Journal of Research High School

Vol. 2024 (7) 252 – 257
ISSN 2688-3651 [online]

Understanding an Extrapolation-based Lossy Compression for Floating-Point Scientific Data

Michael Middlezong,[1] *

[1]Bergen County Academies, Hackensack, NJ, USA
*Corresponding Author: mmiddlezong@gmail.com

Advisor: Dr. Qing Liu, qing.liu@njit.edu

Received October 25, 2023; Revised March 29, 2024; Accepted, May 2, 2024

Abstract

Large-scale scientific instruments generate increasingly large amounts for data analysis. Data compression can reduce the data cost by reducing the size of data saved on disks. This paper explores an extrapolation-based compression algorithm commonly used by many state-of-the-art compressors, investigating its performance. The paper develops two extrapolation schemes in C++, linear and piecewise extrapolation, and evaluates their performance on three real datasets. The key finding of this paper is that extrapolation effectively exposes data redundancy, which means data can be better compressed further. However, the best extrapolation scheme to use depends on data characteristics and needs to be chosen carefully to achieve the best performance.

Keywords: Lossy compression, Extrapolation, Data management

1. Introduction

Large-scale scientific instruments (e.g., supercomputers running calculations across many compute nodes or experimental facilities with high-fidelity sensors) can generate large amounts of data in the pursuit of knowledge. This data often needs to be transported to file systems for long-term storage and later retrieved quickly so that data analysis can be performed in a timely fashion. As compute increases and scientific instruments become more advanced, data input/output (I/O) becomes one of the biggest bottlenecks in end-to-end knowledge discovery. For this reason, research communities have recently developed new data reduction techniques to reduce the amount of data that must be transported to file systems. For example, SZ (Di & Cappello, 2016) uses extrapolation and regression to perform best-fit compression, while MGARD (Chen et al., 2021) uses interpolation along with an orthogonal projection to smooth out the coefficients. Nevertheless, there has been a lack of in-depth study of the design space for interpolation/extrapolation in compression, and more importantly how different schemes can affect downstream operations in compression. For example, in MGARD, only a multilinear interpolation is implemented, and other interpolation schemes have not been further implemented and evaluated. A suboptimal extrapolation will not fully expose the redundancy in data and will affect the effectiveness of the backend entropy encoder, which is Huffman encoding in MGARD.

The idea of compressing floating-point data is to leverage the local smoothness in data. The method employed is to use a form of prediction to transform data to another form that is more compressible. When data points are predicted from earlier ones, it is called extrapolation. This work aims to study the critical component of extrapolation in compression and understand its effectiveness in smoothing out the data. The defined research objectives are:

- To evaluate the effectiveness of extrapolation in lossy compression.
- To evaluate the effectiveness of different types of extrapolation schemes.

The hypothesis is that extrapolation will effectively expose the redundancy of a dataset and result in higher compression ratios when compared to a compression algorithm without any prediction. In addition, different methods

Vol. 2024 (7) 252 – 257
ISSN 2688-3651 [online]

of extrapolation will have varying effectiveness across different datasets.

The following subsections aim to provide a brief overview of the technical background necessary for the rest of this paper.

1.1 Extrapolation

Extrapolation is the prediction of unknown values based on other known values. An input dataset can be represented by a sequence of 1D floating-point values. Given two adjacent values x_i and x_{i+1} in a sequence, linear extrapolation predicts the value for x_{i+2}, denoted as x'_{i+2}, as follows:

$$x'_{i+2} = x_{i+1} + (x_{i+1} - x_i).$$

Piecewise extrapolation is more naive and predicts the next value as being the current value. Given x_i, the next term is predicted as such:

$$x'_{i+1} = x_i.$$

Overall, the goal of extrapolation is to minimize the difference between the predicted value and the actual value. During compression, the differences, instead of the actual values, are stored, and intuitively they are more compressible than the original values. For decompression, the same extrapolation is performed, and the stored difference is added to recover the original value.

Below, the process is described using linear extrapolation. If one would like to compress input data points x_i, x_{i+1}, x_{i+2}, one can extrapolate x_{i+2} based on the previous two values as follows:

$$x'_{i+2} = x_{i+1} + (x_{i+1} - x_i).$$

Then, it suffices to store (and compress) the current term's difference $\Delta_{i+2} = x_{i+2} - x'_{i+2}$.This is because in the reconstruction step, a value for x_{i+2} can be obtained by adding on the difference:

$$x_{i+2} = x'_{i+2} + \Delta_{i+2} = x_{i+1} + (x_{i+1} - x_i) + \Delta_{i+2}.$$

Therefore, given the data points x_i and x_{i+1}, it is sufficient to store (and compress) Δ_{i+2} instead of x_{i+2}. Compressing the difference instead of the data point itself reduces the magnitude of the values to be compressed (assuming the extrapolation can predict the next value well). This makes it more likely that values are close to each other. This will increase the redundancy in the data after the quantization step (described next), leading to a higher compression ratio.

1.2 Quantization

The purpose of quantization is to convert the output from the extrapolation step into integers, as known as quantization levels, so that those values that are close to each other can be represented by the same level. As such, the output of quantization can be well compressed using Huffman encoding to remove the redundancy. However, information loss will be introduced in this step. To control the information loss, the user can set the maximum allowed relative error (also called the relative error bound), denoted as r, to ensure values after decompression are within a certain range from the original values.

The relative error is a fraction of the range of the data. The range R of a sequence of values $\{x_i\}$ is the minimum subtracted from the maximum:

$$\mathrm{R} = \max\{x_i\} - \min\{x_i\}$$

From this, the algorithm calculates the absolute error bound, denoted as e, such that values within e of the original values satisfy the relative error bound:

$$e = rR$$

During quantization, a floating-point number x is mapped to an integer n according to the following function:

$$n = round\left(\frac{x}{2e}\right)$$

Using this formula, values that are close to each other get mapped to the same integer. More precisely, the set of floating-point values that could potentially be mapped to the integer n is the interval $[2en - e, 2en + e)$. During decompression, the following function is used to map an integer n to a quantized floating-point value denoted as x' such that $|x' - x| \leq e$ always holds:

$$x' = 2\text{en}$$

Therefore, this quantization process ensures that the absolute error bound e is respected, which in turn implies that the relative error bound is also respected.

2. Methods

Our analysis uses the following compression algorithm. As referenced in Figure 1, our implementation of Huffman encoding takes in a vector of integers as input. Then, a frequency map is created from the vector. Using the frequency map, the algorithm generates a Huffman tree by first placing every (integer, frequency) pair into a priority queue sorted by frequency in ascending order. Then, until there is only one element left in the priority queue, the following process is repeated. Two elements are dequeued and a binary tree is created with the two dequeued elements as children. The new tree is then inserted back into the priority queue, with its frequency being the sum of the two frequencies of the children. The remaining element is the final Huffman tree. This Huffman tree can be traversed to generate a code table that maps each unique integer to a string of 0s and 1s. Using the code table, the original vector of integers is encoded into a compressed, binary format and saved on disk, along with a buffer to store the Huffman tree for decompression. The time complexity of Huffman encoding is $O(n \log n)$, where n is the length of the input vector (Morris, 1998). In practice, compression throughput depends more on factors such as the relative error bound and data characteristics.

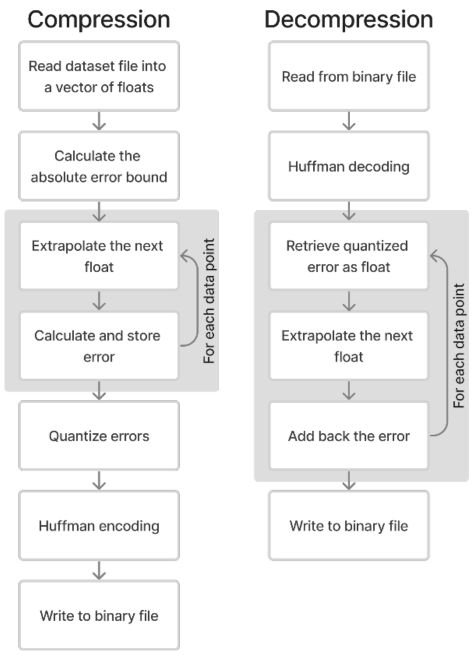

Figure 1. This diagram illustrates the process of compressing and decompressing datasets used in this paper. The shaded gray boxes contain steps that are repeated for every data point.

Decompression (see Figure 1) is the reverse of compression. First, Huffman decoding is used to retrieve the original vector of integers. Huffman decoding goes through the encoded bits and simultaneously traverses the Huffman tree until reaching a leaf node which represents an integer. This repeats until the encoded bits are exhausted.

After retrieving the original vector of integers, each integer is converted back into a floating-point value as described in section 1.2. The floating-point value will be quantized and not exact. The floating-point value represents the error between the extrapolated value and the actual value. Therefore, the final reconstruction step is to extrapolate a value based on previously decompressed values and add the difference to the extrapolated value.

2.1 Experiment Details

For each compression trial, two important metrics were measured:

- Compression ratio: refers to how much the data was compressed by. It is calculated by dividing the original

Vol. 2024 (7) 252 – 257
ISSN 2688-3651 [online]

file size by the compressed file size.

- Throughput: the speed at which compression/decompression occurs, measured in megabytes per second. The trials varied in the dataset used, relative error bound, and extrapolation scheme. As seen in the Results section, only one variable was changed at a time to ensure causality.

2.2 Setup

The tests were run on a Windows 11 PC (Version 22H2, OS Build 22621.2134) with an Intel Core i7-10700K CPU, 32 GB of memory, and a 1 TB NVMe SSD. The tests were run using Windows Subsystem for Linux (WSL) Version 2 with Ubuntu 22.04. The datasets used in the evaluation are from SDRBench (Zhao et al., 2020). All the datasets used consist of single-precision little-endian floating-point values. Listed are some characteristics of the three datasets used:

1. The CESM-ATM dataset consists of 79 fields of 2D climate data with dimensions 1800x3600. Each field is compressed separately in this work.
2. The EXAALT dataset consists of 6 fields of quantities from a molecular dynamics simulation.
3. The dataset of Hurricane ISABEL, or ISABEL for short, consists of 13 fields of 3D weather data with dimensions of 100x500x500.

This work is implemented in C++ and the source code is provided on GitHub (Middlezong, 2023).

3. Results

3.1 Performance Evaluation

Figure 2 highlights the impact of extrapolation on the compression ratio when compared to no extrapolation. For both datasets shown in the figure, linear extrapolation results in a significantly higher compression ratio. In particular, the CESM-ATM dataset shows a nearly 3x higher compression ratio with linear extrapolation compared to no extrapolation. By leveraging extrapolation, one can achieve a remarkable reduction in data size.

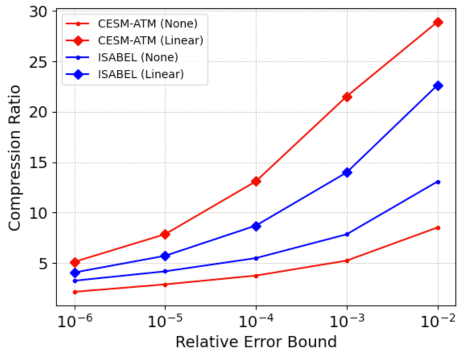

Figure 2. Compression ratio with linear extrapolation vs. with no extrapolation (only quantization) for CESM-ATM and ISABEL datasets, across different relative error bounds. The relative error bounds range from 10^(-6), the strictest bound, to 10^(-2), the loosest bound.

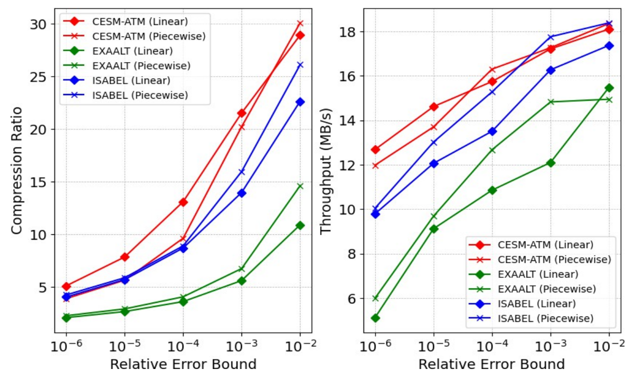

Figure 3. The left graph displays the compression ratio for the two extrapolation schemes at each relative error bound for all three datasets. The relative error bounds range from 10^{-6}, the strictest bound, to 10^{-2}, the loosest bound. The right graph displays the compression throughput (in MB/s).

Figure 3 shows the compression ratio and throughput vs. the relative error bound for linear and piecewise extrapolation schemes, respectively. As the error bound gets looser, a greater compression ratio is achieved. This is because with a larger error bound, it is more likely for different data points to be mapped to the same integer in the quantization step. Then, these data points are more effectively compressed by the Huffman encoding.

Vol. 2024 (7) 252 – 257
ISSN 2688-3651 [online]

It is worth noting that the EXAALT dataset achieved a lower compression ratio than the other two datasets. Figure 4 shows one characteristic difference between the EXAALT and CESM-ATM datasets. The data in CESM-ATM results in quantization levels that are smaller in magnitude. As each quantization level is an integer compressed with Huffman encoding, quantization levels that are closer to each other result in a higher compression ratio. This explains the higher compression ratio of CESM-ATM than EXAALT regardless of the extrapolation scheme. This also suggests that linear extrapolation predicts values of the EXAALT dataset poorly.

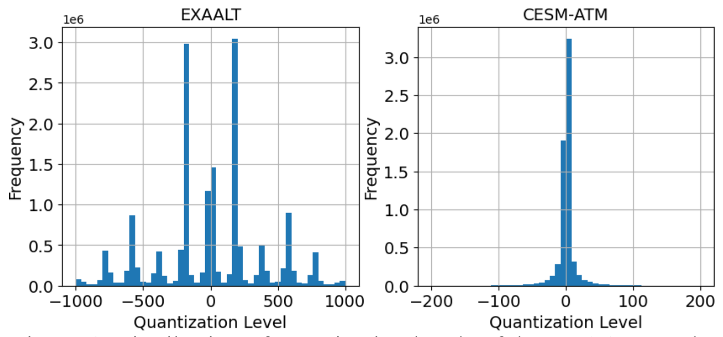

Figure 4. Distribution of quantization levels of the EXAALT and CESM-ATM datasets. They were created by taking a representative file of each dataset and storing the quantization levels calculated during compression using a linear extrapolation scheme (see section 2.2). These quantization levels were plotted in a histogram. The quantization level of a data point measures the difference between its actual value and the extrapolated value.

Linear extrapolation is in general expected to perform better than piecewise extrapolation, because it uses more data points for prediction. However, in Figure 3, it is apparent that piecewise extrapolation leads to a greater compression ratio when compressing the EXAALT dataset. The data visualizations in Figure 5 attempt to explain why. The data in EXAALT appears much choppier than the data in CESM-ATM. In general, smoother data can be more easily predicted using linear extrapolation. However, choppier data cannot be predicted using linear extrapolation, because of the sharp changes between adjacent data points. Therefore, a more relaxed extrapolation scheme, such as piecewise extrapolation, will better predict data. This leads to a higher compression ratio using piecewise extrapolation than linear extrapolation for the EXAALT dataset.

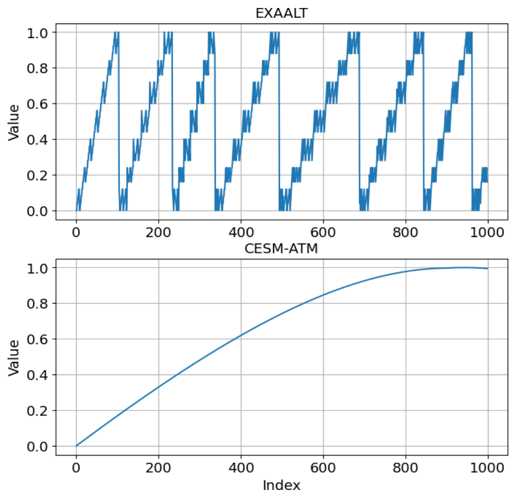

Figure 5. Visualizations of the EXAALT and CESM-ATM datasets. They were created by taking a representative file of each dataset and normalizing and plotting the first 1000 data points of each. The data was normalized to have a minimum of 0.0 and maximum of 1.0.

4. Discussion

Extrapolation can effectively compress data by mapping values that are close to each other to one quantization level, which can be well compressed using Huffman encoding. The extrapolation scheme to be used depends on the data characteristics and needs to be chosen carefully. Data visualization is one way of determining which extrapolation scheme is best for a particular dataset. In general, we expect that relatively smooth data can be better compressed using linear extrapolation.

This research has a few limitations that are to be addressed in the future. The choice of extrapolation scheme (none, linear, or piecewise) does not include more advanced prediction schemes. Future research could investigate more advanced prediction schemes, including pointwise polynomial extrapolation, cubic spline interpolation, and Lorenzo prediction (Ibarria et al., 2003). These prediction schemes may be more effective at predicting certain types of data, leading to even higher compression ratios.

In addition, the extrapolation schemes have only been tested on CPUs. Although this will not affect the results of compression ratio, the architecture has a significant impact on the compression throughput. Throughput is an important factor to consider in real-world data compression. While higher compression ratios are desirable in reducing costs of data transmission and storage, there is a tradeoff between compression throughput and ratio. More effective extrapolation or prediction methods may have an undesirable computational performance. Emerging architectures such as GPUs, which heavily employ parallelization, can potentially resolve the issue. Thus, future research could incorporate emerging architectures, such as GPUs, and investigate the impact that different extrapolation schemes have on compression throughput.

5. Conclusion

This work investigated the use of extrapolation as a technique for lossy compression of floating-point scientific data. Two extrapolation schemes, linear and piecewise, were implemented and their performance was evaluated across three real-world datasets. The metrics of compression ratio and throughput were calculated for each compression trial. The key findings are:

- Extrapolation effectively exposes data redundancy, leading to a better compression ratio.
- The best choice of extrapolation scheme depends on the dataset. Particularly, smoother data is more fit to be predicted using linear extrapolation, while choppier data is better predicted using piecewise extrapolation.

Thus, the extrapolation scheme must be chosen carefully depending on the data characteristics. Future research could incorporate more advanced prediction methods into the comparison and incorporate emerging architectures such as GPUs to further investigate compression throughput.

Acknowledgements

I would like to thank my mentor Professor Liu for guiding me through the research and writing process. This paper could not have been completed without his guidance.

References

Chen, J., et al. (2021). Accelerating multigrid-based hierarchical scientific data refactoring on GPUs. *In 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 922-931). IEEE. https://doi.org/10.1109/IPDPS49936.2021.00095

Di, S., & Cappello, F. (2016). Fast error-bounded lossy HPC data compression with SZ. *In Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 730-739). IEEE.

Ibarria, L., et al. (2003). Out-of-core compression and decompression of large n-dimensional scalar fields. *Eurographics 2003*, 22(3), 343–348. https://doi.org/10.1111/1467-8659.00681

Middlezong, M. (2023). *Understanding an Extrapolation-based Lossy Compression for Floating-point Scientific Data*. GitHub. https://github.com/mmiddlezong/extrapolation-compression/tree/main

Morris, J. (1998). *Huffman Encoding*. School of Computer Science - University of Auckland. https://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html

Zhao, K., et al. (2020). *Scientific Data Reduction Benchmarks*. SDRBench. https://sdrbench.github.io