An In Situ Approach for Explorative Visualization using Temporal Intervals

Nicole Marsaglia
Advisor: Hank Childs

We explore a technique for saving full spatio-temporal simulation data for visualization and analysis. While such data is typically prohibitively large to store, we consider an in situ reduction approach that takes advantage of temporal coherence to make storage sizes tractable in some cases. As I/O constraints continuously increase and hamper the ability of simulations to write full-resolution data to disk, our work presents an in situ data reduction technique with an accuracy guarantee. Rather than limiting our data reduction to individual time slices or time windows, our algorithms act on individual locations and saves data to disk as temporal intervals. Our results show that the efficacy of piecewise approximations, as compared to full spatio-temporal resolution, varies based on the desired error bound guarantee and turbulence of the time-varying data.

1 Introduction

Traditionally, scientific simulations have enabled their data to be visualized by saving many "time slices" to disk. That is, at regular (or irregular) intervals, a simulation code will take the current state of the simulation and store it to a file (or group of files). Visualization programs then operate on this data by loading it from the disk. In this model, the simulation code performs a temporal sampling, meaning that it contains accurate information about the simulation's state for the time slices that were stored, but no information about what happened for the intervals between its time slices. If the scientist running the simulation code determines the temporal sampling will be too sparse, then the scientist can remediate by having the code save more and more time slices.

This traditional model may not be well suited with trends in high-performance computing (HPC). Specifically, the ability to generate data on each new generation of supercomputers is increasing much faster than their ability to store data. One response to this relative decrease in I/O power is to apply all visualization algorithms in situ. This response is reasonable, provided the visualizations to perform are known a priori. Assuming that the desired visualizations are not known a priori (which is the case with explorative visualization), then data needs to be stored for post hoc processing. In this case, a simulation can respond to this relative decrease in I/O power by saving fewer time slices. That said, this would lead to increasingly sparse data from the temporal perspective. This temporal sparsity would, in turn, increase the rate at which interesting phenomena is lost, namely in the cases where the phenomena occurs within intervals that are not saved or in the cases where the phenomena does not have enough temporal resolution to properly observe.

Recent research has focused on a new processing paradigm to address this problem. This research has explored a combination of in situ and post hoc processing. The role of in situ processing is to reduce the simulation data. This reduction often happens through some sort of transformation, which can range from compression (e.g., wavelets) to alternate representations (e.g., topology). The key tension in this process is between data reduction and data integrity — too much reduction creates forms that are not useful, and too much emphasis on integrity can limit possible reductions.

With our research, we introduce a new operator for the in situ reduction / post hoc exploration paradigm. Specifically, we save data as temporal intervals, and focus our data reduction within these intervals. That is, while the traditional approach stores data in groups that all have the same time, our approach stores data in groups that all have the same location. Similarly, while the traditional approach lends itself to reductions that take advantage of spatial coherency, our approach lends itself to reduction that take advantage of temporal coherency.

Our approach is only feasible because of recent changes in HPC. First, deep memory hierarchies are increasingly common on leading-edge supercomputers. These hierarchies provide significantly more memory than was previously available, including places like the "burst buffer," which can be used to store and analyze multiple time slices of simulation state before it is saved to disk. Second, in situ processing is rapidly becoming an accepted processing paradigm by simulation codes. This creates the opportunity for visualization and analysis routines to access more temporal data than ever before, even at the resolution of simulation cycle. This increased access in turn enables visualization and analysis to consider ways to store the data in more efficient forms that were not previously possible.

Temporal interval data is different in nature than the data traditionally stored by simulation codes. Where traditional output has complete accuracy at few time slices, temporal interval data has full temporal resolution. However, temporal interval data is too large to store to disk — just as the traditional model would be if it stored all time slices — so it is only feasible if the intervals are compressed as they are stored. With our approach, we compress the data in a way that guarantees accuracy, e.g. 95% accurate, 99% accurate, etc. Our approach, then, has the benefit of being able to present known error bounds to stakeholders. However, the storage to achieve this accuracy guarantee is variable, and, for some types of data, may require more storage than the stakeholder is willing to allocate. For this research, we decided to embrace this decision (fixed error rate with unknown storage requirements) rather than consider the alternative (fixed storage requirements with unknown error rates), since we feel the former proposition is more desirable to stakeholders.

This work implemented three algorithms that all have a guaranteed error rate, and compress data temporally in situ. For the chosen algorithm and user defined error rate, the algorithm operates individually on every data point within a simulation. Within each data point, for every generated value, the algorithm decides whether to store the subsequent approximation on the SSD, or to write the compressed value to disk. We ran each algorithm on multiple data sets and analyzed the I/O improvement compared to the traditional paradigm of saving time slices. By implementing temporal data reduction in situ, this research has shown an improvement over saving every time slice with full spatial resolution; compressing data as we go temporally, the algorithms provide full temporal resolution of the simulation data and match storage requirements of code that outputs time slices as infrequently as every 500th time step.

The paper is broken up into the following sections: Section 2 discusses related work; Section 3 describes the algorithms utilized in this research as well as the error guarantee and reconstruction process; Section 4 provides an overview of the factors of the experiments; Section 5 lays out the results of this research; Section 6 sums up the conclusion of this research; and Section 7 describes the future work of this research.

2 Related Work

This section classifies related work into categories of compression algorithms for data reduction:

- Individual Time Slice Data
- Multiple Time Slice Data
- Complete Temporal Data

2.1 Individual Time Slice Data

This approach to data reduction disregards the temporal coherence between data points. Most of these data reduction techniques do however use spatial coherence to perform compression.

One approach to compress individual time slice data is to use lossless data reduction. However, lossless compression techniques [2, 16] can only reduce data by modest amounts and is computationally intensive, and thus unfit for in situ data reduction. FPC [2], a lossless technique, is a predictive coding method targeted at 64-bit values. FPC fails to achieve good compression ratios when operating on scientific data or data with significant randomness. The fpzip compression [13] also utilizes predictive coding and achieves good compression as a result of an improved entropy coder. MAFISC [6] is a preconditioner that applies multiple filters and transformations to data before using a standard compression technique; MAFISC achieves better lossless compression on climate data compared to classic lossless techniques such as gzip and lzma. ISOBAR [17] is another preconditioner that statistically separates data into compressible and incompressible, and then determines the optimal combination of standard lossless compression techniques. Another precoditioner by Gomez et al. [5] applies binary masks to mimic zero entropy before compressing.

Another approach to compressing individual time slices is lossy data reduction. ZFP [12] achieves high compression via spatial coherence and bit reduction, but this technique strongly favors smooth data. Similarly, the RBD algorithms [7] utilize spatial locality by viewing data point grids as graphs and representing subgraphs as singular values based on a user defined error bound. Additionally, fpzip can be altered and used for lossy compression. APAX [18], similar to the lossy version of fpzip, also uses predictive encoding but employs an absolute rather than relative error bound.

This work is unfit for in situ compression because these methods are computationally intensive and compress data based solely on spatial coherency. By not considering temporal coherency, while this limits their memory requirements to a single time slice, it also limits their potential compression capabilities.

2.2 Multiple Time Slice Data

This approach to data reduction takes into account multiple time slices, utilizing the temporal coherence present in many scientific data sets, and potentially spatial coherence as well.

ISABELA, developed by Lakshminarasimhan et al. [9], is a lossy in situ preconditioner that can compress tumultuous data by nearly 85% with a 99% average correlation between the original and compressed data with little overhead in runtime. ISABELA takes advantage of monotonic inheritance of points over time by first sorting a time slice, applying B-spline curves, and then representing multiple temporal windows with a reconstructed curve. The representative window is continued temporally until an average error bound has been exceeded. Lehmann [11] extends ISABELA by adding corrective computations to the sorted data and loosening the restrictions of when to write a window to memory. Additionally, it adds support for selective loading of regions with varying levels of resolution.

SZ [4] is a predictive algorithm that converts multidimensional matrices to 1D arrays and applies multiple curve fitting models. Based on a user defined error, SZ only saves a bit corresponding to the model best fitting the current section of values, or if none of the models fall within the user error bounds the value is compressed via a binary representation analysis.

There are several reasons why this work is unfit for in situ data compression on every time slice. For one, these works require a significant amount of memory, namely a number of entire time slices if temporal coherency is to be considered, not to mention the memory requirements for their computations. Computations that could impose significant strain on the simulation if they are conducted for each time slice. Second, if they are going to take into temporal coherency, none of these works know the number of consecutive time slices that would optimize compression.

While these works take into account both spatial and temporal coherency, they do not provide full spatio-temporal resolution as their compression only applies to several consecutive time slices, and not every temporal interval.

2.3 Complete Temporal Data

This section encompasses full spatio-temporal resolution. Our work operates on each grid point's continuous stream of data, attempting to compress every incoming value. Unlike other works, that only provide an average error bound, our work guarantees a user defined error bound relative to every grid point.

Our research includes three in situ algorithms that reduce temporal data into piecewise approximations. It is inspired by the SWAB [8] data mining technique for segmenting time series data.

IDEALEM [10] relies on statistical similarities and breaks streams of data into windows, then compresses those windows based on point distribution to reduce data in situ. IDEALEM does well with tumultuous data as it provides a heavy distribution of points. However, it performs poorly with smooth curves as the data distribution is sparse within each window.

2.4 How Our Approach Differs From Previous Work

All of the works mentioned above, with the exception of RBD, guarantee at most an average error bound. IDEALEM guarantees a user defined statistical similarity but not a definite point by point correlation. Our objective is to guarantee a known error bound per point.

In this work, we compress data at each grid point location over time rather than compressing entire time slices or groups of time slices at once. By employing independent compression operators at each grid point, we can guarantee an accuracy relative to each grid point's time-varying data. Our technique provides infinite temporal resolution of a time series, and full spatio-temporal resolution of a simulation.

3 Compression Operators

The objective of this work is to provide full spatio-temporal resolution with guaranteed error bounds; this is accomplished via a tailored compression technique.

Our algorithms transform grid point values generated in situ into piecewise approximations with a guaranteed error bound. Data is only queued for storage when the difference of the approximated data exceeds the user defined error bound relative to the original data.

The algorithms work as follows. To compress simulation data, there is an instance of a compression object for each data point in the grid. Each such object reduces its respective time series into piecewise approximations. The compression algorithm at a given data point runs independent of the compression algorithms at other data points. For every generated value at a data point there are two possible outcomes. One, the simulation's current value falls within the error bound of the current approximation. Or two, the simulation's current value does not fall within the error bound of

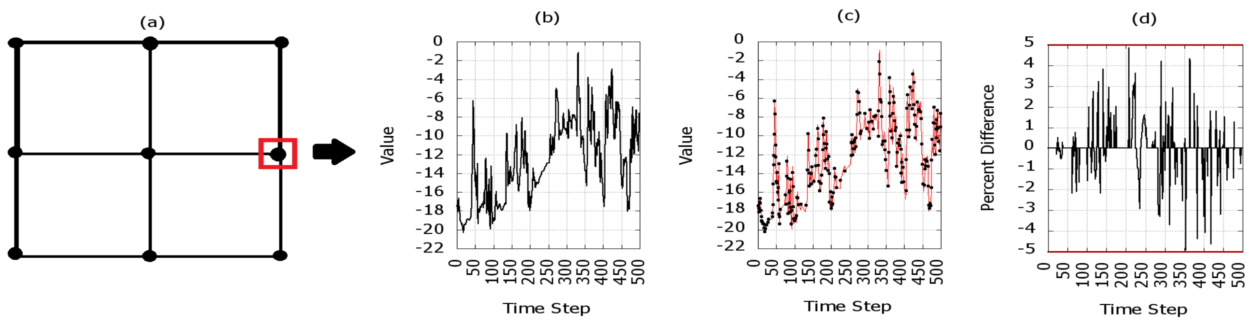

Figure 1: Image (a) represents the simulation and the corresponding data points. As the simulation runs, each data point generates a stream of data. Each data point employs a compression operator that independently compresses its time series into a piecewise approximation. Images (b)-(d) symbolize the . This image shows the original time series of one data point (b), the piecewise approximation (c), and the per point difference of the two (d). Image (d) shows that each point falls within a 5% per point error bound defined by the user.

the current approximation. In this case, the former approximation is written to disk or memory and the process begins anew with the current value.

3.1 Error Bound

For most of the research presented in Section 2, a user defined error is realized as an average error bound, and not as a maximum error bound. Using an average error bound allows for more flexibility within the approximation process, but can lead to unexpected artifacts and errors. By using a maximum error bound, we guarantee an error bound specific to each time series and prevent artifacts when reconstructing the approximations. Maximum error can either mean a maximum absolute error (all reconstructed values must be within X units of the original values) or a relative absolute error (all reconstructed values must be within $X\%$ of the original values). In our case, we focus on relative absolute error, because maximum absolute error can lead to loss of important information when a value is near zero. However, the maximum and minimum values used for normalization can not be known a priori. To tackle this problem, each data point employs a "learning curve": the range of values is initially unset and is updated as new values are generated. This results in a learning overhead as each data point's error bound increases as they encounter a wider range of values.

3.2 Algorithms

The three algorithms considered are:

- Piecewise Linear (PL)
- Piecewise Constant (PC)
- Piecewise Constant Mean (PCM)

While each algorithm is similar in that they all employ a sliding window of size two, they are differentiated by how they approximate the time series. The first element in the window is either the starting value of a new approximation or the current approximated value. Each incoming value becomes the second element in the window and that window is reduced to size one either by compression or saving the first element in the window. If the second element is approximated, the first element is overwritten with the new approximation and the second element is overwritten with the next generated value. In the case that the starting/approximated value is saved to disk or memory, the second element becomes the new starting point, overwriting the first element, and the second element is overwritten with the next generated value. Figure 1 illustrates an instance of a compressor on a single grid point, the resulting approximation, and the difference between the original and approximated time series. Figure 2 shows our three algorithms approximating the same time series. During post-hoc analysis, specific time slices of the simulation can be reconstructed with values that reside within the user defined error bound.

3.2.1 Piecewise Linear (PL)

This algorithm transforms consecutive data point values into piecewise linear approximations as the simulation progresses. The compression object deployed for each data point keeps track of the local maximum and minimum, as well as the set of consecutive values it is currently approximating. The algorithm approximates piecewise linear equations for each data point's time series.

For each piecewise approximation is it necessary to calculate the slope using the starting value of the approximation and the subsequent generated value. From the starting value, the slope is then extended one unit for every time step, creating our approximation. For each time step, the approximation is compared to the current value. If the difference between the approximated value and the current value is within the error bound then this process continues. If the difference between the current value and the approximated value does not fall within the error bound then the value, slope, and current time step are saved to disk. Once an approximated line has been saved, the algorithm then restarts with a new starting point. The new starting point is the last value that was not within the error bound of the approximation; a new slope will then be calculated using the starting point and the subsequent value.

3.2.2 Piecewise Constant (PC)

The second algorithm, Piecewise Constant, is much like Piecewise Linear. Each data point runs its own Piecewise Constant compressor and keeps track of their local minimum, maximum and associated error bound based on the user defined error percentage. However, in the Piecewise Constant algorithm, a single value represents an interval of the time series. The approximated line is extended until the difference between the approximated value and the current value exceeds the error bound. Once the error bound is exceeded, the value that could not be represented by the approximation becomes the reference value going forward. This algorithm saves the representative value and current time step to disk.

3.2.3 Piecewise Constant Mean

The third algorithm, Piecewise Constant Mean, is similar to the first two, but takes the average of the encountered values per approximation interval. This algorithm uses the local extrema of each approximation interval to ensure the error bound guarantee is met.

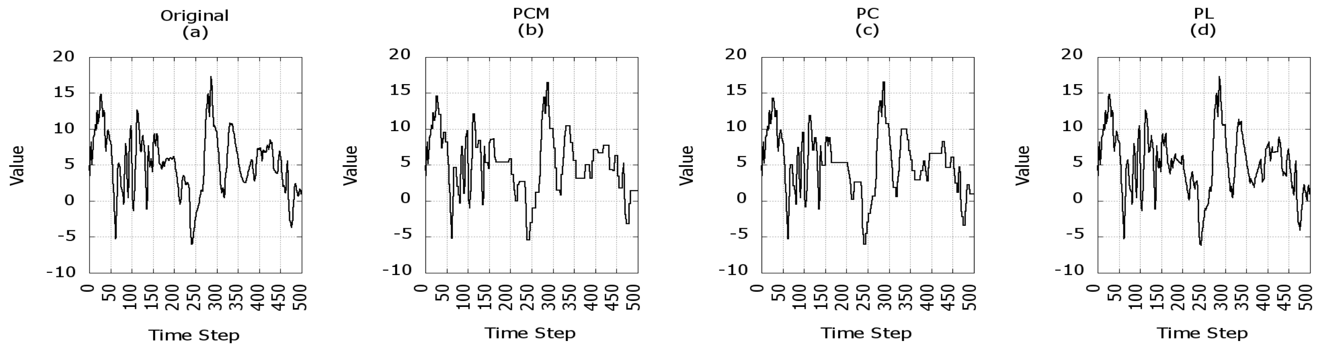

Figure 2: Figure (a) shows the original values. Figures (b)-(d) are the approximations created by our algorithms with a fixed 5% user-defined error bound. Figure (b) is the piecewise approximation created by the PCM algorithm. Figure (c) is the piecewise approximation created by the PC algorithm. And figure (d) is the piecewise approximation created by the PL algorithm.

Initially, and after the error bound has been exceeded, the algorithm restarts and the local maximum and minimum are reset. With each new incoming value, the algorithm calculates an approximated line that is the mean of all the encountered values. The algorithm also checks if each new value is greater than the current local maximum or less than the current local minimum. If so, the local maximum and minimum are set accordingly. The algorithm also makes sure that no previous point falls outside the error bound in relation to the mean. That is, for every new value the algorithm checks to make sure that the current minimum and maximum are within $[mean - \varepsilon, mean + \varepsilon]$ where ε is the error bound.

3.3 Memory Requirements

Within memory, each grid point is required to save several values that facilitate the compression process. Table 1 lists all the necessary data that each grid point must store.

3.4 Reconstruction

The reconstruction process from the compressed data is relatively straightforward. The data from every grid point is written and subsequently available in its own file. This information consists of the saved data values and the corresponding time steps. In addition, the piecewise linear algorithm saves the slope.

To construct an approximation of the original data requires the desired time slice along with the compressed data. Currently, reconstruction is dependent on the entire compressed data being in memory. But our future work plans on eliminating this dependency, allowing for faster reconstruction.

Given the desired time slice, it is straightforward to find the corresponding value within each data point's file. Since each data point saves both the current time step and the associated representative value, we know that the value represents the range of time steps between the previously saved time step to the current. Let t_d be the desired time slice and let t_c be the current time slice with the associated value v and t_p be the previous time slice before t_c. If $t_d \in [t_p + 1, t_c]$ then v is the approximated value for the desired time slice.

For piecewise linear, it is necessary to calculate the value at the desired time step from the saved output. Let t_d be the desired time step for reconstruction, let t_p and t_c be the previously saved time step and the current saved time step, respectively, and let v be the value associated with t_c and within $[t_p + 1, t_c]$. Given the slope s associated with v, if $t_d \in [t_p + 1, t_c]$ then the approximated value is $v + s * (t_d - (t_p + 1))$.

Algorithm	Memory
Piecewise Linear (PL)	4 Byte Min and Max 4 Byte Slope 4 Byte Starting Value 4 Byte Count
Piecewise Constant (PC)	4 Byte Min and Max 4 Byte Value
Piecewise Constant Mean (PCM)	4 Byte Min and Max 4 Byte Approx. Value 4 Byte Local Min and Max 4 Byte Count

Table 1: Depending on the algorithm, each grid point is required to store necessary values in memory. Each grid point calculates it's own error bound tailored to the range of values encountered — this is stored via the minimum and maximum. Also, for each approximated line it is necessary to have either the starting value or the current approximated value. For the PL each grid point stores the slope associated with the current piecewise linear approximation. And the PL and PCM require the count, or count, of time steps their respective lines approximate; the PL needs the count to calculate the current position of the line using the slope and starting value, and the PCM to calculate the mean of all relevant values for the approximated line. Additionally, PCM requires the local extrema to be stored in order to guarantee the error bound criteria.

4 Experiment Overview

Our experiments were designed to quantitatively evaluate the efficacy of data reduction with full temporal resolution. We evaluated several algorithms over several data sets with differing error bound guarantees. This section outlines the parameters of the research conducted; Section 4.1 lists the parameters we varied over our experiments; Section 4.2 goes over the measurements and metrics used in this study; Section 4.3 describes the machine these experiments were conducted on.

4.1 Parameters

We varied the following factors:

- Algorithms (3 options)

- Data Sets (4 options)
- User Defined Error Bounds (3 options)

4.1.1 Algorithms

We evaluated the three algorithms described in Section 3.

- Piecewise Constant (PC)
- Piecewise Linear (PL)
- Piecewise Constant with Local Extrema (PCM)

4.1.2 Data Sets

For this research we tested the efficacy of our compression in two phases. For the first phase, we looked at time series data from individual grid points over four simulation codes. For the second phase, we evaluated our algorithms on a full simulation in situ for one of the data sets. The results from the first phase show the compression ratio of a stream of values over time. Whereas the results of the full simulation shows the compression ratio of the entire simulation per time step, the full simulation compression ratio is a culmination of the compression ratios of all the individual grid points. We considered four data sets for the first phase:

- LULESH [1]: A data set with 8 different grid points each with 4,561 time steps.
- Tornado [15]: A data set with 10 different grid points each with 500 time steps.
- XGC1 Ion Particles [3]: A data set with 25 different grid points each with 818 time steps.
- GHOST [14]: A data set with 20 different grid points each with 9,990 time steps.

The Geophysical High Order Suite for Turbulence, or GHOST, simulates turbulent rotational flows.

For the second phase, we continued with the GHOST simulation data, running in situ compression for a $128 \times 128 \times 128$ grid point simulation run for 250,000 time steps.

4.1.3 User Defined Error Bound

We applied three user defined error bounds:

- 1%
- 3%
- 5%

To guarantee an error bound, our error bound is relative to the range of values encountered at each data point. For example, an error bound of 1% means the approximation will fall within a 1% error window of the original value that is based on the range of values that particular grid point has encountered.

4.2 Measurements and Metrics

In total this work ran 36 experiments each with multiple measurements.

In the first phase comprised of the four data sets, LULESH, Tornado, XGC1 Ion Particles and GHOST, we ran each with the three algorithms and the user defined error bounds of 1, 3, and 5 percent. For each data set we calculated the maximum compression ratio, the minimum compression ratio, average compression ratio and the average Pearson Correlation Coefficient.

The Pearson Correlation Coefficient (PCC) ranges from [1,-1]. A PCC of 1 or -1 indicates that there is a perfect positive or negative linear correlation, respectively. If the PCC is 0 then the two values share no linear relationship. In our case, a PCC close to 1 indicates a near identical correlation between the original data and the reduced data.

In the second phase, each of the three algorithms were run on GHOST in situ with each error bound. Our GHOST simulation setup has 128*128*128 data points oriented as a cube. Each simulation configuration ran for 250,000 time steps. The measurements on GHOST include the number of values saved per time step; the resulting improvement compared to saving an entire time slice every *N* time steps; as well as the average time of computation for all three algorithms at each error bound and the resulting impact on the simulation.

4.3 Hardware

This research was done on Alaska, our in-house research cluster. Alaska is composed of a Xeon E5-2667v3 (16 core) head node and four Intel Xeon E5-1650v3 (6 core) cluster nodes.

5 Results

5.1 Individual Data Points: LULESH, XGC1 Particle Ions, and Tornado

Table 2 lists four metrics — maximum compression ratio (CR), minimum compression ratio, average compression ratio, and average PCC — for each combination of algorithm, *data set*, and error bound. For each data set the piecewise algorithms achieved a high PCC value, meaning the approximated values are closely correlated to the actual values. Each data set having a higher average PCC value when the algorithms employ a smaller error bound percent. Apart from the PL algorithm on the Tornado data set, all algorithms that employed a 1% error bound had an average PCC of $> .999$. Additionally, apart from the PL algorithm on the Tornado data set, all algorithms using the 3% and 5% error bound attained an average PCC of $> .99$ on all data sets.

The four data sets displayed a wide range of compression ratios compared to the size of the original time series. Using a 5% error bound, the XGC1 data set had the lowest compression ratio, varying between 2.06X and 7.18X with an average compression ratio of 2.91X. The Tornado data set had compression ratios only slightly better, ranging from 2.58X and 10.87X with an average of 4.02X. LULESH and GHOST performed much better comparatively. LULESH achieved a significantly higher maximum compression than both Tornado and XGC1 with 49.04X, but had a similar minimum and average compression at 3.95X and 7.46X, respectively. GHOST outperformed the other data sets, having a minimum compression ratio of 95X , a maximum compression ratio of 173.3X, and an average compression ratio of 132.80X.

The compression ratios for each algorithm using 1% and 3% error bounds follow the trends of the 5% error bound, but with smaller compression ratios.

The compression results for these data sets depends tremendously on the turbulence of the data, but also how quickly the algorithm reaches a sufficient range to base its error bound. If a grid point encounters a sufficient minimum and maximum values early on, then the error bound will be based on a wide range of values. Whereas, if a grid point does not encounter a sufficient maximum and minimum until later on, then the error bound is based on an insufficient range, and is saving piecewise approximations less than the guaranteed error bound.

Comparing the graphs of the four data sets' grid points that produced the minimum and maximum compression, it becomes clear that achieving a sufficient range for the error bound quickly helps increase compression capabilities. Additionally, these algorithms work best on time series with high temporal coherence and little fluctuation.

Figure 3 shows the curves that achieved the maximum compression ratio and minimum compression ratios within the data set. The curve that resulted in the maximum compression ratio clearly achieves a wider range of values sooner than the curve that resulted in the minimum compression ratio. The first curve, using

Algorithm	Max CR	Min CR	Avg. CR	Avg. PCC
		LULESH		
PCM				
5%	49.04X	3.95X	7.46X	0.99748
3%	42.63X	3.86X	7.21X	0.99897
1%	29.05X	3.68X	6.52X	0.99986
PC				
5%	42.09X	3.87X	7.17X	0.99899
3%	34.29X	3.78X	6.85X	0.99958
1%	20.83X	3.48X	5.8X	0.99995
PL				
5%	49.04X	3.89X	7.46X	0.99892
3%	45.16X	3.88X	7.35X	0.99965
1%	36.2X	3.85X	7.15X	0.99995
		Tokomak		
PCM				
5%	7.18X	2.06X	2.91X	0.99803
3%	4.65X	1.68	2.15X	0.99933
1%	2.22X	1.22X	1.24X	0.99994
PC				
5%	4.7X	1.63X	2.18X	0.99868
3%	3.33X	1.36X	1.69X	0.99963
1%	1.74X	1.1X	1.26X	0.999977
PL				
5%	4.76X	1.61X	2.07X	0.99480
3%	3.21X	1.35X	1.66X	0.998277
1%	1.76X	1.11X	1.25X	0.999923
		Tornado		
PCM				
5%	10.87X	2.58X	4.02X	0.99403
3%	6.85X	1.87X	2.86X	0.99780
1%	3.07X	1.27X	1.679X	0.99980
PC				
5%	7.24X	1.85X	2.9X	0.99524
3%	4.62X	1.44X	2.12X	0.99852
1%	2.1X	1.13X	1.4X	0.99990
PL				
5%	9.26X	1.85X	2.87X	0.96413
3%	6.76X	1.46X	2.25X	0.98144
1%	3.01X	1.15X	1.48X	0.99807
		GHOST		
PCM				
5%	173.3X	95X	132.80X	0.99771
3%	114.88X	59.16X	84.84X	0.99913
1%	47.5X	22.61X	33.4X	0.99989
PC				
5%	99.80X	52X	73.77X	0.99765
3%	65.43X	32.71X	47.36X	0.99915
1%	28.22X	12.88X	19.24X	0.99990
PL				
5%	143.18X	71.59X	101.02X	0.99880
3%	89.01X	49.9X	66.58X	0.99958
1%	30.78X	18.50X	24.65X	0.99995

Table 2: The minimum, maximum, and average compression ratio (CR) achieved on the four data sets, as well as the average Pearson Correlation Coefficient (PCC) for each algorithm and error bound.

the PCM algorithm and 5% error, achieved a 49.04X compression ratio, whereas the second curve with the same heuristics achieved a compression ratio of 3.95X. A smaller range of values to determine the error bound over the majority of the values coupled with fact that the second curve is more turbulent resulted in a lower compression ratio.

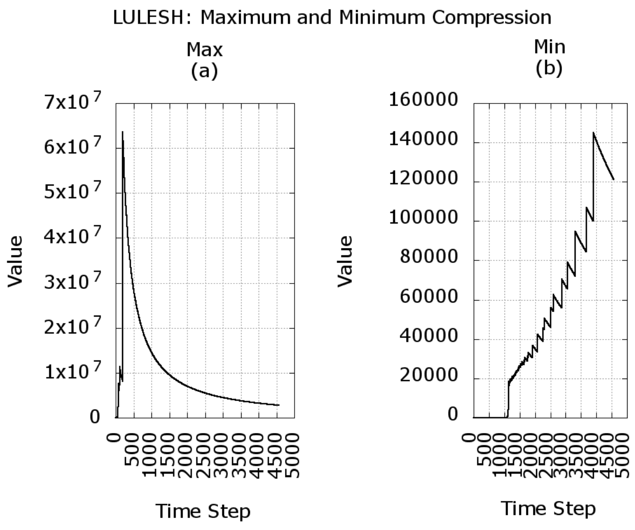

Figure 3: The curves of the LULESH data set that achieved the maximum compression ratio (a) and the curve that achieved the minimum compression ratio (b).

Similarly, Figure 4 shows the curves of the XGC1 Ion Particle data set that achieved the maximum compression ratio and the minimum compression ratio. For these two curves, the first curve, using the PCM algorithm with a 5% error bound, resulted in a compression ratio of 7.18X whereas the second curve resulted in the minimum compression ratio of 2.06X. Again, this difference is due to the first curve reaching a more sufficient range of values sooner and the overall smoothness compared to the second curve.

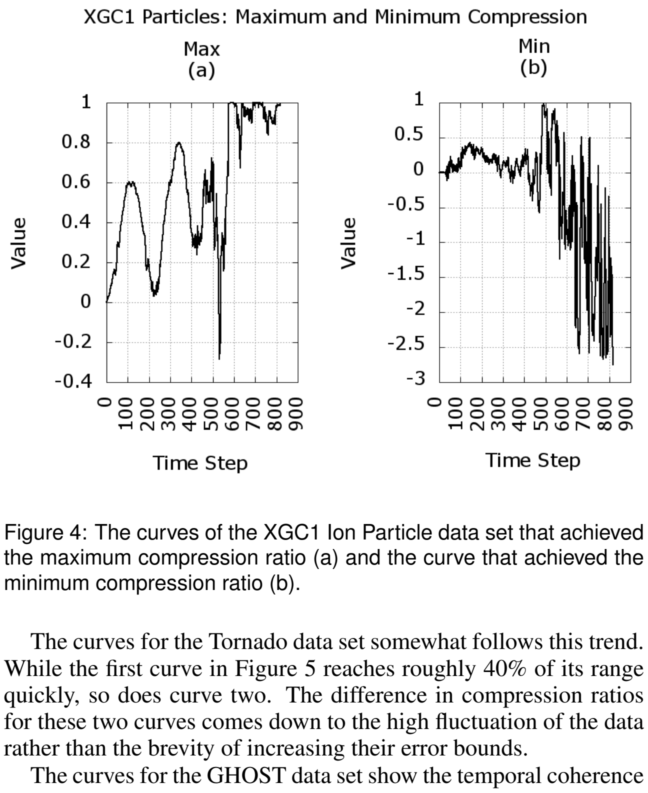

Figure 4: The curves of the XGC1 Ion Particle data set that achieved the maximum compression ratio (a) and the curve that achieved the minimum compression ratio (b).

The curves for the Tornado data set somewhat follows this trend. While the first curve in Figure 5 reaches roughly 40% of its range quickly, so does curve two. The difference in compression ratios for these two curves comes down to the high fluctuation of the data rather than the brevity of increasing their error bounds.

The curves for the GHOST data set show the temporal coherence and smoothness present among the values. The first curve in Figure 6 achieved a compression ratio of 173.3X and the second curve

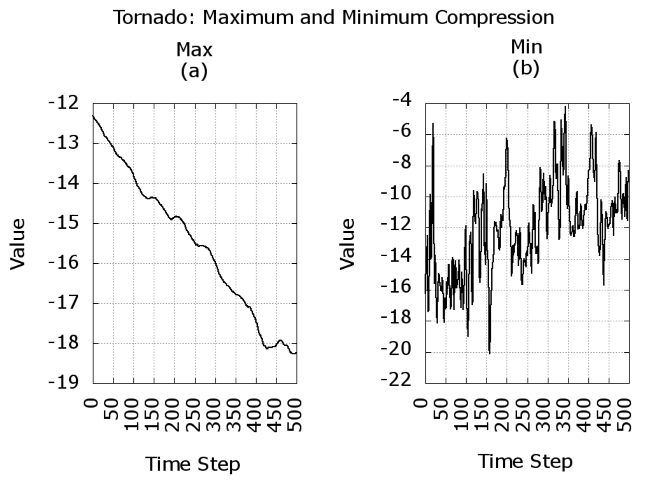

Figure 5: The curves of the Tornado data set that achieved the maximum compression ratio (a) and the curve that achieved the minimum compression ratio (b).

achieved a compression ratio of 95X using the PCM algorithm with a 5% error bound.

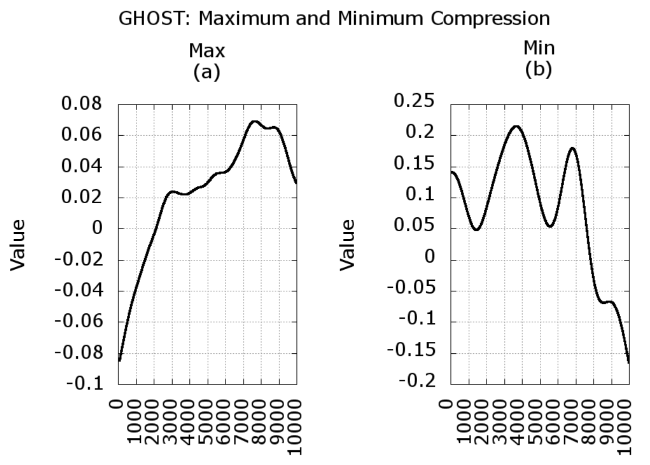

Figure 6: The curves of the GHOST data set that achieved the maximum compression ratio (a) and the curve that achieved the minimum compression ratio (b).

In summary, the results from phase one only produced promising results for GHOST.

The results from the individual data sets show that the algorithms will work well on simulations whose grid points do not have the same level of work. While several grid points may experience tumultuous streams of values, others may remain smooth. The grid points with high compression will even out those grid points with more turbulent data and thus lower compression. This can be seen in the results for the full simulation, Ghost.

5.2 Entire Simulation: GHOST

Each data point has a compressor object that runs the designated algorithm with the user defined error percent. In addition, each data point is responsible for setting their error bound based on their minimum and maximum values, and saving values when dictated by the algorithm.

Our experiments measured the number of values saved per time step, computing compression ratios compared to the size of a time slice with full spatial resolution. Due to the fact that the error bound is based off of the encountered values of each data point, initially the algorithms require a high volume of writes to memory. But once there is a sufficient error bound, each data point saves values less and less frequently. Figure 7 demonstrates that after the initial overhead of their learning curve each algorithm eventually hits an output equilibrium.

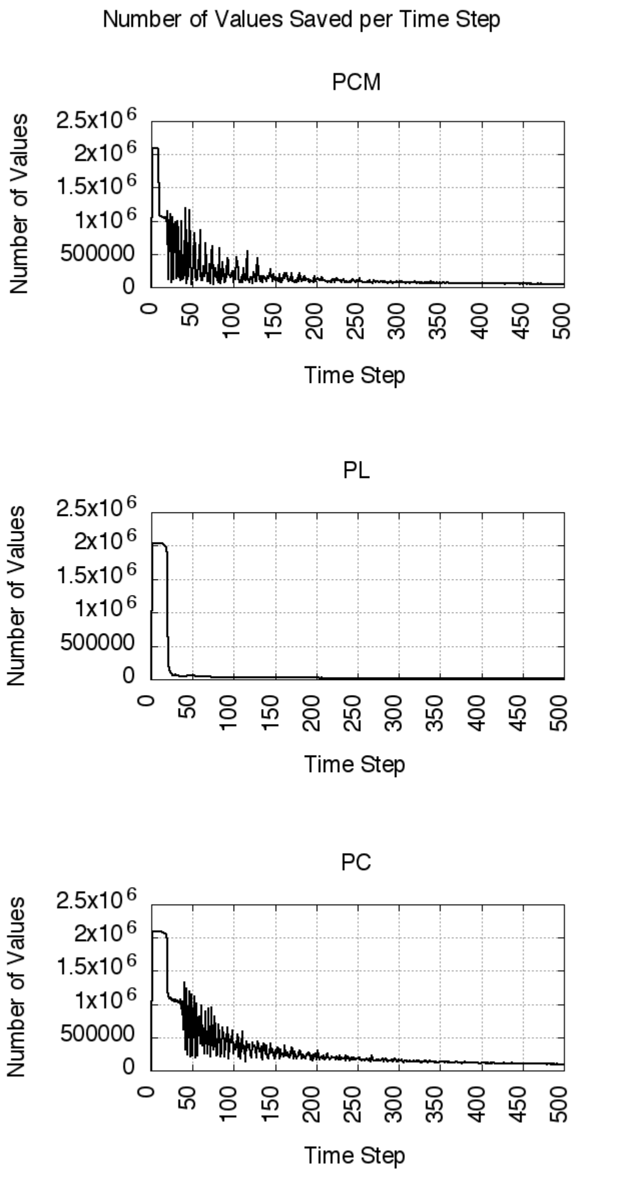

Figure 7: The initial overhead of each algorithm with a 5% user-set error percent. This overhead is referred to as a learning curve, as each algorithm slowly learns the appropriate error bound based on the values each data point encounters.

Figure 8 shows an example of decompressed data and how close it is to the original data, namely within the error bound. For that particular time slice, the reconstructed data is an adequate representation of the original data, with errors within the user defined

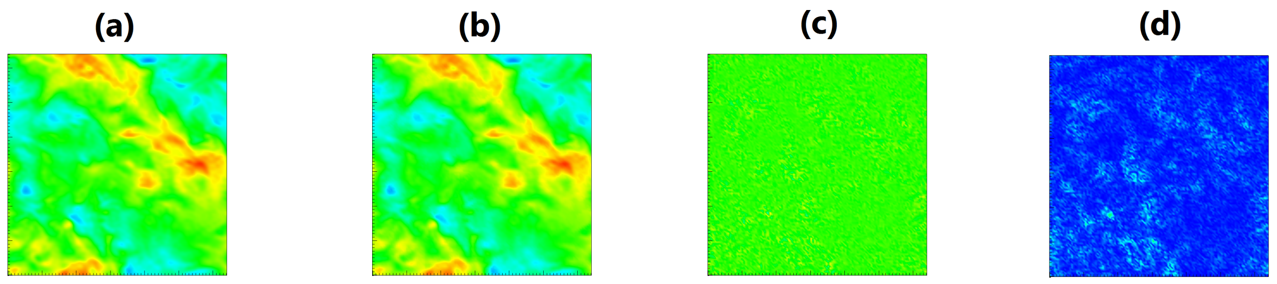

Figure 8: Image (a) is the original data at time slice 205,025. Image (b) is the reconstructed data that used the PCM algorithm with a 5% error. Image (c) is a data comparison of their difference. And image (d) is the absolute value of their difference. Both the dark and light blue in image (d) represents minimal differences between the original and reconstructed data, differences well within the 5% error.

error bound.

With a 5% user defined error, GHOST eventually only saves 2000-5000 values per time stamp depending on the algorithm as shown in Figure 9. With the PCM and PC algorithms this means saving an 4 byte single precision floating point value and a 4 byte integer time step. Whereas the PL algorithm also needs to save the 4 byte single precision floating point value, 4 byte integer time step, as well as an 4 byte single precision floating point value for the slope. This is an improvement 236.96X-582.54X per time step. Table 3 shows what is saved to disk when an approximation is falls outside the user defined error bound. Figure 10 shows an isosurface rendering of a single time slice, an isosurface rendering of the reconstruction of the same time slice and an isosurface rendering of their absolute difference.

Algorithm	Saved to Disk
PCM	4 Byte Approx. Value 4 Byte Round
PC	4 Byte Value 4 Byte Round
PL	4 Byte Slope 4 Byte Starting Value 4 Byte Round

Table 3: When an approximation falls outside the user defined error bound each algorithm saves the necessary information to recreate the approximation. For the PCM and PC algorithms it is necessary to save the approximated value and the last round of the simulation the approximation was valid for. For the PL algorithm it is also required to save the slope of the approximated line in addition to the line's starting value.

Table 4 lists the average save rate each algorithm eventually achieves per error percent after the initial overhead.

Algorithm	1%	3%	5%
PCM	110.37X	343.79X	582.54X
PC	59.91X	201.64X	255.75X
Linear	99.86X	160.70X	236.97X

Table 4: The average compression ratio of the number of pushed values per time step of each algorithm and each user defined error percent. This is after the initial overhead each data point experiences learning their error bound.

If GHOST aims to write out data every time step, this research has a 236.96X-582.54X improvement depending on the algorithm with a 5% error. But if the simulation aims to write out data every 200th time step, the algorithms only have an improvement of 1.27X-2.9X. Figure 11 further illustrates the improvement of each algorithm using each error percentage versus the original simulation with varying number of time steps between output.

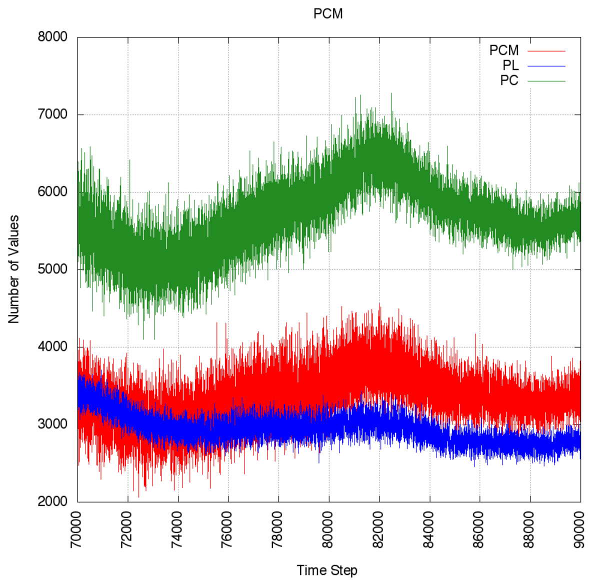

Figure 9: After 70,000 time steps, each algorithm only saves a few thousand out of the 2+ million points of the simulation per time step.

Unfortunately, the algorithms cause significant strain on the simulation. Typically, the algorithms double the run time of the simulation. Table 5 depicts the average time to execute each algorithm at each error percent and the strain this causes the simulation.

6 Conclusion

In this work, we report the compression algorithms over time-varying data. This work presents three piecewise approximation algorithms that guarantee an error bound and provides full temporal resolution. We found that depending on the data set, specifically for GHOST, the compression algorithms can reach a compression ratio greater than 400X for individual grid points, and upwards of 500X for an entire simulation. The compression algorithms achieved these results on a data set with smooth temporal coherence with data fluctuation spread out over time.

These algorithms did not achieve high compression rates on the

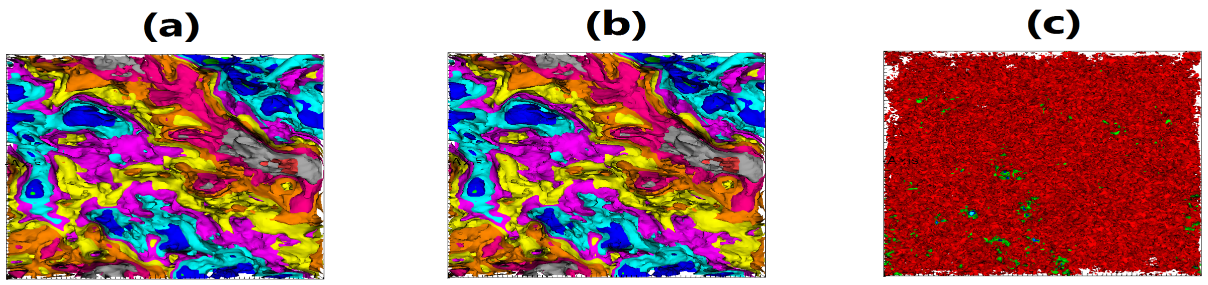

Figure 10: Image (a) is an isosurface rendering of the $125{,}000^{th}$. Image (b) is the reconstruction of that same time slice from the PCM algorithm with a 5% error bound. Image (c) is the isosurface rendering of the absolute difference between the original and reconstruction.

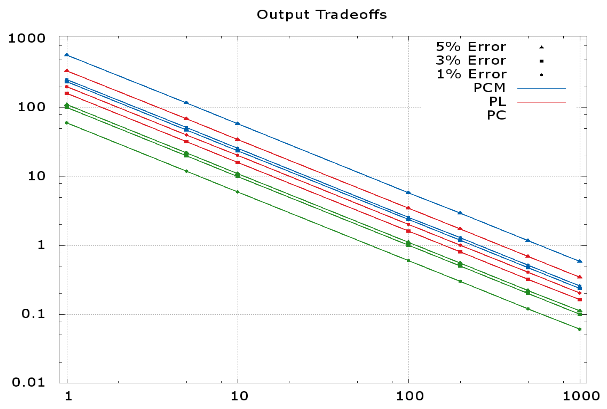

Figure 11: From top to bottom these charts represent the trade off between the algorithms at 5, 3, and 1 percent error bounds, respectively, versus the simulation outputting the entire time slices at varying time steps. The Y-axis represents the improvement of our algorithms whereas the X-axis represents how often the original simulation writes an entire time slice to memory.

Algorithm	Ghost Only	Total Time	Time Increase
PCM			
1%	.6s	1.37s	128%
3%	.6s	1.35s	125%
5%	.6s	1.31s	118%
PC			
1%	.6s	1.15s	91.6%
3%	.6s	1.14s	93%
5%	.6s	1.1s	83%
PL			
1%	.6s	1.3s	116%
3%	.6s	1.16s	93%
5%	.6s	1.1s	83%

Table 5: The average strain that each algorithm causes the simulation without writing to disk.

XGC1 or Tornado data sets, and only showed decent compression with the LULESH times series.

7 Future Work

For future work we plan on parallelizing the work done by the compression operators for each grid point of a simulation to decrease their computation time and lessen the temporal burden on the simulation. Additionally we plan on changing the way files record outputted values so it is no longer necessary to have all of the compressed data files loaded into memory to reconstruct a single time slice.

We will also continue to evaluate these algorithms on other data sets and simulations to determine the extent of their capabilities on both smooth and tumultuous time varying data. While three out of the four time series did not show promising compression in phase one, we'd like to see if there are improvements if we apply the algorithms in situ. Much like how the results for GHOST quadrupled from phase one to phase two, we'd like to see if we can achieve similar improvements for the other data sets in their corresonding simulations.

References

[1] Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254.

[2] M. Burtscher and P. Ratanaworabhan. Fpc: A high-speed compressor for double-precision floating-point data. *IEEE Trans. Comput.*, 58(1):18–31, Jan. 2009.

[3] C. Chang, S. Ku, P. Diamond, Z. Lin, S. Parker, T. Hahm, and N. Samatova. Compressed ion temperature gradient turbulence in diverted tokamak edge). *Physics of Plasmas (1994-present)*, 16(5):056108, 2009.

[4] S. Di and F. Cappello. Fast error-bounded lossy hpc data compression with sz. *Proc. IPDPS. IEEE*, 2016.

[5] L. A. B. Gomez and F. Cappello. Improving floating point compression through binary masks. In *Big Data, 2013 IEEE International Conference on*, pages 326–331. IEEE, 2013.

[6] N. Hübbe and J. Kunkel. Reducing the hpc-datastorage footprint with mafisc—multidimensional adaptive filtering improved scientific data compression. *Computer Science - Research and Development*, 28(2):231–239, 2013.

[7] J. Iverson, C. Kamath, and G. Karypis. Fast and effective lossy compression algorithms for scientific datasets. In *Proceedings of the 18th International Conference on Parallel Processing*, Euro-Par'12, pages 843–856, Berlin, Heidelberg, 2012. Springer-Verlag.

[8] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. *Proceedings 2001 IEEE International Conference on Data Mining*, pages 289–296, 2001.

[9] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6852 LNCS(PART 1):366–379, 2011.

[10] D. Lee, A. Sim, J. Choi, and K. Wu. Novel data reduction based on statistical similarity. In *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*, page 21. ACM, 2016.

[11] H. Lehmann and B. Jung. In-situ multi-resolution and temporal data compression for visual exploration of large-scale sientific simulations.

Journal of Chemical Information and Modeling, 53(9):1689–1699, 2013.

[12] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[13] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, Sept. 2006.

[14] P. Mininni, A. Alexakis, and A. Pouquet. Large-scale flow effects, energy transfer, and self-similarity on turbulence. *Physical Review E*, 74(1):016303, 2006.

[15] L. Orf, R. Wilhelmson, and L. Wicker. Visualization of a simulated long-track ef5 tornado embedded within a supercell thunderstorm. *Parallel Computing*, 55:28–34, 2016.

[16] P. Ratanaworabhan, J. Ke, and M. Burtscher. Fast lossless compression of scientific floating-point data. In *Proceedings of the Data Compression Conference*, DCC '06, pages 133–142, Washington, DC, USA, 2006. IEEE Computer Society.

[17] E. R. Schendel, Y. Jin, N. Shah, J. Chen, C. S. Chang, S. H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. ISOBAR preconditioner for effective and high-throughput lossless data compression. *Proceedings - International Conference on Data Engineering*, pages 138–149, 2012.

[18] A. Wegener. Adaptive Compression and Deocmpression of Bandlimited Signals. *US Patent 7009533*, 1(12):0–4, 2006.