Productive, Performant, and Parallel Generic Lossy Data Compression with LibPressio

VICTORIANA MALVOSO and JON CALHOUN (ADVISOR), Clemson University, USA

Data compression is vital in scientific applications because it reduces the size of files. This is important because running experiments on the data and storing it requires significantly less time and memory. In an effort to create more efficiency, LibPressio was created as a single interface to utilize multiple different compressors. This interface eliminates the need for generating large amounts of code to be able to switch between compressors. This is useful because time spent editing code could be used running more experiments. LibPressio also presents options for getting many different types of metrics so that they are easy to enable. The goal of our testing is to evaluate metrics when using LibPressio as compared to not using it to determine performance. We evaluate how much LibPressio is able to reduce the size of the original file, as well as how long it takes to compress the data. We also evaluate the parallel performance of LibPressio. These are assessed in order to determine if the abstraction improves efficiency.

CCS Concepts: • **Theory of computation** → **Data compression**; • **Computing methodologies** → *Machine learning*; *Massively parallel and high-performance simulations.*

Additional Key Words and Phrases: Data Compression, LibPressio, Error Bounded Lossy Compression, SZ, ZFP

ACM Reference Format:
Victoriana Malvoso and Jon Calhoun (Advisor). 2021. Productive, Performant, and Parallel Generic Lossy Data Compression with LibPressio. In *St. Louis '21: The International Conference for High Performance Computing, Networking, Storage, and Analysis, November 16–18, 2021, St. Louis, MO.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In scientific experiments, generally large amounts of data are produced. Compression can reduce the costs of processing, storing, and calculating this data [1]. However, determining the best compressor to use to compress the data can be time-consuming, as not all compression techniques are appropriate for all data sets. This would require code changes and trial and error to generate the best metrics. Abstractions such as LibPressio reduce the time required for trial and error and minimize the amount of code changes a scientist will need to make in order to utilize new and improved compression techniques. In our work we analyze the impact that the abstraction LibPressio has on runtime, compression ratio, and scalability of compression. LibPressio is a library that allows for the user to easily switch between compressors and analyze their performances. LibPressio allows enabling general utilities for compression such as parallel run times. For many compressors such as MGARD, there is no parallel CPU implementation. Writing custom parallel implementations for each compressor requires more engineering effort, and includes unnecessary code duplication. However, LibPressio has a generic parallel CPU implementation for all of the thread-safe compressors that it supports. Using this may not be best in all cases, but is especially beneficial for thread-safe compressors that currently have no parallel CPU implementation. Since SZ and ZFP already have threading options, these are compared against LibPressio's

Manuscript submitted to ACM

threading implementation for SZ and ZFP to evaluate LibPressio's multi-threaded performance. This paper makes the following contributions:

- Establishes that LibPressio runtime is similar if not better than using SZ, ZFP, or MGARD natively.
- Establishes that LibPressio saves developer time and effort by providing an easy-to-use interface.
- Analyzes scalability of the chunking and threading implementation of LibPressio.

2 BACKGROUND

As technology advances, data collection will become even more thorough, leading to large volumes of data to be processed. Error-Bounded Lossy Compression is useful for large-scale scientific applications because it can generate high compression ratios without distorting past a given error bound. Examples of Error-Bounded Lossy Compression include SZ and ZFP. SZ is prediction based. It has two predictors: the Lorenzo predictor and the regression based predictor. The Lorenzo predictor makes predictions of the next value based on a constant, linear, and quadratic fit of the previous points in n-dimensions. The regression based predictor makes predictions based on a least-squares hyperplane fit through the points in a given block [4]. ZFP is transform based, performing a near orthogonal transform on the data similarly to what JPEG does [5]. However, JPEG is used primarily for image compression.

3 THREAD-SAFE IMPLEMENTATION

The motivation for this testing began with our creation of a thread-safe version of SZ in LibPressio. This was completed in collaboration with the creators of SZ. This included edits to SZ and the LibPressio SZ plugin. We also identified unnecessary overhead with LibPressio's ZFP implementation in collaboration with the creator of LibPressio.

4 EXPERIMENTAL RESULTS

These experiments were run using the computing environment in Table 1.

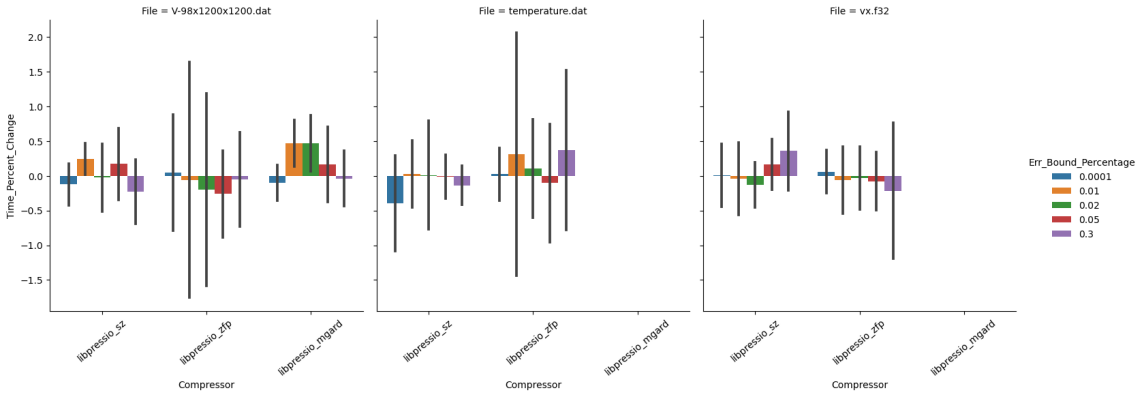

Fig. 1. Time Percent Change comparisons for SZ, ZFP, and MGARD with SDRBench data.

4.1 Serial Performance Test

In this experiment, LibPressio's serial performance is evaluated to determine any timing overhead and difference in compression ratio due to the abstraction. For each compressor, three files from the SDRBench datasets are compressed.

Component	Description	Component	Version
CPU	Intel Xeon 6148G (40 Cores)	Compiler	GCC 8.3.1
RAM	372 GB	OS	CentOS 8.2.2004
Interconnect	100 GB/s HDR Infiniband	MPI	OpenMPI 4.1.1
MGARD	2020-10-01	SZ	v2.1.12
ZFP	v0.5.5	LibPressio	0.70.4

Table 1. Hardware and Software Details. Edits are required to ZFP in order to measure timing using clock_gettime() and CLOCK_MONOTONIC

These are measured with 0.01%, 1%, 2%, 5%, and 30% error bounds, with each configuration being run 30 times. As shown in Figure 1, the time difference is very minimal for each configuration. The minimal change in time is promising, because the reduction in time required to switch between compressors along with low overhead supports the efficiency and usability of LibPressio.

4.2 Parallel Performance Test

Modern super computing nodes have multiple cores, but not all compressors take advantage of this, leading to slower run times. LibPressio is able to parallelize thread-safe compression routines without requiring scientist to generate custom threading implementations. This experiment evaluates LibPressio's generic approach by comparing it to SZ and ZFP's parallel implementations by testing timing speedup and compression ratio. MGARD does not have a threading implementation, and is limited to three-dimensional data sets. The purpose of this experiment is to evaluate the efficiency of LibPressio's performance to support whether or not the abstraction has low overhead. One file is used, with three different error bounds: 1e-6, 1e-4, and 1e-2. Multiple thread counts were used, and the dataset was divided into an equal number of equally sized, contiguous regions. In Figure 2, as the number of threads increases and the chunk size increases, ZFP with LibPressio threads is faster than ZFP with ZFP threads. However,in Table 2, the compression ratio is less than 1, meaning the uncompressed size is smaller than the compressed size. This shows that at a certain chunk size, the performance begins to fail. This can be explained by ZFP's method of compression. It compresses in 4x4 blocks. If the chunk size is not a multiple of 4, ZFP will pad the dataset with zeros until the groups are multiples of 4. This effectively increases the size of the data, leading to a compression ratio below 1. However, at a chunk size of 14, LibPressio threading is faster and the compression ratio is only 13% less than ZFP's threading ratio. This minimal performance difference with an increase in speed without the need for writing threading code for each compressor makes the abstraction worth utilizing.

5 RELATED WORK

Works previous to this such as Foresight offer compression abstractions similar to LibPressio. However, LibPressio respects dimensionality, which gives more accuracy to the compression ratios reported than those by [2]. Another similar work includes the Scientific Compression Library (SCIL), which is another error-bounded lossy compressor interface that allows users to identify the compressor to use [3]. LibPressio differs because it allows the user to enable the parallel implementation for thread-safe compressors rather than a fixed pipeline.

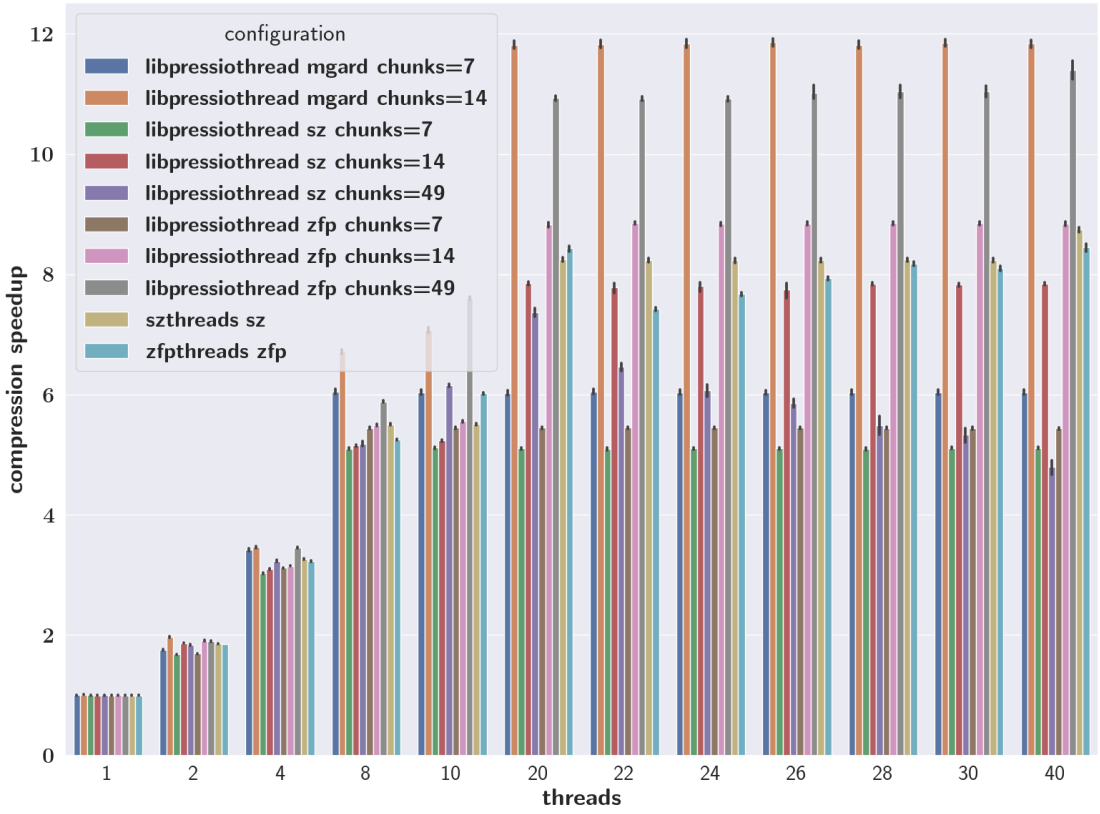

Fig. 2. Compression parallel speedup results at various chunk sizes and numbers of threads.

configuration	chunksize	compression_ratio
libpressiothread MGARD	7	1.389971
	14	1.435876
libpressiothread sz	7	2.245248
	14	2.197810
	49	1.887099
libpressiothread zfp	7	1.427047
	14	1.377799
	49	0.830169
szthreads sz	1	2.256079
zfpthreads zfp	1	1.589013

Table 2. Compression Ratio of different parallel implementations, error_bound=1e-6, 7 chunks, SCALE-LETKF V dataset, chunksize=49 not supported for MGARD due to minimum data size requirements in MGARD

6 CONCLUSIONS

Based on the results of the serial and parallel tests, LibPressio maintains good performance on average while removing the need to write application code for each compressor. There are trade-offs to using LibPressio based on the compressor configuration, but ultimately LibPressio saves time implementing experiments.

ACKNOWLEDGMENTS

Clemson University is acknowledged for generous allotment of compute time on the Palmetto cluster. This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197.

REFERENCES

[1] Franck Cappello, Sheng Di, Sihuan Li, Xin Liang, Ali Murat Gok, Dingwen Tao, Chun Hong Yoon, Xin-Chuan Wu, Yuri Alexeev, and Frederic T Chong. 2019. Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications* 33, 6 (2019), 1201–1220.

[2] Pascal Grosset, Christopher M Biwer, Jesus Pulido, Arvind T Mohan, Ayan Biswas, John Patchett, Terece L Turton, David H Rogers, Daniel Livescu, and James Ahrens. 2020. Foresight: analysis that matters for data reduction. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[3] Julian Martin Kunkel, Anastasiia Novikova, and Eugen Betke. 2017. Towards decoupling the selection of compression algorithms from quality constraints–an investigation of lossy compression efficiency. *Supercomputing Frontiers and Innovations* 4, 4 (2017), 17–33.

[4] 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)*. IEEE, 438–447.

[5] Peter Lindstrom. 2014. Fixed-rate compressed floating-point arrays. *IEEE transactions on visualization and computer graphics* 20, 12 (2014), 2674–2683.