Enabling Explorative Visualization With Full Temporal Resolution Via In Situ Calculation Of Temporal Intervals

Nicole Marsaglia[1], Shaomeng Li[2], and Hank Childs[1]

[1] University of Oregon, Eugene OR 97403, USA
{marsagli,hank}@cs.uoregon.edu
[2] National Center for Atmospheric Research, Boulder CO 80305, USA
shaomeng@ucar.edu

Abstract. We explore a technique for saving full spatiotemporal 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. Rather than limiting our data reduction to individual time slices or time windows, our algorithms act on individual locations and save data to disk as temporal intervals. Our results show that the efficacy of piecewise approximations varies based on the desired error bound guarantee and tumultuousness of the time-varying data. We ran our in situ algorithms for one simulation and experienced promising results compared to the traditional paradigm. We also compared the results to two data reduction operators: wavelets and SZ.

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 store the current state of the simulation 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. As a result, information between time slices is lost, creating a risk of lost science.

The purpose of our research is to enable the storage of full spatiotemporal data, and our research idea is enabled by an emerging processing paradigm for visualization on high-performance computers. This paradigm uses a combination of in situ processing and post hoc processing. First, in situ routines transform — and often reduce — data, which is then saved to disk. Second, the post hoc routines are then used for explorative visualization after the simulation has finished running. While this paradigm has emerged to enable explorative use cases in the face of I/O limitations, we believe it also creates opportunities for better preserving full spatiotemporal data. This is the case with our own work, which is less about data reduction than it is about increasing temporal resolution. That said, saving full spatiotemporal data requires prohibitive storage capabilities, and so data reduction is still necessary; we demonstrate reduction factors that

reduce storage needs to being equal to the storage needs of the traditional paradigm — with the benefit of having full temporal access.

Our research approach depends on the concept of temporal intervals. However, raw 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 an in situ algorithm with the choice of three different compressors that compress data temporally with a guaranteed error rate. For the chosen algorithm and user defined error rate, the algorithm operates individually on every grid point within a simulation. Within each grid point, for every generated value, the algorithm decides whether its current approximation sufficiently captures the value, or whether the approximation should be reset. We ran our algorithm on multiple data sets both post-hoc and in situ. We then analyzed the storage differences of our method compared to other known data reduction techniques. By implementing temporal data reduction in situ, this research has shown an improvement over saving at fixed intervals in the traditional paradigm. By compressing data temporally, the algorithm provides full temporal resolution of the simulation data and is a viable compression option compared to other techniques, namely wavelets and SZ.

2 Related Work

This section begins by classifying related work into categories of compression algorithms and what type of temporal data they operate on:

- Individual Time Slice Data (Section 2.1)
- Multiple Time Slice Data (Section 2.2)
- Complete Temporal Data (Section 2.3)

The remainder of this section considers the impact of introducing error during compression (Section 2.4) and how our approach differs from previous work (Section 2.5).

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 compressors. Such compressors include FPC [7] for 64-bit floating-point scientific data;

FPZIP [33] for generic scientific data; a framework by Fout and Ma [14] for floating-point volumetric data; and the approach by Chen et al. [9] for compressing irregular grids. Certain operations (e.g., sorting, mutation, etc.) can be applied to "precondition" the data to improve compression effectiveness. Such preconditioners include MAFISC [17] which consists of five individual filters; ISOBAR [40] which statistically separates data based on compressability; and a binary mask approach developed by Gomez et al. [15]. MPC [44] and FPcrush [6] are compressors built on top of preconditioners. However, lossless compression techniques can usually only reduce data by modest amounts and can be computationally intensive, and thus less attractive for in situ data reduction.

Another approach to compressing individual time slices is *lossy* data compression. ZFP [31] achieves high compression via spatial coherence and bit reduction, but this technique strongly favors smooth data. Similarly, the RBD algorithms [19] utilize spatial locality by viewing data point grids as graphs and representing subgraphs as singular values based on a user defined error bound. The lossy version of FPZIP [33] uses predictive encoding with a relative error bound. The newly emerged SZ compressors [12, 41] use multiple predictive approaches to achieve effective compression.

Lossy wavelet compression has also been used in many visualization studies. 3D wavelet transforms have been shown to provide random access of voxels [18, 38], level of detail [5, 39, 36], and fast volume rendering [21, 16]. Data reduction achieved by wavelets has also been used to visualize large data in real time, for example, at a cost of 5 seconds per time slice on a $1,024^3$ turbulent-flow data set [42]. Work by Li et al. [28] improves the time cost of wavelet compression using parallel primitives on heterogeneous architectures, making wavelets a viable option for in situ compression [27]. These wavelet-based compression schemes, however, do not guarantee an error bound, and users often need to try their data before knowing the amount of deviation.

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 [22] 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. It is reported to compress tumultuous data by nearly 85% with a 99% average correlation between the original and compressed data with little overhead in runtime [23]. Lehmann et al. [25] extended ISABELA by adding corrective computations to the sorted data and loosening the restrictions of when to write a window to memory. Additionally, their technique added support for selective loading of regions with varying levels of resolution.

Wavelet compression is also reported to operate across multiple time slices, making it a spatio-temporal compressor [30]. Here time slices after traditional spatial wavelet transforms are grouped into "windows" to perform additional temporal wavelet transforms, making use of the temporal coherence. The authors reported an approximately $2:1$ benefit of incorporating temporal compression.

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

2.3 Complete Temporal Data

This section encompasses full temporal resolution at the same spatial resolution as the native grid.

IDEALEM [24] 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.

The work by Fernandes et al. [13] improved upon volumetric depth images (VDIs) by exploiting temporal coherence and utilizing delta encoding, allowing them to achieve both data reduction and explorable time-varying imagery.

Our work also falls under this category of compression.

And finally, we remind our readers that a more comprehensive survey on scientific data reduction techniques was done by Li et al. [29].

2.4 Impact of Error in Compression

Some work has been done to study the extent of error and error propagation on simulations utilizing compression techniques or the errors present in the subsequent analysis.

Lindstrom [32] researched error distributions of lossy compression, and found that the choice of compressors affects the error distribution and subsequent autocorrelation. Baker et al. [4, 3] developed a methodology to evaluate the compression impact on climate data using ensembles of data, and later argued that multiple compression schemes applied on different variables can achieve the best compression on such data [2]. Woodring et al. [43] also looked at climate data compression, and argued that the maximum error (L^{∞}-*norm*) being most effective in communicating compression error.

With respect to visualization, Ma et al. [34] informed users the compression variations by encoding them into marching cubes. Li et al. [26] researched into the impact of data compression to visual analytics in regard to turbulent flow data.

2.5 How Our Approach Differs From Previous Work

Our work is specifically focused on providing complete temporal resolution at the same spatial resolution of the simulation. Since this data is assumed to be too large to store, we compress the data to make storage feasible. This is, of course, related to previous compression work, although we place a particular focus on guaranteeing accuracy — without guaranteed accuracy our approach has little meaning. As an example, it would be possible to use the traditional approach, temporal sampling, and interpolate between adjacent time slices. But such an approach would not actually be especially informative,

since it would miss features that fell outside the interpolation. We experimented with this approach, and found that more than 1% of all interpolated values were more than 5% away from their correct values (see Table 7); we believe this rate of inaccuracy is unacceptable.

Our work operates on each grid point's continuous stream of data, attempting to compress every incoming value. Our research includes three in situ compressors that reduce temporal data into piecewise approximations. Each compressor uses a sliding window similar to the SWAB [20] data mining technique for segmenting time series data.

3 Algorithm

The objective of this work is to provide full temporal resolution with spatial resolution identical to the native grid, that is feasible enough to store on disk, and comes with an error bound guarantee; this is accomplished via a tailored compression technique.

Our algorithm was designed to work in situ and works as follows. Our algorithm creates an instance of a compression object for each grid point in the simulation. Each such object reduces its respective time series into piecewise approximations. The compression object at a given grid point runs independent of the compression objects at other grid points. For every generated value at a grid 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 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 a maximum error bound. 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, by using a maximum relative error bound rather than an absolute error bound, we are imposing a stricter threshold on our compression algorithms. But we believe this guarantee, that results in no post-hoc artifacts, is important to domain scientists. Future work could involve relaxing this error guarantee, which would undoubtedly result in better compression.

3.2 Compression Approaches

As discussed earlier, our overall algorithm instantiates a compressor object for each grid point. We considered three types of compression approaches, each of which could be instantiated as the compressor objects for our algorithm. The three compression approaches are:

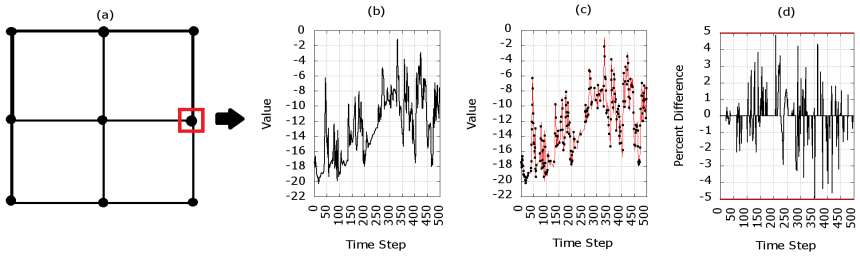

Fig. 1: Image (a) represents a simulation mesh with nine grid points. As the simulation runs, it generates a stream of data at each grid point. In our notional example, we consider the data stream at the grid point indicated with the red square. We apply a compressor object at each grid point which independently compresses its time series into a piecewise approximation. Image (b) shows the original time series of one grid point with 500 values, (c) is the piecewise approximation, and (d) is the per point difference of the two. In the example, the error bound is selected to be 5%, and image (d) shows our approximation to be within that error bound.

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

While each compression approach is similar, they are differentiated by how they approximate the time series. Each compression object approximates a stream of data by attempting to compress each value it encounters one by one.

Figure 1 shows an instance of a compressor object on a single grid point, the resulting approximation, and the difference between the original and approximated time series.

Piecewise Linear (PL) This compression approach transforms consecutive grid point values into piecewise linear approximations as the simulation progresses.

The approximation begins by taking the first value of the stream and assigning it to the starting value of the approximation. When the subsequent value is encountered, the slope (i.e. rate of change) is calculated. Using the starting value and the associated slope, it's possible to calculate the predicted value for a given time step. If $f(n)$ is a valid linear approximation up to time step n with x_i being the start of this approximation, and if t is the different in simulation time between two cycles, then the $slope = \frac{x_{i+1}-x_i}{t}$. Let n be the current time step and i be the starting time step of the approximation, let k be the difference between the current time step and the approximation's starting time step. In this case, $k = (n-i)*t$. The prediction for time step n becomes

$$f(n) = slope * k + x_i \tag{1}$$

From the starting value, the slope is extended one unit for every time step, creating our approximation as seen in Equation (1). For each time step, the prediction is compared to the current value. If the difference between the approximated value and the current value is within the error bound, ε, then this process continues. If the difference between

```
Struct PC_CompressorObject_state contains
  | float prediction;
end
void PiecewiseConstant_AddToStream(PC_CompressorObject_state *s, float cur_value,
                                   int cycle, int grid_point_index, float error_target)
if (cycle == 0) then
  | s→prediction = cur_value;
else
  | float allowable_diff = abs(cur_value)*error_target;
  | float actual_diff = abs(cur_value − s→prediction);
  | if (actual_diff > allowable_diff) then
  |   | s→prediction = cur_value;
  | end
end
```

Algorithm 1: The pseudocode for the PC compressor object for a single grid point. This object will approximate the grid point's stream of values with a guaranteed error bound.

the current value and the approximated value does not fall within the error bound then the starting value, slope, and previous time step are saved. Once an approximated line has been saved, the approach then restarts with a new starting point.

$$f(n+1) = \begin{cases} slope * k + x_i & \text{Error}(x_{n+1}, f(n+1)) \leq \varepsilon \\ x_{n+1} & \text{Error}(x_{n+1}, f(n+1)) > \varepsilon \end{cases} \tag{2}$$

The new starting point is the last value that was not within the error bound of the approximation, as seen in Equation (2). A new slope will then be calculated using the starting point and the subsequent value.

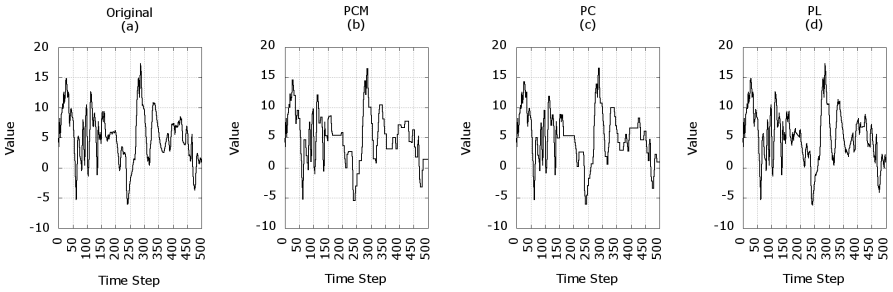

Fig. 2: Image (a) shows the original values. Images (b)-(d) are approximations created by our compression approaches using a 5% error bound. Image (b) is the piecewise approximation created by the PCM compression approach. Image (c) is the piecewise approximation created by the PC compression approach. And image (d) is the piecewise approximation created by the PL compression approach.

Piecewise Constant (PC) Piecewise Constant, our second approach and the approach described in Algorithm 1, is similar to Piecewise Linear. However, in the Piecewise Constant compression approach, a single value represents an interval of the time series, as in Equation (3). If $f(n)$ is a valid constant approximation up to time step n and if x_i is the start of this approximation, then the prediction for time step n becomes

$$f(n) = x_i \tag{3}$$

The approximated line is extended until the difference between the predicted value and the current value exceeds the error bound, ε.

$$f(n+1) = \begin{cases} x_i & \text{Error}(x_{n+1}, f(n+1)) \leq \varepsilon \\ x_{n+1} & \text{Error}(x_{n+1}, f(n+1)) > \varepsilon \end{cases} \tag{4}$$

Once the error bound is exceeded, the compression approach saves the previous reference value and last valid time step, and the value that could not be represented by the approximation becomes the reference value going forward, as in Equation (4).

Algorithm 1 shows the pseudocode of the PC compressor object at a single grid point and how this particular approach approximates each individual value encountered. For each incoming point in the stream, the compressor object makes a decision based on the error bound. If the incoming point and the predicted value are within an error bound then the prediction is valid and no actions are taken. Otherwise, the compression object saves the previous prediction and begins a new approximation with the current value.

Piecewise Constant Mean The third compression approach, Piecewise Constant Mean, is similar to the first two, but takes the average of the encountered values per approximation interval. If $f(n)$ is a valid constant approximation up to time step n and if x_i is the start of this approximation, then the approximation is the average of all encountered points from x_i to x_n. The prediction for time step n becomes

$$f(n) = AVG(x_i, x_{i+1}, \ldots, x_{n-1}, x_n) \tag{5}$$

With each new incoming value, the compression approach calculates an approximation. The approximation is the mean of all the encountered values in the current interval, as in Equation (5). The compression approach also makes sure that no point in the interval range falls outside the error bound in relation to the mean, as in Equation (6).

$$f(n+1) = \begin{cases} & \text{Error}(range\_max, f(n+1)) \leq \varepsilon \\ \text{AVG}[x_i, x_{n+1}] & \quad \textbf{and} \\ & \text{Error}(range\_min, f(n+1)) \leq \varepsilon \\ x_{n+1} & \text{o.w.} \end{cases} \tag{6}$$

That is, for every new value the approach checks to make sure that all represented points of the interval are within $[mean - \varepsilon, mean + \varepsilon]$ where ε is the error bound. This error bound needs to be checked for the minimum and maximum value of the current interval mean approximation. If the current minimum and maximum are within the error bound of the mean, which can change with every added value, then all values within the minimum and maximum are within their relative error bound to the approximated mean.

Figure 2 shows our three compression approaches approximating the same time series.

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

Constructing an approximation of the original data requires the desired time slice along with the compressed data.

The reconstruction process from the compressed data works as follows. The data from every grid point is written and subsequently available in its own file. Better organization of this data, for example, in one file, is future work. This information consists of the saved data values and their corresponding time steps. In addition to the time step and value, the PL approach also saves the associated slope.

Given the desired time slice, it is straightforward to find the corresponding value within each grid point's file. If the desired time slice falls within an approximation interval, we know that the value associated with that interval represents the desired time slice at some grid point. For reconstructing the PL approach, it is necessary to determine where the desired time slice falls within the approximation interval, this information can then be used to calculate the approximation value using the starting value and slope for that interval.

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

Table 1: The data members for each compressor object. For each approximation, it is necessary to have either the starting value or the current approximation value. For the PL approach, each object stores the slope associated with the current approximation. The PL and PCM also require the count, or number, of time steps they are approximating. The PL needs the count for post-hoc reconstruction, in order to calculate the current position of the line using the slope and starting value, and the PCM needs it to calculate the mean of all relevant values for the line. Note that an unsigned short could be sufficient for the count variable. Additionally, PCM requires the local range to be stored in order to guarantee the error bound criteria.

Reconstruction Time Due to the rudimentary file organization, with each location having its own approximation file, the reconstruction time is proportional to the size of each file. With 250,000 time steps, reconstructing the later time steps takes longer than reconstructing earlier time steps due to searching individual files for the desired approximation interval and respective value. The reconstruction time of the compressed data set in Phase 3 of our research ranged from 30 seconds to 2 minutes. We aim to decrease reconstruction time in future work by implementing a more sophisticated way to save the approximation data.

4 Evaluation

Our experiments were designed to quantitatively evaluate our technique. This section outlines the parameters of the research conducted.

Our study was divided into three phases. In the first phase, we evaluated our technique in various configurations, including multiple data sets and error bounds. Phase one helped determine the potential and viability of our approaches as well as what type of data is best suited for our experiments. In the second phase, we compared our method with data reduction techniques that focus solely on spatial coherence. In the third phase, we ran an in situ study using our technique to demonstrate its viability as an in situ compressor.

The discussion of experiment factors is organized into configuration parameters (Section 4.1), measurement and metrics (Section 4.2), computing environment (Section 4.3), and software (Section 4.4).

4.1 Experiment Configuration

We varied the following factors:

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

In the first phase of our study, the cross product of these factors was explored. In the second and third phases, only a subset of the cross product was considered.

Compression Approaches We evaluated the three compression approaches described in Section 3: Piecewise Constant (PC), Piecewise Linear (PL), and Piecewise Constant Mean (PCM).

Data Sets The first phase focused on time series data, with each time series coming from a spatial location within a simulation. The times series data we considered is as follows:

- LULESH [1]: A collection of 8 time series (i.e. from 8 different spatial locations) from an explosion data set. Each time series has 4,561 single-precision data points; each time series is 18.244kB.

- Tornado [37]: A collection of 10 time series from a tornado data set. Each time series has 500 single-precision data points; each time series is 2kB.
- XGC1 Ion Particles [8]: A collection of 25 time series from a tokamak data set. Each time series has 818 single-precision data points; each time series is 3.272kB.
- GHOST [35]: A collection of 15 time series from a turbulent rotational flow data set. Each time series has 5,000 single-precision data points; each time series is 20kB.

For the second phase we compared our algorithm results on 10,000 time slices of GHOST with two spatial compression techniques: wavelets and SZ-1.4. Each time slice of GHOST is roughly 8MB in size, with each time slice being comprised of 128^3 single-precision floating point values.

For the third phase, we ran our algorithm in situ on the GHOST simulation for 250,000 time steps.

User Defined Error Bound We applied three user defined error bounds: 1%, 3%, and 5%. Our error bound is a point-wise relative error bound, and is the same error bound that SZ-1.4 provides as a compression option. The point-wise relative error bound acts on individual grid points. If a grid point has value v, and the point-wise error bound is set at 5%, then the reconstructed value will be $\texttt{abs}(0.05 * v)$ away from v.

From here on we will refer to this error bound approach as *point-wise error bound*.

4.2 Phase Overview and Measurements

The first phase was comprised of applying our algorithm to the four data sets, LULESH, Tornado, XGC1 Ion Particles and GHOST. For each data set we ran our algorithm with each compression approach and each user defined error bound. This means we ran 36 experiments each with multiple measurements. For each data set we calculated the maximum compression ratio, the minimum compression ratio, and average compression ratio.

For the second phase, we compared our compression approach with two other compressors: wavelets and SZ-1.4. In this phase, we fixed certain aspects of each compression operator, compressed 10,000 time slices of GHOST, and measured the results. Because our method produces results with full temporal resolution that allow for post-hoc exploration, we applied wavelets and SZ-1.4 to every time step to mimic the temporal resolution that our algorithm produces.

For the wavelet compressor, we fixed the compression ratio and measured the average point-wise error, maximum point-wise error, and average compression time. We ran four wavelet configurations that guaranteed compression size, namely, 32:1, 64:1, 128:1, and 256:1 compression.

For SZ-1.4, we fixed the point-wise error bound to be 1%, 3%, and 5% and then measured the average point-wise error, maximum point-wise error, average compression ratio, and average compression time.

We also applied our approach to the 10,000 time slices of GHOST and measured the average point-wise error bound, maximum point-wise error bound, average compression ratio, and average compression time. We note that we did not run the SZ-1.4

or wavelet compression operators in situ, but applied the compressors to each of the 10,000 time slices of GHOST post-hoc.

In the third phase, our algorithm was run in situ on GHOST with each compression approach and each error bound. The GHOST simulation setup has 128^3 grid points oriented as a cube. We instantiated a tightly coupled compressor object on each grid point and ran each simulation configuration for 250,000 time steps. The measurements on GHOST include the number of values saved per time step and resulting average compression, as well as the average computation time and average I/O time for all three compression approaches at each error bound.

For all experiments, we only consider a single scalar value per grid point per time step. While this may not be in line with current simulations, we decided to explore the feasibility of in situ compression on individual scalar values and save implementing in situ compression on multiple scalar values for future work.

4.3 Hardware

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

4.4 Software

Besides our proposed algorithm, we used two other compressors as comparators: a wavelet compressor from VAPOR [10, 11] and SZ-1.4 [41].

VAPOR is an open-source visualization package for the the geoscience community that adopts wavelet transforms to achieve compression. It utilizes coefficient prioritization as its compression strategy, which means wavelet coefficients are prioritized based on their information content, and the ones containing less information are discarded. We used the best wavelet configurations reported in [26] in our study.

SZ-1.4 is an open-source lossy compression code that uses a multidimensional prediction model. The model compresses each data point using the prediction of nearby values in multiple dimensions. From here on we will refer to SZ-1.4 as SZ.

For all three softwares, we compiled them using `gcc` with the `-O2` optimization on for a fair comparison.

5 Results

This results section is organized as follows: Phase One (Section 5.1) of our evaluation applies our compression algorithm on time series data. Phase Two (Section 5.2) compares our compression results with two other compression methods: wavelets and SZ. Phase Three (Section 5.3) evaluates our algorithm in situ for the GHOST simulation and compares our compression results with temporal sampling.

5.1 Phase One: GHOST, LULESH, XGC1 Particle Ions, and Tornado

The four data sets displayed a wide range of compression ratios compared to the size of the original time series data. Table 2 displays the minimum, average, and maximum compression ratios achieved for each data set using the PCM compressor with a 5% error bound.

The best results were for GHOST. Over the fifteen time series, the worst compression was 11:1 and the best compression was 2500:1. The average over the time series was just under 41:1. Assuming the time series were representative, this means we could save all temporal data using the same storage requirements as the traditional temporal sampling technique when saving every 41^{st} time slice.

The results for the Tornado and XGC1 data sets were much worse. The time series for these data sets were much more turbulent temporally, and they do not seem to be appropriate for our technique. As a result, we eliminated them from further consideration, recognizing that our technique works best with smooth data. Further, although the LULESH results were better than the Tornado and XGC1 data sets, we decided to focus on GHOST for our in situ experiments, since it produced the best results.

Data Set	Min CR	Max CR	Avg. CR
GHOST	11:1	2500:1	40.99:1
LULESH	16.89:1	21.51:1	18.6:1
Tornado	1.11:1	3.7:1	1.07:1
XGC1	.63:1	2.11:1	.845:1

Table 2: The minimum, maximum, and average compression ratios of all four data sets using the PCM approach and an error bound of 5%.

5.2 Phase Two: Comparison with Wavelets & SZ

Wavelets are fundamentally different from our approach in that wavelets allow the end user to pick an arbitrary compression level with no consideration to accuracy. In contrast, our approach and SZ both guarantee an error bound, but they are unable to guarantee a given level of data reduction. We found the best way to compare wavelets and our approach was to find configurations where they achieve similar data reductions, and then compare their respective accuracies. Results for wavelets are in Table 3 and for our compression algorithm using the PCM approach are in Table 5.

Time	0.327s	0.327s	0.327s	0.327s
Space	32:1	64:1	128:1	256:1
Avg Error	10.89%	15.46%	23.67%	34.6%
Max Error	352k%	443k%	776k%	980k%

Table 3: The results from Phase 2 using wavelet compression on 10,000 time slices of the GHOST data set.

For 5% error, our PCM approach achieved a compression ratio of 34:1, whereas the wavelet approach with fixed 32:1 compression had an average error of 10.89%. This is a bigger contrast than initially appears, as the maximum error for our approach is twice as good as the average error for wavelets in this case. As a result, we conclude that our method is superior for spatiotemporal data with high temporal frequency.

Our approach produced better compression rates for each error bound compared to SZ (Table 4). We believe this is due to the fact that SZ is based on spatial coherency, whereas our method is based on temporal coherency. For example, with a 5% error guarantee, SZ achieved a compression ratio of 4:1, whereas our approach using PCM achieved 34:1 compression.

Time	0.155s	0.156s	0.156s
Space	2.37:1	3.46:1	4.01:1
Relative Error	1%	3%	5%
Avg Error	.0254%	0.051%	0.102%
Max Error	1.01%	3%	5%

Table 4: SZ

Time	0.364s	0.325s	0.304s
Space	9.61:1	21.0:1	35.0:1
Relative Error	1%	3%	5%
Avg Error	0.481%	1.79%	2.3%
Max Error	1%	3%	5%

Table 5: Intervals: PCM

Fig. 3: Results from Phase 2. In this phase, 10,000 time slices of the GHOST data set were compressed using SZ and using our interval compression with the PCM approach.

SZ had the best runtime, taking roughly half the compression time experienced by both the parallelized wavelet implementation and our interval approach. Overall, all three compression approaches have runtime overhead that we feel are acceptable for in situ.

5.3 Phase Three: In Situ Experimentation

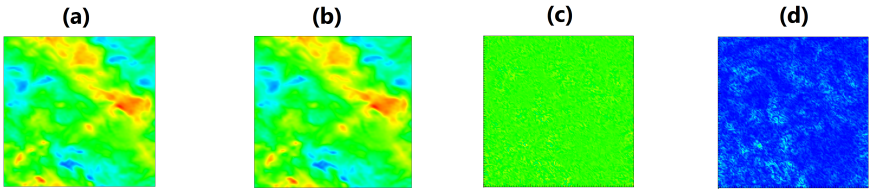

Fig. 4: Image (a) is the original data at time slice 25,000. Image (b) is the reconstructed data that used the PCM algorithm with a 5% error. Image (c) is a data comparison of their difference. And image (d) is the absolute value of their difference. Both the dark and light blue in image (d) represents minimal differences between the original and reconstructed data, differences well within the 5% error.

Phase Three ran our algorithm in situ and measured the number of values saved per time step. We then calculated the compression ratio based on the size of an entire time slice.

Figure 4 shows an example of decompressed data and how close it is to the original data. With our technique, the reconstructed data is always within the user defined error bound.

Algorithm	1%	3%	5%
PCM	11.27:1	29.6:1	45.59:1
PL	14.56:1	49.9:1	63.55:1
PC	6.64:1	16.38:1	25.45:1

Table 6: The average compression ratio of the reduced data per time step of each algorithm and each user defined error percent over 250,000 time steps.

With a 5% user defined error, GHOST, on average, saves between 11,000 and 41,200 values per time step depending on the approach, as shown in Figure 5. With the PCM and PC algorithms this means saving a 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 of 25.45:1-63.55:1 per time step as opposed to saving the entire time slice.

Table 6 lists the average compression ratio each approach achieves per error percent.

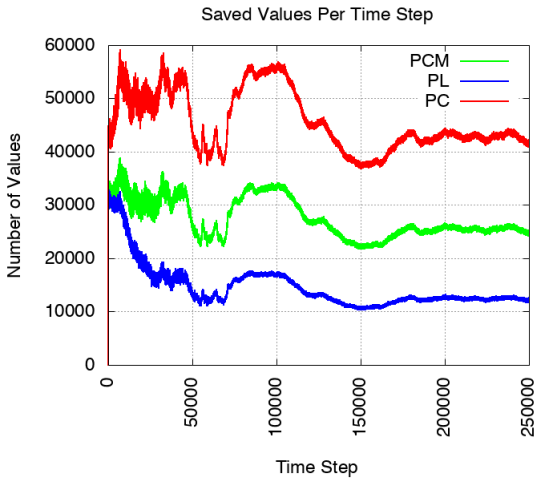

Fig. 5: For each of the 250,000 time steps, the three methods, with a 5% error bound, saves out reduced data that is a fraction of the original data. For instance, out of the total grid of 128^3, the PCM method saves out between 20,000 to 40,000 approximation values per time step.

Runtime Our algorithm does affect simulation runtime. Typically, the algorithm adds roughly 30-35% to the runtime depending on the compression approach when running on a single core. On average, each time step of GHOST executes in 1.10 seconds per cycle. With minor percentage variations depending on the error bound, the PCM algorithm increased the runtime by an average of 34% to a total runtime of 1.48 seconds per cycle. The PC algorithm increased the runtime of each time step by an average of 31% to a total runtime of 1.44 seconds per cycle. And the PL algorithm increased the runtime by an average of 27% to a total runtime of 1.40 seconds per cycle.

These results are based on a single scalar value being approximated at each grid point. But many simulations have multiple scalars at each grid point. We ran our algorithms with two scalar values being approximated independently at each grid point. We found that an additional scalar approximation added negligible strain to the simulation. For instance, the PL run time increased from a total of 1.40 seconds to 1.44 seconds when approximating one scalar value and two scalar values,

respectively. We believe this is due to the fact that adding scalar values does not increase the number of cache misses since each approximation can be in the same data structure. But approximating additional scalars means adding, and in most cases doubling, the memory requirements for the approximations.

I/O We consider efficient I/O to be an area of future work for this method. Since our current implementation has many small writes, I/O times could be as much as six seconds per time step. We believe this slowdown is unacceptable in practice, but could be mitigated in the future by staging data in a deep memory hierarchy with occasional saves of large numbers of temporal intervals en masse.

Scalability Our algorithm is embarrassingly parallel, i.e., it can proceed on a per node basis with no coordination across nodes. Therefore, we expect excellent scalability. One point of contention, however, would be in coordinating the way we write to disk.

Our Approach vs. Temporal Sampling Although users rarely want to output every time slice, our approach, with a 5% error bound, would lead to a 63.55:1 improvement. In a more common case of temporal sampling, such as saving every 50^{th} time step, the algorithm has an improvement of 1.27:1. This is calculated by comparing the amount of data of necessary to save a complete time slice every 50^{th} time step, as in the traditional model, compared to saving intervals intermittently for 50 time steps with our in situ compression. But our method provides full temporal resolution at the same spatial resolution as the native grid for the entirety of the simulation with a guaranteed error bound. Whereas temporal sampling does not provide full spatiotemporal resolution and can only accurately depict the saved time slices.

With temporal sampling, linear interpolation can be used to reconstruct the missing time slices. We reconstructed a time slice using linear interpolation between time slices fifty time steps apart and measured the error compared to the original time slice, as shown in Table 7. While the majority of points in the reconstruction had a point-wise relative error less than or equal to 1%, there was still an occurrence of errors greater than 5%. With linear interpolation there is no guarantee that the reconstructed slices will be similar to the ground truth and may introduce higher than anticipated error rates. In the context of this example, our algorithm provides information (with guaranteed error) for an additional 49 time slices for every individual time slice saved using the traditional model, all while saving less data overall.

Error	Percentage of Points
$<= 1\%$	98.1146%
$<= 3\%$	0.475788%
$<= 5\%$	0.141287%
$> 5\%$	1.26834%

Table 7: The percentage of points with a point-wise error less than or equal to 1%, 3%, 5%, and greater than 5% for the reconstructed data using linear interpolation between two time slices of GHOST fifty time steps apart.

6 Conclusion

With this work, we consider a new approach for visualization of time-varying data, namely temporal intervals. We introduced an algorithm that uses several compressors that use the temporal interval approach. Our findings show that the algorithm is effective enough to create feasible storage requirements for some data (but not others). Temporal intervals are a different paradigm than the traditional technique. They provide complete temporal information, as opposed to the traditional method of temporal sampling. But, to make the technique practical, the displayed results include error. Our approach adapts to this error by providing firm limits on the total amount of error that can be presented to an end user. In all, we believe this approach has merit, as it provides domain scientists certainty that they are not missing important science — which can be a pitfall with temporal sampling.

In terms of specific findings, this work presents three piecewise approximation compressors that guarantee an error bound and provides full temporal resolution for post-hoc exploration. We found that, depending on the data set and specifically for GHOST, the compression algorithms can reach a compression ratio greater than 2500:1 for individual grid points, and upwards of 63:1 for an entire simulation. The compression approaches achieved these results on a data set with smooth temporal coherence with data fluctuation spread out over time. On the other hand, this algorithm did not achieve high compression rates on the XGC1 or Tornado data sets, although it did show decent compression with the LULESH times series.

7 Future Work

The biggest area of future work is to develop new compression operators that improve the value proposition for stakeholders. New compression operators could include more elaborate encoding strategies with higher-order reconstruction, this would undoubtedly require more memory per grid point but may result in better compression. If compression operators can be developed that produce, for example, 1000:1 reductions in data storage, then we believe this technique could be very useful in the future to address shortcomings in supercomputer I/O and scenarios where domain scientists would typically opt to save data less and less often.

We will continue to evaluate these approaches on other data sets and simulations to determine the extent of their capabilities on both smooth and tumultuous time varying data. We will also work on reducing the I/O and reconstruction time, in particular the way we store our approximations to disk. A final point of evaluation is on the effects resulting from transitioning from one temporal interval to another. In the worst case, this could lead to "flickering" when animating over time. We have not observed this in our own experiments, and we believe that the phenomenon is prevented by our error bound guarantee. Regardless, more evaluation is needed.

References

1. Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Tech. Rep. LLNL-TR-490254

2. Baker, A., Xu, H., Hammerling, D., Li, S., Clyne, J.: Toward a Multi-method Approach: Lossy Data Compression for Climate Simulation Data. In: Proceedings of ISC workshops on Data Reduction for Big Scientific Data (DRBSD-1). Frankfurt, Germany (Jun 2017)
3. Baker, A.H., Hammerling, D.M., Mickelson, S.A., Xu, H., Stolpe, M.B., Naveau, P., Sanderson, B., Ebert-Uphoff, I., Samarasinghe, S., De Simone, F., Carbone, F., Gencarelli, C.N., Dennis, J.M., Kay, J.E., Lindstrom, P.: Evaluating lossy data compression on climate simulation data within a large ensemble. Geoscientific Model Development **9**(12), 4381–4403 (2016)
4. Baker, A.H., Xu, H., Dennis, J.M., Levy, M.N., Nychka, D., Mickelson, S.A., Edwards, J., Vertenstein, M., Wegener, A.: A methodology for evaluating the impact of data compression on climate simulation data. In: Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing. pp. 203–214. HPDC '14, ACM, New York, NY, USA (2014)
5. Bertram, M., Duchaineau, M.A., Hamann, B., Joy, K.I.: Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization. In: Proceedings of the conference on Visualization'00. pp. 389–396. IEEE Computer Society Press (2000)
6. Burtscher, M., Mukka, H., Yang, A., Hesaaraki, F.: Real-time synthesis of compression algorithms for scientific data. In: SC16: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 264–275 (Nov 2016)
7. Burtscher, M., Ratanaworabhan, P.: Fpc: A high-speed compressor for double-precision floating-point data. IEEE Trans. Comput. **58**(1), 18–31 (Jan 2009)
8. Chang, C., Ku, S., Diamond, P., Lin, Z., Parker, S., Hahm, T., Samatova, N.: Compressed ion temperature gradient turbulence in diverted tokamak edge). Physics of Plasmas (1994-present) **16**(5), 056108 (2009)
9. Chen, D., Chiang, Y.J., Memon, N., Wu, X.: Lossless Geometry Compression for Steady-State and Time-Varying Irregular Grids. In: Santos, B.S., Ertl, T., Joy, K. (eds.) EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization. The Eurographics Association (2006)
10. Clyne, J., Mininni, P., Norton, A., Rast, M.: Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. New Journal of Physics **9**(8), 301 (2007)
11. Clyne, J., Rast, M.: A prototype discovery environment for analyzing and visualizing terascale turbulent fluid flow simulations. In: Electronic Imaging 2005. pp. 284–294. International Society for Optics and Photonics (2005)
12. Di, S., Cappello, F.: Fast error-bounded lossy hpc data compression with sz. Proc. IPDPS. IEEE (2016)
13. Fernandes, O., Frey, S., Sadlo, F., Ertl, T.: Space-time volumetric depth images for in-situ visualization. In: 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV). pp. 59–65 (Nov 2014)
14. Fout, N., Ma, K.L.: An adaptive prediction-based approach to lossless compression of floating-point volume data. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2295–2304 (Dec 2012)
15. Gomez, L.A.B., Cappello, F.: Improving floating point compression through binary masks. In: Big Data, 2013 IEEE International Conference on. pp. 326–331. IEEE (2013)
16. Guthe, S., Strasser, W.: Real-time decompression and visualization of animated volume data. In: Proceedings of IEEE Visualization (VIS'01). pp. 349–572 (Oct 2001)
17. Hübbe, N., Kunkel, J.: Reducing the hpc-datastorage footprint with mafisc—multidimensional adaptive filtering improved scientific data compression. Computer Science - Research and Development **28**(2), 231–239 (2013)

18. Ihm, I., Park, S.: Wavelet-based 3d compression scheme for interactive visualization of very large volume data. In: Computer Graphics Forum. vol. 18, pp. 3–15. Wiley Online Library (1999)
19. Iverson, J., Kamath, C., Karypis, G.: Fast and effective lossy compression algorithms for scientific datasets. In: Proceedings of the 18th International Conference on Parallel Processing. pp. 843–856. Euro-Par'12, Springer-Verlag, Berlin, Heidelberg (2012)
20. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. Proceedings 2001 IEEE International Conference on Data Mining pp. 289–296 (2001)
21. Kim, T.Y., Shin, Y.G.: An efficient wavelet-based compression method for volume rendering. In: Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications. pp. 147–156. IEEE (1999)
22. Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: 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)
23. Lakshminarasimhan, S., Shah, N., Ethier, S., Ku, S.H., Chang, C.S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: ISABELA for effective in situ compression of scientific data. Concurrency and Computation: Practice and Experience **25**(4), 524–540 (2013)
24. Lee, D., Sim, A., Choi, J., Wu, K.: Novel data reduction based on statistical similarity. In: Proceedings of the 28th International Conference on Scientific and Statistical Database Management. p. 21. ACM (2016)
25. Lehmann, H., Jung, B.: In-situ multi-resolution and temporal data compression for visual exploration of large-scale scientific simulations. In: 2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV). pp. 51–58 (Nov 2014)
26. Li, S., Gruchalla, K., Potter, K., Clyne, J., Childs, H.: Evaluating the efficacy of wavelet configurations on turbulent-flow data. 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV) pp. 81–89 (2015)
27. Li, S., Larsen, M., Clyne, J., Childs, H.: Performance impacts of in situ wavelet compression on scientific simulations. In: Proceedings of the In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization Workshop. ISAV2017, ACM, New York, NY, USA (2017)
28. Li, S., Marsaglia, N., Chen, V., Sewell, C., Clyne, J., Childs, H.: Achieving Portable Performance For Wavelet Compression Using Data Parallel Primitives. In: Telea, A., Bennett, J. (eds.) Eurographics Symposium on Parallel Graphics and Visualization. The Eurographics Association (2017). https://doi.org/10.2312/pgv.20171095
29. Li, S., Marsaglia, N., Garth, C., Woodring, J., Clyne, J., Childs, H.: Data reduction techniques for simulation, visualization and data analysis. Computer Graphics Forum (Mar 2018). https://doi.org/10.1111/cgf.13336
30. Li, S., Sane, S., Orf, L., Mininni, P., Clyne, J., Childs, H.: Spatiotemporal wavelet compression for visualization of scientific simulation data. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 216–227 (Sept 2017)
31. Lindstrom, P.: Fixed-rate compressed floating-point arrays. IEEE Transactions on Visualization and Computer Graphics **20**(12), 2674–2683 (2014)
32. Lindstrom, P.: Error Distributions of Lossy Floating-Point Compressors. Joint Statistical Meetings (October 2017)
33. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. IEEE Transactions on Visualization and Computer Graphics **12**(5), 1245–1250 (Sep 2006)
34. Ma, J., Murphy, D., O'Mathuna, C., Hayes, M., Provan, G.: Visualizing uncertainty in multi-resolution volumetric data using marching cubes. In: Proceedings of the International Working Conference on Advanced Visual Interfaces. pp. 489–496. ACM (2012)
35. Mininni, P., Alexakis, A., Pouquet, A.: Large-scale flow effects, energy transfer, and self-similarity on turbulence. Physical Review E **74**(1), 016303 (2006)

36. Olanda, R., Pérez, M., Orduña, J.M., Rueda, S.: Terrain data compression using wavelet-tiled pyramids for online 3d terrain visualization. International Journal of Geographical Information Science **28**(2), 407–425 (2014)
37. Orf, L., Wilhelmson, R., Wicker, L.: Visualization of a simulated long-track ef5 tornado embedded within a supercell thunderstorm. Parallel Computing **55**, 28–34 (2016)
38. Rodler, F.F.: Wavelet based 3d compression with fast random access for very large volume data. In: Seventh Pacific Conference on Computer Graphics and Applications. pp. 108–117. IEEE (1999)
39. Sakai, R., Sasaki, D., Obayashi, S., Nakahashi, K.: Wavelet-based data compression for flow simulation on block-structured cartesian mesh. International Journal for Numerical Methods in Fluids **73**(5), 462–476 (2013)
40. Schendel, E.R., Jin, Y., Shah, N., Chen, J., Chang, C.S., Ku, S.H., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: ISOBAR preconditioner for effective and high-throughput lossless data compression. Proceedings - International Conference on Data Engineering pp. 138–149 (2012)
41. Tao, D., Di, S., Chen, Z., Capello, F.: Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. IEEE International Parallel and Distributed Processing Symposium (2017), (to appear)
42. Treib, M., Burger, K., Reichl, F., Meneveau, C., Szalay, A., Westermann, R.: Turbulence visualization at the terascale on desktop pcs. IEEE Transactions on Visualization and Computer Graphics **18**(12), 2169–2177 (2012)
43. Woodring, J., Mniszewski, S., Brislawn, C., DeMarle, D., Ahrens, J.: Revisiting wavelet compression for large-scale climate data using jpeg 2000 and ensuring data precision. In: IEEE Symposium on Large Data Analysis and Visualization (LDAV). pp. 31–38. IEEE (2011)
44. Yang, A., Mukka, H., Hesaaraki, F., Burtscher, M.: Mpc: A massively parallel compression algorithm for scientific data. In: 2015 IEEE International Conference on Cluster Computing. pp. 381–389 (Sept 2015)