEC.1957.5221586.

[37] Gregory K. Wallace. The JPEG still picture compression standard. *Commun. ACM*, 34(4):30–44, April 1991. https://doi.org/10.1145/103085.103089.

[38] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402 Vol.2, 2003. https://doi.org/10.1109/ACSSC.2003.1292216.

[39] 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*, 13(4):600–612, 2004. https://doi.org/10.1109/TIP.2003.819861.

[40] Kai Zhao, Sheng Di, Maxim Dmitriev, Thierry-Laurent D. Tonellot, Zizhong Chen, and Franck Cappello. Optimizing Error-Bounded Lossy Compression for Scientific Data by Dynamic Spline Interpolation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1643–1654, 2021. https://doi.org/10.1109/ICDE51399.2021.00145.

[41] Kai Zhao, Sheng Di, Xin Lian, Sihuan Li, Dingwen Tao, Julie Bessac, Zizhong Chen, and Franck Cappello. SDRBench: Scientific Data Reduction Benchmark for Lossy Compressors. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2716–2724, 2020. https://doi.org/10.1109/BigData50022.2020.9378449.