# Foresight: Analysis That Matters for Data Reduction


1st Pascal Grosset
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
pascalgrosset@lanl.gov

2nd Christopher M. Biwer
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
cmbiwer@lanl.gov

3nd Jesus Pulido
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
pulido@lanl.gov

4th Arvind T. Mohan
*CCS-2: Computational Physics and Methods*
*Los Alamos National Laboratory*
Los Alamos, USA
arvindm@lanl.gov

5th Ayan Biswas
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
ayan@lanl.gov

6th John Patchett
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
patchett@lanl.gov

7th Terece L. Turton
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
tlturton@lanl.gov

8th David H. Rogers
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
dhr@lanl.gov

9th Daniel Livescu
*CCS-2: Computational Physics and Methods*
*Los Alamos National Laboratory*
Los Alamos, USA
livescu@lanl.gov

10th James Ahrens
*Data Science at Scale, CCS-7*
*Los Alamos National Laboratory*
Los Alamos, USA
ahrens@lanl.gov



***Abstract*—As the computation power of supercomputers increases, so does simulation size, which in turn produces orders-of-magnitude more data. Because generated data often exceed the simulation's disk quota, many simulations would stand to benefit from data-reduction techniques to reduce storage requirements. Such techniques include autoencoders, data compression algorithms, and sampling. Lossy compression techniques can significantly reduce data size, but such techniques come at the expense of losing information that could result in incorrect post hoc analysis results. To help scientists determine the best compression they can get while keeping their analyses accurate, we have developed Foresight, an analysis framework that enables users to evaluate how different data-reduction techniques will impact their analyses. We use particle data from a cosmology simulation, turbulence data from Direct Numerical Simulation, and asteroid impact data from xRage to demonstrate how Foresight can help scientists determine the best data-reduction technique for their simulations.**



***Index Terms*—Data compression, Performance evaluation, Multi-layer neural network**


## I. Introduction

The push toward exascale has given us extremely powerful supercomputers, with simulations consequently generating massive amounts of data at an unprecedented rate. However, I/O speeds and data storage capabilities have not increased at the same rate as processing speeds, and hence, High Performance Computing (HPC) centers struggle to store data generated from these extreme-scale simulations. One solution to address this "big data" issue is to look at data-reduction techniques. A data-reduction scheme that provides a 5X reduction in terms of data size would enable us to store 5X as much data. For a scientist, such a scheme also means saving data at every 5 timesteps instead of at every 25 timesteps, resulting in a higher fidelity capture of the simulation. Moreover, a fast data-reduction technique would allow simulations to spend less time doing I/O and more time performing actual computation.

Many different options now exist for data reduction, such as autoencoder, data compression algorithms, and sampling. (***Note:*** *we will use the term data compression algorithms in this paper to denote "traditional" data-reduction algorithms such as Lempel-Ziv (lz) [1] and JPEG [2].*) Ideally, we would not want to lose any data. Although lossless compression is an option for reducing data sizes, the best compression ratio from lossless data compression algorithms is usually less than two [3], [4]. Lossy data compression algorithms, sampling, and autoencoders are much more effective at reducing data sizes but such techniques do so at the expense of losing some of the information. For many scientists, performing a large run on a supercomputer is an opportunity that comes every couple of years, and the data generated from such a run will be analyzed over several months. If the data is lossily compressed to the point at which analysis is no longer accurate, it is a disaster for the scientist. Resolving this issue leads to three main questions: (i) how much precision/accuracy


SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

is needed to ensure that any post hoc analysis (analysis that is run by the scientist on data generated by the simulation) is still accurate? (ii) which data-reduction technique is best at preserving enough accuracy for post hoc analysis? (iii) which data-reduction technique will give the best compression ratio and throughput for the desired accuracy level?

Currently, there is no framework that enables scientists to evaluate systematically how data-reduction schemes will impact their post hoc analysis. Compression benchmarking tools, such as Z-checker [5] and Squash [6], are more focused on metrics of interest for the compression community, such as Mean Square Error (MSE) and Peak Signal-to-Noise Ratio (PSNR), rather than a scientist's concerns and interests. For example, a cosmologist would rather know whether the distribution of halos (clusters of particles) is the same after the data have been downsampled rather than the decompression speeds. Moreover, such benchmarks only exist for compressing algorithms. No framework exists that enables a scientist to decide whether he/she should use sampling over autoencoders.

The key contribution of this paper is Foresight, a framework that enables us to compare different data-reduction schemes and evaluate how they impact post hoc analysis of simulation data. Foresight's evaluation has three stages:

- **Data reduction**: the data is reduced using data compression algorithms or sampling or autoencoders and then are reconstructed. Foresight also comes with CBench, a C++ tool that enables scientific data compression algorithms to run at scale on simulation data.
- **Analysis**: runs the simulation's post hoc analysis on the reconstructed and original data for comparison. Foresight provides a simple API that enables scientists to describe how to run their post hoc analysis.
- **Visualization for comparison**: analysis results from the reconstructed and original data is visualized, and compression ratio and other metrics are provided such as throughput and PSNR (if scientific data compression algorithms are used). Again, an API is provided that enables the user to specify how to plot their results.

We demonstrate the usefulness of Foresight by running post hoc analysis using data from three different simulations: HACC [7], a Lagrangian cosmology code; CFDNS [8], an Eulerian turbulence code; and xRage [9], a multidimensional hydrodynamics code with continuous adaptive mesh refinement, as testbeds to evaluate and pick the best data reduction method. The ultimate power of Foresight is its ability to sweep the huge parameter space of data reduction methods, (both lossy and lossless data reduction techniques) and offer scientists all the information they need to make the best decision about what data reduction method is the most suited to their task. While Verification & Validation are commonplace in basic sciences such as physics, biology, and chemistry, it has lagged behind in the HPC community. In this paper, we are introducing an effort to create validation among data reduction techniques by establishing guidelines for designing data reduction analysis software and presenting a comparison between "conventional" data reduction techniques based on entropy and "newer" techniques such as autoencoders and sampling. As far as we know, Foresight is currently the only framework that allows comparison of data reduction techniques to be done for scientific analysis.

The paper is organized as follows: In section II, we present and discuss different data reduction techniques. In section III, we describe the architecture of Foresight. In section IV, we present use cases from HACC, CFDNS, and xRage, where there is a need for data reduction, and show how Foresight helped understand the impact of data reduction schemes. Finally, in section V, we summarize our results and point to future work that could be done in this area.

## II. Previous Work

### A. Data-Reduction Evaluation Frameworks

Evaluating the performance and impact of data-reduction techniques is as important as reducing data size; data that have been compressed too much are useless for analysis. Currently, only data compression algorithms offer benchmarks, which usually focus on compression metrics such as:

- Compression Ratio: $\frac{Original\ Size}{Compressed\ Size}$
- Compression Throughput: $\frac{Compressed\ Size}{Runtime\ Of\ Algorithm}$
- Absolute Error: $|Original\ Value - Compressed\ Value|$
- Relative Error: $\left|\frac{Original\ Value - Compressed\ Value}{Original\ Value}\right|$
- Mean Squared Error (MSE): $\frac{1}{n}\sum_{i=1}^{n}(Absolute\ Error)^2$ where $n$ is the total number of data values.
- Peak Signal to Noise Ratio (PSNR): $10\ log_{10}(\frac{R^2}{MSE})$ where $R$ is the maximum possible value of the variable.

Z-checker, by Tao *et al.* [5], is dedicated to evaluating the performance of compression algorithms. Z-checker evaluates a number of lossy data compression algorithms using measures such as PSNR and compression ratio, and it has a web interface to visualize these metrics. Squash [6] is an online tool that analyzes the performance of data compression algorithms, but it omits commonly used compression algorithms used for scientific data, such as SZ or ZFP. Libpressio [10] is a new compression tool that focuses on introspection of compression algorithms to facilitate their use. None of these frameworks enable the user to conduct any evaluation of the impact of lossy compression on domain-specific post hoc analysis. As mentioned before, these frameworks are usually developed by the compression community, and thus their evaluation does not seem geared toward the scientists who want to use them.

### B. Data-Reduction Evaluation

Scientists have already started to look beyond standard compression metrics when it comes to choosing the best data compression algorithm for their data. Baker *et al.* [11] investigated the impact of lossy compression, using the fpzip data compression algorithm, on climate data from the CESM (Community Earth System Model) Large Ensemble project, and concluded that the data loss is not significant enough to affect analysis, so long as the number of bits used for fpzip is carefully chosen. In their follow-up work [12], they

also performed a user study to validate how much data they can lose before the user notices any visual difference. Lu *et al.* [13] performed a somewhat more thorough analysis, where they compared ISABELA, SZ, and ZFP on scientific data, but the only evaluation they presented is visual blob detection. Hoang *et al.* [14] investigated the interplay of precision and resolution, and investigated how that can increase data quality while keeping the size of the data low. They also pointed out the need to adjust to application goals, as some applications might be more concerned with quality of the data, whereas other applications might need more data reduction, and are hence more willing to sacrifice data quality for such reduction. The Foresight framework enables users to systematically evaluate data reduction techniques for the different needs of an application. For example, visualization typically has lower accuracy requirements than a power spectrum based analysis. So, a user can specify different analyses to evaluate these tasks separately and ultimately pick the best data reduction method.

### C. Data-Reduction Methods

*1) Autoencoders:* Convolutional Neural Networks (CNNs) are well positioned to ingest high-dimensional datasets because they utilize *parameter-sharing* [15], where a filtering kernel is convolved across the domain to learn the correlation structure in the data. Early layers in the CNN are sensitive to short-range correlations, and later layers build on this to learn long-range correlation as well. Because the same parameters are re-used (shared) in convolutions across the spatial domain, CNNs need orders of magnitude fewer parameters than fully-connected NNs. This is paramount for 2D and 3D flow problems, where the amount of data is exponential in the spatial dimension of the problem, and a fully-connected approach thus requires exponentially more parameters. The reader is referred to Refs [16], [15], and [17] for details of CNNs.

*2) Data Compression Algorithms:* A portion of this study examines the application of lossy data compression algorithms, such as SZ [18], ZFP [19], fpzip [20], ISABELA [21], [22], and MGARD [23]. These algorithms have different compression strategies, but many of them can be driven by a similar series of input parameters: absolute error tolerance, relative error tolerance, or the amount of bit truncation. As a point of reference, we also compare to a lossless compressor, BloscLZ [24], that has proven to be effective in previous work [4].

Blosc [24] is a meta compressor (it can use a number of compression algorithms) optimized for binary data. Its default compressor is BloscLZ, which is a streamlined implementation of FastLZ [25], a variant of the LZ77 compression algorithm. Blosc is the defacto standard lossless compressor used in scientific applications.

fpzip is a predictive compressor that enables lossless or lossy reduction, specializing in 2D and 3D floating-point scalar fields. fpzip uses entropy coding of the residuals between the Lorenzo predictions [26] and original data points to reduce the dataset size. fpzip was designed as a lossless compressor but it includes a lossy extension.

ZFP is another predictive compressor for floating-point arrays that targets lossy compression to achieve very high throughput. Optionally using error-bounded support, the ZFP compressor has three modes: fixed rate (fixed number of bits), fixed accuracy (variable number of bits but fixed number of bit planes), and fixed precision (within specified absolute error tolerance).

ISABELA (In-situ Sort-And-B-spline Error-bounded Lossy Abatement) is a specialized lossy compression algorithm primarily designed for spatio-temporal scientific datasets. By first sorting the data point values, this method performs an approximation using a cubic B-spline fit for dataset points.

MGARD (Multi-grid Adaptive Reduction of Data) [23] is a recent, under-development method for multi-level lossy compression. Based on the theory of multigrids, MGARD employs a hierarchical scheme or an orthogonal decomposition method to produce multiple levels of partial decompression of the data within user-defined levels of tolerance.

SZ is an error-bounded data compression method that enables the user to set either an absolute or a relative error tolerance, as well as use the min or max of the two user-specified tolerances. SZ aims to predict data points using a curve-fitting model represented as a bit-array which is losslessly compressed. Any data points that cannot be predicted within the error bounds are stored at their original quality.

*3) Sampling Methods:* Sampling is a popular statistical method to select a subset from a larger collection. We will focus on generic sampling methods (unlike feature-based sampling methods) that do not rely on user intervention to reduce data sizes for Eulerian datasets.

Random sampling is a method where a random number, $\eta$ (where $0 \leq \eta \leq 1$), is generated at each cell of an Eulerian dataset. To select $n\%$ of the original data, all the locations where $\eta < \frac{n}{100}$ are selected. Although this method is space-filling and unbiased, it is not intended to prioritize the features of the data that are often useful for visualization and data reduction in the scientific community.

Regular sampling is a method where given $N$ data points, if $2\times$ reduction is required, every other data point is selected to produce the subsampled dataset. In addition to being space-filling and unbiased, regular sampling also preserves the topology of the data. However, like random sampling, it does not prioritize the features of the data, and often produces visualization artifacts caused by the regular nature of selection.

Histogram-driven sampling is a relatively new method, proposed by Biswas *et al.* [27], that uses the concept of entropy-maximization to generate samples with maximal entropy; more importance is given to data from the low probability regions (interesting features are usually rare in scientific datasets) and less importance to the high probability regions. Although this method does not retain the distributional properties in the resulting samples, it has proven to be more useful in capturing the salient data properties and in creating meaningful data summaries from large scale scientific datasets.

TABLE I: The table below shows the functional and non-functional requirements that drove the Foresight design.

| Functional Requirements | Non-Functional Requirements |
|---|---|
| F1 - Supports different data reduction techniques | NF1 - Provenance: Allows recreation of the pipeline for reproducible results |
| F2 - Ability to split data on multiple ranks | NF2 - API: Provides a simple API for specifying data reduction, analysis, and visualization |
| F3 - Run post hoc analysis on supercomputers | NF3 - Logging: Provides detailed logging of every step for debugging purposes |
| F4 - Support visualization and exchange of results | NF4 - Extensibility: Ability to add new features without changing the whole code base |
| | NF5 - Usability: Ease of building and running on supercomputer |
| | NF6 - Self-Verification: Have some degree of confidence that the framework results are correct |

### D. Visualization

Once we have compressed and analyzed data, we have two types of information for the user: (1) metrics such as compression ratio, MSE, compression, and decompression throughput; and (2) comparative analysis of output such as power spectra and halo distributions. Showing the compressors and the metrics is a multi-dimensional data analysis visualization problem. The two main tools for visualizing multi-dimensional data are scatter plot matrices [28] and parallel coordinates. Scatter plot matrices are basically tables of scatter plots, where in each scatter plot a metric is plotted against another. One of the issues of using such a representation for Foresight is that one of the assets of Foresight is its ability to sweep through, let's say 30 different compression parameters, and then enable the user to review the results for each. Using a scatter plot does not scale in this case. On the other hand, using parallel coordinates will enable us to see all the parameters at once, and it will also enable us to highlight, or only show, parameters that are relevant to the analysis. Moreover, we decided to not use plots such as radar or star plot, as their circular nature is not conducive for showing compression metrics—and such plots might even be misleading.

In addition to compression metrics, we also want to show the result of domain relevant post hoc analysis. Most simulation codes have tools to plot power spectrum ratios and related quantities. Because we want to show how data reduction affects the post hoc analysis, we should ideally be combining the compression metrics with the analysis results. Many generic tools exist to create dashboards for presenting data. Tableau [29], Plotly [30], and D3 [31] are widely used although, in many instances, a server of some kind is needed to host these visualizations. Eventually, we chose the Cinema Explorer tool [32]. Cinema Explorer is built on D3 and can easily render the contents of a comma-separated value (CSV) file in an interactive web-based viewer. Moreover, if the CSV file contain images (e.g., power spectrum ratio plots), these can be displayed in an image spread linked to the parallel coordinates view. Finally, Cinema Explorer does not rely on any server since it is built using only client-side operations. Consequently, this facilitates the sharing of the results with other scientists; Cinema Explorer databases can even be embedded in GitHub pages, where they can be browsed by anyone without installing any software.

## III. Architecture

To come up with a useful framework to evaluating data reductions, we must first determine what are the needs of the application. In this case, we should be able to run different data reduction techniques on large datasets at scale on HPCs. Then, we should be able to run post hoc analysis on the results of the previous stage and finally compute and visualize the differences between analyses run on reconstructed data vs. original data. These needs help establish the functional requirements of our framework. Most tools developed up to now can only run compression algorithms; i.e., F1 in Table I.

When creating such frameworks, non-functional requirements may be as important as the functional requirements. Although the ease of building software on supercomputers is improving with tools such as Spack [33] and CMake, keeping the dependencies to a minimum is always a good idea; odd dependencies, such as relying on LaTeX for generating reports (as is the case with Z-Checker), should be avoided. Other prominent features, as shown in Table I, include provenance, logging, usability, extensibility, and an easy-to-use API. In the next section, we present the design of Foresight, and show how we address the functional and non-functional requirements.

Foresight's evaluation process has the following stages:

1) Data Reduction: where data compression algorithms or sampling or autoencoder are used to reduce the amount of stored data.
2) Analysis: where post hoc simulation analysis is run on the reduced data.
3) Comparison and Visualization: where the results of the simulation analysis are plotted and Cinema databases are generated (as needed).

As shown in Fig. 1, at the heart of Foresight—and the main driver of these three stages—is the Data Reduction and Analysis Workflow (DRAW) tool. DRAW coordinates all the activities in Foresight and is responsible for launching and managing jobs. The two other tools provided by Foresight are (1) CBench, a C++ tool to run compression at scale on scientific data, and (2) Cinema, a Python tool for creating a Cinema Explorer database. Input to Foresight is through JSON files with the following sections (Listing 1):

Listing 1: Structure of the JSON input

```
{
  "input" : {},
  "data-reduction" : {},
```

```
  "analysis" : {},
  "visualization" : {
    "plots" : {},
    "cinema" : {}
  }
}
```

where:

- input: specifies the location of the input file and the variables to be compressed
- data-reduction: specifies the data reduction methods to be used; it will, for example, contain the parameters to CBench if the latter is being used
- analysis: specifies the analysis to be run; it usually points to the post hoc analysis code to be run
- visualization: specifies the type of plotting to be done
  - plots: points to the plotting routines used by the simulation
  - cinema: specifies the parameters to create a Cinema database for Foresight

The JSON file format has been chosen to specify the input parameters because it is easy to understand and can be easily edited remotely using screen-oriented editors such as vim. JSON files facilitate usability of Foresight. Once the whole pipeline of Foresight is complete, a new JSON is created that shows all the modifications that the pipeline made to the original JSON file (if any) and adds a git hash to the file. Using that git hash, a user can check out that specific tag from the git repository and run the exact same pipeline (same code, same number of MPI ranks, etc.) to replicate the results. These features allow us to satisfy NF1 and NF5 mentioned in Table I, which deal with the reproducibility and criterion. We will now look at the different components of Foresight.

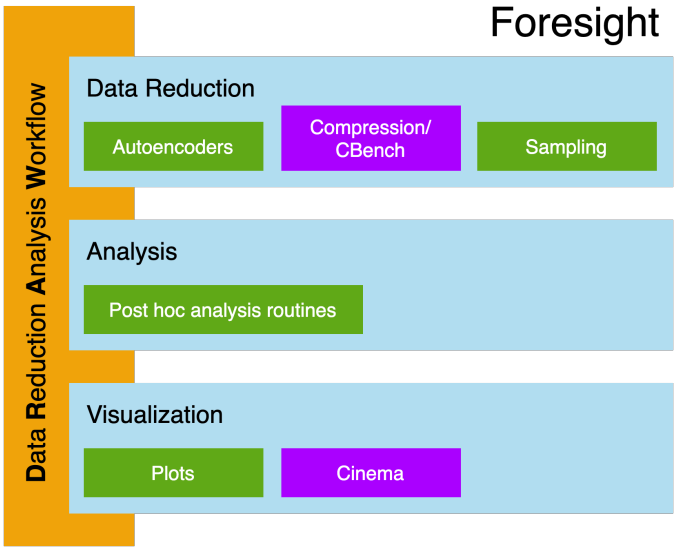



Fig. 1: Architecture of Foresight framework, with CBench and DRAW modules provided shown in purple.

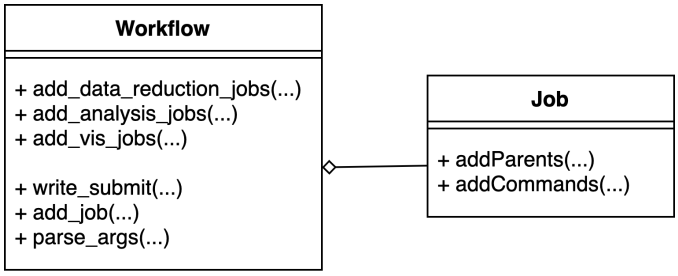



Fig. 2: UML diagram of the DRAW workflow. DRAW has two classes: Workflow and Job.

### A. *DRAW*

DRAW is a Python workflow tool providing simple, yet powerful workflow features such as creating SLURM jobs and setting up a dependency graph for data reduction jobs. As shown in Fig. 2, DRAW has two classes: Workflow and Job. The Job class stores all the parameters needed to run a job. Listing 2 shows the input parameters used to create a job.

Listing 2: Structure of the JSON job input

```
{
  "name": "job_name",
  "path": "run_command path_to_job",
  "params": ["argument_1",
             "argument_2", ...],
  "evn_path": "scripts/bash.darwin",
  "pre-run-command": ["...", "...", ...],
  "post-run-command": ["...", "...", ...],
  "configuration":
  {
    "partition": "general",
    "nodes": 8,
    "ntasks-per-node": 12
  }
}
```

In Listing 2, *path* points to the job to be run (as well as the command to run it); *evn_path* points to the script that loads the environment parameters for the job; *configuration* describes the resources to allocate to that job; and *pre-run-command* and *post-run-command* specify any commands that must be run before and after the SLURM job.

As shown in Fig. 2, the Workflow class parses the input JSON file and creates SLURM jobs with proper dependencies. When using Foresight, add_data_reduction_jobs, add_analysis_jobs, and add_vis_jobs must be implemented. These specify the reduction technique, post hoc analysis, and visualization jobs to be used respectively. Moreover, while implementing those, the SLURM job dependency is also specified. Job dependencies can be specified as depending on all previous jobs, one specific job, or a category of jobs. For example, the visualization jobs must wait for all analysis jobs to be finished, whereas the analysis jobs must wait on a single data reduction job to finish. A typical dependency is shown in Fig. 3. The workflow input structure shown in Listing 2 is used for all jobs (CBench runs, analysis runs, and visualization runs). This is part of the effort to provide a

consistent yet simple API to facilitate job submission, thereby satisfying NF2.

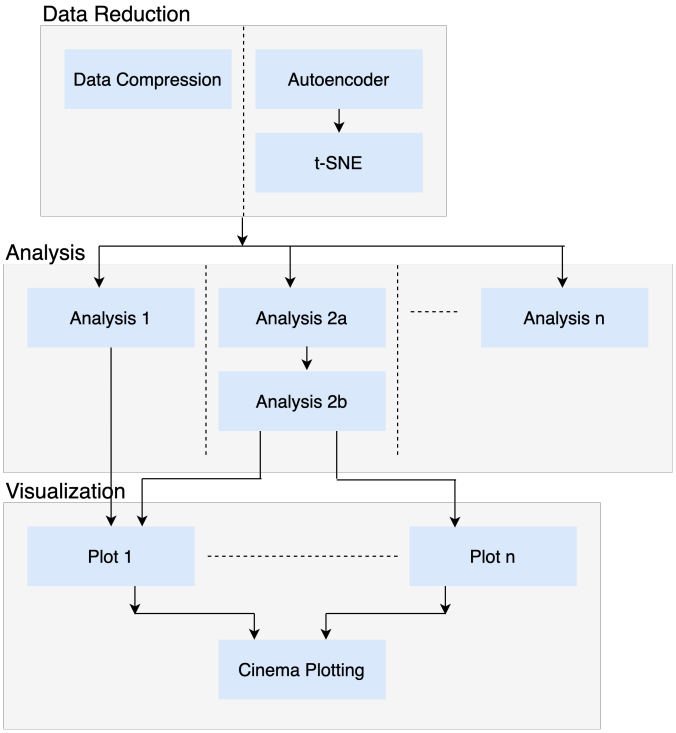



Fig. 3: Dependency graph of a standard Foresight workflow with data reduction, analysis, and visualization components.

Foresight has been designed to be modular and extensible, the reason being that very often data reduction, analysis, and visualization often need several stages. For example, let's consider the two-stage data reduction in Fig. 3—an autoencoder followed by t-SNE [34], which is often used for compression [35]. Doing that in Foresight would involve creating a new Job instance for the autoencoder, and then another Job for the t-SNE calculation with a dependency on the Autoencoder job. The analysis would then depend on the Data Reduction job group. This is fully supported by Foresight's DRAW workflow.

### *B. Compression Bench (CBench)*

CBench is the compression component of Foresight. Running data compression algorithms is probably the most commonly used data reduction technique, and so we have added it to Foresight to facilitate the running of scientific compression algorithms, as well as to allow newer reduction techniques, such as autoencoders and sampling, to be compared against state-of-the-art data reduction techniques.

Written in C++ and MPI, CBench is designed to be easy-to-use, portable, and flexible such that it scales across many platforms and architectures ranging from local desktops to supercomputers. CBench uses the Abstract Factory design pattern, a creational design pattern that allows the easy addition of new data compression algorithms, compression metrics, and data loaders with minimal changes to the CBench main code.

Although CBench is an integral component of the Foresight framework for distributed-computing workflows, it can also be run on its own to produce metrics such as compression ratio, compression throughput, and relative error. As shown in Fig. 4, CBench has three main components:

- The **Data Loader** module enables the addition of new data readers. It provides an interface that requires the implementation of *loadData()*, *saveCompressedData()*, and write data *writeData()* methods. Because CBench is designed to run in parallel, to mimic how a simulation will use a compressor, a data partition module is provided. This module will equally split blocks of data among multiple MPI ranks for distributed loading. The output of the data loader module is a pointer to the data loaded. Currently, CBench supports loaders for raw binary data, GenericIO for the HACC simulation, HDF5 for the Nyx simulation [36], and the VTI file format from VTK [37].
- The **Compressor** module supports the addition of scientific compressors through a simple interface that requires the implementation of a *compress()* and *decompress()* function. This interface makes the addition of compressors quite trivial since most data compression algorithms have a compress and decompress function where the parameters are: the data to be compressed/decompressed, compression parameters, and size of the dataset. All a user needs to do is get the pointer from the data loader module, and the input parameters from the JSON file. The compressor module will in turn have a pointer to the decompressed dataset. We currently support most existing data compression algorithms, namely the following: BLOSC, fpzip, ISABELA, MGARD, SZ, and ZFP. These compressors can be linked by pointing CBench to the static (*.a*) or dynamic library (*.so*). If a user chooses to build a compressor from source, the CMake interface can be used to point to the libraries.
- The **Metrics** module supports the addition of custom-made metrics through the *execute()* function. It receives a pointer to the data that have been compressed and an MPI communicator, enabling the computation of both local and global metrics with cross-rank communication. Currently implemented metrics are absolute and relative error, MSE, PSNR, compression ratio, and compression and decompression throughput.

Input to CBench is through a JSON file that specifies the file to compress, scalars to compress, data compression algorithms, parameters to use, and where to output the compression metrics. The two main outputs from CBench are a CSV file that contains compression metrics and the lossy compressed data for each compressor that has been used. Some metrics, such as compression ratio, compression throughput, and decompression throughput, are always generated. The user can specify other metrics to output, such as absolute error and MSE, PSNR, in addition to easily implementing their own. The CSV file format was chosen because most data analytics packages can inherently read CSV files.

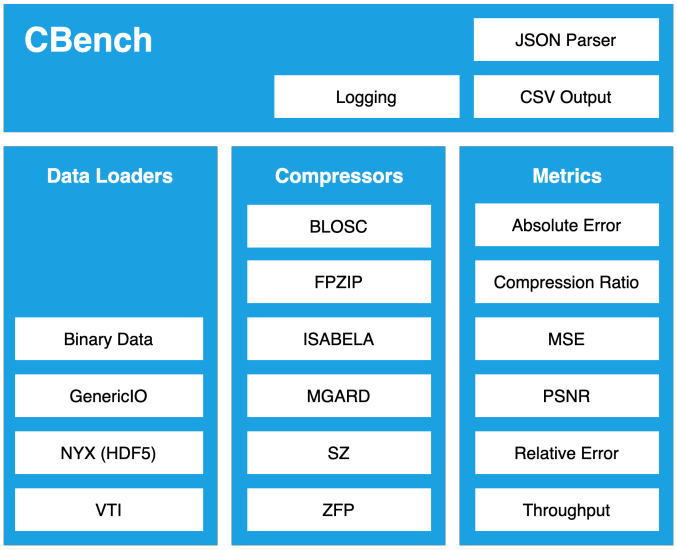



Fig. 4: CBench is composed of three main components: the Data Loader (I/O), Compressors (data compression algorithms), and Metrics (statistics). The abstract nature and simplicity of CBench enables the addition of virtually any data-type, compressor, or metric.

The flow of control through CBench is shown in Algorithm 1. For each timestep, and for each data field to be compressed, the user specifies which compression parameters to use and the name of the resulting output file. Then, each scalar specified in the input is compressed using the specified compressor settings. Finally, the reconstructed data is outputted to disk. Throughout the execution of CBench, logs are being created (satisfying NF3) for each MPI rank (this can be turned off by going into release mode); metrics and lossily reconstructed data are being saved to disk. All these features enable CBench to be easily used by developers; for example, Jin *et al.* [38] extended CBench by adding the GPU compressors from SZ and ZFP for cosmology data. **Note:**

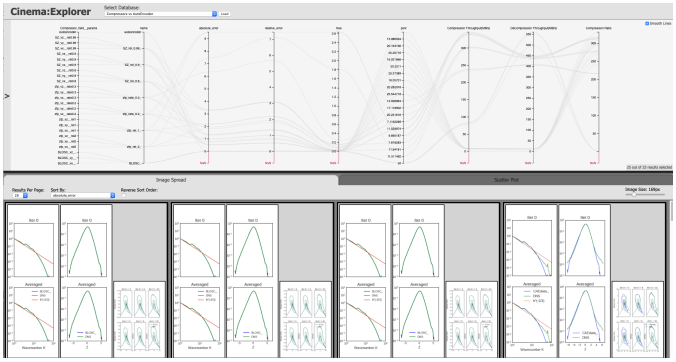



Fig. 5: Cinema Explorer with Parallel Coordinates view of a CFDNS dataset compressed using SZ and ZFP with different input parameters. The top view shows compression metrics and the bottom view shows the result of analysis.

**Algorithm 1:** CBench algorithm.

```
read input JSON file;
initialize reader;
for each timestep do
    for each compressor do
        for each data field to be processed do
            load compressor parameter;
            compress data;
            for each compressor metric to be measured
              do
                compute metric;
            output metrics to disk;
        Save decompressed data to disk;
```

As shown in Fig. 4, BLOSC, a lossless compressor, is also included in CBench. BLOSC has been chosen as it is often the defacto standard for lossless compression in simulations such as HACC. While no evaluation is needed when BLOSC is used as a compressor since no data is lost, including BLOSC serves two purposes: firstly, to allow scientists to see how much more compression a lossy compressor will give over a lossless compressor, and secondly, to validate that there are no bugs in the framework. BLOSC follows the exact path that a lossy data reduction method will, but since no data is lost, in HACC all halos should be preserved and in the CFDNS dataset, all the small, medium, and large scale features should be preserved. If this happens not to be the case, this is an indication of a bug in our framework. This satisfies NF6 and helps the user build confidence that they can trust the results from the framework. The use of BLOSC for the CFDNS dataset is shown in Fig. 5.

### C. Cinema Visualization

Fig. 5 shows an example of the output for Foresight. The top window will contain all the different data-reduction metrics gathered for the dataset, whereas the bottom window will contain the results of the post hoc analysis. The metrics are generally gathered from the data-reduction portion of the pipeline and will usually have at least three dimensions: compression ratio, compression throughput, and the input compressor settings. To generate a Cinema database, the user can choose to implement the Cinema class of DRAW that will provide an API and some utilities to facilitate the creation of Cinema Explorer databases. However, given that a cinema database is really just a CSV file within which each row has the information for one scalar and compression settings, the user could perform this process manually too. Another feature of the Cinema Explorer tool is that it provides a scatter plot window, where the user can decide to visualize one metric against another; e.g., the user may want to see which compressor has the best compression ratio and lowest MSE for each variable-compression parameter combinations.

However, the notable benefit of the Cinema Explorer tool is its ability to filter out results based on user selection. For

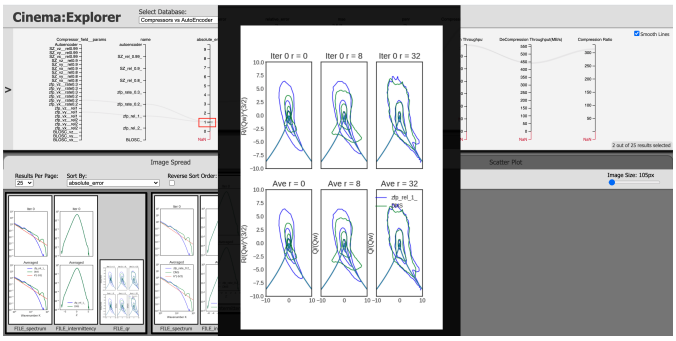



Fig. 6: Using the Cinema Explorer to examine the QR-Plot.

example, let's say that the user wants to find the characteristics of the QR-plot at an absolute error of 1. This is done by selecting 1 in the parallel coordinates window for the absolute error bar, which ultimately narrows the plots in the image spread. Clicking the QR-plot, as shown in Fig 6, allows us to zoom in on the plot for further examination.

## IV. Analysis and Evaluation

The main supercomputer used for testing is Darwin, a heterogeneous supercomputer at the Los Alamos National Laboratory, with about 400 nodes. For large datasets, we used the Haswell partition, that has 2,388 Intel E5-2698 v3 nodes, of the Cori, a Cray XC40, supercomputer at NERSC. Foresight is an Open Source software available at https://github.com/lanl/VizAly-Foresight.

### A. Simulation Use Cases

*1)* ***CFDNS****:* The dataset consists of a 3D Direct Numerical Simulation (DNS) of homogeneous, isotropic turbulence, in a box of size $128^3$. We denote this dataset as *HIT* for the remainder of this work. We provide a brief overview of the simulation and its physics in this section, and a detailed discussion can be found in Daniel *et. al* [39]. The HIT dataset is obtained using the incompressible version of the CFDNS code ([39] and references therein), which uses a classical pseudo-spectral algorithm. We solve the incompressible Navier-Stokes equations:

$$\partial_{x_i} v_i = 0, \qquad \partial_t v_i + v_j \partial_{x_j} v_i = -\frac{1}{\rho}\partial_{x_i} p + \nu \Delta v_i + f_i^v,$$

where $f^v$ is a low band forcing, restricted to small wavenumbers $k < 1.5$. The $128^3$ pseudo-spectral simulations are dealiased using a combination of phase-shifting and truncation to achieve a maximum resolved wavenumber of $k_{max} = \sqrt{2}/3 \times 128 \sim 60$. Based on the sampling rate, each eddy turnover time $\tau$ consists of 33 snapshots. The training dataset uses 22 snapshots in the time range $\approx 0 - 0.75\tau$. The test dataset also consists of 22 snapshots but in the time range $\approx 4 - 4.75\tau$.

Data Reduction Requirement: Turbulence datasets are extremely large, which causes inefficient archival storage, and in many cases, inefficient transmission of data [40], [41]. Furthermore, turbulence is highly multiscale in nature, and loss of information at various scales caused by compression algorithms must be effectively regulated and quantified, since it might lead to deviations from physical laws that were obeyed in the raw dataset. Another major application of compression is to construct reduced order/surrogate models, which need to be computationally efficient and yet resolve the scales required for the specific application. In most engineering applications, compression and modeling of turbulence data require that large and inertial scale features be accurately retained, although there is considerable leeway in discarding small scale information. Some of the key aspects we aim to evaluate are as follows:

- ***CFDNS-A:*** Which data compression algorithm will give the best throughput, PSNR, and MSE for a very large compression ratio (e.g., 300X compression), typical of autoencoders?
- ***CFDNS-B:*** How do data compression algorithms compare with autoencoders for capturing not just the large scale portion of the energy spectrum, but also the 3D morphology of the flow?

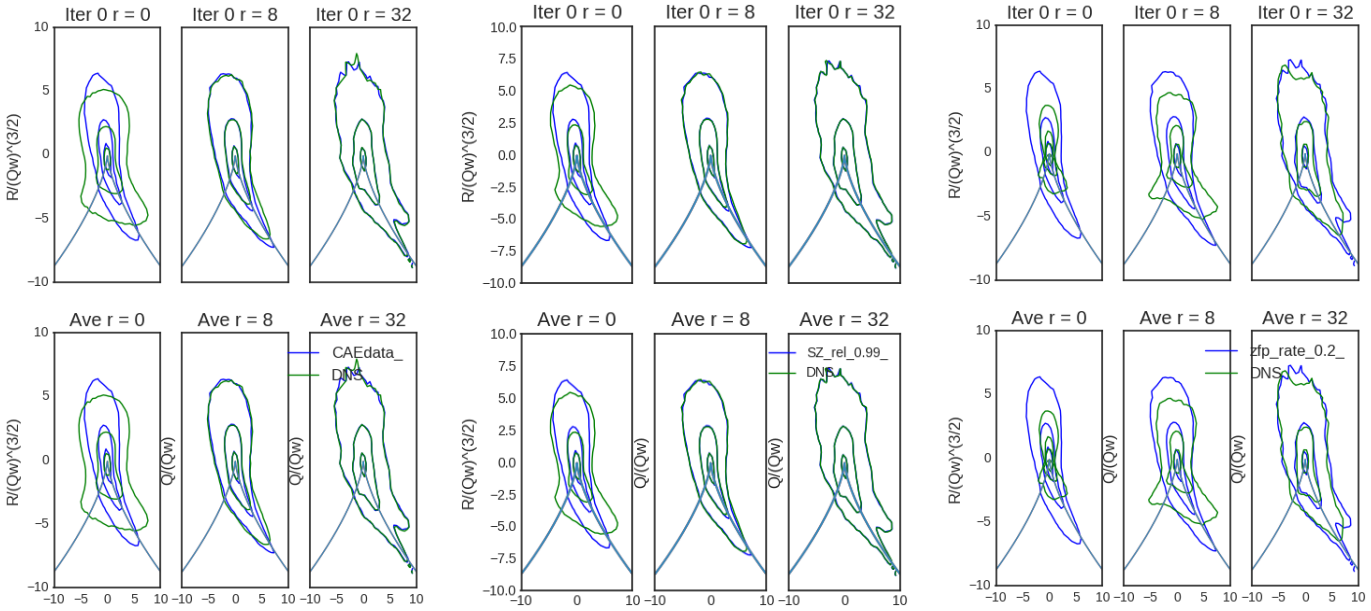
*2)* ***HACC****:* HACC [7] is an extreme-scale cosmological simulation code used to simulate the evolution of the universe. It uses an N-Body Lagrangian framework for modeling the action of gravity on dark matter and a Conservative Reproducing Kernel Smoothed Particle Hydrodynamics (CRKSPH) [42] scheme for gas dynamics. The raw output of HACC is a massive quantity of dark matter particles, each having an *index*, position ($x$, $y$, $z$), and velocity ($v_x$,$v_y$,$v_z$). It generates TB (PB for large runs) of data, and often runs out of disk space on HPC systems. The analysis suite of HACC, called CosmoTools, is used to compute the power spectrum and dark matter halos (clusters of dark matter particles). The power spectrum is useful for characterizing the clustering properties of N-body simulations [43]. Dark halos are of interest because, according to the leading theories, they play a crucial role in galaxy formation and dark matter is said to compose 80% to 90% of the universe. For halos, the values of interest are mass, number of particles, and radius.

Data Reduction Requirement: Cosmology is an observational science. So, it is important that the theoretical simulation error does not exceed the observational error used in current and future observational campaigns. As long as the data loss is within 3 kpc (kiloparsec) in position, which in HACC translates to absolute error of 0.003, the data will be within the bounds of observational error and post hoc analysis will still be able to produce meaningful results. HACC already uses BLOSC for lossless compression and the developers would like to add lossy compression to the simulation. Before picking a lossy compressor, they would like to know the following:

- ***HACC-A:*** Which compressor will give the best compression ratio and throughput for a maximum absolute error of 0.003?
- ***HACC-B:*** What is the impact of an absolute error of 3 kpc on the power spectrum and halos?

TABLE II: Data Compressing algorithm metrics for the HIT CFDNS dataset, answering ***CFDNS-A*** with a compression ratio of about 300X. SZ was run in relative error mode, whereas ZFP was run in both relative error and fixed rate modes.

| Data compression Algorithm | Parameters | Variable | Absolute Error | Relative Error | PSNR | MSE | Throughput (MB/s) | Compression Ratio |
|---|---|---|---|---|---|---|---|---|
| SZ | relative error 0.99 | vx | 3.29 | 3.09 | 0.08 | 25.21 | 7.44 | 330.71 |
| | | vy | 0.52 | 0.52 | 0 | 20.16 | 7.53 | 307.4 |
| | | vz | 1.35 | 1.35 | 0.02 | 15.99 | 7.53 | 269.19 |
| ZFP | relative error 1.0 | vx | 9.53 | 7.45 | 2.63 | 9.9 | 346.11 | 310.9 |
| | | vy | 0.99 | 0.99 | 0.01 | 11.53 | 345.57 | 312.11 |
| | | vz | 2.17 | 2.17 | 0.15 | 7.12 | 340.5 | 311.8 |
| ZFP | fixed rate 0.2 | vx | 9.53 | 7.45 | 2.63 | 9.9 | 338.29 | 315.08 |
| | | vy | 0.99 | 0.99 | 0.01 | 11.53 | 342.25 | 315.08 |
| | | vz | 2.17 | 2.17 | 0.15 | 7.12 | 338.73 | 315.08 |




(a) Using Autoencoder (compression ratio: ≈ 300X)

(b) Using SZ with 0.99 relative error (compression ratio: 269X - 330X)

(c) Using ZFP with compression rate of 0.2 (compression ratio: ≈ 315X)

Fig. 7: QR-plots of the HIT CFDNS dataset after lossy reconstruction with an autoencoder and the data compression algorithms. The top row shows typical instantaneous plots (i.e., comparisons for one snapshot) and the bottom row shows averages over the testing range.

- ***HACC-C:*** How much can we increase the compression ratio before the data become unusable?

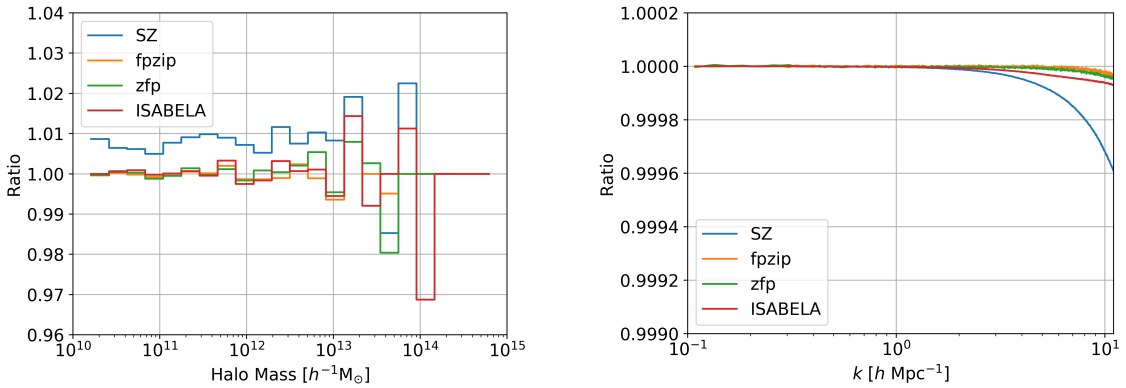
*3)* ***xRage:*** xRage [44] is a parallel multi-physics Eulerian hydrodynamics code developed at the Los Alamos National Laboratory. It uses an Adaptive Mesh Refinement technique that allows higher and lower resolution areas of the simulation grid (where appropriate) for computation. Through ParaView Catalyst [45], it outputs VTK unstructured grid data, which is resampled to a regular grid of $500^3$ before being written to disk. xRage was used to produce the Deep Water Impact Ensemble dataset [46], which was created to study the propensity of asteroids impacting deep ocean water to create tsunamis. The crater formed during the impact is of great interest to scientists because tsunami waves originate from the collapsing crater, and the radius of the impact crater is directly proportional to the kinetic energy of the impact [47].

Data Reduction Requirement: The deep water impact ensemble dataset is a massive dataset that very often needs to be downsampled for sharing. However, while downsampling the dataset, important features such as the crater and spray of water ejected must be preserved so that any post hoc analysis run on the dataset is still viable. Given that different analyses require different accuracy levels, it is difficult to set a hard limit on any quality metric, but knowing which compression method (sampling or data compression algorithm) will give the best compression quality (the meaning of quality will differ for different analysis) at a certain compression level is important. So for this dataset, the questions to be answered are as follows:

- ***xRage-A:*** Which data reduction scheme will give the best compression quality when targeting 50X compression?
- ***xRage-B:*** How does sampling compare to data compression algorithms at the asteroid impact crater?

TABLE III: Compression of the HACC-256 particle dataset answering ***HACC-A***, showing the best set of parameters to reach a maximum absolute error target of 0.003 for each position scalar fields from a run on Cori at NERSC.

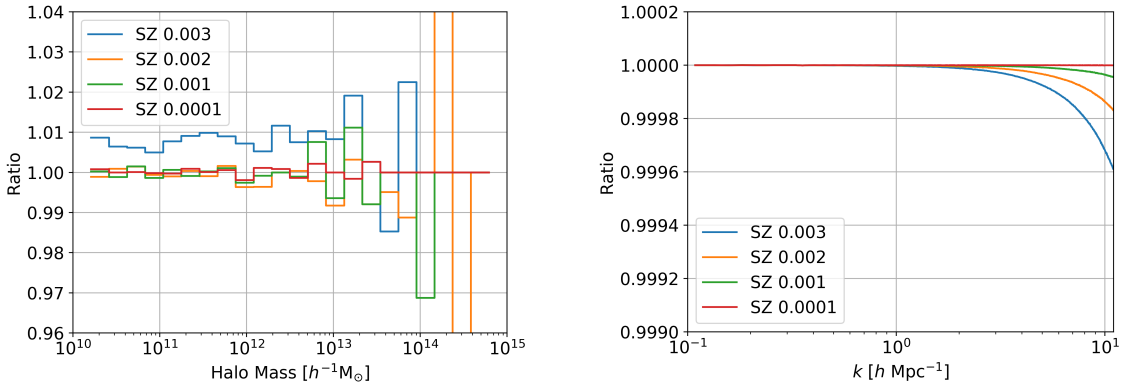
| Data compression Algorithm | Parameters | Variable | Absolute Error | PSNR | MSE | Throughput (MB/s) | Compression Ratio |
|---|---|---|---|---|---|---|---|
| fpzip | Truncate: 26 bits | x | 0.000961 | 115.72 | 1.71e-07 | 14.48 | 3.18 |
| | | y | 0.000961 | 115.52 | 1.79e-07 | 14.06 | 3.12 |
| | | z | 0.000961 | 115.52 | 1.78e-07 | 13.95 | 3.04 |
| ISABELA | Window Size: 2048<br>Tolerance: 0.001<br>Coefficients: 50 | x | 0.00256 | 112.85 | 3.40e-07 | 1.24 | 2.21 |
| | | y | 0.00256 | 111.96 | 4.17e-07 | 1.03 | 2.09 |
| | | z | 0.00256 | 110.69 | 5.60e-07 | 0.85 | 1.49 |
| SZ | Absolute Error Bound: 0.003 | x | 0.00299 | 103.42 | 2.99e-06 | 77.52 | **4.93** |
| | | y | 0.00299 | 103.40 | 2.99e-06 | 76.89 | **4.78** |
| | | z | 0.00299 | 103.41 | 2.99e-06 | 73.95 | **4.60** |
| ZFP | Absolute Error Bound: 0.007 | x | 0.00291 | 111.76 | 4.37e-07 | 46.41 | 2.07 |
| | | y | 0.00243 | 111.761 | 4.37e-07 | 47.55 | 2.06 |
| | | z | 0.00242 | 111.76 | 4.37e-07 | 46.63 | 2.04 |




(a) Halo mass distribution

(b) Power spectrum ratio

Fig. 8: Halo analysis meeting the requirements for ***HACC-B***. Ratio of mass distributions (a) and the power spectra ratio (b) verify that both derivative quantities are well preserved.




(a) Halo mass distribution

(b) Power spectrum ratio evolution

Fig. 9: Compression parameter scaling addresses ***HACC-C*** using derivative halo mass distribution (a) and power spectra ratio (b).

### *B. Evaluation Results*

*1)* ***CFDNS****:* The HIT sample dataset has 50 timesteps with a total size of $\approx$ 2.5 GB. For this application, autoencoders were being used to compress the data, with the compression ratio set to 300X because that struck the right balance of data size and accuracy during post hoc analysis [48].

- ***CFDNS-A***: Compared to autoencoders, data compression algorithms are not usually set up to target a specific compression ratio. So, using Foresight, a parameter sweep

was performed to determine which compressor would have minimal signal loss at a 300X compression ratio. Moreover, because of the presence of large rare deviations from the mean caused by small scale intermittency, relative error was used instead of absolute error to drive the data compression algorithms. After running these tests, it was found that SZ and ZFP were the best compressors to use. Table II shows the results from the best data compression algorithms. Although ZFP has a very high throughput, SZ has better MSE and PSNR, which probably means that it has fewer errors. There is no clear answer to ***CFDNS-A***, as which compressor to use depends on whether throughput or PSNR and MSE is more important.

- ***CFDNS-B***: The joint probability density function of Q and R, the second and third invariants of the velocity gradient tensor (known as QR-plot), can be used to describe the characteristic structure of turbulence at different scales. This is accomplished by coarse-graining the 3D velocity field using a Lagrangian tetrad representation, before computing the QR-plot ([49] and references therein). Here, $r$ is related to the volume of the tetrahedron and defines the scale of the observation. Comparing the QR-plot is a stringent test of 3D turbulence topology at different scales $r$, with the characteristic tear drop shape at intermediate scales being one of the hallmarks of 3D turbulence. Fig. 7 shows the QR diagrams for the autoencoder compared to the SZ and ZFP results. While it expected that small scales are not well represented compared to the DNS results due to the lossy compression, as it can be seen, ZFP struggles to preserve large scale features ($r$=32) and medium scale features ($r$=8) as well as SZ and the autoencoder; the results from SZ and autoencoders are virtually indistinguishable. To answer ***CFDNS-B***, SZ is on par with the autoencoder, but ZFP at 300X compression lags behind.

TABLE IV: Timings for analysis in Table III.

| | **fpzip** | **ISABELA** | **ZFP** | **SZ** |
|---|---|---|---|---|
| Compression Time (s) | 1.61 | 26.85 | 0.22 | 0.28 |
| Decompression Time (s) | 1.40 | 1.77 | 0.24 | 0.33 |

These results show the importance of validating compression results with analysis. ZFP has a very high throughput and could be attractive for in situ compression but this comes at the risk of losing a lot of information about the simulation since ZFP fails to preserve large scale features for analysis.

*2) **HACC**:* The HACC dataset used for testing is a 45 GB file distributed in 256 partitions.

- ***HACC-A***: To know which compressor is the best for a maximum absolute error of 0.003, we ran a parameter sweep using fpzip, ISABELA, SZ, and ZFP to determine which of these compressors would give the best compression ratio for an absolute error of 3 kpc, and show the results in Table III. Only SZ and ZFP allow us to specify absolute error as an input parameter. Furthermore, of these two algorithms, SZ's error distribution is able to achieve the closest to this target absolute error bound; for a survey on error distributions from lossy compressors see Lindstrom [50].
- ***HACC-B***: Fig. 8(a) shows the halo mass distribution for each of the compressors in Table III, and Fig. 8(b) shows the power spectrum ratios for them. As we can see from Fig. 8(b), all the compressors from Table III are well within the 1% limit, which makes them all valid for post hoc analysis. The halos mass distribution from Fig. 8(a) is also within the acceptable deviation range for analysis, further validating that an absolute error of 0.003 would not damage any post hoc analysis.
- ***HACC-C***: For this analysis, we focused on SZ because it has the best compression ratio for HACC. Fig. 9 shows how the halo mass distribution and power spectrum evolve with various compression parameters set for SZ. SZ was near the 1% limit with an absolute error tolerance of 0.003, placing an upper limit on the absolute error tolerance for ***HACC-C***. If the absolute error tolerance is lowered to 0.002, 0.001, or 0.0001 we note that the error decreases to well within the error budget; however, the compression ratios also reduced to 4.55, 3.98, and 2.78, respectively, for the $x$-position.

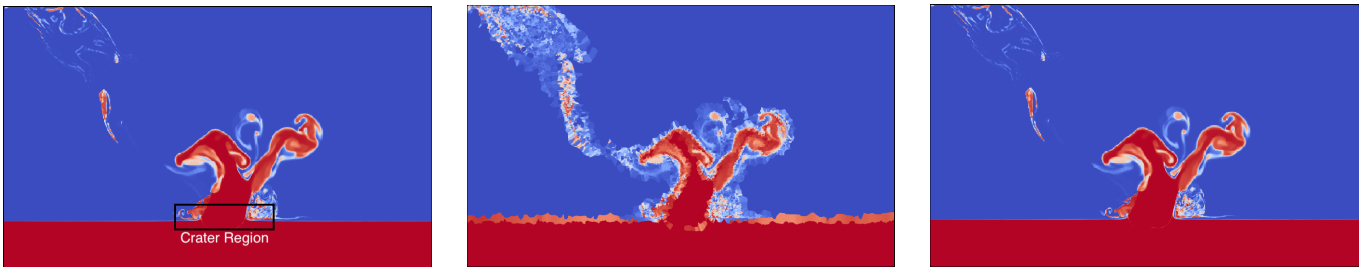
For the HACC compression experiments, a maximum absolute error of 0.003 is seen fit for each of the position values. While SZ and ZFP (ZFP does not strictly abide by the absolute error specified) allow us to do that, with fpzip and ISABELA, we cannot directly request such a bound. The ability to clearly specify the limits of tolerance is an important factor for scientific applications since it gives the scientist some peace of mind that the data will never be affected by more than what is specified. This evaluation was run on the Cori supercomputer at NERSC using 128 MPI ranks; 32 nodes and 4 cores per node, and took on average 190.6s. Load time for each scalar was 2.91s and metrics computation took an average of 4.81s. Compression and decompression time varied per compressor and their timings are shown in Table IV. All the timings quoted are from an average of 5 runs.

*3) **xRage**:* The asteroid impact dataset used from the xRage simulation has a total of 475 timesteps, stored in a compressed VTK format that has already been sampled from its original unstructured format onto a regular grid. For this application, features of interest (craters) appear after the asteroid hits the ocean, which occurs at timestep 99 in this dataset.

- ***xRage-A***: A sweep of different compression parameters was done to determine data reduction methods that came close to giving 50X compression. From Table V, we can see that data compression algorithms have a higher (better) PSNR and throughput and lower (better) MSE and absolute error. Looking at Fig. 10, where we only showed the best of the sampling and data compression algorithm, we can see SZ preserves the details much better than sampling. In response to ***xRage-A***, compression should be used instead of sampling.

TABLE V: Data reduction of the xRage asteroid impact dataset answering ***xRage-A***. Both sampling and compression methods target a compression ratio of 50X, on the water volume fraction (scalar field v02 in the simulation).

| Reduction Method | Parameters | Absolute Error | PSNR | MSE | Throughput (MB/s) | Compression Ratio |
|---|---|---|---|---|---|---|
| Regular Sampling | Rate: 0.015 | 1.0 | 23.796 | 0.00417 | 73.5 | 48.23 |
| Random Sampling | Rate: 0.015 | 1.0 | 23.158 | 0.00483 | 72.8 | 49.94 |
| Histogram Sampling | Rate: 0.010 | 1.0 | 19.0459 | 0.01245 | 7.93 | 49.52 |
| SZ | abs 0.0003 | 0.000300 | 107.202 | 1.904e-11 | 151.576 | 52.15 |
| ZFP | abs 0.01 | 0.002262 | 81.49 | 7.096e-09 | 250.806 | 46.73 |




(a) Original (b) Histogram Sampling (c) SZ Compression

Fig. 10: Comparison of sampling against SZ lossy compression. The crater region for which the SNR is being calculated is highlighted in (a).

- ***xRage-B***: When analyzing the structure of the asteroid impact crater, the primary metric for evaluation is the Signal-to-Noise Ratio (SNR) of the region. Generally, a higher SNR denotes a better preservation of crater features. When analyzing the asteroid crater region, shown inside a rectangle in Fig. 10(a), we found that the different data reduction methods produced the following SNR results: Regular sampling: 13.359; Random sampling: 12.570; Histogram sampling: 14.400; SZ: 19.2790; and ZFP: 19.2787. In response to ***xRage-A***, we can see that data compression algorithms perform better than sampling both visually and in terms of SNR.

The results show that despite recent developments, sampling techniques still lag behind data compression algorithms with low data throughput and poor reconstruction quality.

## V. Conclusion and Future Work

In this paper, we introduce Foresight, a framework that enables users to compare different data-reduction methods and evaluate their impact on post hoc analysis of simulation data. To showcase the capabilities of Foresight, we showed how it helped explore data-reduction results from three different simulation codes (HACC, CFDNS, and xRage) applied to three different domains (cosmology, turbulence, and hydrodynamics). The comparison was performed using an autoencoder, data compression algorithms, and sampling methods. Foresight's ability to seamlessly run on HPC systems (where scientific simulations are run), and its ability to share data through Cinema databases, makes it very easy for scientists to use it and share the results of their evaluations. No such tool currently exists for evaluating analysis of data-reduction techniques to help scientists pick the right compression method for their data. As future work we would like to create a feedback loop based on specified quality constraints (e.g., a 1% maximum difference in power spectrum ratio) so that Foresight would iterate on the input compression parameters to find the best parameters to satisfy the stated quality constraints. Another interesting future work would be to run Foresight along with a simulation and perform a limited parameter study in situ, which would allow us to choose the best compressor configuration based on the data that the simulation is generating.

## Acknowledgment

This work has been authored by employees of Triad National Security, LLC which operates Los Alamos National Laboratory under Contract No. 89233218CNA000001 with the U.S. Department of Energy/National Nuclear Security Administration.

The authors would like to thank the National Energy Research Scientific Computing Center (NERSC) for providing access to the Cori supercomputer and LANL for access to the Darwin Supercomputer. They would also like to thank the HACC team at Argonne National Laboratory for granting us access to cosmology datasets.

This research was supported by the Exasky Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. A.T.M. and D.L. have been supported by LANL's LDRD program, project number 20190058DR.

## References

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[2] G. K. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[3] S. Son, Z. Chen, W. Hendrix, A. Agrawal, W. Liao, and A. Choudhary, "Data compression for the exascale computing era - survey," *Supercomput. Front. Innov.: Int. J.*, vol. 1, pp. 76–88, July 2014.

[4] M. Zeyen, J. Ahrens, H. Hagen, K. Heitmann, and S. Habib, "Cosmological particle data compression in practice," in *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, ISAV'17, (New York, NY, USA), pp. 12–16, ACM, 2017.

[5] D. Tao, S. Di, H. Guo, Z. Chen, and F. Cappello, "Z-checker: A framework for assessing lossy compression of scientific data," *CoRR*, vol. abs/1707.09320, 2017.

[6] E. Nemerson, "Squash compression benchmark," 2020. [ONLINE] https://quixdb.github.io/squash-benchmark/, Last accessed on 2020-03-02.

[7] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, and W. keng Liao, "Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49 – 65, 2016.

[8] D. Livescu, J. Mohd-Yusof, M. R. Petersen, and J. W. Grove, "CFDNS: A computer code for direct numerical simulation of turbulent flows," tech. rep., Los Alamos National Laboratory, 2009. LA-CC-09-100.

[9] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, D. Ranta, and R. Stefan, "The RAGE radiation-hydrodynamic code," *Computational Science & Discovery*, vol. 1, p. 015005, nov 2008.

[10] R. Underwood, "Libpressio," 2020. Codesign Center for Online Data Analysis and Reduction [ONLINE]. https://github.com/CODARcode/libpressio/, Last accessed on 2020-03-02.

[11] A. H. Baker, D. M. Hammerling, S. A. Mickelson, H. Xu, M. B. Stolpe, P. Naveau, B. Sanderson, I. Ebert-Uphoff, S. Samarasinghe, F. De Simone, F. Carbone, C. N. Gencarelli, J. M. Dennis, J. E. Kay, and P. Lindstrom, "Evaluating lossy data compression on climate simulation data within a large ensemble," *Geoscientific Model Development*, vol. 9, no. 12, pp. 4381–4403, 2016.

[12] A. H. Baker, D. M. Hammerling, and T. L. Turton, "Evaluating Image Quality Measures to Assess the Impact of Lossy Data Compression Applied to Climate Simulation Data," *Computer Graphics Forum*, 2019.

[13] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu, and Z. Qiao, "Understanding and modeling lossy compression schemes on hpc scientific data," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 348–357, May 2018.

[14] D. Hoang, P. Klacansky, H. Bhatia, P. Bremer, P. Lindstrom, and V. Pascucci, "A study of the trade-off between reducing precision and reducing resolution for data analysis and visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, pp. 1193–1203, Jan 2019.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[18] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 730–739, IEEE, 2016.

[19] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.

[20] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[21] 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," in *European Conference on Parallel Processing*, pp. 366–379, Springer, 2011.

[22] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C.-S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Isabela for effective in situ compression of scientific data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, pp. 524–540, 2013.

[23] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data—the univariate case," *Computing and Visualization in Science*, vol. 19, pp. 65–76, Dec 2018.

[24] F. Alted, "Blosc, an extremely fast, multi-threaded, meta-compressor library," 2019.

[25] A. Hidayat, "Fastlz, free, open-source, portable real-time compression library," *URL http://www.fastlz.org*, 2019.

[26] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak, "Out-of-core compression and decompression of large n-dimensional scalar fields," *Computer Graphics Forum*, vol. 22, no. 3, pp. 343–348, 2003.

[27] A. Biswas, S. Dutta, J. Pulido, and J. Ahrens, "In situ data-driven adaptive sampling for large-scale simulation data summarization," in *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV '18, (New York, NY, USA), p. 13–18, Association for Computing Machinery, 2018.

[28] N. Elmqvist, P. Dragicevic, and J.-D. Fekete, "Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation," *IEEE transactions on visualization and computer graphics*, vol. 14, pp. 1141–8, 11 2008.

[29] D. Murray, *Tableau Your Data!: Fast and Easy Visual Analysis with Tableau Software*. Wiley Publishing, 1st ed., 2013.

[30] C. Sievert, *plotly for R*, 2018.

[31] M. Bostock, V. Ogievetsky, and J. Heer, "$D^3$ data-driven documents," *IEEE Transactions on Visualization & Computer Graphics*, no. 12, pp. 2301–2309, 2011.

[32] J. Woodring, J. P. Ahrens, J. Patchett, C. Tauxe, and D. H. Rogers, "High-dimensional scientific data exploration via cinema," in *2017 IEEE Workshop on Data Systems for Interactive Analysis (DSIA)*, pp. 1–5, Oct 2017.

[33] T. Gamblin, M. P. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and W. S. Futral, "The Spack Package Manager: Bringing order to HPC software chaos," in *Supercomputing 2015 (SC'15)*, (Austin, Texas), November 15-20 2015.

[34] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[35] E. Vaiciukynas, M. Ulicny, S. Pashami, and S. Nowaczyk, "Learning low-dimensional representation of bivariate histogram data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 11, pp. 3723–3735, 2018.

[36] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukic, and E. Van Andel, "Nyx: A massively parallel amr code for computational cosmology," *Astrophysical Journal*, vol. 765, 3 2013.

[37] W. Schroeder, k. Martin, and W. Lorensen, *The Visualization Toolkit*. Kitware, 4th ed., 2006.

[38] S. Jin, P. Grosset, C. Biwer, J. Pulido, J. Tian, D. Tao, and J. Ahrens, "Understanding gpu-based lossy compression for extreme-scale cosmological simulations," *IEEE International Parallel & Distributed Processing Systems*, 05 2020.

[39] D. Daniel, D. Livescu, and J. Ryu, "Reaction analogy based forcing for incompressible scalar turbulence," *Physical Review Fluids*, vol. 3, no. 9, p. 094602, 2018.

[40] K. Kanov, R. Burns, C. Lalescu, and G. Eyink, "The johns hopkins turbulence databases: an open simulation laboratory for turbulence research," *Computing in Science & Engineering*, vol. 17, no. 5, pp. 10–17, 2015.

[41] J. Pulido, D. Livescu, K. Kanov, R. Burns, C. Canada, J. Ahrens, and B. Hamann, "Remote visual analysis of large turbulence databases at multiple scales," *Journal of Parallel and Distributed Computing*, vol. 120, pp. 115–126, 2018.

[42] N. Frontiere, C. D. Raskin, and J. M. Owen, "Crksph–a conservative reproducing kernel smoothed particle hydrodynamics scheme," *Journal of Computational Physics*, vol. 332, pp. 160–209, 2017.

[43] S. Colombi, A. H. Jaffe, D. Novikov, and C. Pichon, "Accurate estimators of power spectra in n-body simulations," *MNRAS*, vol. 393, 11 2008.

[44] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, *et al.*, "The rage radiation-hydrodynamic code," *Computational Science & Discovery*, vol. 1, no. 1, p. 015005, 2008.

[45] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, and J. Mauldin, "Paraview catalyst: Enabling in situ data analysis and visualization," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV2015, (New York, NY, USA), p. 25–29, Association for Computing Machinery, 2015.

[46] J. Patchett and G. Gisler, "Deep water impact ensemble data set," *Technical Report LA-UR-17-21595, Los Alamos National Laboratory*, 2017.
[47] K. Wunnemann and R. Weiss, "The meteorite impact-induced tsunami hazard," *The Royal Society*, 2015.
[48] A. Mohan, D. Daniel, M. Chertkov, and D. Livescu, "Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence," *arXiv preprint arXiv:1903.00033*, 2019.
[49] A. Mohan, D. Tretiak, M. Chertkov, and D. Livescu, "Spatio-temporal deep learning models of 3d turbulence with physics informed diagnostics," *Journal of Turbulence*, p. in press, 2020.
[50] P. Lindstrom, "Error distributions of lossy floating-point compressors," in *Joint Statistical Meetings*, vol. 2017, pp. 2574–2589, 2017.

# Appendix: Artifact Description/Artifact Evaluation

## SUMMARY OF THE EXPERIMENTS REPORTED

We ran the HACC CBench experiments on the CORI supercomputer at NERSC using C++ and MPI. The analysis runs were done on the Darwin cluster at Los Alamos National Lab since many of the analysis code is proprietary.

The HACC dataset is public as well as the original asteroid impact dataset but not the turbulence dataset.

## ARTIFACT AVAILABILITY

*Software Artifact Availability:* All author-created software artifacts are maintained in a public repository under an OSI-approved license.

*Hardware Artifact Availability:* There are no author-created hardware artifacts.

*Data Artifact Availability:* There are no author-created data artifacts.

*Proprietary Artifacts:* There are associated proprietary artifacts that are not created by the authors. Some author-created artifacts are proprietary.

*Author-Created or Modified Artifacts:*

```
Persistent ID:
↪ https://github.com/lanl/VizAly-Foresight
Artifact name: Source code repository

Persistent ID: http://doi.org/10.5281/zenodo.3875675
Artifact name: Zenodo DOI of the latest stable release

Persistent ID: http://doi.org/10.5281/zenodo.3875684
Artifact name: DOI of the current dev branch

Persistent ID:
↪ https://lanl.github.io/VizAly-Foresight/
Artifact name: Cinema Explorer database of results

Persistent ID: http://dx.doi.org/10.21227/zg3m-8j73
Artifact name: HACC Dataset

Persistent ID:
↪ https://oceans11.lanl.gov/deepwaterimpact/
Artifact name: Deep Water Impact Asteroid dataset
```

## BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

*Relevant hardware details:* Cori at NERSC, Darwin at LANL

*Operating systems and versions:* The OS on these supercomputers

*Compilers and versions:* C++ 14

*URL to output from scripts that gathers execution environment information.*

```
https://github.com/lanl/VizAly-Foresight/tree/v1.3.1
↪ /scripts
```

## ARTIFACT EVALUATION

*Verification and validation studies:* We ran analysis on three different kinds of simulation/data. The results for HACC were validated by the HACC team at Argonne national lab. The other results were validated by scientists at Los Alamos National Lab.

*Accuracy and precision of timings:* Timing is not what we are measuring in the paper. What we are measuring is compression quality and impact on analysis. The compression algorithms are deterministic; running them several times should give the same compression ratio. We did test the validity of the framework by using BLOSC which is lossless and make sure that the results from BLOSC match the original uncompressed data. The only metric that could vary on each run is the throughput and this is averaged over the many ranks that these compressors are running.

*Used manufactured solutions or spectral properties:* Not applicable

*Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment:* Not applicable

*Controls, statistics, or other steps taken to make the measurements and analyses robust to variability and unknowns in the system.* Not applicable