2021 IEEE International Conference on Big Data (Big Data) | 978-1-6654-3902-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/BigData52589.2021.9671954

Improving Lossy Compression for SZ by Exploring the Best-Fit Lossless Compression Techniques

Jinyang Liu, [†] Sihuan Li,[†] Sheng Di,[*] Xin Liang,[‡] Kai Zhao,[†] Dingwen Tao,[§] Zizhong Chen,[†] Franck Cappello[*¶]
[*]Argonne National Laboratory, IL, USA
[†] University of California, Riverside, CA, USA
[‡] Missouri University of Science and Technology, Rolla, MO, USA
[§]Washington State University, Pullman, WA, USA
[¶]University of Illinois at Urbana-Champaign, IL, USA
jliu447@ucr.edu, sli049@ucr.edu, sdi1@anl.gov, xliang@mst.edu, kzhao016@ucr.edu,
dingwen.tao@wsu.edu, chen@cs.ucr.edu, cappello@mcs.anl.gov

Abstract—**In the past decades, various lossy compressors have been studied broadly due to the ever-increasing volume of data being produced by today's scientific applications. SZ has been one of the best error-bounded lossy compressors ever raised, and it has a flexible framework that includes four adjustable steps: prediction, quantization, variable-length encoding, and lossless compression. In this paper, we improve the lossy compression performances of the SZ compression model by exploring different existing lossless compression techniques using the Squash data compression benchmark. Specifically, we first characterize the bytes outputted by the first three steps in SZ, then we investigate the best lossless compressor with different datasets and different error bounds. We perform our exploration by testing 8 widely used lossless compressors under different configurations together with SZ over five well-known scientific simulation datasets. Our experiments show that adopting the best-fit lossless compressor selected based on our analysis can improve the overall compression speed by up to 40% compared to the previous lossless compression technique used in SZ with the comparable quality of reconstructed data.**

I. Introduction

Vast volumes of data are being produced by today's scientific simulations on supercomputers, introducing a big challenge to the data storage not only because of limited storage space but also limited I/O bandwidth of the parallel file system (PFS). The Hardware/Hybrid Accelerated Cosmology Code (HACC) [1], for instance, may produce 60 PB of data to store with up to 3.5 trillion particles to simulate in one simulation. HACC researchers rely on decimation to store data (storing snapshots selectively in the simulation), inevitably losing valuable information for post-analysis. Error-controlled lossy compression techniques have been considered a better solution than the simple decimation method for reducing the data size significantly while guaranteeing that the distortion of compression data is acceptable by users [2]–[5].

According to recent studies, SZ [6]–[8] has been one of the best error-controlled lossy compressors on multiple simulation datasets across different scientific domains. SZ adopts a flexible prediction-based compression model, which includes four adjustable steps: (1) data prediction, (2) error quantization, (3) variable-length encoding, and (4) lossless compression. The initial design of SZ [6], adopts three types of 1D data prediction methods in Step 1 and a 2-bit code to approximate each floating-point value by the best-fit prediction method in Step 2. We further improved the compression quality by developing a multidimensional one-layer prediction method [7], [8] in Step 1 and a linear-scaling quantization method in Step 2. For two-electron integral datasets in quantum chemistry simulation, we customized an effective predictor called Pastri [9] by leveraging the scaled pattern of the dataset to improve the prediction accuracy in Step 1 and developing a lightweight variable-length coding algorithm in Step 3. There are several recent researches which present new derivations of SZ. For example, SZauto [10] introduces second-order predictors into SZ and SZinterp [11] leverages dynamic spline interpolation predictors. However, how to improve the last step, lossless compression, is still a left question in SZ.

In this work, we further improve the lossy compression quality based on the SZ compression model, by exploring the best-fit lossless compression technique. Specifically, our contribution is threefold:

- We characterize the bytes output generated by the first three steps in SZ's compression pipeline.
- Based on a well-known lossless compression benchmark, Squash [12], we analyze different aspects of the lossy compression qualities, for SZ combined with different lossless compression techniques.
- We summarize several takeaways based on our performance characterization. Our experiments show that the overall best-fit lossless compressor for SZ is zstd [13], which can significantly improve the compression/decompression speed with only negligible compression ratio loss compared to the previous lossless compression technique. Specifically, the compression performance of SZ can be improved by up to 40% in most of cases. by using zstd compared with the previous lossless compressor Zlib.

The rest of the paper is organized as follows. In Section II, we briefly review the SZ compression model. In Section

III, we introduce some state-of-the-art lossless compressors as well as their pros and cons. In Section IV, we present the evaluation results using different lossless compressors in the SZ compression model, over multiple real-world simulation datasets. We discuss the related work in Section V. In Section VI, we briefly summarize our conclusions and discuss future work.

II. SZ Lossy Compression Model

SZ [6], [7] is the state-of-the-art error-bounded lossy compressor for significantly reducing the data size of extreme-scale scientific simulations. SZ compression contains four critical steps: (1) value prediction on each data point for the sake of decorrelation, (2) linear-scaling quantization surrounding the predicted value with equal-sized bins, (3) variable-length encoding used to encode the integer indices of the bins, and (4) lossless compression technique to further shrink the data size. In Step 1, SZ performs a single-dimensional or multidimensional prediction for each data point based on its neighbor data points (the dimension of the prediction depends on the dimension of the dataset). A set of consecutive bins with each twice the error bound in length are constructed in Step 2; and the index of the bin containing the real value of the data point, called the *located bin*, are encoded by a customized Huffman encoding in Step 3. Steps 1 and 3 are both lossless procedures, which means that these two steps will not introduce data loss during their corresponding decompression steps. Step 4 adopts some lossless compressor such as Gzip [14] or Zstd [13]. In this paper, we explore many lossless compressors by characterizing their impact on the lossy compression quality, and we select the best-fit lossless compressor for SZ.

III. State-of-the-Art Lossless Compressors

After considering numerous existing lossless compressors, we selected 8 of them, then ran Squash [12] with the combination of SZ and each based on a relatively small set of data. We investigated them thoroughly with more datasets and various error bounds for selecting the best-fit one. In the following, we briefly describe the widely used lossless compressors; more details can be found in [15].

1. *brieflz* [16]: A lightweight implementation of the Lempel-Ziv (LZ) compression algorithm. It focuses on fast compression rate with comparable compression ratio.

2. *compress* [17]: A fast compressor based on the Lempel-Ziv-Welch (LZW) algorithm. It is the de facto file compression standard in the UNIX community.

3. *deflate*: One of the methods in zlib [18] that uses LZ77 and Huffman coding. Its data format is portable across platforms, and it never expands the data as do some of the LZW algorithms, which may double or triple the data size in worst cases. Zlib has been an important component of software platforms such as Linux, MAC OS X, and iOS and even gaming platforms such as PlayStation and Xbox.

4. *fari* [19]: An arithmetic compressor with extremely high compression/decompression speeds.

5. *gipfeli* [20], [21]: A high-speed compression library based on LZ77 and an improved entropy coding instead of relatively slow Huffman or arithmetic coding. *gipfeli* achieves 3X the compression speed of deflate or zlib.

6. *lzfse* [22]: An LZ style compression algorithm using finite-state entropy coding. It has a compression ratio similar to that of deflate or zlib but has better compression and decompression speed. *lzfse* has been open sourced by Apple, and it is in the compression library beginning with iOS 9 and OSX 10.11 El Capitan.

7. *zling* [23]: An implementation of order-1 ROLZ (reduced offset LZ) and Huffman coding.

8. *zstd* [13]: A fast lossless compression using finite-state entropy coding by Facebook.

IV. Exploring Best Lossless Compressor for SZ

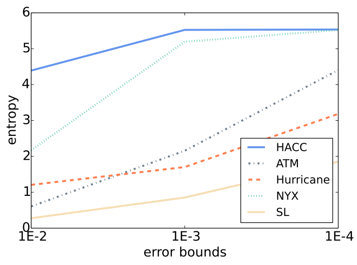
In this section, we investigate compression quality of multiple different lossless compressors and identify the best-fit one(s) for SZ. Our experiments are based on the latest public stable version of SZ, which is SZ2.1 [3], [8] (shorted as SZ in the following text).

A. Characterization of bytes output from SZ's first three steps

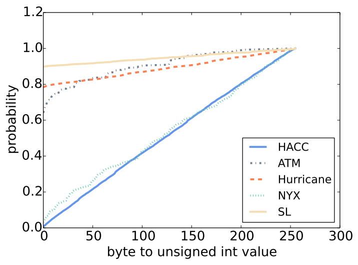
Different byte inputs may have different compressors perform differently, thus we first investigate the characteristics of the input of the lossless compressors (i.e., the output of the former three steps of SZ), by analyzing its entropy, cumulative distribution function (CDF), and autocorrelation. These measurements are calculated based on the output generated by SZ's first three steps, which is an array of bytes (unsigned characters) valuing from 0 to 255. The entropy is used to measure the randomness of the numbers. Higher entropy implies higher randomness, which usually means the data is harder to compress. The CDF shows the distribution of the numbers. Generally, the data with a sharper CDF increase means their distribution is more clustered and thus it should be easier to compress. The autocorrelation coefficient (its value always ranges in [0,1]) indicates the correlation between the array and the array with some lag when the array is treated as time series. A higher autocorrelation value generally implies that the data is easier to compress.

We present the characterization results in Figure 1 (one representative field for each dataset because of space limitations). The representative field selected here has the closest compression ratio to the overall compression ratio on all fields in the dataset. Specifically, the selected representative fields for HACC, ATM, Hurricane, NYX and SCALE-LETKF are *vx*, *FLDSC*, *QICEf48*, *baryon_density* and *QC* respectively. Entropy is plotted by setting the error bound of SZ to [1E-2, 1E-4]. The CDF and autocorrelation are shown only with error bound 1E-3 because of space limitations. The evaluation results of lossless compressors based on the bytes outputted by SZ's first three steps are presented in Figure 2. We summarize two valuable findings as follows.

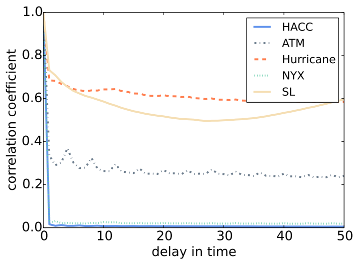
- Some datasets such as SCALE-LETKF, ATM and Hurricane should have some potential to be further com-



(a) Entropy



(b) Cumulative Distribution Function (CDF)



(c) Auto-Correlation

Fig. 1. Characterization of the output by SZ's former 3 steps

pressed by using lossless compressors, compared with other datasets such as HACC or NYX. All results in Figure 1 consistently imply that SCALE-LETKF, ATM and Hurricane are easier to compress than HACC and NYX. This is also verified by results in Figure 2.

- We can see that the entropy of the bytes outputted by SZ's first three steps is relatively low, especially with high error bounds such as 1E-2. This inspires developers to explore a better variable-length encoding (Step 3) in SZ such that the output bytes could already be compressed well after the first three steps.

B. Lossless compressor selection

The Squash benchmark [12] consists of 43 different compression algorithms. Since comparing all the algorithms is not feasible, we first select several good candidates with quick experiments then perform comprehensive experiments to select the best-fit one from among the good candidates. To select the good candidate lossless compressors, we run all 43 compressors on the output of SZ's first three steps on HACC data and select the ones that are ranked in the top 25 for both compression ratio and compression rate. These two factors are critical to a fast and efficient lossy compression. The

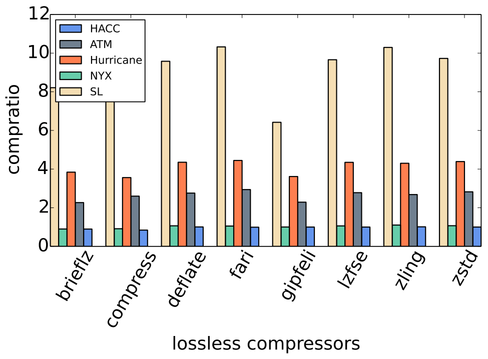


Fig. 2. Lossless compressor compression ratio on the bytes outputted by SZ's first 3 steps on the representative field for each dataset with error bound 1E-3

selection yields 9 good candidates: *brieflz*, *compress*, *deflate*, *fari*, *gipfeli*, *lzfse*, *lzvn*, *zling*, and *zstd*. The compressor *lzvn* was subsequently removed from the candidate list because it cannot work for large input sizes based on our experiments.

The following experiments were all executed based on five scientific datasets: HACC, ATM, Hurricane, NYX and SCALE-LETKF (short for SL). The input of all the 8 selected lossless compressors is the output of SZ's former three steps. We adopt the default compression levels for all the lossless compressors. Table I presents the fundamental information about the datasets. The data type for all data is single precision floating point number.

TABLE I
DATASET INFORMATION

	#Fields	Dimensions	Total Size (GB)
HACC	6	1D: 280,953,867	6.3
ATM	77	2D: 1800 X 3600	1.9
Hurricane	13	3D: 100 X 500 X 500	1.2
NYX	6	3D: 512 X 512 X 512	3
SL	6	3D: 98 X 1200 X 1200	3.2

C. Lossless compressor performance

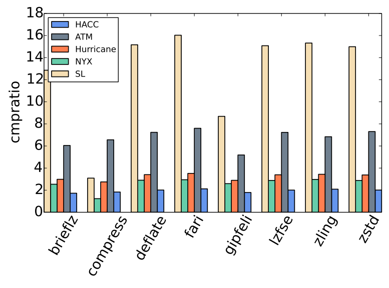
1) Compression ratio: Based on Figure 3, we can observe that under the error bound of 1E-2, the compressor *fari* has the best overall compression ratio, followed by *zling*, *deflate*, and *zstd*. Their compression ratios decrease with decreasing error bounds from 1E-2 to 1E-4.

2) Compression rate: Based on Figure 4, we notice that the compressor *zstd* has the best compression rate, followed by *gipfeli*, *brieflz*, and *deflate*. In absolute terms, zstd is 50% $\sim$ 300% faster than Deflate.

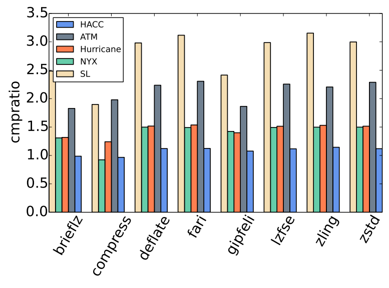
3) Decompression rate: Based on Figure 5, we can see that *zstd* has the best decompression rate except for SL data under error bound 1E-2, followed by *deflate*, *lzfse*, and *gipfeli*.

D. Improvement to SZ

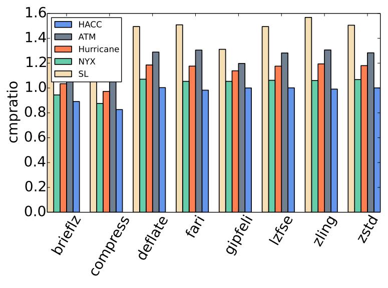
The preceding evaluation focuses on the individual performance of the lossless compressors. In this section, we investigate the overall improvement for the whole compression of SZ



(a) error bound = 1E-2

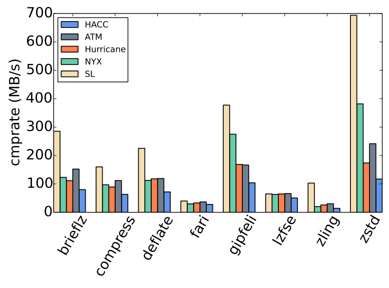


(b) error bound = 1E-3

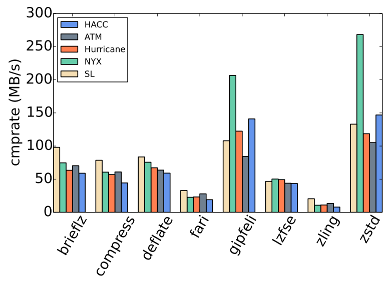


(c) error bound = 1E-4

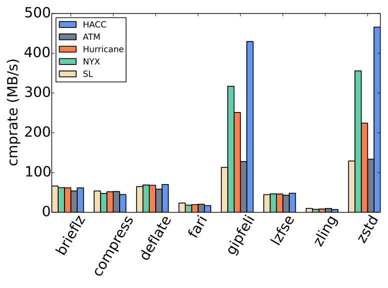
Fig. 3. Lossless compression ratios with varied error bounds of SZ



(a) error bound = 1E-2

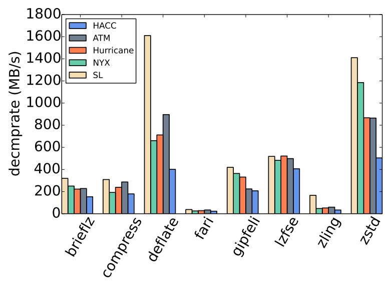


(b) error bound = 1E-3

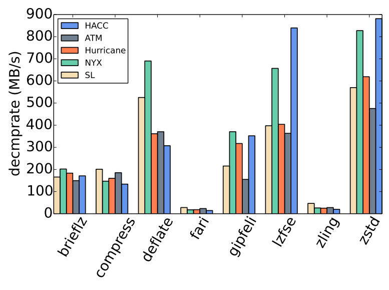


(c) error bound = 1E-4

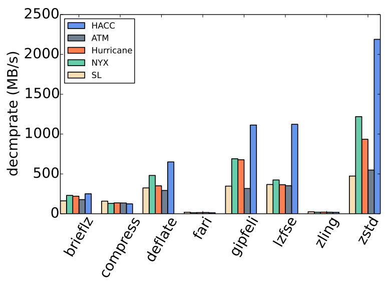
Fig. 4. Lossless compression rate with varied error bounds of SZ



(a) error bound = 1E-2



(b) error bound = 1E-3



(c) error bound = 1E-4

Fig. 5. Decompression rate with varied error bounds of SZ

by using different lossless compressors. The relationship of the compression performance between lossless compressors and SZ is as follows. If the compression ratio, compression rate, and decompression rate of SZ's first three steps are denoted as r_0, c_0, and d_0 respectively, and the corresponding metrics of the lossless compressors are r_1, c_1 and d_1 respectively, then we can derive the overall compression ratio, compression rate, and decompression rate of SZ as follows.

$$r = r_0 r_1 \tag{1a}$$

$$c = \frac{c_0 c_1 r_0}{r_0 c_1 + c_0} \tag{1b}$$

$$d = \frac{d_0 d_1 r_0}{r_0 d_1 + d_0} \tag{1c}$$

With Equation (1), we can evaluate the overall performances based on the performances of SZ's first three steps and the performances of lossless compressors without integrating the lossless compressors in SZ physically. In our experiments, we run SZ's first three steps on different datasets and measure r_0, c_0, and d_0. Then, we treat the output by SZ's first three steps as the input to the 8 lossless compressors, in order to measure the lossless compressors' r_1, c_1, and d_1. The overall compression performance is calculated by Equation (1). The compression ratio r_0, compression rate c_0, and decompression rate d_0 of the first three steps of SZ are listed in Table II. The overall improvement to SZ without a lossless compressor is listed in Table III to Table VII for the five datasets. If the value in the table is negative, it means performance degradation in percentage. Observing the tables, we see that although *fari* and *zling* have the best compression ratios, they incur too much computation overheads on SZ in terms of both

TABLE II
PERFORMANCE OF SZ'S FIRST 3 STEPS WITH DIFFERENT ERROR BOUNDS (COMPRESSION AND DECOMPRESSION RATE MEASURED IN MB/S)

	1E-2			1E-3			1E-4		
	r_0	c_0	d_0	r_0	c_0	d_0	r_0	c_0	d_0
HACC	15.72	181	324	8.84	173	193	5.06	163	127
ATM	19.82	200	419	16.31	162	338	11.66	143	249
Hurricane	16.17	198	400	12.18	167	305	8.71	139	203
NYX	14.90	201	370	11.23	174	269	7.29	139	169
SL	20.08	191	446	15.00	168	368	14.82	142	283

TABLE III
OVERALL IMPROVEMENT PERCENTAGE TO SZ WITH DIFFERENT LOSSLESS COMPRESSORS ON HACC DATA

	1E-2			1E-3			1E-4		
	r	c	d	r	c	d	r	c	d
brieflz	73	-13	-12	-1	-25	-11	-1	-34	-9
compress	83	-15	-10	-3	-31	-14	-2	-42	-17
deflate	**101**	**-14**	**-5**	**12**	**-25**	**-7**	**0**	**-32**	**-4**
fari	112	-29	-47	12	-51	-60	-2	-66	-67
gipfeli	79	-10	-9	8	-12	-6	0	-7	-2
lzfse	100	-19	-5	11	-31	-3	0	-40	-2
zling	109	-45	-38	14	-71	-52	-1	-82	-58
zstd	**101**	**-9**	**-4**	**12**	**-12**	**-2**	**0**	**-6**	**-1**

compression rate and decompression rate. The *zstd* and *deflate* have similar compression ratios and are much faster than *fari* and *zling*. Comparing *zstd* with *deflate*, we observe that they have very close compression ratios but *zstd* is always faster than *deflate*. Moreover, the gap of compression rate (or decompression rate) between *zstd* and *deflate* increases as the error bound decreases, as highlighted in the table. We therefore set *zstd* as our default lossless compressors in SZ because of its comparable performance in compression ratio and much higher compression rate on all datasets.

In addition, we present two other interesting findings.

1. We found if the former 3 steps have better compression ratios on dataset A than B, then the lossless compressor will have better compression ratios on A than B as well. Specifically, SL, ATM, Hurricane, NYX, HACC are ordered by increasing difficulties for lossless/lossy compression.

2. For the 5 lossless compressors (deflate, fari, lzfse, zling, zstd) with top compression ratios and the 5 datasets, if compressor A has a better compression ratio than compressor B has on dataset D1, it will have a no worse compression ratio than B on dataset D2. This tells us that it is not necessary to select a lossless compressor dynamically for different datasets.

V. RELATED WORK

Error-bounded lossy compressors have been studied for years. In general, such compressors are designed based on either a transform-based compression model or a prediction-based compression model. ZFP [24] is a typical example designed in terms of the transform-based model. It splits the whole data into non-overlapped blocks with an edge size of 4, performs a data transform for each block, and then extracts the most important bits from the transformed data by an embedded coding. FPZIP [25], for example, adopts the Lorenzo [26] predictor to predict each data point in a dataset based on its

TABLE IV
OVERALL IMPROVEMENT PERCENTAGE TO SZ WITH DIFFERENT LOSSLESS COMPRESSORS ON ATM DATA

	1E-2			1E-3			1E-4		
	r	c	d	r	c	d	r	c	d
brieflz	505	-2	-8	83	-12	-12	11	-19	-11
compress	556	-8	-7	98	-14	-10	9	-19	-14
deflate	**624**	**-8**	**-2**	**124**	**-14**	**-5**	**29**	**-17**	**-7**
fari	660	-22	-38	131	-26	-47	30	-38	-58
gipfeli	419	-6	-9	86	-11	-12	20	-9	-6
lzfse	623	-13	-4	126	-18	-5	28	-22	-6
zling	584	-25	-26	121	-42	-43	31	-56	-52
zstd	**630**	**-4**	**-2**	**129**	**-7**	**-4**	**28**	**-8**	**-4**

TABLE V
OVERALL IMPROVEMENT PERCENTAGE TO SZ WITH DIFFERENT LOSSLESS COMPRESSORS ON HURRICANE DATA

	1E-2			1E-3			1E-4		
	r	c	d	r	c	d	r	c	d
brieflz	198	-10	-10	32	-18	-12	3	-21	-10
compress	175	-12	-9	24	-19	-14	-3	-24	-15
deflate	**241**	**-9**	**-3**	**52**	**-17**	**-6**	**19**	**-19**	**-6**
fari	252	-27	-47	54	-37	-58	18	-45	-61
gipfeli	189	-7	-7	40	-10	-7	14	-6	-3
lzfse	239	-16	-5	51	-22	-6	18	-26	-6
zling	244	-32	-32	53	-55	-50	19	-66	-54
zstd	**238**	**-7**	**-3**	**51**	**-10**	**-4**	**18**	**-7**	**-2**

neighboring data values and then shrinks the data size by truncating the insignificant mantissa bits for the difference between the original value and predicted value. MGARD [27] is another example of error-bounded lossy compressor which has a hierarchical structure of data prediction. SZ [6]–[8] involves four critical steps, which mainly predict each data point by the Lorenzo [26] predictor and perform a linear-scaling quantization method and a customized Huffman encoding algorithm to reduce the data based on user-set error bound. In our research, we focus on the SZ compression framwork because of its loosely coupled design. For example, the prediction method could be customized by users based on specific datasets, and the Huffman encoding could also be replaced by other variable-length encoding algorithms based on specific data features. Several new derivations from SZ are SZauto [10] which includes second-order predictors, and SZinterp [11] leveraging dynamic spline interpolations. In this paper, we explore the best-fit lossless compression techniques for the SZ compression model in terms of different scientific datasets.

VI. CONCLUSION AND FUTURE WORK

In this paper, we evaluate 8 widely used lossless compressors on the bytes generated by SZ's first three steps on five scientific datasets, in order to investigate the best lossless compressor for SZ. We present the following findings that are very helpful in improving SZ's overall performance with respect to compression ratio, compression rate, and decompression rate.

- The best-fit lossless compressor is *zstd* for all the datasets in different error bounds: it has similar compression ratios

TABLE VI
OVERALL IMPROVEMENT PERCENTAGE TO SZ WITH DIFFERENT LOSSLESS COMPRESSORS ON NYX DATA

	1E-2			1E-3			1E-4		
	r	*c*	*d*	*r*	*c*	*d*	*r*	*c*	*d*
brieflz	154	-10	-9	31	-17	-11	-6	-24	-9
compress	23	-12	-11	-8	-20	-14	-12	-29	-15
deflate	**190**	**-11**	**-4**	**50**	**-17**	**-3**	**7**	**-22**	**-5**
fari	195	-31	-50	49	-41	-57	5	-51	-63
gipfeli	159	-5	-6	42	-7	-6	5	-6	-3
lzfse	188	-18	-5	49	-24	-4	6	-29	-5
zling	197	-40	-34	50	-59	-48	6	-72	-55
zstd	**188**	**-3**	**-2**	**50**	**-5**	**-3**	**7**	**-5**	**-2**

TABLE VII
OVERALL IMPROVEMENT PERCENTAGE TO SZ WITH DIFFERENT LOSSLESS COMPRESSORS ON SL DATA

	1E-2			1E-3			1E-4		
	r	*c*	*d*	*r*	*c*	*d*	*r*	*c*	*d*
brieflz	1186	-3	-6	148	-10	-13	24	-13	-11
compress	209	-6	-7	90	-12	-11	30	-15	-11
deflate	**1416**	**-4**	**-1**	**198**	**-12**	**-4**	**49**	**-13**	**-6**
fari	1503	-19	-37	212	-25	-47	51	-29	-51
gipfeli	768	-2	-5	141	-9	-10	31	-8	-5
lzfse	1408	-13	-4	199	-19	-6	49	-18	-5
zling	1432	-8	-12	215	-35	-34	57	-49	-43
zstd	**1399**	**-1**	**-2**	**200**	**-8**	**-4**	**51**	**-7**	**-4**

to those of zlib but improves the overall compression rate by up to 40% on HACC data under error bound of 1E-4.
- Characterization of the bytes outputted by SZ's first three steps shows that the bytes are not fully compressed on the third step of SZ.

We plan to the explore more effective encoding or compression techniques for SZ's third step, because the current last step can improve the compression significantly in some cases, meaning that the third step has much potential for improvement. We will also investigate more scientific datasets to check whether *zstd* is always the best lossless compressor.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation's exascale computing imperative. The material was supported by the U.S. Department of Energy (DOE), Office of Science and DOE Advanced Scientific Computing Research (ASCR) office, under contract DE-AC02-06CH11357, and supported by the National Science Foundation under Grant OAC-2003709, OAC-2003624/2042084, SHF-1617488, and OAC-2104023. We acknowledge the computing resources provided on Bebop, which is operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

REFERENCES

[1] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, V. Vishwanath, T. Peterka, J. Insley *et al.*, "Hacc: extreme scaling and performance across diverse architectures," *Communications of the ACM*, vol. 60, no. 1, pp. 97–104, 2016.
[2] D. Tao, S. Di, Z. Chen, and F. Cappello, "Exploration of pattern-matching techniques for lossy compression on cosmology simulation data sets," in *DRBSD2017*, 2017, pp. 43–54.
[3] ——, "In-depth exploration of single-snapshot lossy compression techniques for N-body simulations," in *2017 IEEE International Conference on Big Data*. IEEE, 2017, pp. 486–493.
[4] ——, "In-depth exploration of single-snapshot lossy compression techniques for n-body simulations," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017, pp. 486–493.
[5] D. Tao, S. Di, X. Liang, Z. Chen, and F. Cappello, "Improving performance of iterative methods by lossy checkponting," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 52–65.
[6] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *IEEE International Parallel and Distributed Processing Symposium (IEEE IPDPS)*. IEEE, 2016, pp. 730–739.
[7] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *IEEE International Parallel and Distributed Processing Symposium (IEEE IPDPS)*. IEEE, 2017, pp. 1129–1139.
[8] X. Liang, S. Di, D. Tao, S. Li, S. Li, H. Guo, Z. Chen, and F. Cappello, "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," in *2018 IEEE International Conference on Big Data*. IEEE, 2018.
[9] A. M. Gok, S. Di, Y. Alexeev, D. Tao, V. Mironov, and F. Cappello, "Pastri: Error-bounded lossy compression for two-electron integrals in quantum chemistry," in *IEEE International Conference on Cluster Computing (IEEE Cluster)*, 2018, pp. 1–11.
[10] K. Zhao *et al.*, "Significantly improving lossy compression for HPC datasets with second-order prediction and parameter optimization," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20, 2020, pp. 89–100.
[11] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cappello, "Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 1643–1654.
[12] Squash Compression Benchmark, https://quixdb.github.io/squash-benchmark/, online.
[13] Zstd, https://github.com/facebook/zstd/releases, online.
[14] L. P. Deutsch, "Gzip file format specification version 4.3," 1996.
[15] M. Mahoney, "Data compression explained," *mattmahoney. net, updated May*, vol. 7, p. 1, 2012.
[16] brieflz, https://github.com/jibsen/brieflz, online.
[17] compress, https://github.com/vapier/ncompress, online.
[18] deflate, http://www.zlib.net/, online.
[19] fari, https://github.com/davidcatt/FastARI, online.
[20] gipfeli code, https://github.com/google/gipfeli, online.
[21] R. Lenhardt and J. Alakuijala, "Gipfeli – high speed compression algorithm," in *Data Compression Conference (DCC), 2012*. IEEE, 2012, pp. 109–118.
[22] lzfse, https://github.com/lzfse/lzfse, online.
[23] zling, https://github.com/richox/libzling, online.
[24] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.
[25] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.
[26] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large n-dimensional scalar fields," in *Computer Graphics Forum*, vol. 22, no. 3. Wiley Online Library, 2003, pp. 343–348.
[27] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the univariate case," *Computing and Visualization in Science*, vol. 19, no. 5-6, pp. 65–76, 2018.