Lessons Learned on the Path to Guaranteeing the Error Bound in Lossy Quantizers

ALEX FALLIN, Texas State University, United States
MARTIN BURTSCHER, Texas State University, United States

Rapidly increasing data sizes in scientific computing are the driving force behind the need for lossy compression. The main drawback of lossy data compression is the introduction of error. This paper explains why many error-bounded compressors occasionally violate the error bound and presents the solutions we use in LC, a CPU/GPU compatible lossy compression framework, to guarantee the error bound for all supported types of quantizers. We show that our solutions maintain high compression ratios and cause no appreciable change in throughput.

CCS Concepts: • **Information systems → Data compression**; • **Software and its engineering → Correctness**.

Additional Key Words and Phrases: lossy data compression, error-bounded compression, floating-point data, CPU/GPU compatibility

1 INTRODUCTION

Many scientific instruments and simulations generate more data than can reasonably be handled, both in terms of throughput and in terms of total size [6]. There are two types of data compression, lossy and lossless, to alleviate these problems. Lossless compressors exactly reproduce the original data bit-for-bit. However, they are often not able to deliver the desired compression ratios. In contrast, lossy compression can yield much higher compression ratios, but with the caveat that the data is not exactly reproduced. High compression ratios without knowing the quality of the reconstructed data is not useful, thus lossy compression is often error-bounded. This means that the original value and the decompressed value do not deviate by more than a preset threshold. The three most frequently used error-bound types are point-wise absolute error (ABS), point-wise relative error (REL), and point-wise normalized absolute error (NOA). Bounding the error is important for scientific analysis as it gives the users that decompress and analyze the data confidence that the information is reasonably preserved. Otherwise, they may not be able to draw valid conclusions from the decompressed data.

In today's heterogeneous HPC environments, scientific data is often generated and compressed on one type of device (i.e., a CPU or a GPU) but needs to be decompressed on a different type of device. For example, GPU-based compression may be critical for applications that produce data at a very high throughput, while CPU-based compression may be sufficient in other environments. Independently, the resulting data may be decompressed and analyzed by various users who may or may not have a GPU. Hence, cross-device compression and decompression is important, but it is rarely supported by today's state-of-the-art lossy compressors.

Beyond differences in hardware, issues with software, mainly surrounding the use of floating-point values and operations, can be the reason for failing to meet the error bound. Rounding issues, lack of associativity, compiler optimizations, and special values all have the potential to cause lack of parity or error-bound violations.

Allowing for error in compression begs two important questions. What factors affect the error bounding when reconstructing data? How do we guarantee matching results across different hardware (i.e., parity)? In this paper, we discuss our answers to these questions and describe the solutions we implemented in the guaranteed-error-bounded lossy quantizers of our open-source LC framework [1].

This paper makes the following main contributions.

- It presents an evaluation of state-of-the-art lossy compressors on all possible single-precision floating-point values and many double-precision values showing that most of them violate the error bound on some values.
- It describes problems with floating-point arithmetic that can cause such violations.
- It discusses difference between CPUs and GPUs that cause them to produce different compressed data when running the same compression algorithm.
- It explains the code changes we had to make to guarantee the error bound and inter-device parity in all cases.
- It analyzes the impact of these changes on the compression ratio and throughput.

The rest of the paper is organized as follows. Section 2 describes the common types of error-bounding. It further describes the problems we encountered, in terms of correctness, with floating-point arithmetic and parity between the CPU and GPU when implementing our compressor. Section 3 presents the code changes we made in our quantizers to address these floating-point arithmetic and parity issues. Section 4 summarized related work on lossy compression and explains how various prior compressors run into correctness problems. Section 5 describes our evaluation methodology. Section 6 measures and discusses the impact of our solutions on the compression ratio and throughput. Section 7 concludes the paper with a summary.

2 BACKGROUND

There are three point-wise error-control metrics that are commonly used in the literature: point-wise absolute (ABS), point-wise relative (REL), and point-wise normalized absolute (NOA). In this subsection, we define these error bounds and describe their uses [21].

2.1 Common Error-bound Types

2.1.1 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 [16]. The absolute error of a data value x is defined as $e_{abs} = |x_{original} - x_{reconstructed}|$. To guarantee an absolute error bound of ε, each value in the reconstructed file must satisfy $e_{abs} \leq \varepsilon$. Hence, each reconstructed value must be in the range $x_{original} - \varepsilon \leq x_{reconstructed} \leq x_{original} + \varepsilon$.

2.1.2 Point-Wise Relative Error (REL). The point-wise *relative* error of a data value is the *ratio* between the difference of the original and its reconstructed value and the original value [8]. The relative error of a value x is expressed as $e_{rel} = |\frac{x_{original} - x_{reconstructed}}{x_{original}}| = |1 - \frac{x_{reconstructed}}{x_{original}}|$. To guarantee a relative error bound of ε, every value in the reconstructed file must satisfy $e_{rel} \leq \varepsilon$. In other words, each reconstructed value must have the same sign as the original value and be in the range $|x_{original}|/(1+\varepsilon) \leq |x_{reconstructed}| \leq |x_{original}| \times (1+\varepsilon)$.

2.1.3 Point-Wise Normalized Absolute Error (NOA). The point-wise *normalized absolute* error is the point-wise absolute 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 data value x is defined as $e_{noa} = |\frac{e_{abs}}{R}|$. To guarantee an error bound of ε, each value in the reconstructed file must satisfy $e_{noa} \leq \varepsilon$. Hence, each reconstructed value must be in the range $x_{original} - \varepsilon R \leq x_{reconstructed} \leq x_{original} + \varepsilon R$. Since NOA is a variant of and has the same issues as ABS, we do not separately evaluate it in this paper.

2.2 Floating-point Arithmetic

Floating-point values cannot precisely represent all numbers. Values that cannot be represented are rounded to a representable value. This behavior is important to note as rounding issues during data reconstruction are a common cause of error-bound violations. For example, ABS quantization is generally performed by multiplying the input value by the inverse of twice the error bound and rounding the result to the nearest (integer) bin number. This operation should be completely safe and yield a reconstructed value (i.e., the center of the value range represented by the bin) that differs from the original value by no more than the error bound. Small rounding issues, however, can cause an error-bound violation by placing a value that is very close to the border of one bin into the neighboring bin. Note that this is a problem even if the rounding error is much smaller than the error bound used for the quantization.

Infinity (INF), not-a-number (NaN), and denormal floating-point values pose additional challenges. INF and NaN values propagate when used in floating-point computations. Denormals are particularly susceptible to rounding issues as they are unable to retain the same precision as normal values. These special values, while problematic, must be preserved. For example, with an ABS error bound, normal and denormal values can be binned, but infinities and NaNs must be separately handled because, for example, $NaN \pm 0.001$ is still NaN and $INF \pm 0.001$ is still INF. For a REL error bound, even denormals may require special handling.

2.3 Result Parity

In the process of developing our quantizers, we encountered many problems related to result parity between CPUs and GPUs. In this subsection, we describe these issues and give examples of code that causes them.

A fused multiply add (FMA) is a special machine instruction that performs both a multiplication and an addition without rounding the intermediate result, thus sometimes producing a different answer than a multiplication followed by a separate addition does. Since the FMA is faster, optimizing compilers try to use them when possible. For example, consider the following partial check of whether the ABS error bound has been exceeded (where `eb2` is twice the error bound): `bin * eb2 + eb < orig_value`. The left-hand-side expression may be compiled into an FMA depending on the many factors taken into account when optimizing the code. This optimization changes the rounding error as described above. What is more, different compilers make different optimization decisions as is the case for our CPU and GPU compilers, causing a disparity between the CPU compressed file and the GPU compressed file. Additionally, as compilers evolve, code that does not currently yield FMA instructions may do so in the future.

Another major problem with supporting both CPUs and GPUs is the difference in libraries, and thus the results of some of the basic functions that would normally be expected to match. While developing the quantizers of LC, we encountered such a mismatch. REL uses the `log()` function in the quantization and the `pow()` function in the reconstruction step. Interestingly, these two functions do not produce the same result when passed the same argument on a CPU and a GPU. An actual example is the GPU producing a `log()` result of 88.5 when the CPU produces 88.4999.... Whereas this mismatch seems small, it may result in one code choosing a different bin than the other, removing the guarantee of parity between the CPU and GPU.

2.4 Edge Cases

Beyond floating-point arithmetic and result parity, it is important to handle edge cases. For instance, we found that we cannot use `std::abs()` in our quantizers. We originally used the check `if ((std::abs(bin) >= maxbin) ...)` to determine if a bin number was valid. Since the range of twos-complement integers is $-2,147,483,648$ to $2,147,483,647$

(note the difference in the last digit), `std::abs()` does not work for $-2,147,483,648$ as there is no corresponding positive value. While this is a 1-in-4-billion edge case, we encountered it on a real scientific input.

3 APPROACH

The previous section describes the three types of issues we had to contend with while developing the quantizers in LC [1], that is, the rounding of floating-point values, differences between CPUs and GPUs, and corner cases. In this section, we describe the solutions we implemented to create a CPU/GPU-compatible lossy compressor that provides a true error-bound guarantee and discuss the impacts these solutions have.

3.1 Floating-point Arithmetic

To address the rounding issues inherent to floating-point operations, we employ "double-checking" in the quantization step, meaning we immediately reconstruct each value and check whether it is within the error bound. To this end, we included the following lines of code, where `bin` is the quantized bin number, `eb` the error bound, `eb2` twice the error bound, `origval` the original value, and `recon` the reconstructed value (we only show the relevant `if` conditions):

```
const float recon = bin * eb2;
if (fabsf(origval - recon) > eb) ...
```

By performing this check, we catch any floating-point issues that would cause the requested bound to be violated. If the condition is met, the value is preserved losslessly as we cannot quantize it within the error bound. For all three error bound types, we found that most of the input files contain at least one outlier that is caught by this test and preserved losslessly. We store these losslessly preserved outliers in-line with the bin numbers, which simplifies the program parallelization. This is in contrast to, for example, SZ3 [9, 12, 20], which does not commingle outliers and bin numbers but instead stores outliers in a separate list and uses bin number 0 to indicate an outlier.

We handle infinity by explicitly checking for it in our REL quantizer. In the ABS quantizer, the check is implicit; infinities are encoded losslessly because they cause checks that handle other error-bound issues to fail. Both quantizers explicitly check for and handle NaNs. Denormals are treated like normal values.

3.2 Result Parity

The precision- and performance-increasing fused-multiply-add instructions can sometimes be avoided by tricking the compiler into thinking the intermediate value is used when it actually is not. This is not a reliable fix, however, as compiler improvements may be able to determine that the intermediate value goes unused. Therefore, we use the compiler flag `-mno-fma` for *g++* and the similar flag `-fmad=false` for *nvcc* to disable the use of FMAs. Note that these flags may potentially reduce the achievable precision. However, this is not a problem because of the aforementioned double checking. On the off chance that the reduced precision yields a wrong bin number, the corresponding value is simply encoded losslessly. This may lower the compression ratio, but it will not violate the error bound. Employing these flags, in combination with only using fully IEEE 754-compliant floating-point operations, results in code that produces the same compressed and decompressed values on CPUs and GPUs.

The differing `log()` and `pow()` functions were particularly challenging to fix. The solution we ultimately adopted is to write our own approximation functions. Our `log2()` and `pow2()` code for single-precision data is as follows:

```
float log2approxf(const float orig_f) {
  const int mantissabits = 23;
  const int orig_i = *((int*)&orig_f); // extract bit pattern
  const int expo = (orig_i >> mantissabits) & 0xff; // isolate exponent
  const int frac_i = (127 << mantissabits) | (orig_i & ~(~0 << mantissabits)); // isolate fraction
  const float frac_f = *((float*)&frac_i); // convert fraction back to float
  const float log_f = frac_f + (expo - 128); // add de-biased exponent
  return log_f;
}
float pow2approxf(const float log_f) {
  const int mantissabits = 23;
  const float biased = log_f + 127; // re-bias exponent
  const int expo = biased; // get exponent
  const float frac_f = biased - (expo - 1); // recreate fraction
  const int frac_i = *((int*)&frac_f); // extract fraction
  const int exp_i = (expo << mantissabits) | (frac_i & ~(~0 << mantissabits)); // combine exp & frac
  const float recon_f = *((float*)&exp_i); // convert back to float
  return recon_f;
}
```

These approximations guarantee matching solutions between the CPU and GPU because every operation within them is fully IEEE 754-compliant or an integer operation. As shown in Section 6, this solution hurts the compression ratio a little because the approximation is not particularly accurate. As before, it does not affect correctness because results that exceed the error bound are discarded and the corresponding values losslessly encoded.

3.3 Edge Case

We handle the problem with `std::abs()` by breaking the single `(std::abs(bin) >= maxbin)` check into two separate checks, namely `((bin >= maxbin) || (bin <= -maxbin))`. This fixes the edge case but requires an additional check.

4 RELATED WORK

As lossy compression is a widely researched domain, this section focuses on the lossy floating-point compressors we evaluate in Section 6. Table 1 summarizes these compressors and their support for the widely-used error-bound types. A '✓' indicates that the compressor supports that error-bound type whereas a '○' shows that it does not.

Table 1. All tested compressors and the error-bound types they support at a glance (ordered by initial release date)

Compressor	ABS	REL	NOA	Guaranteed error bound
ZFP	✓	✓	○	○
SZ2	✓	✓	✓	○
SZ3	✓	○	✓	✓
MGARD-X	✓	○	✓	○
SPERR	✓	○	○	○
FZ-GPU	○	○	✓	○
cuSZp	✓	○	✓	○
LC	✓	✓	✓	✓

There are four main versions of SZ. They all use prediction in their compression pipeline. SZ2 [10] employs Lorenzo prediction [5] and linear regression followed by quantization and lossless compression. SZ3 [9, 12, 20] is an improvement that typically produces better compression ratios with similar throughput. It adds preprocessing before the prediction and entropy coding to the lossless compression stage. SZ2 and SZ3 are both CPU-only compressors. As discussed in Section 2, outliers are likely to occur. While LC leaves these outliers in-line, SZ3 uses the '0' bin as a reserved value for outliers, which are grouped outside of the quantized portion. cuSZ [17, 18] is a CUDA implementation that employs a different, more GPU-friendly algorithm. It performs Lorenzo prediction and quantization followed by multi-byte Huffman coding. FZ-GPU [19] is a specialized version of cuSZ that fuses multiple kernels together for better throughput. cuSZp [4] splits the data into blocks and then quantizes and predicts the values in all nonzero blocks. Next, it losslessly compresses the result. Similar to LC, SZ2 and SZ3 control the error by reconstructing the value in the compression stage. They tighten the error bound for values that would otherwise exceed the error bound. FZ-GPU and cuSZp both quantize in the same way that LC does. Unlike LC, however, they do not double-check whether the quantization is within the requested error bound. All versions of SZ support ABS error-bounding, but only SZ2 supports REL error-bounding. They also all support single-precision data, and only FZ-GPU does not support double-precision values.

ZFP [3, 13] is a widely used compression tool that is based on a custom decorrelating transform. 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, performs the aforementioned decorrelation, reorders the data, and converts the values to negabinary representation. Then, it shuffles the bits and losslessly compresses them. ZFP controls the error during the transformation into an integer. The theorem used to support error guarantees assumes infinite precision. Due to this assumption, ZFP is susceptible to floating-point arithmetic errors in some cases. It supports the ABS error-bound and both single- and double-precision data.

MGARD [2, 11] is the only other compressor we found that also supports compatible compression and decompression between CPUs and GPUs. This compressor uses multi-grid hierarchical data refactoring to decompose the data into coefficients with correction factors for reconstruction. The error is controlled during decompression by selectively loading the correct hierarchy of decomposed data based on the requested error bound. It supports the ABS error bound and both single- and double-precision data.

SPERR [7], which is an evolution of SPECK [15], uses advanced wavelet transforms that are applied recursively to the input. SPERR detects outliers that do not meet the error bound and stores correction factors for those values. This correction appears to be susceptible to floating-point arithmetic errors, especially as outliers are refined in further steps. SPERR supports ABS error-bounding and both types of floating-point data.

5 EXPERIMENTAL METHODOLOGY

We evaluated the compressors described in Section 4 on a system based on an AMD Ryzen Threadripper 2950X CPU with 16 cores. Hyperthreading is enabled, that is, the 16 cores can simultaneously run 32 threads. The main memory has a capacity of 64 GB. The operating system is Fedora 37. The 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. The GPU driver version is 525.85.05. The GPU codes were compiled with *nvcc* version 12.0.140 using the "-O3 -arch=sm_89" flags. Unless otherwise specified by the build process, we compiled the C++ codes using the "-O3 -march=native" flags.

When evaluating throughput, we measured the execution time of the compression and decompression functions, excluding any time spent reading the input file, verifying the results, and, for the GPU codes, transferring data to and from the GPU. We run each experiment 9 times and collect the compression ratio, median compression throughput, and median decompression throughput.

Table 2. Information about the used input datasets

Name	**Description**	**Format**	**Files**	**Dimensions**
CESM-ATM	Climate	Single	33	26 × 1800 × 3600
EXAALT Copper	Molecular Dynamics	Single	6	Various 2D
HACC	Cosmology	Single	6	280,953,867
Hurricane ISABEL	Weather Simulation	Single	13	100 × 500 × 500
NYX	Cosmology	Single	6	512 × 512 × 512
QMCPACK	Quantum Monte Carlo	Single	2	33,120 × 69 × 69
SCALE	Climate	Single	12	98 × 1200 × 1200

We used the 7 single-precision suites shown in Table 2 as inputs. They stem from the SDRBench repository [14, 21], which hosts scientific datasets from different domains. For the throughput evaluation, we use only one file from each input set because the performance of our compressor does not change significantly between individual inputs within a suite. For the compression-ratio evaluation, we use all the inputs and report the geometric mean within each suite.

Additionally, we generated sets of single- and double-precision inputs that cover a wide range of values, including positive and negative infinity (INF), not-a-number (NaN), and denormal values, which sometimes cause issues in floating-point compressors. As mentioned, we only test ABS and REL error bounds as NOA is similar to ABS.

We report the throughput and compression results in bar charts where the bars are the metric in question normalized to the non-correctness-guaranteed metric. We use the REL quantizer for the `pow()` and `log()` comparisons because only REL requires these functions. We use the ABS quantizer to evaluate the rounding-error protection.

6 RESULTS

Table 3 summarizes which kinds of values each tested compressor can handle. The results are for ABS only, with the exception of SZ2 and LC, which support both REL and ABS. A '✓' indicates that the compressor successfully handles this kind of value. A '○' shows that the compressor does not guarantee the error bound but also does not crash. Finally, a '×' denotes a crash when supplied that kind of value.

Table 3. Values that meet the error bound. '✓' indicates no issues, '○' indicates error-bound violations, and '×' indicates a crash.

	Single				Double		
Compressor	Normal	INF	NaN	Denormal	INF	NaN	Denormal
ZFP	○	○	○	✓	○	○	✓
SZ2	○	✓	✓	○	✓	✓	○
SZ3	✓	✓	✓	✓	✓	✓	✓
MGARD-X	○	✓	✓	✓	✓	✓	✓
SPERR	○	×	×	✓	×	×	✓
FZ-GPU	○	✓	✓	✓	n/a	n/a	n/a
cuSZp	○	×	✓	✓	×	×	✓
LC	✓	✓	✓	✓	✓	✓	✓

All tested compressors can handle normal values, though most of them do not guarantee the error bound. These error-bound violations are likely due to the rounding issues discussed in Section 2. Every compressor except for SZ2 correctly handles and error-bounds denormal values. The reasons SZ2 does not properly handle these values is due to it being the only compressor (aside from LC) that supports REL. When a small denormal value is bound using REL, it is highly susceptible to rounding issues. Three compressors have problems with INF and NaN, on which SPERR and cuSZp occasionally crash and ZFP is unable to guarantee the error bound.

The table highlights that even state-of-the-art compressors have problems with some values. By implementing the fixes discussed in Section 3, LC is able to avoid crashing or violating the error bound on any of these values. In fact, we exhaustively tested it on all roughly 4 billion possible 32-bit floating-point values with several error bounds to ensure that it handles all values correctly.

Figure 1 and Table 4 show the compression ratio effects of our `pow()` and `log()` replacements in the REL quantizer for an error bound of 1E-3, and Figure 2 and Tables 5 and 6 show the throughput effects. In the figures, the bar height indicates the performance relative to using the library versions of `pow()` and `log()` (higher is better). Each bar represents the geometric mean over all files in one input dataset for the compression ratios. For the throughput results, each bar represents the median GPU throughput for the representative file from that suite.

When switching to our less-accurate but parity-protected approximation functions, the compression ratio is affected negatively, as expected. This drop in compression ratio is due to more values being stored losslessly as they are unable to be quantized within the error bound. Losslessly stored float values are harder to compress in the later stages of LC. While the compression loss is significant at 5.2% on average, without it the compressor would be unable to produce the same result on both CPUs and GPUs. Note that this only affects the REL quantizer as the ABS and NOA quantizers do not use these approximation functions.

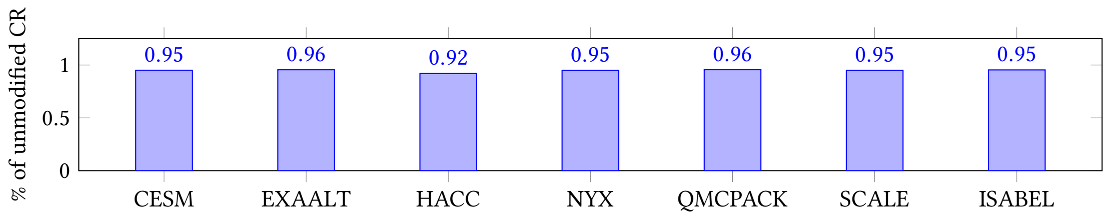

Fig. 1. Compression ratios of the parity-ensured $pow()$ and $log()$ REL compressor normalized to the non-ensured ratios.

Table 4. Compression ratios with and w/o the parity-ensured $pow()$ and $log()$ functions in the REL compressor.

	CESM	EXAALT	HACC	NYX	QMCPACK	SCALE	ISABEL
Original Functions	7.2	3.8	5.1	4.0	2.6	7.4	5.2
Replaced Functions	6.8	3.6	4.7	3.8	2.5	7.1	4.9

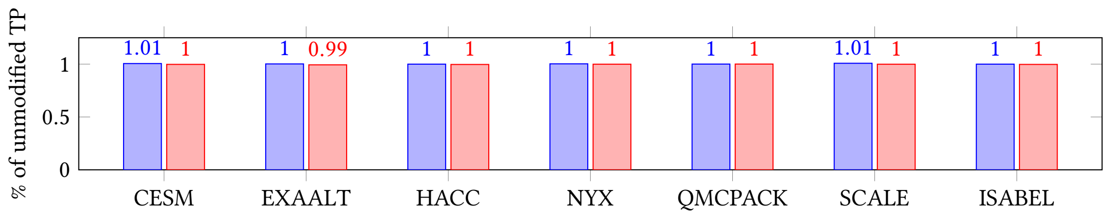

Fig. 2. GPU throughput of the parity-ensured $pow()$ and $log()$ REL compressor normalized to the non-ensured throughputs. Blue bars are compression, red bars are decompression.

Interestingly, the throughput of LC remains within ±1% when switching to our approximate functions. There are several reasons for why there is essentially no change in throughput. First, these functions only represent a small

Table 5. GPU compression throughput in GB/s with and w/o the parity-ensured $pow()$ and $log()$ functions in the REL compressor.

	CESM	EXAALT	HACC	NYX	QMCPACK	SCALE	ISABEL
Original Functions	143.5	146.5	143.0	144.0	141.2	145.2	138.7
Replaced Functions	144.3	146.8	143.0	144.4	141.3	146.4	138.7

Table 6. GPU decompression throughput in GB/s with and w/o the parity-ensured $pow()$ and $log()$ functions in the REL compressor.

	CESM	EXAALT	HACC	NYX	QMCPACK	SCALE	ISABEL
Original Functions	132.8	134.7	131.8	132.0	130.5	133.9	127.4
Replaced Functions	132.5	133.8	131.4	131.9	130.6	133.8	127.1

fraction of the overall execution time. Second, compression and decompression are memory-bound operations that may hide some of the computation latency. Third, the native `pow()` and `log()` are also quite slow and probably not much faster than our approximate functions, which exclusively use fast operations.

Figure 3 and Table 7 show the GPU throughput changes due to the rounding-error protection in our code using the ABS quantizer with an error bound of 1E-3 while Figure 4 and Table 8 show the compression ratio changes. For the bar charts, the height of the bar is normalized to the performance of the code without rounding-error protection. We do not show decompression results as the "double checking" is not present in the decompressor.

The addition of the extra checks to prevent an error-bound violation does not significantly affect throughput. The reasoning is likely the same as above. These checks represent very little of the total runtime and may be hidden under the memory-access latency. A bigger difference is observed in compression ratio. The version of the compressor with the double checking yields ratios that are about 5% worse than the compressor that does not include the check.

Table 9 sheds light on the reason for this loss, where the most pronounced decrease in compression ratio corresponds to the highest percentage of values incurring rounding errors that must be mitigated. The loss in compression ratio is most pronounced in the EXAALT input set, which includes a file where 11.2% of the values fail the verification. Nevertheless, every dataset compresses well, even though they all incur some rounding errors. They still compress well because all values, even the ones found to be non-quantizable due to a rounding error, are compressed losslessly. This helps mitigate the effect of non-quantizable values on compression ratio. Overall, this small loss in compression ratio is the cost of guaranteeing the error bound when floating-point arithmetic is involved.

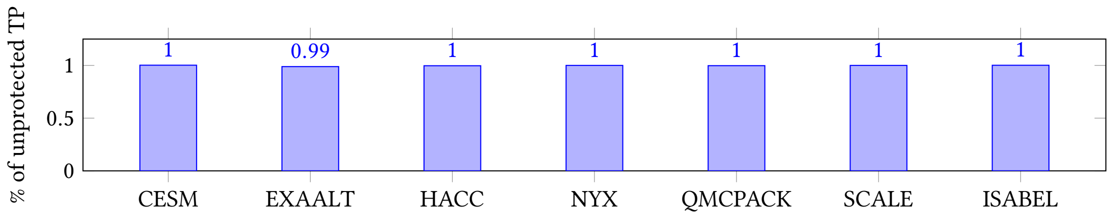

Fig. 3. GPU compression throughput of the rounding-error protected ABS compressor normalized to the non-protected throughput.

In summary, the solutions to the correctness problems we discovered while developing LC do not adversely affect the throughput but do lower the compression ratio noticeably. However, they guarantee the error bound for both ABS and REL (and NOA) and ensure that the CPU and GPU results are bit-for-bit identical.

Table 7. GPU compression throughput in GB/s of the rounding-error protected ABS compressor vs. the non-protected compressor.

	CESM	EXAALT	HACC	NYX	QMCPACK	SCALE	ISABEL
Protected	156.0	145.4	138.9	144.7	143.7	190.6	141.8
Unprotected	155.7	147.1	139.3	144.7	144.0	190.6	141.6

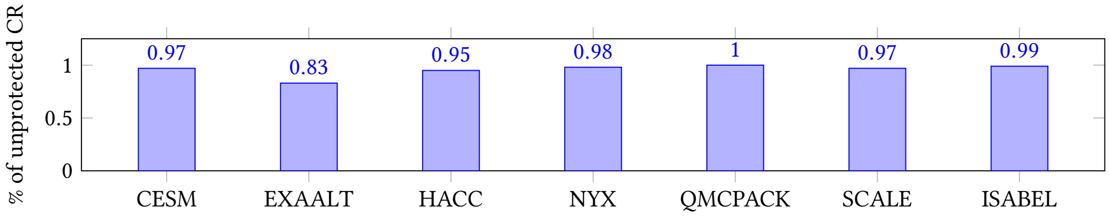

Fig. 4. GPU compression ratio of the rounding-error protected ABS compressor normalized to the non-protected compression ratio.

Table 8. Compression ratio of the rounding-error protected ABS compressor vs. the non-protected compressor.

	CESM	EXAALT	HACC	NYX	QMCPACK	SCALE	ISABEL
Protected	122.0	3.3	2.3	1.9	4.3	81.1	140.8
Unprotected	126.1	4.0	2.4	1.9	4.3	83.8	142.4

Table 9. Percentage of the input values affected by rounding errors in the ABS quantizer.

	Average	Maximum
CESM	0.12%	1.68%
EXAALT	3.41%	11.16%
HACC	0.25%	0.40%
NYX	0.89%	5.29%
QMCPACK	0.00%	0.00%
SCALE	0.16%	1.38%
ISABEL	0.05%	0.63%

7 SUMMARY AND CONCLUSIONS

This paper explores correctness in error-bounded lossy quantizers. We describe problems that affect the ability to guarantee specific error bounds. We show code examples of how we addressed these issues in the LC compression framework we are developing and demonstrate that our fixes do not affect throughput but degrade the compression ratio (5% on average). We hope our solutions will be helpful to others who work on lossy compression and will result in increased availability of guaranteed-error-bounded lossy compressors.

ACKNOWLEDGMENTS

This work has been supported in part by the Department of Energy, Office of Science under Award Number DE-SC0022223 as well as by an equipment donation from NVIDIA Corporation. We thank Sheng Di and Franck Cappello for their invaluable help with domain knowledge during the development of LC.

REFERENCES

[1] Noushin Azami, Alex Fallin, Brandon Burtchell, Andrew Rodriguez, Benila Jerald, Yiqian Liu, and Martin Burtscher. 2024. LC Git Repository. https://github.com/burtscher/LC-framework. Accessed: 2024-04-12.

[2] Jieyang Chen, Lipeng Wan, Xin Liang, Ben Whitney, Qing Liu, David Pugmire, Nicholas Thompson, Jong Youl Choi, Matthew Wolf, Todd Munson, Ian Foster, and Scott Klasky. 2021. Accelerating Multigrid-based Hierarchical Scientific Data Refactoring on GPUs. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 859–868. https://doi.org/10.1109/IPDPS49936.2021.00095

[3] James Diffenderfer, Alyson L. Fox, Jeffrey A. Hittinger, Geoffrey Sanders, and Peter G. Lindstrom. 2019. Error Analysis of ZFP Compression for Floating-Point Data. *SIAM Journal on Scientific Computing* 41, 3 (2019), A1867–A1898. https://doi.org/10.1137/18M1168832 arXiv:https://doi.org/10.1137/18M1168832

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

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

[6] 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. 2015. 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 (2015), 1333 – 1349. https://doi.org/10.1175/BAMS-D-13-00255.1

[7] Shaomeng Li, Peter Lindstrom, and John Clyne. 2023. Lossy Scientific Data Compression With SPERR. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1007–1017. https://doi.org/10.1109/IPDPS54959.2023.00104

[8] Xin Liang, Sheng Di, Dingwen Tao, Zizhong Chen, and Franck Cappello. 2018. An Efficient Transformation Scheme for Lossy Data Compression with Point-Wise Relative Error Bound. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. 179–189. https://doi.org/10.1109/CLUSTER.2018.00036

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

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

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

[12] 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. 2023. SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors. *IEEE Transactions on Big Data* 9, 2 (2023), 485–498. https://doi.org/10.1109/TBDATA.2022.3201176

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

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

[15] Xiaoli Tang and William A. Pearlman. 2006. *Three-Dimensional Wavelet-Based Compression of Hyperspectral Images*. Springer US, Boston, MA, 273–308. https://doi.org/10.1007/0-387-28600-4_10

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

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

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

[19] Boyuan Zhang, Jiannan Tian, Sheng Di, Xiaodong Yu, Yunhe Feng, Xin Liang, Dingwen Tao, and Franck Cappello. 2023. FZ-GPU: A Fast and High-Ratio Lossy Compressor for Scientific Computing Applications on GPUs. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing* (Orlando, FL, USA) *(HPDC '23)*. Association for Computing Machinery, New York, NY, USA, 14 pages. https://doi.org/10.1145/3588195.3592994

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

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