International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

*Research Paper*

The International Journal of High Performance Computing Applications
2023, Vol. 37(5) 539–559

DOI: 10.1177/10943420231180504
journals.sagepub.com/home/hpc
Sage

# Accelerated dynamic data reduction using spatial and temporal properties

**Megan Hickman Fulp[1], Dakota Fulp[1], Changfeng Zou[1], Cooper Sanders[1], Ayan Biswas[2], Melissa C. Smith[1] and Jon C. Calhoun[1]**

## Abstract

Due to improvements in high-performance computing (HPC) capabilities, many of today's applications produce petabytes worth of data, causing bottlenecks within the system. Importance-based sampling methods, including our spatio-temporal hybrid data sampling method, are capable of resolving these bottlenecks. While our hybrid method has been shown to outperform existing methods, its effectiveness relies heavily on user parameters, such as histogram bins, error threshold, or number of regions. Moreover, the throughput it demonstrates must be higher to avoid becoming a bottleneck itself. In this article, we resolve both of these issues. First, we assess the effects of several user input parameters and detail techniques to help determine optimal parameters. Next, we detail and implement accelerated versions of our method using OpenMP and CUDA. Upon analyzing our implementations, we find 9.8× to 31.5× throughput improvements. Next, we demonstrate how our method can accept different base sampling algorithms and the effects these different algorithms have. Finally, we compare our sampling methods to the lossy compressor cuSZ in terms of data preservation and data movement.

## Keywords

Data reduction, data sampling, importance sampling, temporal selection, feature preservation, parameter optimization, GPU, CUDA, OpenMP, accelerated hardware, performance

## Introduction

Modern high-performance computing (HPC) systems have increasingly high computation capabilities, allowing scientific simulations to solve previously intractable problems. These intensive simulations are capable of producing petabytes of data (Strand 2015; Habib et al., 2013), yet current HPC system I/O capabilities are not capable of handling such data efficiently (Cappello et al., 2019). As a result, conventional post hoc data analysis is less tractable, as storing all output data is costly (Tikhonova et al., 2010; Nouanesengsy et al., 2014; Dutta et al., 2017; Ahrens et al., 2014). Many researchers use data reduction techniques to reduce data size before any I/O operations to alleviate this bottleneck.

Data sampling is a popular data reduction method that saves a subset of data values and uses this subset to reconstruct missing data when the whole dataset is needed. Some existing sampling methods utilize uniform random selection techniques to determine which points to keep (Woodring et al., 2011; Childs 2015; Wei et al., 2018), while others focus on preserving specific regions of interest (ROI) in the data (Biswas et al. 2018, 2020a; Nouanesengsy et al., 2014). In our prior work, we develop a spatio-temporal hybrid data sampling method that biases rare data values and leverages a dataset's temporal aspect to achieve higher post-reconstruction quality (Fulp et al., 2020).

While our hybrid data sampling method has been shown to outperform existing reduction schemes, there exist three areas of improvement that we must address. First, the effectiveness of our method relies heavily on user input parameters, which highly affect the reduction throughput and data quality (see Section 4). However, in our previous work, the extensive effects of these parameters have yet to be explored. Second, while our method aims to work with HPC applications, our CPU implementation is ill-prepared due to

[1]Holcombe Department of Electrical and Computing Engineering, Clemson University, Clemson, SC, USA
[2]Los Alamos National Laboratory, Los Alamos, NM, USA

**Corresponding author:**
Megan Hickman Fulp, Holcombe Department of Electrical and Computing Engineering, Clemson University, Clemson 29634, SC, USA.
Email: mlhickm@clemson.edu

the lower throughput of the overall reduction process (see Section 5). Last, as our method uses a specific sampling algorithm at its core (Biswas et al., 2018), this may cause our method to be dated as the field of sampling grows (see Section 6).

In this work, we aim to resolve these issues by providing methods to aid the user in finding an optimal set of parameters, improving our method's throughput with OpenMP and CUDA, and detailing how to use other sampling cores within our method. Specifically, our novel contributions are as follows:

- We analyze the impacts of various user input parameters on the effectiveness of our hybrid data sampling method and detail methods to assist the user in determining an optimal set of input parameters.
- We describe, implement, and assess accelerated versions of our hybrid data sampling method in OpenMP and CUDA. Upon evaluating our GPU-based implementations, we find throughput improvements of 9.8× to 31.5× when compared to the OpenMP implementations.
- We demonstrate how our hybrid data sampling method accepts different sampling core algorithms and analyze the effects of using different core algorithms.

## Related works

### Spatial and temporal data reduction methods

While reducing the size of scientific datasets is critical, these datasets often include features that are more important to scientists; thus, not all data values are equally important. To meet the needs of domain scientists, data reduction techniques that significantly reduce data size while maintaining these features are necessary. Importance-based sampling preserves these features by assuming that rare values are more important and by giving these rare values a sampling bias. Biswas et al.'s importance-based sampling method uses this approach and, as such, over-represents rare values, without ignoring more common values (Biswas et al. 2018, 2020a; Nouanesengsy et al., 2014). Their method uses the distribution of data values to calculate an importance factor ($0 \leq I_F \leq 1$) for each data point ($p_i$) such that rare values have a higher $I_F$ and more frequent values are assigned a lower priority. Upon deciding $I_F(p_i)$, a random number $\xi$ is generated for each data point. Their method then determines if $\xi < I_F(p_i)$ for each data point, and, if so, it includes the data point in the sample.

Another approach to data reduction is through time-step selection (Akiba et al., 2006). This type of data reduction analyzes the differences between sequential time-steps to determine which time-steps provide a representative overview of the entire data series. These few representative time-steps are selected to be saved while the other time-steps are discarded, thus reducing the overall data size. For example, assuming the previous time-step ($t_{k-1}$) has been selected, we need to decide whether to select the current time-step ($t_k$) as well. Upon comparing the two, if $t_k$ is similar enough to $t_{k-1}$, we do not need to select it as $t_{k-1}$ is a sufficient representation.

Using concepts from Biwas et al.'s importance-based sampling (Biswas et al., 2020b) and existing time-step selection methods, we develop our spatio-temporal hybrid data sampling method in prior work (Fulp et al., 2020). This method provides a bias to more rare data values while leveraging the temporal aspect of the data to use its sampling budget efficiently. This approach has been shown to achieve higher post-reconstruction data quality.

### Accelerated data reduction methods

To keep pace with existing and future HPC applications, many researchers use GPUs to improve algorithm throughput. Currently, many works have studied the use of GPUs to reduce big data (Tian et al., 2020; Wang et al., 2009; Fogal et al., 2013; Gutiérrez et al., 2017). While there are various forms of both sophisticated and straightforward data sampling for GPUs, a majority of them are specifically designed for computer graphics (Wang et al., 2009; Fogal et al., 2013) or machine learning (Gutiérrez et al., 2017). While their specializations make them great for their specific fields, they also make them ill-suited for all scientific datasets requiring high data fidelity.

The first example of these works utilizes data sampling on a GPU for simulating global illumination (Wang et al., 2009). Next, SMOTE-GPU is a GPU implementation of the Synthetic Minority Oversampling Technique (SMOTE) that performs data sampling algorithms based on the SMOTE algorithm. However, this work is based on a machine-learning sampling aspect and studies how imbalanced data samples can affect machine learning (Gutiérrez et al., 2017). Finally, Fogal et al. implement a GPU sampling algorithm that focuses on identifying regions that should be densely sampled (Fogal et al., 2013). Similar to our methods of dividing a dataset into regions, their algorithm subdivides the volume into pieces, then loops over each "brick" and samples the data. However, their work comes from a volume renderer point of view; thus, they focus on using concepts of empty space skipping for renders rather than choosing samples that will maintain data fidelity post-reconstruction.

To the best of our knowledge, our approach is the first work to leverage OpenMP and GPUs to improve existing importance-based sampling methods. Furthermore, our approach enables CUDA HPC applications to retain their high throughput performance by removing the need to return to the host to perform sampling operations.

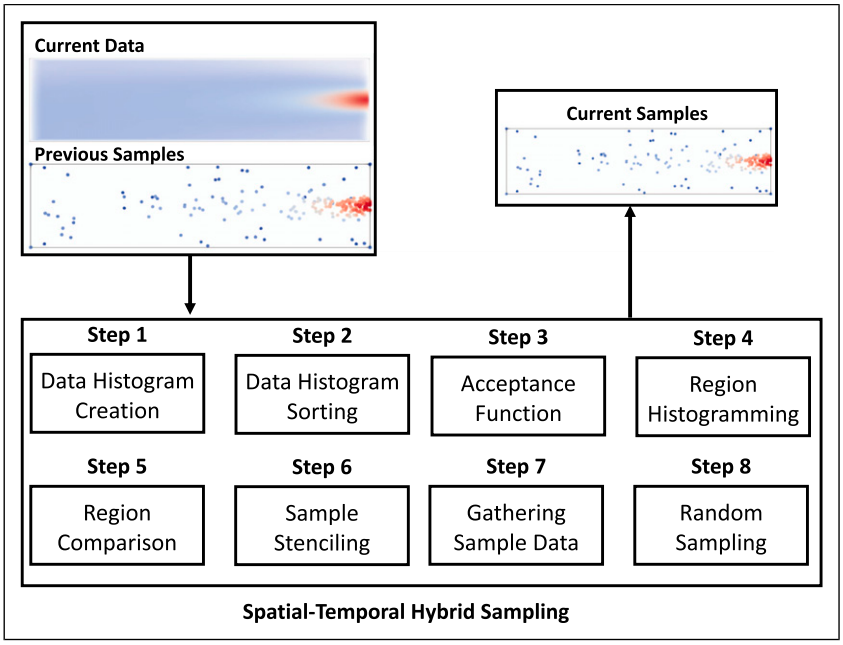


**Figure 1.** An overview of our spatio-temporal hybrid data sampling method process.

## Background

### *Importance-based sampling*

In many scientific simulations, there exist rapidly changing regions of interest (ROI) that should be kept with higher fidelity, as they are more important to domain scientists. Biswas et al.'s importance-based sampling method (Biswas et al., 2018) and our spatio-temporal hybrid data sampling method (Fulp et al., 2020) take a biased sampling of data that over-represents rare values without neglecting common values. However, our hybrid sampling method also leverages the dataset's temporal properties to enable higher post-reconstruction data quality, overall and within the ROI. It accomplishes this by comparing data regions of consecutive time-steps using histogram intersection or root-mean-squared error (RMSE) and reuses similar previous regions.

Overall, Biswas et al.'s non-reuse sampling method consists of five key steps, as detailed in Figure 1: 1, 2, 3, 6, and 7. In contrast, our spatio-temporal hybrid data sampling method consists of all eight steps in Figure 1.

**Step 1. Data Histogram Creation:** The first step in both methods is to create a histogram of all data values in the current time-step. When creating this histogram, the user defines the number of histogram bins, and the histogram range is set to the minimum and maximum values of the current time-step's data.

**Step 2. Data Histogram Sorting:** Both methods then sort the resulting histogram bins from least to greatest. The resulting list of bins is used to develop an acceptance function that biases more rare data values.

**Step 3. Acceptance Function Development:** Using the sorted histogram, both methods develop an acceptance function that determines the acceptance rate for values that are within each bin's range. Using the user-defined sample ratio, each method determines the target number of samples to keep and divides this among all bins. Following this, each method iterates over the sorted histogram bins to determine whether the data values in the bin's range are more than the target number of samples per bin. If there are more values than the target, each method sets the total samples for that bin equal to the target number of samples. However, if there are fewer data values than the target, the total samples for that bin are set to the number of data values. When a bin does not utilize its entire sample budget, the unused budget is redistributed among all remaining bins. Once each method determines the number of samples to keep per bin, they divide this number by the data values in the corresponding bin to create an acceptance rate between 0 and 1. This ensures a majority of rare data values are kept while still allowing for some of the more common values.

**Step 4. Region Histogram Construction:** Before sampling, the hybrid data sampling method divides each time-step into equally sized regions. Then, the data within a region of the current time-step is compared to the corresponding region of the previous time-step. This comparison either uses histogram intersection or RMSE, depending on the user's specification. When using histogram intersection, the hybrid data sampling method requires an extra step to create histograms for each region of the current time-step.

These histograms use the same number of bins as before, but the histogram range is set to the minimum and maximum values one expects throughout the simulation's lifetime. While the exact minimum and maximum are not always known before the simulation finishes, we use a speculative range for the purposes of our work. If our algorithm encounters a value outside the range, it places the values into the first or last bin depending on which side of the range the value falls outside of. Overall, using a range closest to the true lifetime range will yield the best quality. These results are saved for the next time-step's comparison step to avoid excess computations.

**Step 5. Region Comparison and Reuse:** Using the histogram intersection or RMSE metric, the hybrid data sampling method next compares each region of the current time-step with the corresponding region of the prior time-step. If the histogram intersections are above the intersection threshold or if the error is below the user-defined threshold, the hybrid method flags the region for reuse. When previous time-step information is not available, the hybrid data sampling method uses Biswas et al.'s non-reuse sampling method.

**Step 6. Random Number Generation and Sample Stenciling:** At this stage, both methods determine which samples to save. For each data value, both methods generate a random value between 0 and 1. If this value is less than the acceptance rate and the data value is not in a region previously planned to be reused, the sampling method flags the data value to be kept in the sampling stencil.

The non-deterministic behavior of random sampling may raise concerns among some users who require consistent and reproducible results, as the nature of random operations may affect the comparability of results. If the results differ significantly depending on the random seed or the particular subset of data selected by the random sampling, it may be challenging to make valid comparisons between the algorithms. Our method resolves the non-deterministic nature of random sampling by using the multi-criteria importance analysis from Biswas et al. to ensure it clusters the samples it takes around the region of interest. Utilizing a random sample approach in this step of the ExaAM dataset (see Figure 1; 100 trials, 5% sample ratio) introduces a 2.7 dB standard deviation and an average PSNR of 38.5 dB. Using the deterministic systematic sampling method ensures the consistency and reproducibility of the results. The same experiments had a standard deviation of zero; however, they only reached a PSNR of 18.3 dB because they do not sufficiently sample the region of interest leading to large errors, which degrade accuracy. Therefore, having some randomness included in the process is preferred despite its non-deterministic nature, as it is designed to balance the need for randomness with the need for representative samples.

**Step 7. Gathering Sample Data:** Our method then uses the resulting sampling stencil to collect information on the chosen samples and appends them to the sample data array. When storing information on a value's location, our method saves a single global index value rather than three individual coordinate values. Our method calculates this global index value as $int(x + y*XDIM + z*XDIM*YDIM)$. By operating in this manner, our method introduces a storage overhead of only 2× in the number of samples, as opposed to the 4× overhead of storing the coordinates separately. We consider this overhead when we compare our method to other data reduction methods in later sections.

**Step 8. Additional Random Sampling:** As the hybrid data sampling method reuses regions, it may not use the entire sampling budget. To rectify this, it determines the number of extra samples it should collect and randomly distributes this amount among all regions that it sampled this time-step. Similar to before, it uses this information to determine a new acceptance rate and begins a random sampling pass. This ensures that it uses all of the sampling budget to achieve the highest post-reconstruction quality possible.

### Reconstruction

In our workflow, to visualize and analyze the effectiveness of our samples, we first restore each time-step back to full resolution. We use a linear interpolation-based reconstruction using a Delaunay (Delaunay et al., 1934) triangulation to reconstruct. To begin, this reconstruction method uses the stored sample points to build interpolation triangles. It then uses these triangles to interpolate each missing data point. The reconstruction method uses a weighted average of the three triangle vertices to rebuild a missing data point. The weight associated with each vertex is dependent on the distance from the target data point. Finally, the reconstruction method reconstructs the data point using the vertex values and their weights. We use this method as a balanced trade-off between quality and speed, whereas generally, the higher order the interpolation is, the better the quality but the slower the process.

### Optimization strategies

To improve application performance, many developers leverage thread-level parallelism interfaces such as OpenMP and CUDA. OpenMP[1] is a CPU multi-threading programming interface that enables users to improve the throughput of highly parallelizable tasks. CUDA[2] is a parallel programming interface that enables users to improve the throughput of highly parallelizable tasks through a CUDA-enabled graphics processing unit (GPU). Each of these widely used interfaces provides different advantages and disadvantages.


**Figure 2.** ExaAM time-step 64 with highlighted ROI.

### Data sets

*ExaAM.* The Exascale Additive Manufacturing Project uses exascale simulations to design Additive Manufacturing components (Belak et al., 2019; Jibben 2020). We use 108 time-steps with its full spatial resolution of 20 × 200 × 50. Figure 2 shows time-step 64, with highlighted ROI, the hottest portion of the visual.

*Hurricane Isabel.* The Hurricane Isabel Data models the 2003 hurricane in the west Atlantic region (Hurricane ISABEL Simulation Data 2019). In the following experiments, we use the pressure variable, as it provides a distinct representation of the eye of the hurricane, the ROI of this data set (Figure 3(c)). We use 48 time-steps of the full resolution, 500 × 500 × 100.

*Asteroid Impact.* The Deep Water Impact Ensemble data set represents the study of the impact of an asteroid in deep ocean water to learn the limit of dangerous asteroids (Patchett and Gisler 2017). We use the full spatial resolution of 300 × 300 × 300 over 100 time-steps. We use the water volume variable, V02, as it visualizes the water splash. The data values range from 0.0 to 1.0, where 1.0 is pure water. Figure 4 shows time-step 22,388 of this data set, with highlighted ROI (the water splash).

## Determining input configurations

Our spatio-temporal hybrid sampling method has multiple configurable parameters to allow the user to best select a combination to yield a higher quality of reconstructed data or higher reduction throughput, depending on their data set and situation. This section examines the impact of various input configurations and helps the user determine an optimal configuration for their situation. We refer to our findings as "an optimal" configuration, as there is often not one sole configuration that yields the overall highest quality and bandwidth, but rather a range of trade-offs.

### Number of bins

Both our hybrid method and Biswas et al.'s non-reuse method rely heavily on the histogramming of the input data. This is especially true for our hybrid method, as it uses histogram intersection to determine whether to reuse regions or not.

As simulations develop, the optimal number of histogram bins may change. However, as our hybrid method uses histogram intersection to determine whether to reuse regions or not, we must maintain a consistent number of bins to ensure we make an accurate comparison. Furthermore, as both methods are designed to run in situ, we only have access to the first time-step to determine an optimal number of bins. With this in mind, both methods are designed and operate best with smooth simulations.

While the user could manually set this parameter if they had extensive knowledge of the dataset, we aim to provide a generic approach such that no previous knowledge of the data is required. If the user does not know how to set the number of bins properly, we provide an optimization step as detailed in Section 4.1.1.

To better understand how the number of bins affects our hybrid method, we evaluate the three regions specified in Figures 3(a) and (b) from the Hurricane Isabel data set. We first analyze the amount of intersection between two histograms at the same region in neighboring time-steps, as we vary the number of bins used. Figure 5 shows that there is less intersection as we increase the number of bins. Using more bins, we parse values out to more specific bins, reducing the areas where both histograms overlap. This correlation is especially true within regions of high entropy, like region A. This is concerning as, without enough intersections, our hybrid method is left unoptimized and rarely utilizes previous samples. However, while fewer bins enable our hybrid method to group more items, too few bins lead to excess intersections and high levels of error in the resulting data.

While the user can specify any percentage of histogram intersection as a threshold for determining whether to reuse samples from a previous region, we choose to reuse only when both histograms are identical. This approach enables us to maintain high data fidelity. We conduct an in-depth study on how histogram intersection threshold affects the number of regions reused and the resulting reconstructed data quality in Section 4.2. Figure 6 showcases how the

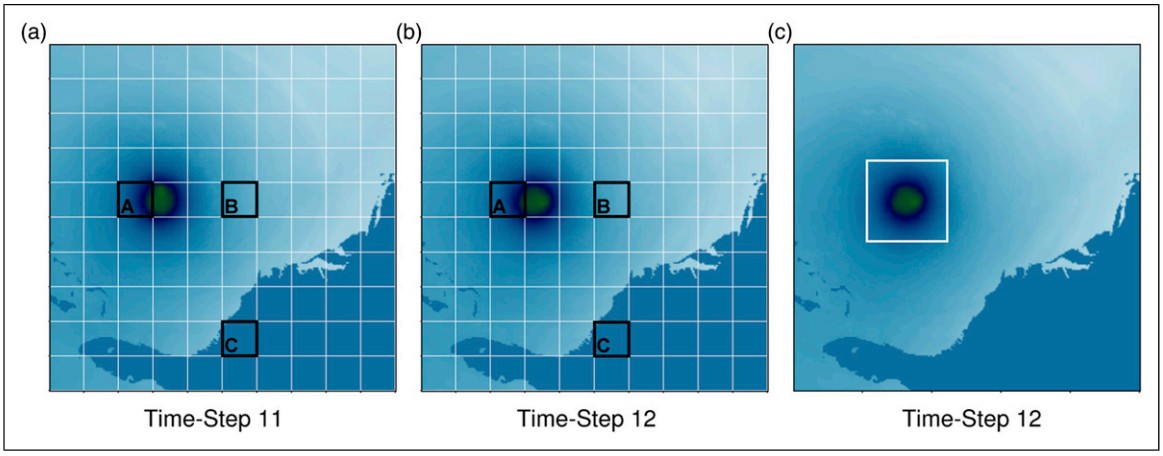


**Figure 3.** Hurricane Isabel Pressure Visualizations. Figures (a) and (b) show the dataset divided into regions of dimension 25 × 25 × 25. Figure (c) highlights our definition of Region of Interest for this dataset.

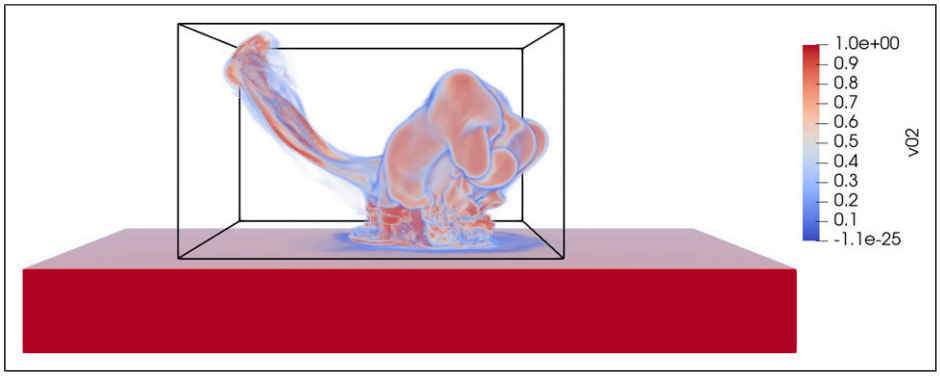


**Figure 4.** Impact time-step 22,388 with highlighted ROI.

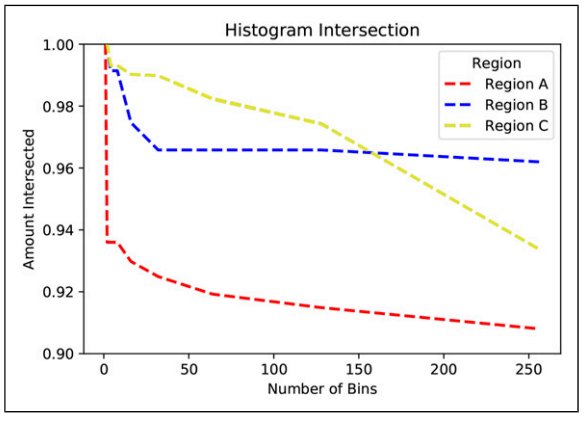


**Figure 5.** Amount of Intersection with varying bins at three regions of Hurricane Isabel (see Fig. 4).

number of bins used affects the percentage of reused regions, independently of dataset and region size. From this figure, our results show that using a higher number of bins leads to much lower levels of reuse for all regions sizes tested. Similarly, when using too few bins, too many regions are reused.

*Determining an optimal configuration.* To assist users in determining an optimal number of histogram bins when using our hybrid method or Biswas et al.'s non-reuse method, we run a pre-processing step using existing algorithms to suggest an optimal number of bins based on data from the first time-step. While Sturges' rule for estimating an optimal number of bins is widely recommended, it is only optimal for Gaussian data (Scott 2009). Therefore, we use Doane's rule (Hyndman 1995; Doane 1976), a modification to Sturges' rule that works better with non-normal data sets. Secondly, we use Scott's rule (Scott 2010), which is more statistically rooted and takes both data size and variability into account, and works well with large datasets.

We use Doane's and Scott's rules to provide users with a range of bins to consider using with their dataset. To better assist users, our pre-processing step also collects a few samples and reconstructs the data to estimate what resulting data quality using each of these bins will yield. Using this information, our pre-processing step recommends the

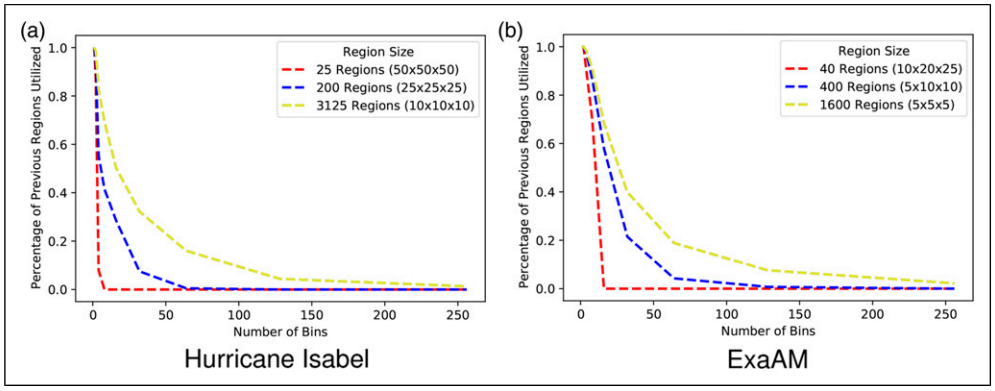


**Figure 6.** Percentage of previous regions utilized, varying number of bins and region sizes.

number of bins that yield the highest quality. Since we only want an estimation of the quality to assess which number of bins is most optimal, we use an OpenMP accelerated version of nearest neighbor's reconstruction, which is faster but lower quality than other reconstruction methods. Thus, the overhead of this process depends heavily on the time it takes to sample and reconstruct the first time-step of the dataset twice.

With a sample ratio of 1%, this pre-processing step has an average overhead of 0.2% of the entire sampling process, slightly varying as the input data set size and number of time-steps increases. The majority (≥93%) of this step is spent in the reconstruction and quality analysis phase. This is an acceptable temporal trade-off, as this configuration process only needs to be run once per data series and, as determining the number of bins to use is a complex problem, this step assists users in determining the input that leads to the highest overall quality.

Since our pre-processing step determines an optimal number of bins based only on the first time-step of the series, it is possible that it does not yield the absolute optimal number of bins for the average time-step in the series. However, our Hybrid sampling method is designed to utilize temporal similarities; thus, it works best with datasets that change smoothly over time. Assuming the input data is smooth, Figure 7 shows that choosing the optimal number of bins for the first time-step and using that throughout sampling the rest of the series yields the same quality as if we were to recompute the optimal number of bins for each time-step independently. Thus, while our method does not always yield the absolute optimal number of bins, it does provide a more near-optimal option than a user could choose at random.

Figure 8 demonstrates the effects of using this pre-processing step with Biswas et al.'s non-reuse method, while a similar experience is seen with our hybrid method. This figure shows that the quality achieved with a certain number of bins and the first time-step reflects the average quality of the first 10 time-steps. We compare against the first 10 time-steps only as our method aims to work with

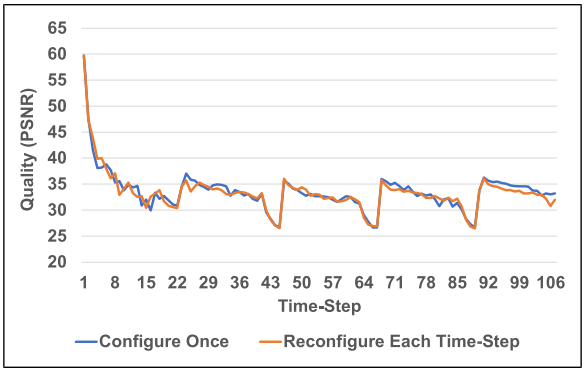


**Figure 7.** Using a consistent number of bins yields equivalent quality to varying number of bins per time-step (ExaAM).

smooth simulations and, as a result, these first ten time-steps are representative of the overall simulation. Therefore, while our pre-processing step only uses the first time-step to determine what number of bins to use, it continues to be an optimal configuration as the data series progresses. For example, Figure 8(a) shows that given the ExaAM dataset, a user could choose to use an input of 10 bins, resulting in an overall PSNR of 42 dB. However, by using our pre-processing step, the user would find using 633 bins to be more optimal in terms of quality, yielding a PSNR of 60 dB.

We use the number of bins recommended by this pre-processing step for each dataset, as listed in Table 1.

## *Histogram intersection threshold*

Histogram intersection uses a user-set histogram threshold to determine whether to reuse the region or not for the following time-step, as long as the region's histogram intersection is higher than the threshold. In this work, we ran all datasets on a 100% intersection threshold, meaning we only reused samples in a region where the histograms are 100% identical. This ensures that the distribution of data values are the same in the

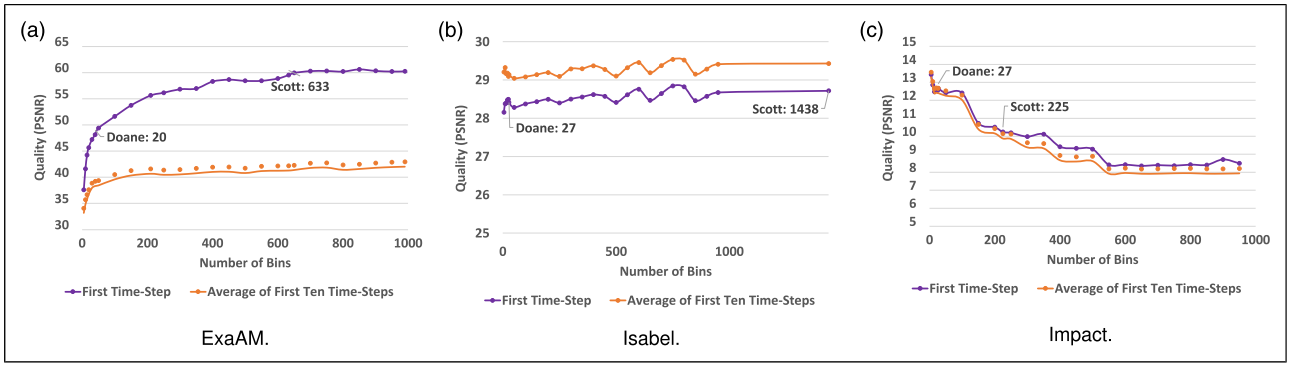


**Figure 8.** The average quality of the first ten time-steps, varying number of bins (Non-Reuse Method).

**Table 1.** Datasets and configurations used in experimental evaluations.

| Dataset | Variable | Dimensions | Data size, MB | Steps | Region dimensions | BINS | Error threshold |
|---|---|---|---|---|---|---|---|
| ExaAM | - | 20 × 200 × 50 | 0.8 | 108 | 10 × 40 × 10 | 633 | 0.0 |
| Isabel | Pressure | 500 × 500 × 100 | 95 | 48 | 25 × 25 × 25 | 27 | 28.0 |
| Impact | V02 | 300 × 300 × 300 | 108 | 130 | 50 × 50 × 50 | 27 | 0.0 |

regions that are reused and yields a higher post-reconstruction quality. As the histogram threshold is made less strict, more blocks are reused. Reusing more regions introduces more error, lowering the post-reconstruction quality. However, there is a trade-off where we can lower the tolerance, reuse more regions, and meet nearly the same PSNR. The more regions reused, the fewer new samples that need to be taken for that time-step, meaning less time is spent sampling regions.

To showcase the relationship between histogram threshold, quality, and block re-usage, we test various histogram intersection tolerances for various datasets, as shown in Figure 9. For the ExaAM dataset, as the histogram intersection threshold approaches 60%, there is no change in PSNR because there is no change in the number of blocks reused. However, between 60% and 75%, as the percentage of blocks reused decreases, less error is introduced, and the quality increases. Between a threshold of 75% and 100%, the PSNR only has an increase of 5%. This trend signifies some optimum trade-off between the speed at which the sampling operation can be performed by performing fewer sampling operations and reusing more previously gathered samples, but at the cost of data quality. However, for all datasets, the most optimum threshold, in terms of quality, exists at the fewest number of blocks reused. Since this article focuses on achieving high quality of a post-reconstruction dataset, we continue to use a threshold of 100% to maximize PSNR for our tests to determine the effectiveness of our method. Still, the effectiveness of histogram intersection heavily depends on the dataset itself.

The impact of the histogram intersection threshold is more apparent for a dataset similar to the ExaAM dataset, where regions rapidly change through time. On the other hand, the Asteroid Impact dataset shows no improvement in performance when the histogram intersection decreases. As a matter of fact, for that dataset, in particular, the best histogram threshold is at 100% since the regions across time-step remain similar. Thus, if the domain sees little change between time-steps, we suggest a histogram intersection near 100%. As the domain changes more rapidly, a lower histogram intersection value is recommended.

### *Error threshold*

When using RMSE to compare regions over time-steps, we use a user-set error threshold to limit the amount of error allowed when utilizing previous samples. As a generic standard to determining this threshold, we calculate the difference of each corresponding data value between the first two time-steps. We use the third quartile of this list of errors as the error threshold. This allows enough error to utilize previous samples while not negatively affecting the reconstructed quality.

With a sample ratio of 1%, this pre-processing step has an average overhead of 0.03% of the entire sampling process, varying as data set size and number of time-steps varies. As with the previous pre-processing step, providing the user with a configuration that leads to better overall data quality outweighs the small amount of overhead introduced.

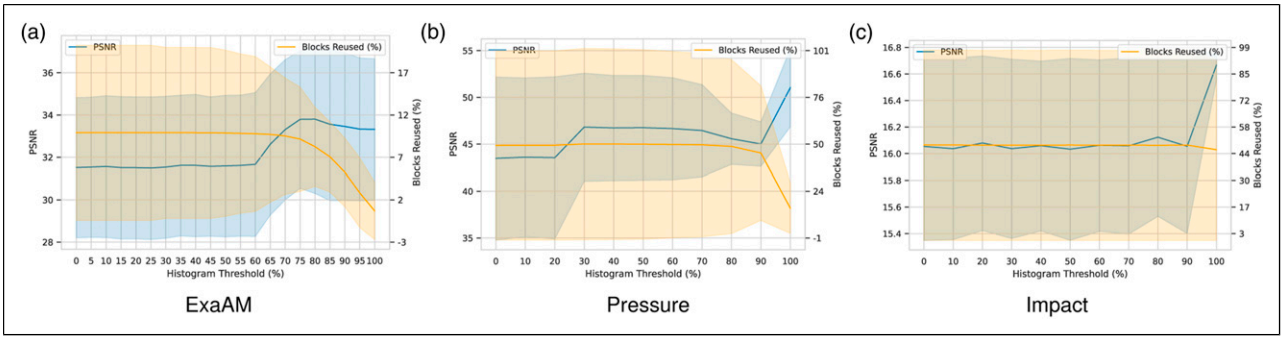


**Figure 9.** Reconstruction accuracy and percent of previous regions utilized as the histogram threshold changes.

Figure 10 shows how the quality of the reconstructed data is affected by varying the error threshold when gathering samples. This figure shows that the suggested error threshold yields one of the highest qualities for the first time-step and across the first ten time-steps.

We use the error threshold recommended by this pre-processing step for each dataset, as listed in Table 1. With two of the datasets tested, the third quartile of the error distribution is equal to zero because they change smoothly over time, causing the difference between the first two time-steps to be very low. Even with an error threshold of 0.0, there are still enough similarities between regions to consider samples reusable.

### *Number of regions*

Setting an appropriate region size is critical for optimal performance of our hybrid method, as a region size too small or too large affects overall efficiency. We evaluate the effects of different sized regions and quantify their impact in Figure 11. In this assessment, we compare the average percentage of regions utilized from time-step tk−1 when gathering samples for time-step tk, of the first 10 time-steps. Our method utilizes more samples from tk−1 when we divide the dataset into more regions than when using fewer regions. The more regions we divide the dataset into, the smaller and more specific the amount of data within them becomes, allowing us to be more specific with the information we are reusing. By utilizing more regions, we have access to more samples, which generally correlates to a higher post-reconstruction quality. Thus, if the user wants the highest qualities possible, they would use a larger number of regions. Consequently, breaking the data into more regions also increases the number of similarity comparison computations across each time-step. Figure 11 shows that by introducing more comparisons, the sampling algorithm is drastically slowed down. Therefore, when determining the number of regions to divide the data set, the user must set focus on higher post-reconstruction quality, higher bandwidth, or some trade-off of both.

*Determining an optimal configuration.* To aid the user in choosing an optimal number of regions for their situation, we provide plots of the percentage of regions utilized and bandwidth versus the number of regions, for a subset of the number of regions possible. Given these plots, the user visualizes the quality-bandwidth trade-off and chooses their number of regions according to whether they value accuracy or speed more.

While an exhaustive study of providing results for every possible region dimension would yield a more specific plot, this would introduce a costly overhead that outweighs its benefits. Thus, we use a distributed subset of 10 region dimensions, as it lowers the temporal overhead while adequately representing the trends.

With a sample ratio of 1%, the average introduced overhead of this pre-processing step is ≤3% of the entire sampling process. Similar to before, we deem this overhead as an acceptable trade-off, as this process provides the user with configuration parameters better suited to their data set.

For example, given the ExaAM dataset, if the user were to use 1000 regions, they would reuse between 90% and 100% of their regions. While this results in higher data quality due to the space saved from reuse, it does lead to lower overall bandwidth, as seen in Figure 11(a). Our method provides the user with the trade-off graph, and, given this information, they are well-equipped to determine the most suitable number of regions for this current work.

## Throughput improvements

As our hybrid data sampling method aims to work with HPC applications, we must address its lower throughput. Often, researchers use parallelization strategies such as OpenMP and CUDA to improve application performance. With this in mind, we develop two distinct parallelization approaches to our method. The first approach uses OpenMP to optimize key steps within our method. Similarly, in the second parallelization approach, we create separate CUDA kernels for each key step within our method, which we run on an

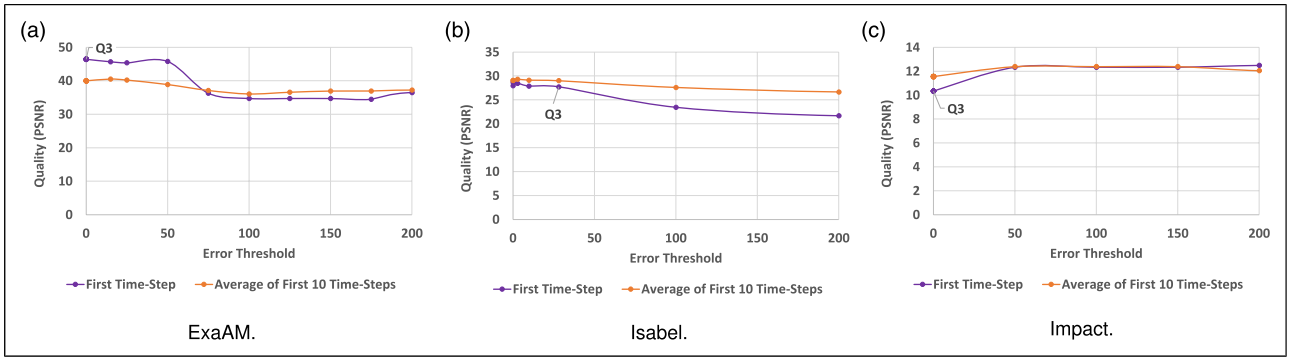


**Figure 10.** The average quality of the first ten time-steps, varying error threshold (Error-Reuse Method).

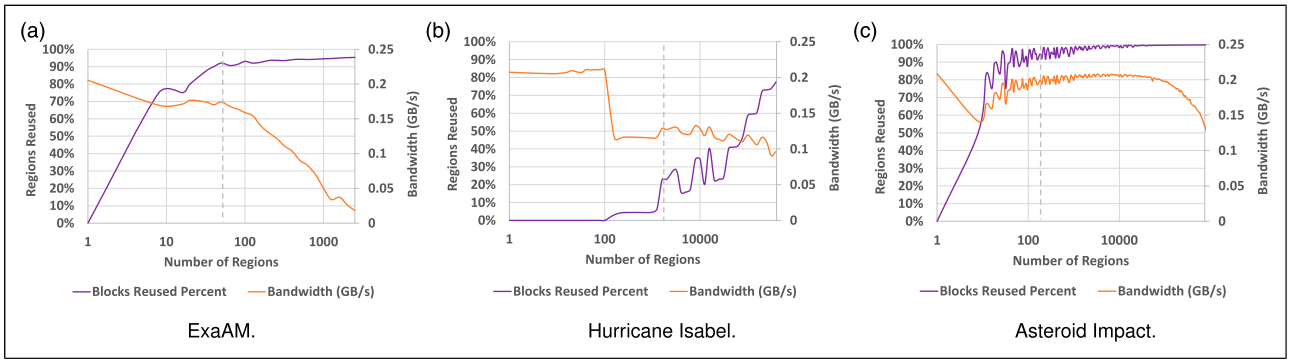


**Figure 11.** Sampling bandwidth and percent of previous regions utilized as the number of regions varies (Histogram-Reuse Method).

NVIDIA Telsa V100 GPU. The design of both of our parallelization approaches is seen in Figure 12. We also parallelize Biswas et al.'s non-reuse importance-based sampling method for later comparison using this same design.

## Design methodology

**Step 1. Data Histogram Creation:** As histograms are critical to both methods, parallelizing their creation is crucial. In our OpenMP approach, we parallelize the creation by dividing the data values equally among all threads. Each thread constructs a private data histogram and, upon processing all data values, atomically adds its results to a global histogram. In our CUDA approach, the histogram kernel assigns each input element to a thread, using 1,024 threads per block. As CUDA threads run in warps of 32, it is important that this thread block size be a multiple of 32 so that the warps do not have any idle threads. Then, each block zeros a privatized histogram array in shared memory, and threads compute an address and increment the appropriate bin using atomic operations. This use of shared memory means fewer contending threads on each histogram bin and shorter access latency to the bins than a more straightforward approach using only a global histogram. Finally, each block accumulates its local histogram into the global histogram using atomic operations.

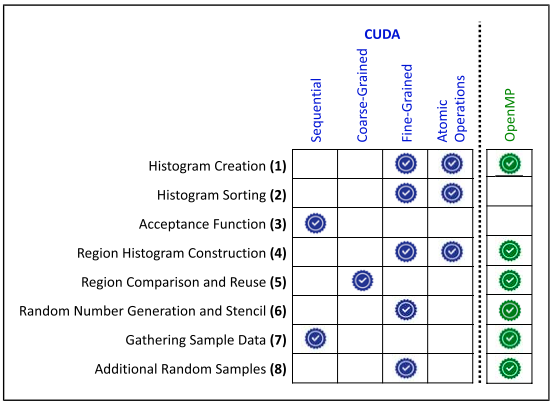


**Figure 12.** Optimization strategies implemented for each key step of our spatio-temporal hybrid data sampling method.

**Step 2. Data Histogram Sorting:** When sorting histograms, we choose to leverage existing histogram

sorting libraries. In our OpenMP approach, this step is done sequentially to leverage the built-in stable sort algorithm. In our CUDA approach, our histogram sort CUDA kernel uses a fine-grained level of parallelism with the standard Thrust libraries Radix sort algorithm, which has been shown to be "considerably faster than alternative comparison-based sorting algorithms such as Merge Sort" (Bell and Hoberock 2012).

**Step 3. Acceptance Function Development:** One step that we cannot parallelize is the acceptance function step, due to its iterative nature. We leave this step as sequential as each iteration of the development process relies on the previous iteration, as we detail in Step 3 in Section 3.1. Even though this step is sequential, this does not limit our throughput improvement much as this step represents a very small portion of both sampling methods, as it is only dependent on the sorted histogram.

**Step 4. Region Histogram Construction:** Before our hybrid data sampling method compares regions for reuse, it must first construct histograms for each region of the current time-step. In both of our parallelization approaches, we reuse the same process as the initial histogram creation from Step 1.

**Step 5. Region Comparison and Reuse:** The process of comparing regions for reuse consists of two sub-steps and depends on whether we compare regions with histograms or error. When comparing histograms, we calculate the histogram intersection of each current time-step region histogram with the corresponding region histogram of the previous time-step. If the intersection is above the intersection threshold, we mark the region for reuse. When comparing error, we calculate the RMSE between each current time-steps region and the corresponding region of the previous time-step. If the error is below the error threshold, we mark the region for reuse. Once each region is marked for reuse or not, we move to the utilization decision kernel, which analyzes the results and sets the necessary information for reuse regions so that they are not sampled in the sampling step.

This process is highly parallelizable, at a per-region level, as each region of subsequent time-steps can be analyzed independently. Thus, in our OpenMP approach, we divide the data regions among all threads. Each thread then determines whether to mark their regions for reuse or not and sets the necessary information for reuse regions. In our CUDA approach, we use a similar coarse-grained level of parallelism and assign each set of regions to a separate CUDA thread.

**Step 6.** Random Number Generation and Sample Stenciling The sampling process of each method is also highly parallelizable and consists of two sub-processes: generating random values and setting the sample stencil. In our OpenMP approach, we generate random numbers in parallel, varying the random seed per-thread id. When creating the stencil, we divide the data values equally among all threads and use the random values to determine the stencil value for each of their data values.

In our CUDA approach, we use a fine-grained level of parallelism and assign a single data value to each CUDA thread with 1000 threads to a single thread block. Following this, each thread generates a random number using the CURAND library (Nvidia, 2010). We use this library when generating random values as this library ensures each thread has a separate unique sequence of random values. Each thread uses its random value and data value to determine whether to keep the data value as a sample and, if so, sets the necessary value in the sampling stencil.

**Step 7.** Gathering Sample Data Using the resulting stencil, both methods must next gather the corresponding samples. In our OpenMP approach, the stencil is equally divided among all threads. Each thread uses the stencil to collect its respective sample data into private sample data arrays. Once the thread finishes processing the stencil, the resulting sample data arrays are concatenated to assemble the final sample data array. This process is kept sequential in our CUDA approach as a varying number of samples are collected for each time-step.

**Step 8. Additional Random Sampling:** As a final step, our hybrid method fills any empty space in the sampling budget with further random samples. In our OpenMP approach, the desired number of samples is split among all threads, and a similar process to the stenciling process occurs. Similarly, in our CUDA approach, we use a fine-grained level of parallelism and assign a single data value and a new random number to each CUDA thread with 1000 threads to a single thread block. Each thread calculates the new sample rate and uses the random number to determine whether to keep the data value as an additional sample. In both cases, all new samples must not occur within a reuse region.

## *Parallelization analysis*

Applying our OpenMP and CUDA approaches to Biswas et al.'s non-reuse method and our hybrid data sampling method, we move to evaluate the resulting throughput of each.

*Datasets and system.* When conducting our experiments, we use an NVIDIA V100 GPU (NVIDIA, 2020) on Clemson's Palmetto Cluster (Palmetto Cluster, Clemson University, 2021) along with two 20-core Intel Xeon 6148G CPUs with 372 GB of memory. When using OpenMP, version 3.1.4, we use 10 threads as we found this to be an optimal number of threads, as seen in Figure 13. We use CUDA version 10.0.130 and GCC

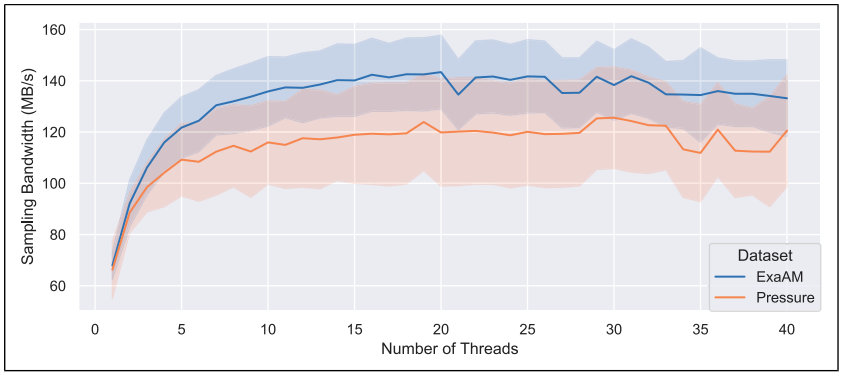


**Figure 13.** Average Bandwidth (MB/s) of OpenMP accelerated sampling process with varying number of threads.

version 7.1.0, as this is the maximum supported GCC version for our CUDA version.

In our experiments, we evaluate each method using three HPC data sets and three sampling ratios: 0.5%, 1%, and 2%. Table 1 describes the details of each data set and the input parameters we use for sampling (as chosen in Section 4).

### *Optimization effects on process throughput*

Using our three data sets, we evaluate the throughput of each of our eight sampling sub-process kernels. Figure 14 shows the average bandwidth of steps 1–8 for each dataset, sampling method, and optimization strategy, calculated as the average of all time-steps in each series, using a 2% sample ratio.

**Step 1. Data Histogram Creation Kernel:** With the larger datasets, the accelerated CPU implementation of Step 1 has a 3× to 8× improvement over the serial version. However, with smaller datasets, such as ExaAM (0.8 MB per time-step), we see lower throughput due to the OpenMP overhead not being outweighed by the data size. When using CUDA, we find a 700× to 900× improvement in bandwidth to a serial implementation, as shown in 19b. This is a significant speedup to even the previous CUDA approach, due to its optimized use of shared memory.

Specifically, due to our histogram kernel optimizations explained in Section 5.1, we are capable of reaching speeds of 200 GB/s. This throughput of histogramming is comparable to Tian et al., who experimented with a state-of-the-art design, reaching speeds of 252 GB/s with similarly sized data (Tian et al., 2021).

**Step 2. Data Histogram Sorting Kernel:** When evaluating our parallelized histogram sorting step, we find our OpenMP implementation yields a 1.2×, except for the Isabel dataset, in which it performs as fast as serial. However, the CUDA Thrust Radix sort does not achieve as high of a throughput as serial, due to the small number of bins needed to be sorted. With more bins to be sorted, we would better leverage the capabilities of this sorting algorithm.

**Step 3. Acceptance Function Development Kernel:** Step 3 is performed sequentially in all experiments, however, when this sequentially limited process is run on a single GPU thread, the overall throughput is lower than the CPU and accelerated CPU versions. While we could transfer the data back to the CPU to achieve better performance, we choose to keep the process on the GPU to avoid the memory transfer overhead, as it would reduce the resulting overall throughput improvement.

**Step 4. Region Histogram Construction Kernel:** When evaluating our parallelized region histogram creation step, we find that both OpenMP and CUDA achieve higher throughput than serial. When using OpenMP, we find a consistent 3× improvement, but as the OpenMP version must aggregate the results of all threads at the end of each histogram creation step, its throughput improvement is limited. In contrast, the CUDA implementation does not have this issue and achieves between 17.12× and 54.89× improvement over serial, with more improvement with smaller datasets.

**Step 5. Region Comparison and Reuse Kernel:** When assessing our parallelized region comparison and reuse step, we find unique results for both the Error-Reuse and Histogram-Reuse variants of the hybrid method. While in serial and with OpenMP, the Error-Reuse version leads to the lowest throughput, this is not the case on a GPU as CUDA threads efficiently handle the RMSE and utilization processes. For the error-based methods, we calculate the RMSE between two lists of sample values. We break this down into a thread per region, such that the RMSE for each region is calculated in parallel. However, that still results in a small calculation with very little data.

The opposite is true for the Histogram-Reuse version, which sees lower CUDA throughput due to the extra

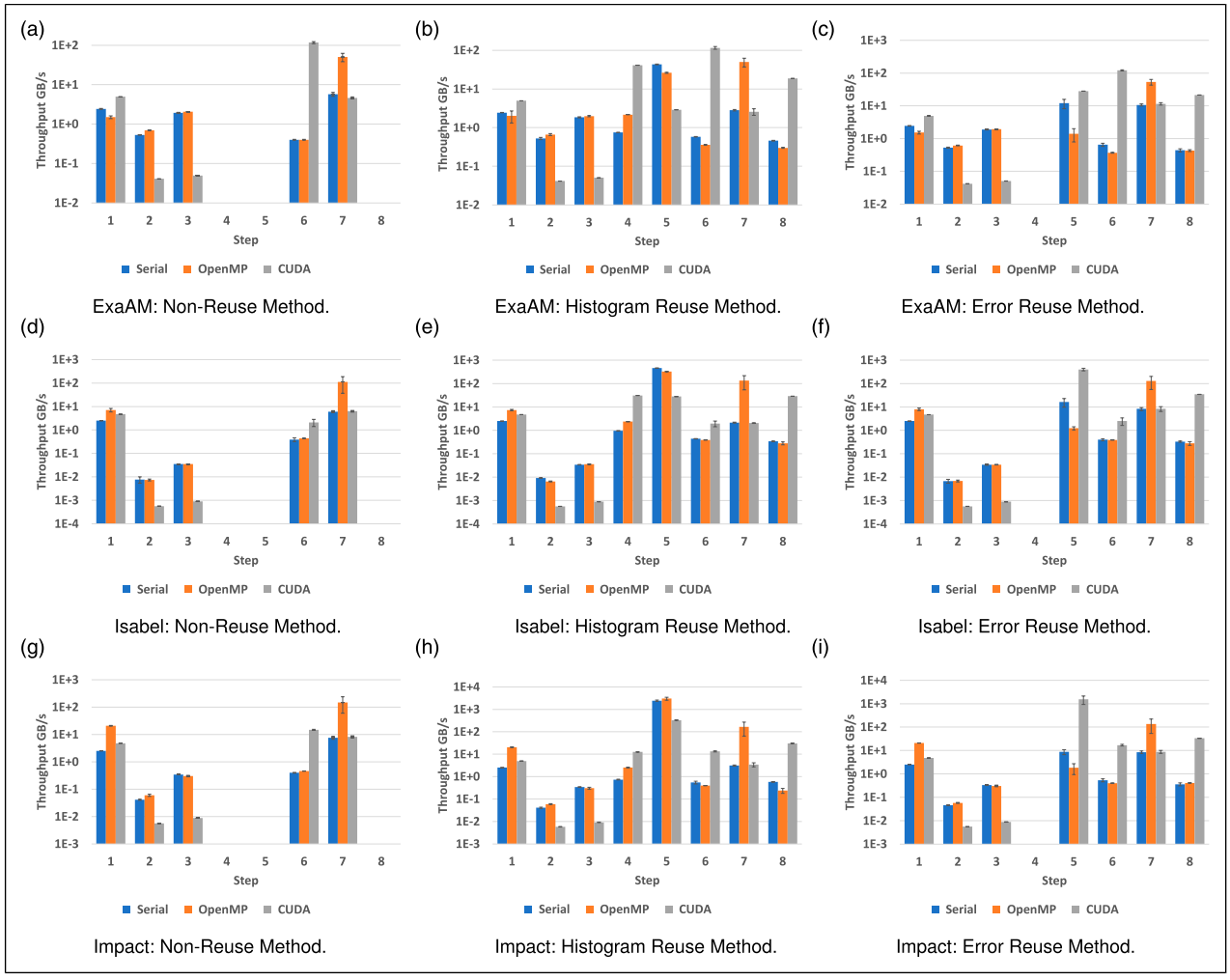


**Figure 14.** Average throughput per sub-process steps 1–8 per dataset and optimization technique (2% sample ratio).

necessary computations and data accesses needed to compare the histograms of regions. We see lower CUDA throughput due to the limitations this step has. In Step 5, we do not access all of the data in the region. With Histogram-Reuse, we only compare the histograms of each region to the corresponding region in the following time-step, yielding significantly fewer data elements to compare. For the histogram-based method, we only need to perform a single operation per region: calculating the intersection between the two histograms. This calculation uses a relatively small number of bins, with the Isabel and Impact datasets only using 27 bins each.

Both methods result in a single comparison operation being performed per region. Due to the low intensity of operations needed to compare a region, parallelizing the comparison process into a coarse-grained breakdown yields the most improvement possible. However, to calculate the histogram intersection and RMSE, we need to access two groups of data since we are comparing two lists of data from two time-steps, located in different memory locations. This is what significantly slows down the step overall.

**Step 6. Random Number Generation and Sample Stenciling Kernel:** Upon assessing our parallelized random number and stencil step, we find our CUDA implementation using CURAND drastically outperforms both serial and OpenMP. Specifically, we find our CUDA implementation achieves between 4.45× and 186.7× improvement over serial, with more improvement with smaller datasets.

**Step 7. Gathering Sample Data Kernel:** Upon assessing our parallelized sample-gathering step, we find our OpenMP implementation outperforms other implementations, with an average 24× improvement over serial. Due to the need for resizable arrays, we leave the sample-gathering process as a sequential process for CUDA. Step 5 involves arrays with an unknown, non-static size, creating the need for resizable arrays. In

CUDA, adding this functionality would introduce significant overhead, making it inefficient overall. Thus, we find similar performance levels for CUDA and serial. However, we use OpenMP to divide and conquer this step, leading to higher throughput.

**Step 8. Additional Random Sampling Kernel:** When evaluating our parallelized additional random sampling step, we find our CUDA implementation outperforms both OpenMP and serial. Specifically, we find our CUDA implementation achieves between 40.74× and 105.6× improvement over serial (Figure 15).

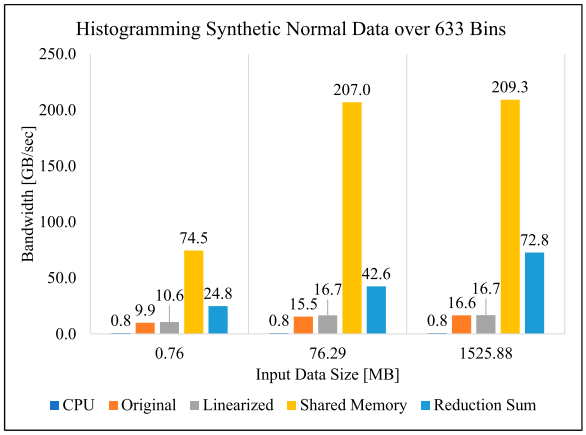


**Figure 15.** Histogram bandwidth improvements found with varying GPU kernel implementation.

### *Parallel performance observations*

From our evaluation above, we find that using CUDA or OpenMP drastically improves many steps in the sampling process of both importance-based methods. However, it is also critical to understand how each parallelization strategy affects each sampling method's overall performance. Figure 16 shows the average overall bandwidth for each dataset and parallelization technique, while Figures 17 and 18 show each parallelization strategies performance improvement over the serial implementation.

From these figures, we find that using CUDA improves the sampling processes' overall throughput tremendously. When looking at Figure 16 we find CUDA throughput range from 1211 to 3993 MB/s for all data sets, while running these methods in serial only achieves between 63 and 196 MB/s. Additionally, previous implementations of Biswas et al.'s importance-based sampling method were only capable of achieving throughput on the order of tens of megabytes (Biswas et al., 2018, 2020b).

First, looking at the specific improvements with the importance-based method, OpenMP reaches a 1.17 to 1.64× improvement. The CUDA implementation reaches between 15.75 to 25.26×, as even though some steps are slower on the GPU, they are outweighed by the improvements in the target slowdown steps (Steps 4, 5, 6 and 8).

With the Histogram-Reuse method, the OpenMP version only reaches a sampling throughput comparable to (1.13×) or lower than (0.76×) the serial version. This is due to Steps 5, 6, and 8, as discussed in the previous section. However, the CUDA version improves Steps 4, 6, and 8, yielding an overall 8.40× to 24.65× speedup over serial.

Likewise, the OpenMP version of error-based reuse suffers with Steps 5 and 6. However, since it does not have to compute histograms per method (Step 4), it is able to achieve an overall improvement between 1.17 and 1.93× when using 10 threads. As the number of samples increases, more computations are needed to calculate the error between each sample and the coordinating new value at the next time-step. Since this task will take a long time in serial, we are able to better leverage OpenMP, yielding greater improvements with higher sample ratios. The CUDA version targets Steps 5, 6, and 8, yielding a 175×, 31×, 92.69× improvement over the serial implementations, respectively (with the Impact dataset). These speedups allow the GPU implementation to reach an overall improvement greater than OpenMP, reaching a 22.54× to 52.70× speedup over serial.

Our results show that the OpenMP implementation does not achieve as high throughput as the CUDA version. Many of our method's steps depend on the results from previous steps. With this dependence, it is critical that when we can parallelize our method that we do so to the greatest extent possible. Using 1024 CUDA threads to parallelize the parallelizable steps of our method is a more effective solution than using the 10 OpenMP threads we found to be optimal for sampling in Section 5.2. We attribute this difference to the low amount of thread divergence that occurs within these parallelizable steps, making CUDA the optimal choice. Overall, while the OpenMP implementation achieves higher throughput in some steps, our CUDA version achieves the greatest improvement over the serial version of each method.

## Changing the core sampling algorithm

The novelty of our spatio-temporal hybrid sampling method is that it utilizes samples from a previous time-step to capitalize on similarities between neighboring time-steps. The method by which we gather these samples, however, is independent of our algorithm. We refer to this base method as the "Core Sampling Algorithm" (CSA). Our method uses the CSA when gathering samples for the first time-step and in regions that do not reuse previous samples.

We have the ability to switch the CSA to any existing sampling algorithm, which allows our method to maintain relevance as the data reduction field grows. To show the variance and usefulness of this ability, we use our sampling

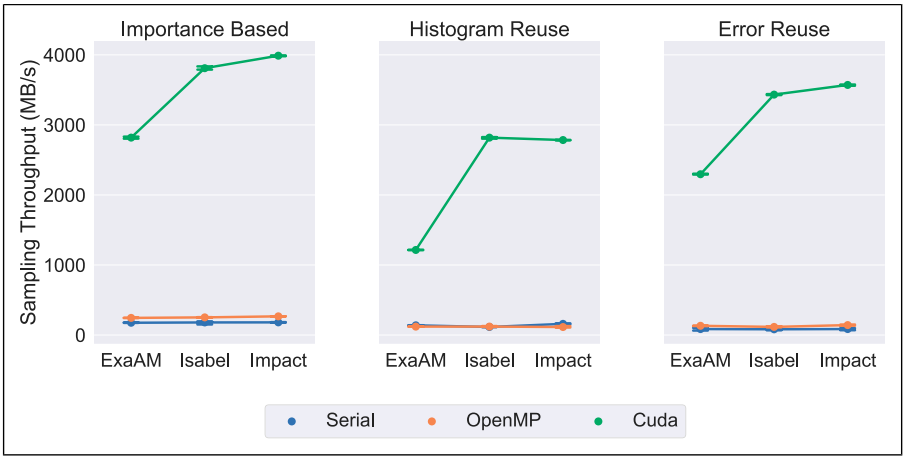


**Figure 16.** Average Total Sampling Process Bandwidth for each dataset and parallelization technique.

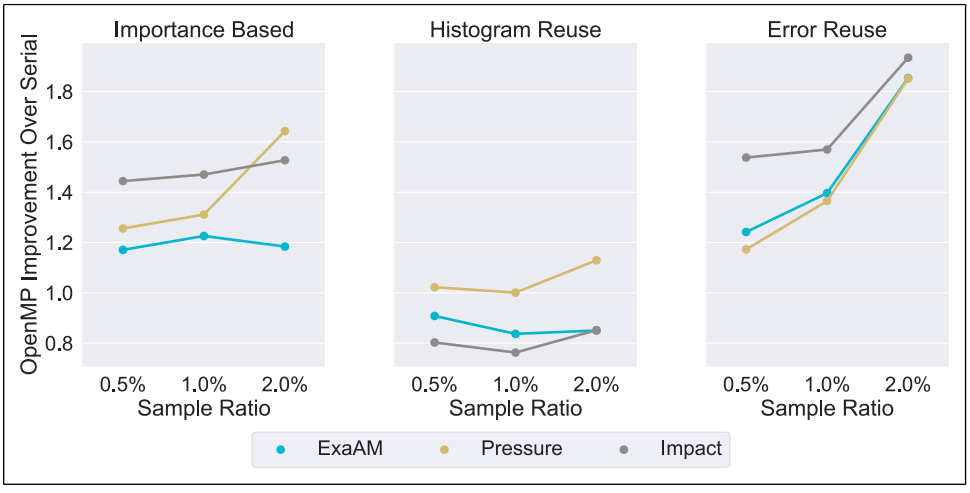


**Figure 17.** OpenMP performance improvement over serial.

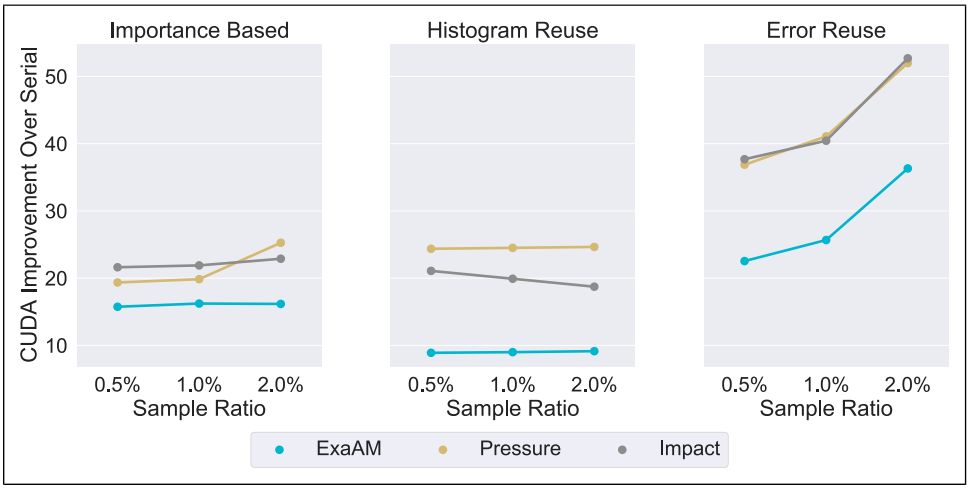


**Figure 18.** Cuda performance improvement over serial.

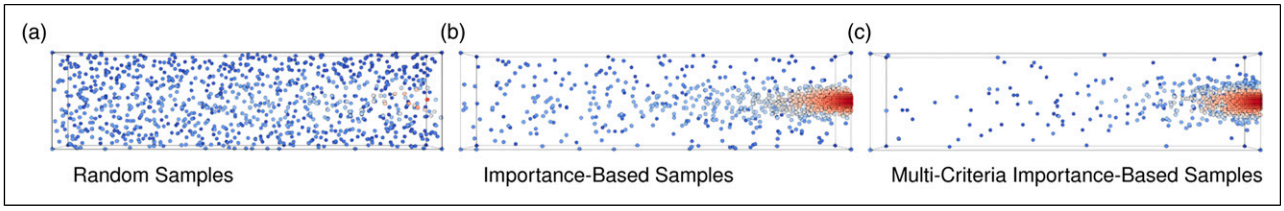


**Figure 19.** Samples gathered by different Core Sampling Algorithms, Using ExaAM time-step 64 and a sample rate of 0.5%.

algorithm configurations with three different cores: Simple Random, Importance-Based (Biswas et al., 2018), and Multi-Criteria Importance-Based Sampling (Biswas et al., 2020b). We describe Importance-Based sampling in Section and Biswas et al. describe a multi-criteria importance-based sampling algorithm that utilizes both the histogram of all data values and the local gradient that gives both areas of rare data values and abrupt change a higher priority of being sampled (Biswas et al., 2020b).

To understand the effect the CSA has on our algorithm, we first show how they distribute their samples, as shown in Figure 19. Simple random sampling produces a uniformly distributed sample set. The importance-based method biases rare data values, so the samples are clustered around the ROI. The multi-criteria importance-based sampling method has the most samples in the ROI, as it gives a bias to rare data values and data values with a high change in gradient.

The location of the samples heavily affects post-reconstruction quality. Since the data values within the ROI have high variance, taking more samples from this area will better maintain its quality. However, areas of low variance require fewer samples to maintain quality. Thus, the multi-criteria importance-based sampling method yields the highest post-reconstruction quality overall, as it takes the majority of its samples from the high-variance areas.

Regardless of CSA, our hybrid sampling method can be applied to increase post-reconstruction quality, as shown in Figure 20(a). Figure 20(b) shows that there is a trade-off between quality and bandwidth. Overall, the more sophisticated the CSA, the higher the quality, but the lower the bandwidth.

Unlike the other CSAs used, the multi-criteria importance-based sampling control method does not have the highest bandwidth. This method on its own takes the longest to sample, as it takes both value and gradient into consideration. When the histogram-based and error-based reuse methods determine a region that will reuse previous samples, we do not use this computationally heavy algorithm for that region, which leads to faster sampling overall.

Through this study of core sampling algorithms, we have shown that our spatio-temporal hybrid sampling method is able to be used with any existing method of gathering samples. The implication of this is that our approach stays relevant as the field of data sampling grows. Thus, the performance of our method is permanently influenced by the newest and upcoming sampling algorithms. The minimum quality our method achieves is the maximum of the CSA. Likewise, the throughput of our method is limited by the speed of the CSA. Thus, as faster and more accurate sampling techniques emerge, our method can continue to be used to improve quality.

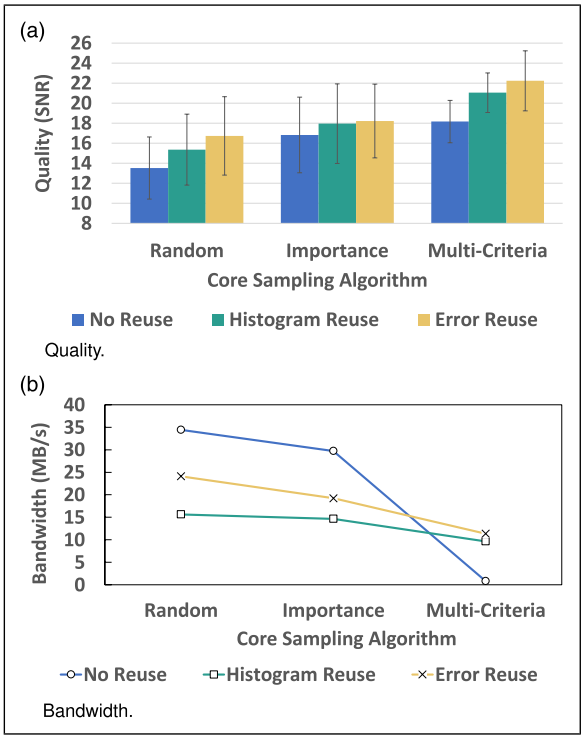


**Figure 20.** Quality and Bandwidth comparison with 1% sample rate of ExaAM data set with varying core sampling algorithm.

## Comparison to lossy compression

### *Quality*

Our spatio-temporal method utilizes sampling methods; therefore, we use sample ratio to refer to the amount of data reduction, rather than compression ratio. This allows us to achieve an exact overall reduction ratio and meet user constraints, which modern compressors are unable to do without trial-and-error or outside tools like FRaZ (Underwood et al., 2020). In order to ensure a fair

**Table 2.** Data quality with varying data reduction methods at compression ratios 100:1 (equivalent sample ratio: 0.5%).

| Dataset | Method | Reduction ratio | Error bound | PSNR | Write time (s) | Read time (s) |
|---|---|---|---|---|---|---|
| ExaAM | No reduction | 1:1 | — | Inf | 0.097450 | 0.006124 |
| — | Histogram reuse | 100:1 | — | 35.62 | 0.013151 | 0.000400 |
| — | cuSZ | 22:1 | Rel. $1 \times 10^{2}$ | 14.93 | 0.000706 | 0.000197 |
| Isabel | No reduction | 1:1 | — | Inf | 12.265770 | 0.180772 |
| — | Histogram reuse | 100:1 | — | 39.12 | 0.696782 | 0.064536 |
| — | cuSZ | 22:1 | Rel. $1 \times 10^{1}$ | 61.20 | 0.002616 | 0.002107 |
| Impact | No reduction | 1:1 | — | Inf | 6.534960 | 0.190949 |
| — | Histogram reuse | 100:1 | — | 15.64 | 0.175427 | 0.006367 |
| — | cuSZ | 29:1 | Abs. $1 \times 10^{-1}$ | 37.00 | 0.004804 | 0.001965 |

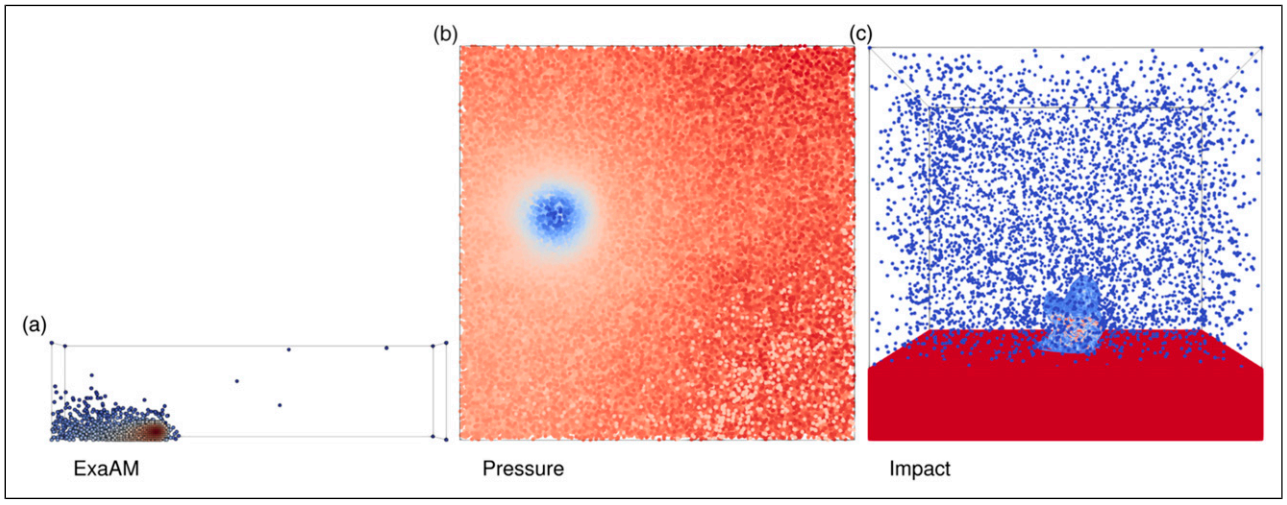


**Figure 21.** Samples gathered by Our Sampling Algorithm, using varying datasets and a sample rate of 0.5%.

comparison between compressors, we incorporate an overhead factor that takes into account the additional storage requirements of our sampling-based approach. We convert a sample ratio (SR) to a compression ratio as 1/(*SR**2). This conversion factor reflects the fact that our approach requires the storage of both the value and index of each sample, hence the need to double the SR value.

Table 2 details the quality of each reconstructed dataset interpolated from a set of samples at a rate of 0.5%, yielding an overall reduction ratio of 100 : 1. We compare the resulting interpolated quality with the decompressed quality resulting from the GPU-optimized lossy compressor cuSZ (Tian et al., 2020). Input parameters for cuSZ were chosen to meet the reduction ratio of 100 : 1 as closely as possible. Input parameters for cuSZ were chosen to meet the reduction ratio of 100 : 1 as closely as possible. However, due to the compressibility limitations of the data, such compression ratios are not feasible. We detail the maximum achievable compression ratios using this compressor, and the resulting PSNR of the decompressed values. The Error Bound column in Table 2 lists the error bounding mode and tolerance used per dataset. Any error bound larger than listed introduces an unfeasible amount of error into the dataset. For example, we compress the ExaAM dataset with a relative error bound of 100 because anything beyond that returns the same compression ratio of 22:1.

For the ExaAM dataset, we achieve a 2.4× improvement over what cuSZ achieves in terms of quality. Our method achieves higher quality because it excels with datasets with a concentrated region of interest that moves in spatial location quickly over time. With the Isabel and Impact datasets, however, cuSZ yields a higher PSNR. Lossy compression is able to preserve the data better because it was not capable of meeting the target reduction ratio. Since compression retained more of the true values when decompressed, it yields a higher quality.

Moreover, while data sampling aims to preserve the region of interest, this is not necessarily reflected in the

**Table 2.** Data quality with varying data reduction methods at compression ratios 100:1 (equivalent sample ratio: 0.5%).

| Dataset | Method | Reduction ratio | Error bound | PSNR | Write time (s) | Read time (s) |
|---|---|---|---|---|---|---|
| ExaAM | No reduction | 1:1 | — | Inf | 0.097450 | 0.006124 |
| — | Histogram reuse | 100:1 | — | 35.62 | 0.013151 | 0.000400 |
| — | cuSZ | 22:1 | Rel. $1 \times 10^{2}$ | 14.93 | 0.000706 | 0.000197 |
| Isabel | No reduction | 1:1 | — | Inf | 12.265770 | 0.180772 |
| — | Histogram reuse | 100:1 | — | 39.12 | 0.696782 | 0.064536 |
| — | cuSZ | 22:1 | Rel. $1 \times 10^{1}$ | 61.20 | 0.002616 | 0.002107 |
| Impact | No reduction | 1:1 | — | Inf | 6.534960 | 0.190949 |
| — | Histogram reuse | 100:1 | — | 15.64 | 0.175427 | 0.006367 |
| — | cuSZ | 29:1 | Abs. $1 \times 10^{-1}$ | 37.00 | 0.004804 | 0.001965 |

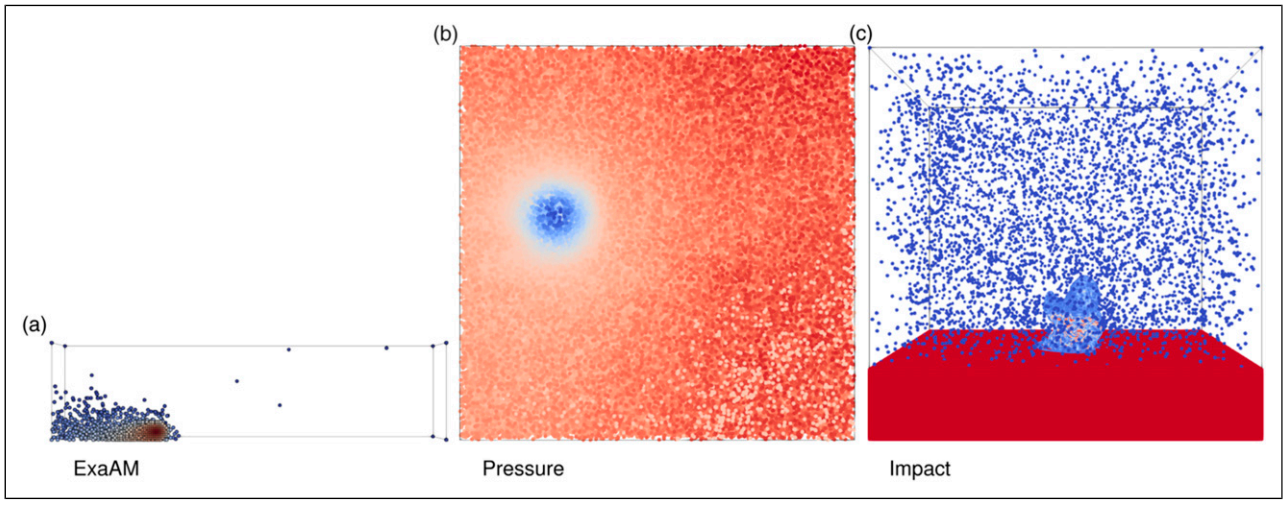


**Figure 21.** Samples gathered by Our Sampling Algorithm, using varying datasets and a sample rate of 0.5%.

comparison between compressors, we incorporate an overhead factor that takes into account the additional storage requirements of our sampling-based approach. We convert a sample ratio (SR) to a compression ratio as 1/(*SR**2). This conversion factor reflects the fact that our approach requires the storage of both the value and index of each sample, hence the need to double the SR value.

Table 2 details the quality of each reconstructed dataset interpolated from a set of samples at a rate of 0.5%, yielding an overall reduction ratio of 100 : 1. We compare the resulting interpolated quality with the decompressed quality resulting from the GPU-optimized lossy compressor cuSZ (Tian et al., 2020). Input parameters for cuSZ were chosen to meet the reduction ratio of 100 : 1 as closely as possible. Input parameters for cuSZ were chosen to meet the reduction ratio of 100 : 1 as closely as possible. However, due to the compressibility limitations of the data, such compression ratios are not feasible. We detail the maximum achievable compression ratios using this compressor, and the resulting PSNR of the decompressed values. The Error Bound column in Table 2 lists the error bounding mode and tolerance used per dataset. Any error bound larger than listed introduces an unfeasible amount of error into the dataset. For example, we compress the ExaAM dataset with a relative error bound of 100 because anything beyond that returns the same compression ratio of 22:1.

For the ExaAM dataset, we achieve a 2.4× improvement over what cuSZ achieves in terms of quality. Our method achieves higher quality because it excels with datasets with a concentrated region of interest that moves in spatial location quickly over time. With the Isabel and Impact datasets, however, cuSZ yields a higher PSNR. Lossy compression is able to preserve the data better because it was not capable of meeting the target reduction ratio. Since compression retained more of the true values when decompressed, it yields a higher quality.

Moreover, while data sampling aims to preserve the region of interest, this is not necessarily reflected in the

quantifiable PSNR of the overall dataset. As detailed by Grosset et al. (Grosset et al., 2020), sampling alone is unable to effectively preserve the details of this dataset, and compression is recommended oversampling when trying to achieve high quality for highly detailed datasets. However, achieving a high-quality post-reconstruction is not the sole goal of data sampling.

Figure 21 shows only the samples gathered with our method for each dataset. This shows our purpose of taking samples is not necessary to reproduce the exact input data, but rather to obtain a summary of the data. Once the end user is able to quickly analyze the summary, the samples have the capability of being interpolated to reflect the full-resolution data. However, if it is not necessary, we do not have to spend the temporal overhead in reconstructing the data. For example, with the Impact dataset, the domain scientists are explicitly interested in the asteroid during the entry into the water (Patchett et al., 2016; Ahrens et al., 2014). Likewise, the ExaAM and Hurricane Isabel datasets contain features (hottest portion and the hurricane eye, respectively) that are the crux of what needs to be studied. Patchett et al. (Patchett and Ahrens 2018) found that while data sampling was too coarse to capture fine details of the data, it could be applied to summarize the feature at a coarse-level detail until a timestep was found that gained enough interest to warrant a full-resolution version. This is a direct contrast to compressors, which need to decompress the data before the data can provide any useful analysis.

### *Data movement*

When comparing the read and write times of our sampling process, we do not include the reconstruction step. As part of the in situ visualization process, the time to store, load, and render data offsets the amount of information gained by studying the data. The set of samples gathered yields a summary of the data, rather than the memory-intensive, high-resolution dataset. While higher-resolution data are necessary for presentations and visualizations, it is important for end users to be able to quickly determine whether a feature is of scientific interest by working with smaller, more quickly stored, loaded, and visualized intermediate data (Patchett and Ahrens 2018). After the low sampling rates are used to quickly display key features across timesteps, the most important visuals can be downloaded at higher resolutions.

Table 2 details the time to perform data movement operations. We have broken these operations down into the time to perform a read and a write operation with the data. The Write Time column records the number of seconds necessary to write the data to a binary file. In the no-reduction case, we have to write all of the data values to the file. However, with our Histogram-Reuse method, we only store 0.5% of the total data points. This allows for faster write times, as less data needs to be moved. We record this time as the number of seconds needed to reduce the data, then write the reduced data to a binary file. Our sampling method yields an average 20.75× speedup in data movement over the no-reduction times, with increasing improvement in larger datasets.

We also compare the time to compress and move the reduced data with the GPU lossy compressor cuSZ (Tian et al., 2020). Using cuSZ achieves an average speedup of 1546.6× over using no reduction. The drawback of using data sampling instead of compression is that both the values of samples and the spatial location of each sample needs to be saved. Since we have to write two different lists of data, our method sees longer write times than lossy compression that only needs to write its compressed data to one file.

Second, we analyze the data movement of reading the data back in. We record the Read Time column as the time it takes to read in the reduced data to be analyzed. Since our sampling method needs to read in a smaller file, we see an average improvement in read times of 48.1× over no reduction. Moving the data with cuSZ however, yields an average improvement of 71.4×. Just as with the disadvantage of sampling and writing to files, reading data samples from files has a disadvantage because we have to gather the sample data from two files. Reading and writing from a single file is generally faster than working with two smaller files due to the overhead of managing multiple files. Each file requires the system to open a file handle, read the contents, and then close the handle, which can slow down the operation. Despite the fact that the contents of the two smaller files may be smaller in total size, the overhead involved in managing multiple files can offset any potential advantages in file size. We need to take the time to read in data for the values and separate information on the sample locations before we have enough useful information to analyze the data.

In conclusion, our data sampling method is capable of meeting a user-specified target reduction ratio and effectively speeds up data movement on HPC systems from 2.8× to 37.25×. In comparison to a leading lossy compressor, data sampling has the advantage of meeting the user's storage constraint requirements. However, the need to store both sample values and locations yields less improvement in data movement speed than achieved by a GPU-optimized lossy compressor.

## Conclusion

The I/O bottleneck found on HPC systems has made intelligent data reduction schemes necessary. These methods need to achieve high post-reconstruction quality while also being as fast as possible to meet HPC workflow demands.

In this article, we enhance our existing sampling algorithm by introducing multiple ways to improve quality and

throughput. By assisting users in setting the sampling constraints, they have more flexibility while still being confident that they will achieve an optimal sample set that has been shown to improve quality and/or throughput. By using CUDA, we achieve average throughput improvements between 9.8× to 31.5× while maintaining the same level of post-reconstruction data quality as the sequential counterparts. Last, we show that the core sampling algorithm can be interchanged, allowing the user to achieve a higher quality or higher throughput, depending on their dataset and situation. Overall, we have introduced and analyzed several ways to improve the overall performance of our spatio-temporal hybrid sampling algorithm.

In the future, we aim further to improve the post-reconstruction quality that our sampling method can achieve. Specifically, we will evaluate how the information between multiple variables of the same data series can be leveraged to yield a more cohesive group of samples. As our methods' output is a highly compressible list of floats, instead of competing with compression methods, we propose that our sampling method be used as a pre-processing step for compression. This combination will enable us to achieve even higher accuracy while still maintaining high compression ratios.

## Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197 and SHF-1943114. This work was funded by Los Alamos National Laboratory under Information Science and Technology Student Fellowship program. The publication has been assigned the LANL identifier LA-UR-20-27478. We would like to thank our ECP collaborators on the ExaAM project. Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster. The ExaAM research is supported by the 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.

## Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Los Alamos National Laboratory and National Science Foundation (SHF-1910197; SHF-1943114).

## ORCID iDs

Megan Hickman Fulp https://orcid.org/0000-0001-5630-049X
Cooper Sanders https://orcid.org/0009-0005-4879-3622
Jon C. Calhoun https://orcid.org/0000-0001-7191-4422

## Notes

1. https://www.openmp.org/
2. https://developer.nvidia.com/cuda-toolkit

## References

Ahrens J, Jourdain S, OLeary P, et al. (2014) An image-based approach to extreme scale in situ visualization and analysis. *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 424–434.

Akiba H, Fout N and Ma KL (2006) Simultaneous classification of time-varying volume data based on the time histogram. *EuroVis* 6: 1–8.

Belak J, Turner J and Team ET (2019) Exaam: Additive manufacturing process modeling at the fidelity of the microstructure. *APS* 2019.

Bell N and Hoberock J (2012) Thrust: A productivity-oriented library for cuda. *GPU computing gems Jade edition*. Elsevier, pp. 359–371.

Biswas A, Dutta S, Lawrence E, et al. (2021) Probabilistic data-driven sampling via multi-criteria importance analysis. *IEEE Transactions on Visualization and Computer Graphics* 27: 4439–4454.

Biswas A, Dutta S, Lawrence E, et al. (2021) Probabilistic data-driven sampling via multi-criteria importance analysis. *IEEE Transactions on Visualization and Computer Graphics* 27: 4439–4454. DOI: 10.1109/TVCG.2020.3006426.

Biswas A, Dutta S, Pulido J, et al. (2018) In situ data-driven adaptive sampling for large-scale simulation data summarization. *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization - ISAV '18*. ACM Press, pp. 13–18. DOI: 10.1145/3281464.3281467.

Cappello F, Di S, Li S, et al. (2019) Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications* 33(6): 1201–1220. DOI: 10.1177/1094342019853336.

Childs H (2015) Data exploration at the exascale. *Supercomputing Frontiers and Innovations* 2(3): 5–13.

Delaunay B, et al. (1934) Sur la sphere vide, Izv. Akad. *Nauk SSSR, Otdelenie Matematicheskikh i Estestven- nykh Nauk*, 7: 793–800.

Doane DP (1976) Aesthetic frequency classifications. *The American Statistician* 30(4): 181–183.

Dutta S, Chen C, Heinlein G, et al. (2017) In situ distribution guided analysis and visualization of transonic jet engine simulations. *IEEE Transactions on Visualization and Computer Graphics* 23(1): 811–820.

Fogal T, Schiewe A and Krüger J (2013) An analysis of scalable gpu-based ray-guided volume rendering. In: *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*. IEEE, pp. 43–51.

Fulp MH, Biswas A and Calhoun JC (2020) Combining spatial and temporal properties for improvements in data reduction. *2020 IEEE International Conference on Big Data (Big Data)*, pp. 2654–2663. DOI: 10.1109/BigData50022.2020.9378457.

Grosset P, Biwer CM, Pulido J, et al. (2020) Foresight: analysis that matters for data reduction. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–15.

Gutiérrez PD, Lastra M, Benítez JM, et al. (2017) Smote-gpu: Big data preprocessing on commodity hardware for imbalanced classification. *Progress in Artificial Intelligence* 6(4): 347–354.

Habib S, Morozov V, Frontiere N, et al. (2013) Hacc: extreme scaling and performance across diverse architectures. *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–10.

Hurricane ISABEL Simulation Data (2019) http://vis.computer.org/vis2004contest/data.html.

Hyndman RJ (1995) *The Problem with Sturges' Rule for Constructing Histograms*. Monash University. 1–2.

Jibben Z (2020) truchas-pbf. https://gitlab.com/truchas/truchas-pbf/

NVIDIA (2020) *GPU*. https://www.nvidia.com/en-us/data-center/v100/ (accessed 23 April 2021).

Nouanesengsy B, Woodring J, Patchett J, et al. (2014) Adr visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In: *2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)*. pp. 43–50.

Nvidia CUDA (2010) Curand library.

Palmetto Cluster, Clemson University (2021) Online. http://citi.clemson.edu/palmetto/ (accessed 24 June 2021).

Patchett J and Ahrens J (2018) Optimizing scientist time through in situ visualization and analysis. *IEEE Computer Graphics and Applications* 38(1): 119–127.

Patchett J and Gisler G (2017) *Deep Water Impact Ensemble Data Set*. Los Alamos, NM: Los Alamos National Laboratory. https://datascience.dsscale.org/wp-content/uploads/2017/03/DeepWaterImpactEnsembleDataSet.pdf

Patchett J, Samsel F, Tsai KC, et al. (2016) *Visualization and Analysis of Threats from Asteroid Ocean Impacts*. Los Alamos, NM: Los Alamos National Laboratory.

Scott DW (2009) Sturges' rule. *WIREs Computational Statistics* 1(3): 303–306.

Scott DW (2010) Scott's rule. *Wiley Interdisciplinary Reviews: Computational Statistics* 2(4): 497–502.

Strand G (2015) The cesm workflow re-engineering project. *AGUFM* 2015.

Tian J, Di S, Zhao K, et al. (2020) Cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data. *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pp. 3–15.

Tian J, Rivera C, Di S, et al. (2021) Revisiting huffman coding: Toward extreme performance on modern gpu architectures. *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, pp. 881–891.

Tikhonova A, Correa CD and Ma KL (2010) Explorable images for visualizing volume data. *PacificVis* 10: 177–184.

Underwood R, Di S, Calhoun JC, et al. (2020) *Fraz: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data*.

Wang R, Wang R, Zhou K, et al. (2009) An efficient gpu-based approach for interactive global illumination. *ACM SIGGRAPH 2009 papers*, pp. 1–8.

Wei TH, Dutta S and Shen HW (2018) Information guided data sampling and recovery using bitmap indexing. In: *2018 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, pp. 56–65.

Woodring J, Ahrens J, Figg J, et al. (2011) In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. *Computer Graphics Forum* 30(3): 1151–1160. DOI: 10.1111/j.1467-8659.2011.01964.x.

## Author biographies

*Megan Hickman Fulp* is a Lecturer of Computer Science at Coastal Carolina University, holding a Bachelor of Science degree in Computer Science from the same institution (2019). She obtained a Master of Science degree in Computer Engineering from Clemson University (2021), where her research focused on big data reduction for High-Performance Computing systems. Her current research interests lie in effectively teaching computer programming and algorithmic thinking.

*Dakota Fulp* is an associate professor in the Computer Technology department at Horry-Georgetown Technical College. He received a B.S. degree in Computer Science and Applied Physics from Coastal Carolina University in 2015 and 2017, respectively. Upon graduating, he researched HPC resiliency with Los Alamos National Laboratory from 2017 to 2020. Following this, he received an M.S. degree in Computer Engineering from Clemson University, during which his research focused on data resiliency and lossy data compression for HPC systems. His research interests lie in fault tolerance and resilience for HPC systems, data compression, and effective teaching methodologies.

*Changfeng Zou* is an undergraduate student studying Computer Engineering at Clemson University. He is driven by a passion for creating lasting solutions and maximizing efficiency in computing. He holds the position of president at IEEE-HKN Zeta Iota Chapter.

*Cooper Sanders* is an undergraduate computer engineering student at Clemson University with interests in parallel computing and numerical algorithms, and a talented young

software engineer. He has worked for several years as an undergraduate researcher at Clemson University, as well as interning with Los Alamos National Lab for a summer. On top of engineering, he is also an accomplished jazz trombonist.

*Ayan Biswas* is a scientist at Los Alamos National Laboratory. He is currently leading LANL's sampling-based data reduction efforts under the Exascale Computing Project (ECP) for ALPINE. He was also the co-PI for a recently concluded LANL funded LDRD-DR project (total funding $5M over 3 years) that focused on in situ data modeling and statistical inference. He is an expert of visualization and big data analysis with also considerable experience in information theoretic approaches, vector field exploration, uncertainty visualization, and machine learning for large-scale scientific data sets.

*Melissa C. Smith* is a professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University. She received her B.S. (1993) and M.S. (1994) degrees in Electrical Engineering from Florida State University and a Ph.D. (2003) in Electrical Engineering from the University of Tennessee. She has over 25 years of experience developing and implementing scientific workloads and machine-learning applications across multiple domains, including 12 years as a research associate at Oak Ridge National Laboratory. Her current research focuses on the performance analysis and optimization of emerging heterogeneous computing architectures (GPGPU- and FPGA-based systems) for various application domains, including machine learning, high-performance or real-time embedded applications, and image processing. She is a Senior Member of the IEEE.

*Jon C. Calhoun* is an associate professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University. He received a B.S. in Computer Science from Arkansas State University in 2012, a B.S. in Mathematics from Arkansas State University in 2012, and a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2017. He was awarded a prestigious NSF CAREER award in 2020. His research interests lie in fault tolerance and resilience for high-performance computing (HPC) systems and applications, lossy and lossless data compression, scalable numerical algorithms, power-aware computing, and approximate computing. He is a Senior Member of the IEEE.

quantifiable PSNR of the overall dataset. As detailed by Grosset et al. (Grosset et al., 2020), sampling alone is unable to effectively preserve the details of this dataset, and compression is recommended oversampling when trying to achieve high quality for highly detailed datasets. However, achieving a high-quality post-reconstruction is not the sole goal of data sampling.

Figure 21 shows only the samples gathered with our method for each dataset. This shows our purpose of taking samples is not necessary to reproduce the exact input data, but rather to obtain a summary of the data. Once the end user is able to quickly analyze the summary, the samples have the capability of being interpolated to reflect the full-resolution data. However, if it is not necessary, we do not have to spend the temporal overhead in reconstructing the data. For example, with the Impact dataset, the domain scientists are explicitly interested in the asteroid during the entry into the water (Patchett et al., 2016; Ahrens et al., 2014). Likewise, the ExaAM and Hurricane Isabel datasets contain features (hottest portion and the hurricane eye, respectively) that are the crux of what needs to be studied. Patchett et al. (Patchett and Ahrens 2018) found that while data sampling was too coarse to capture fine details of the data, it could be applied to summarize the feature at a coarse-level detail until a timestep was found that gained enough interest to warrant a full-resolution version. This is a direct contrast to compressors, which need to decompress the data before the data can provide any useful analysis.

### *Data movement*

When comparing the read and write times of our sampling process, we do not include the reconstruction step. As part of the in situ visualization process, the time to store, load, and render data offsets the amount of information gained by studying the data. The set of samples gathered yields a summary of the data, rather than the memory-intensive, high-resolution dataset. While higher-resolution data are necessary for presentations and visualizations, it is important for end users to be able to quickly determine whether a feature is of scientific interest by working with smaller, more quickly stored, loaded, and visualized intermediate data (Patchett and Ahrens 2018). After the low sampling rates are used to quickly display key features across timesteps, the most important visuals can be downloaded at higher resolutions.

Table 2 details the time to perform data movement operations. We have broken these operations down into the time to perform a read and a write operation with the data. The Write Time column records the number of seconds necessary to write the data to a binary file. In the no-reduction case, we have to write all of the data values to the file. However, with our Histogram-Reuse method, we only store 0.5% of the total data points. This allows for faster write times, as less data needs to be moved. We record this time as the number of seconds needed to reduce the data, then write the reduced data to a binary file. Our sampling method yields an average 20.75× speedup in data movement over the no-reduction times, with increasing improvement in larger datasets.

We also compare the time to compress and move the reduced data with the GPU lossy compressor cuSZ (Tian et al., 2020). Using cuSZ achieves an average speedup of 1546.6× over using no reduction. The drawback of using data sampling instead of compression is that both the values of samples and the spatial location of each sample needs to be saved. Since we have to write two different lists of data, our method sees longer write times than lossy compression that only needs to write its compressed data to one file.

Second, we analyze the data movement of reading the data back in. We record the Read Time column as the time it takes to read in the reduced data to be analyzed. Since our sampling method needs to read in a smaller file, we see an average improvement in read times of 48.1× over no reduction. Moving the data with cuSZ however, yields an average improvement of 71.4×. Just as with the disadvantage of sampling and writing to files, reading data samples from files has a disadvantage because we have to gather the sample data from two files. Reading and writing from a single file is generally faster than working with two smaller files due to the overhead of managing multiple files. Each file requires the system to open a file handle, read the contents, and then close the handle, which can slow down the operation. Despite the fact that the contents of the two smaller files may be smaller in total size, the overhead involved in managing multiple files can offset any potential advantages in file size. We need to take the time to read in data for the values and separate information on the sample locations before we have enough useful information to analyze the data.

In conclusion, our data sampling method is capable of meeting a user-specified target reduction ratio and effectively speeds up data movement on HPC systems from 2.8× to 37.25×. In comparison to a leading lossy compressor, data sampling has the advantage of meeting the user's storage constraint requirements. However, the need to store both sample values and locations yields less improvement in data movement speed than achieved by a GPU-optimized lossy compressor.

## Conclusion

The I/O bottleneck found on HPC systems has made intelligent data reduction schemes necessary. These methods need to achieve high post-reconstruction quality while also being as fast as possible to meet HPC workflow demands.

In this article, we enhance our existing sampling algorithm by introducing multiple ways to improve quality and

throughput. By assisting users in setting the sampling constraints, they have more flexibility while still being confident that they will achieve an optimal sample set that has been shown to improve quality and/or throughput. By using CUDA, we achieve average throughput improvements between 9.8× to 31.5× while maintaining the same level of post-reconstruction data quality as the sequential counterparts. Last, we show that the core sampling algorithm can be interchanged, allowing the user to achieve a higher quality or higher throughput, depending on their dataset and situation. Overall, we have introduced and analyzed several ways to improve the overall performance of our spatio-temporal hybrid sampling algorithm.

In the future, we aim further to improve the post-reconstruction quality that our sampling method can achieve. Specifically, we will evaluate how the information between multiple variables of the same data series can be leveraged to yield a more cohesive group of samples. As our methods' output is a highly compressible list of floats, instead of competing with compression methods, we propose that our sampling method be used as a pre-processing step for compression. This combination will enable us to achieve even higher accuracy while still maintaining high compression ratios.

## Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197 and SHF-1943114. This work was funded by Los Alamos National Laboratory under Information Science and Technology Student Fellowship program. The publication has been assigned the LANL identifier LA-UR-20-27478. We would like to thank our ECP collaborators on the ExaAM project. Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster. The ExaAM research is supported by the 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.

## Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Los Alamos National Laboratory and National Science Foundation (SHF-1910197; SHF-1943114).

## ORCID iDs

Megan Hickman Fulp https://orcid.org/0000-0001-5630-049X
Cooper Sanders https://orcid.org/0009-0005-4879-3622
Jon C. Calhoun https://orcid.org/0000-0001-7191-4422

## Notes

1. https://www.openmp.org/
2. https://developer.nvidia.com/cuda-toolkit

## References

Ahrens J, Jourdain S, OLeary P, et al. (2014) An image-based approach to extreme scale in situ visualization and analysis. *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 424–434.

Akiba H, Fout N and Ma KL (2006) Simultaneous classification of time-varying volume data based on the time histogram. *EuroVis* 6: 1–8.

Belak J, Turner J and Team ET (2019) Exaam: Additive manufacturing process modeling at the fidelity of the microstructure. *APS* 2019.

Bell N and Hoberock J (2012) Thrust: A productivity-oriented library for cuda. *GPU computing gems Jade edition*. Elsevier, pp. 359–371.

Biswas A, Dutta S, Lawrence E, et al. (2021) Probabilistic data-driven sampling via multi-criteria importance analysis. *IEEE Transactions on Visualization and Computer Graphics* 27: 4439–4454.

Biswas A, Dutta S, Lawrence E, et al. (2021) Probabilistic data-driven sampling via multi-criteria importance analysis. *IEEE Transactions on Visualization and Computer Graphics* 27: 4439–4454. DOI: 10.1109/TVCG.2020.3006426.

Biswas A, Dutta S, Pulido J, et al. (2018) In situ data-driven adaptive sampling for large-scale simulation data summarization. *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization - ISAV '18*. ACM Press, pp. 13–18. DOI: 10.1145/3281464.3281467.

Cappello F, Di S, Li S, et al. (2019) Use cases of lossy compression for floating-point data in scientific data sets. *The International Journal of High Performance Computing Applications* 33(6): 1201–1220. DOI: 10.1177/1094342019853336.

Childs H (2015) Data exploration at the exascale. *Supercomputing Frontiers and Innovations* 2(3): 5–13.

Delaunay B, et al. (1934) Sur la sphere vide, Izv. Akad. *Nauk SSSR, Otdelenie Matematicheskikh i Estestven- nykh Nauk*, 7: 793–800.

Doane DP (1976) Aesthetic frequency classifications. *The American Statistician* 30(4): 181–183.

Dutta S, Chen C, Heinlein G, et al. (2017) In situ distribution guided analysis and visualization of transonic jet engine simulations. *IEEE Transactions on Visualization and Computer Graphics* 23(1): 811–820.

Fogal T, Schiewe A and Krüger J (2013) An analysis of scalable gpu-based ray-guided volume rendering. In: *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*. IEEE, pp. 43–51.

Fulp MH, Biswas A and Calhoun JC (2020) Combining spatial and temporal properties for improvements in data reduction. *2020 IEEE International Conference on Big Data (Big Data)*, pp. 2654–2663. DOI: 10.1109/BigData50022.2020.9378457.

Grosset P, Biwer CM, Pulido J, et al. (2020) Foresight: analysis that matters for data reduction. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–15.

Gutiérrez PD, Lastra M, Benítez JM, et al. (2017) Smote-gpu: Big data preprocessing on commodity hardware for imbalanced classification. *Progress in Artificial Intelligence* 6(4): 347–354.

Habib S, Morozov V, Frontiere N, et al. (2013) Hacc: extreme scaling and performance across diverse architectures. *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–10.

Hurricane ISABEL Simulation Data (2019) http://vis.computer.org/vis2004contest/data.html.

Hyndman RJ (1995) *The Problem with Sturges' Rule for Constructing Histograms*. Monash University. 1–2.

Jibben Z (2020) truchas-pbf. https://gitlab.com/truchas/truchas-pbf/

NVIDIA (2020) *GPU*. https://www.nvidia.com/en-us/data-center/v100/ (accessed 23 April 2021).

Nouanesengsy B, Woodring J, Patchett J, et al. (2014) Adr visualization: A generalized framework for ranking large-scale scientific data using analysis-driven refinement. In: *2014 IEEE 4th Symposium on Large Data Analysis and Visualization (LDAV)*. pp. 43–50.

Nvidia CUDA (2010) Curand library.

Palmetto Cluster, Clemson University (2021) Online. http://citi.clemson.edu/palmetto/ (accessed 24 June 2021).

Patchett J and Ahrens J (2018) Optimizing scientist time through in situ visualization and analysis. *IEEE Computer Graphics and Applications* 38(1): 119–127.

Patchett J and Gisler G (2017) *Deep Water Impact Ensemble Data Set*. Los Alamos, NM: Los Alamos National Laboratory. https://datascience.dsscale.org/wp-content/uploads/2017/03/DeepWaterImpactEnsembleDataSet.pdf

Patchett J, Samsel F, Tsai KC, et al. (2016) *Visualization and Analysis of Threats from Asteroid Ocean Impacts*. Los Alamos, NM: Los Alamos National Laboratory.

Scott DW (2009) Sturges' rule. *WIREs Computational Statistics* 1(3): 303–306.

Scott DW (2010) Scott's rule. *Wiley Interdisciplinary Reviews: Computational Statistics* 2(4): 497–502.

Strand G (2015) The cesm workflow re-engineering project. *AGUFM* 2015.

Tian J, Di S, Zhao K, et al. (2020) Cusz: An efficient gpu-based error-bounded lossy compression framework for scientific data. *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pp. 3–15.

Tian J, Rivera C, Di S, et al. (2021) Revisiting huffman coding: Toward extreme performance on modern gpu architectures. *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, pp. 881–891.

Tikhonova A, Correa CD and Ma KL (2010) Explorable images for visualizing volume data. *PacificVis* 10: 177–184.

Underwood R, Di S, Calhoun JC, et al. (2020) *Fraz: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data*.

Wang R, Wang R, Zhou K, et al. (2009) An efficient gpu-based approach for interactive global illumination. *ACM SIGGRAPH 2009 papers*, pp. 1–8.

Wei TH, Dutta S and Shen HW (2018) Information guided data sampling and recovery using bitmap indexing. In: *2018 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, pp. 56–65.

Woodring J, Ahrens J, Figg J, et al. (2011) In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. *Computer Graphics Forum* 30(3): 1151–1160. DOI: 10.1111/j.1467-8659.2011.01964.x.

## Author biographies

*Megan Hickman Fulp* is a Lecturer of Computer Science at Coastal Carolina University, holding a Bachelor of Science degree in Computer Science from the same institution (2019). She obtained a Master of Science degree in Computer Engineering from Clemson University (2021), where her research focused on big data reduction for High-Performance Computing systems. Her current research interests lie in effectively teaching computer programming and algorithmic thinking.

*Dakota Fulp* is an associate professor in the Computer Technology department at Horry-Georgetown Technical College. He received a B.S. degree in Computer Science and Applied Physics from Coastal Carolina University in 2015 and 2017, respectively. Upon graduating, he researched HPC resiliency with Los Alamos National Laboratory from 2017 to 2020. Following this, he received an M.S. degree in Computer Engineering from Clemson University, during which his research focused on data resiliency and lossy data compression for HPC systems. His research interests lie in fault tolerance and resilience for HPC systems, data compression, and effective teaching methodologies.

*Changfeng Zou* is an undergraduate student studying Computer Engineering at Clemson University. He is driven by a passion for creating lasting solutions and maximizing efficiency in computing. He holds the position of president at IEEE-HKN Zeta Iota Chapter.

*Cooper Sanders* is an undergraduate computer engineering student at Clemson University with interests in parallel computing and numerical algorithms, and a talented young

software engineer. He has worked for several years as an undergraduate researcher at Clemson University, as well as interning with Los Alamos National Lab for a summer. On top of engineering, he is also an accomplished jazz trombonist.

*Ayan Biswas* is a scientist at Los Alamos National Laboratory. He is currently leading LANL's sampling-based data reduction efforts under the Exascale Computing Project (ECP) for ALPINE. He was also the co-PI for a recently concluded LANL funded LDRD-DR project (total funding $5M over 3 years) that focused on in situ data modeling and statistical inference. He is an expert of visualization and big data analysis with also considerable experience in information theoretic approaches, vector field exploration, uncertainty visualization, and machine learning for large-scale scientific data sets.

*Melissa C. Smith* is a professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University. She received her B.S. (1993) and M.S. (1994) degrees in Electrical Engineering from Florida State University and a Ph.D. (2003) in Electrical Engineering from the University of Tennessee. She has over 25 years of experience developing and implementing scientific workloads and machine-learning applications across multiple domains, including 12 years as a research associate at Oak Ridge National Laboratory. Her current research focuses on the performance analysis and optimization of emerging heterogeneous computing architectures (GPGPU- and FPGA-based systems) for various application domains, including machine learning, high-performance or real-time embedded applications, and image processing. She is a Senior Member of the IEEE.

*Jon C. Calhoun* is an associate professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University. He received a B.S. in Computer Science from Arkansas State University in 2012, a B.S. in Mathematics from Arkansas State University in 2012, and a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2017. He was awarded a prestigious NSF CAREER award in 2020. His research interests lie in fault tolerance and resilience for high-performance computing (HPC) systems and applications, lossy and lossless data compression, scalable numerical algorithms, power-aware computing, and approximate computing. He is a Senior Member of the IEEE.