Accelerated Dynamic Data Reduction Using Spatial and Temporal Properties

Journal Title
XX(X):1–12

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Megan Hickman Fulp [1], Dakota Fulp [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 spatial-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 paper, 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. Finally, we demonstrate how our method can accept different sampling core algorithms and the effects different core algorithms have.

Keywords

Data Reduction, Data Sampling, Importance Sampling, Temporal Selection, Feature Preservation, Parameter Optimization, GPU, CUDA, OpenMP, Accelerated Hardware, Performance

1 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 (1; 2), yet current HPC system I/O capabilities are not capable of handling such data efficiently (3). As a result, conventional post-hoc data analysis is less tractable, as storing all output data is costly (4; 5; 6; 7). 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 (8; 9; 10), while others focus on preserving specific regions of interest (ROI) in the data (11; 5; 12). In our prior work, we develop a spatial-temporal hybrid data sampling method that biases rare data values and leverages a dataset's temporal aspect to achieve higher post-reconstruction quality (13).

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 the lower throughputs of the overall reduction process (see Section 5). Lastly, as our method uses a specific sampling algorithm at its core (12), 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 methods 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.

[1]Holcombe Department of Electrical and Computing Engineering - Clemson University
[2]Los Alamos National Laboratory

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

- We demonstrate how our hybrid data sampling method accepts different sampling core algorithms and analyze the effects of using different core algorithms.

2 Related Works

2.1 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 (11; 5; 12). 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 ξ 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 (14). 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 and existing time-step selection methods, we develop our spatial-temporal hybrid data sampling method in prior work (13). 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.

2.2 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 (15; 16; 17; 18; 19). While there are various forms of both sophisticated and straightforward data sampling for GPUs, a majority of them are specifically designed for computer graphics (17; 18) or machine learning (19). 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 (17). 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(19). Finally, Fogal et al. implement a GPU sampling algorithm that focuses on identifying regions that should be densely sampled(18). 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.

3 Background

3.1 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 (12) and our spatial-temporal hybrid data sampling method (13) 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 and reuses similar previous regions.

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

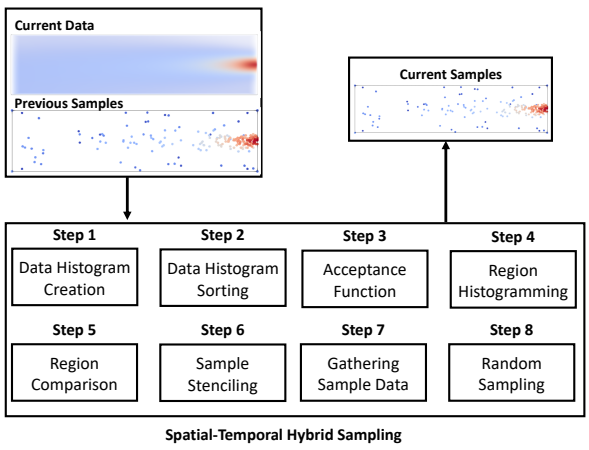


Figure 1. An overview of our spatial-temporal hybrid data sampling method process.

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 root-mean-squared error, 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. 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 root-mean-squared error (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.

Step 7: Gathering Sample Data Using the resulting sampling stencil, each method begins to collect the information for the chosen samples and appends them to the sample data array.

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.

3.2 Reconstruction

To visualize and analyze the effectiveness of our samples, we need to restore each time-step back to full resolution. We use a linear interpolation-based reconstruction using a Delaunay triangulation reconstruct. We use this method as a balanced trade-off between quality and speed, whereas generally, the higher-order the interpolation, the better the quality but the slower the process.

3.3 Optimization Strategies

To improve application performance, many developers leverage thread-level parallelism interfaces such as OpenMP and CUDA. OpenMP * is a CPU multi-threading programming interface that enables users to improve the throughput of highly parallelizable tasks. CUDA † 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.

3.4 Data Sets

3.4.1 ***ExaAM*** The Exascale Additive Manufacturing Project uses exascale simulations to design Additive Manufacturing components (20; 21). We use 108 time-steps with its full spatial resolution of $20 \times 200 \times 50$. Figure 2 shows time-step 64, with highlighted ROI, the hottest portion of the visual. As determined in Section 4, we use 633 bins,

*https://www.openmp.org/

†https://developer.nvidia.com/cuda-toolkit

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

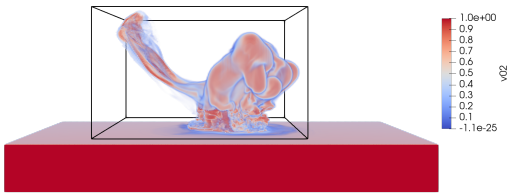
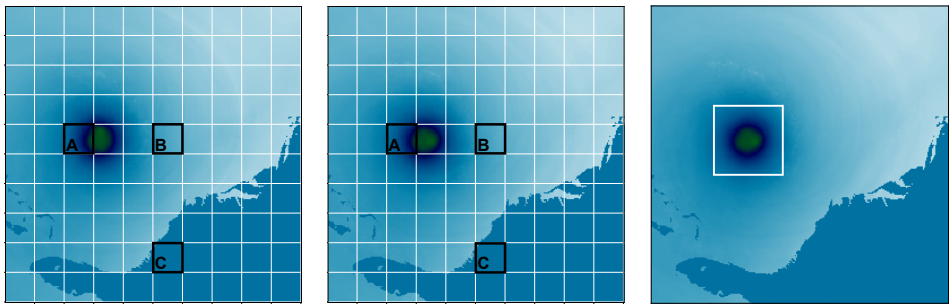


Figure 3. Impact time-step 22388 with highlighted ROI.



(a) Time-Step 11 **(b)** Time-Step 12 **(c)** Time-Step 12

Figure 4. Hurricane Isabel Pressure Visualizations. Figures a) and b) show the dataset divided into regions of dimension $25 \times 25 \times 25$. Figure c) highlights our definition of Region of Interest for this dataset.

an error threshold of 0.0, and 250 regions with dimensions $4 \times 20 \times 10$.

3.4.2 ***Hurricane Isabel*** The Hurricane Isabel Data models the 2003 hurricane in the west Atlantic region (22). 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 (Fig. 4c). We use 48 time-steps of the full resolution, $500 \times 500 \times 100$. As determined by the pre-processing step described in Section 4, for the following experiments we use 27 bins, an error threshold of 28.0, and 1600 regions with dimensions $25 \times 25 \times 25$.

3.4.3 ***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 (23). We use the full spatial resolution of $300 \times 300 \times 300$ 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 3 shows time-step 22,388 of this data set, with highlighted ROI (the water splash). As determined by the pre-processing step described in Section 4, for the following experiments we use 27 bins, an error threshold of 0.0, and 216 regions with dimensions $50 \times 50 \times 50$.

4 Determining Input Configurations

Our Spatial-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.

4.1 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. Therefore, determining an optimal number of histogram bins is very important early on to ensure consistency throughout the data reduction process.

To better understand how the number of bins affects our hybrid method, we evaluate the three regions specified in Figures 4a and 4b 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 un-optimized 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. Figure 6 showcases how the 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.

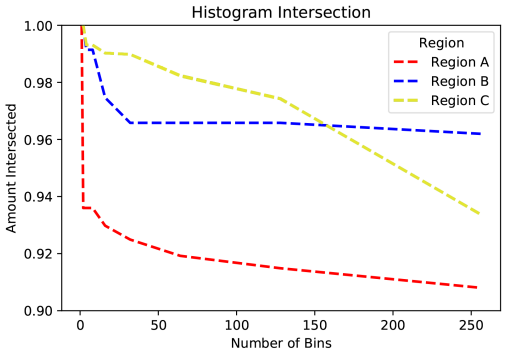


Figure 5. Amount of Intersection with varying bins at three regions of Hurricane Isabel (see Fig. 4)

DATASET	VARIABLE	DIMENSIONS	DATA SIZE	STEPS	REGION DIMENSIONS	BINS	ERROR THRESHOLD
ExaAM	-	$20 \times 200 \times 50$	0.8 MB	108	$10 \times 40 \times 10$	633	0.0
Isabel	Pressure	$500 \times 500 \times 100$	95 MB	48	$25 \times 25 \times 25$	27	28.0
Impact	V02	$300 \times 300 \times 300$	108 MB	130	$50 \times 50 \times 50$	27	0.0

Table 1. Datasets and configurations used in experimental evaluations.

4.1.1 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. Therefore, we use Doane's rule, a modification to Sturges' rule that works better with non-normal data sets (24). Secondly, we use Scott's rule, 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 small number of 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 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 neighbors 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 three times.

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 ($\leq$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.

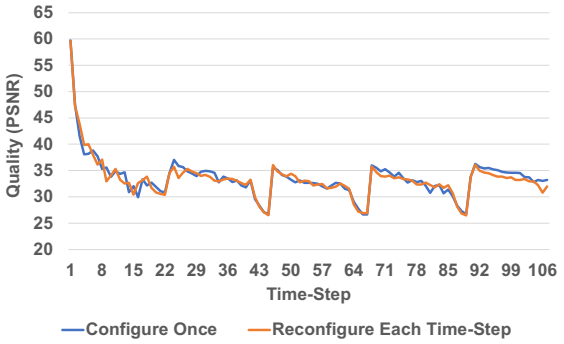
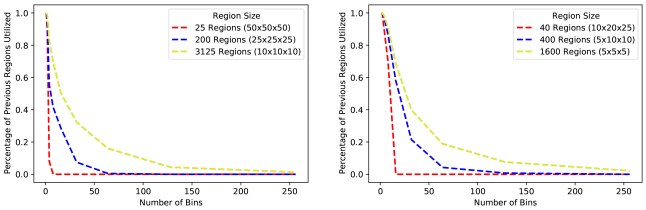


Figure 7. Using a consistent number of bins yields equivalent quality to varying number of bins per time-step.

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 ten time-steps. 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 8a 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 an SNR of 60 dB.

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

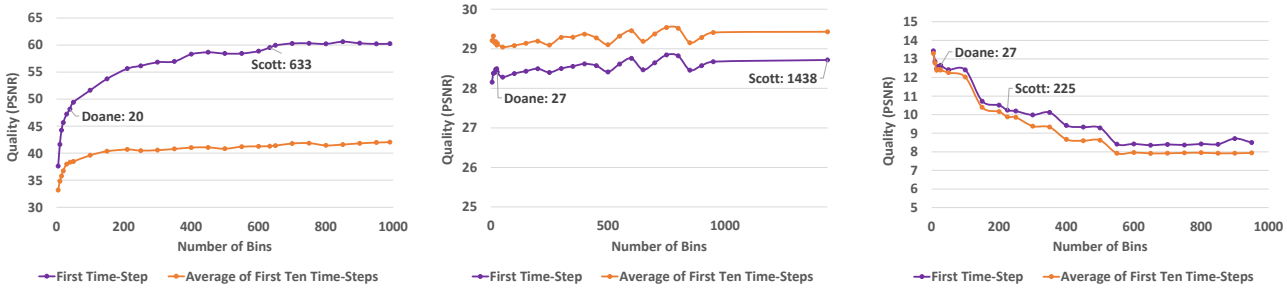


(a) Hurricane Isabel. **(b)** ExaAM.

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

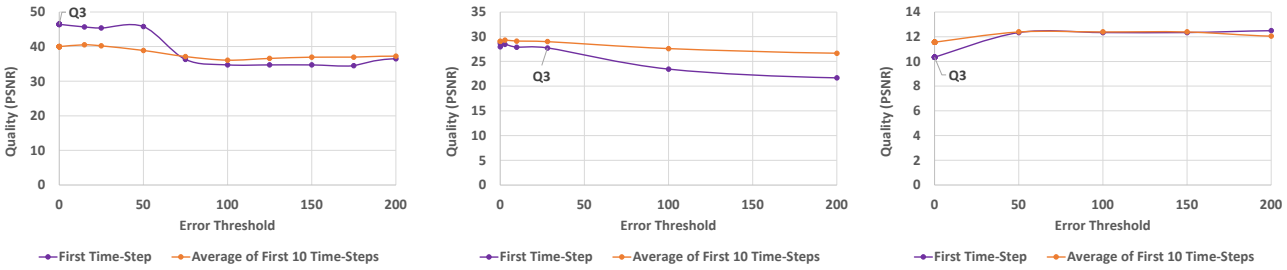
4.2 Error Threshold

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.



(a) ExaAM. **(b)** Isabel. **(c)** Impact.

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



(a) ExaAM. **(b)** Isabel. **(c)** Impact.

Figure 9. The average quality of the first ten time-steps, varying error threshold (Error Reuse Method).

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 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 each of the datasets tested, the third quartile of the error distribution is less than one, 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.

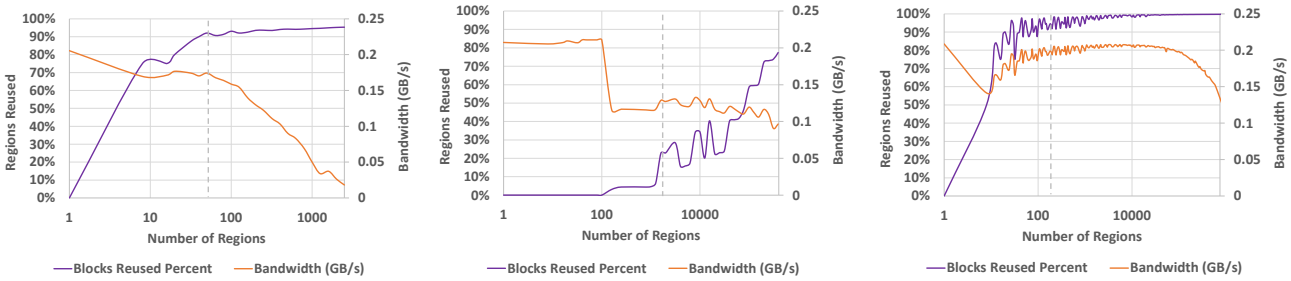
4.3 Number of Regions

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 10. In this assessment, we compare the average percentage of regions utilized from time-step t_{k-1} when gathering samples for time-step t_k, of the first ten time-steps. Our method utilizes more samples from t_{k-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 10 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 a focus on higher post-reconstruction quality, higher bandwidth, or some trade-off of both.

4.3.1 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 ten 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 $\leq$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.



(a) ExaAM. **(b)** Hurricane Isabel. **(c)** Asteroid Impact.

Figure 10. Sampling bandwidth and percent of previous regions utilized as the number of regions varies (Histogram Reuse Method).

For example, given the ExaAM dataset, if the user were to use 1,000 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 10b. 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.

5 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 NVIDIA Telsa V100 GPU. The design of both of our parallelization approaches can be seen in Figure 11. We also parallelize Biswas et al.'s non-reuse importance-based sampling method for later comparison using this same design.

5.1 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, our histogram creation kernel uses a fine-grained level of parallelism and assigns each data value to a separate CUDA thread with 1000 threads to a single thread block. Each thread then decides which histogram bin its data value belongs to and uses an atomic operation to increment the bin count.

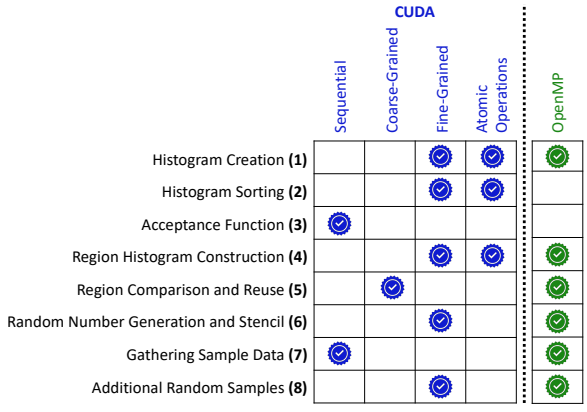


Figure 11. Optimization strategies implemented for each key step of our spatial-temporal hybrid data sampling method.

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" (25).

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 root-mean-squared error 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 (26). 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 their 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, and dynamic memory allocation is not permitted inside CUDA kernels.

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.

5.2 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.

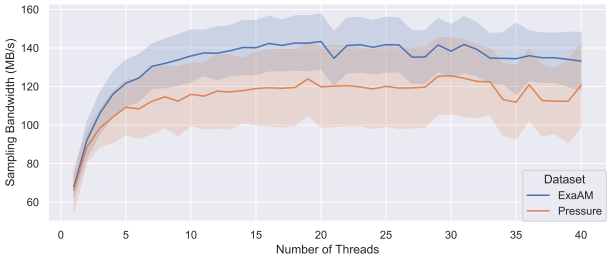


Figure 12. Average Bandwidth (MB/s) of OpenMP accelerated sampling process with varying number of threads.

5.2.1 Datasets and System When conducting our experiments, we use an NVIDIA V100 GPU (27) on Clemson's Palmetto Cluster (28) along with an R740 model 40 core Intel Xeon CPU with 372 GB of memory. When using OpenMP, version 3.1.4, we use ten threads as we found this to be an optimal number of threads, as seen in Figure 12. We use CUDA version 10.0.130 and GCC 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).

5.3 Optimization Effects on Process Throughput

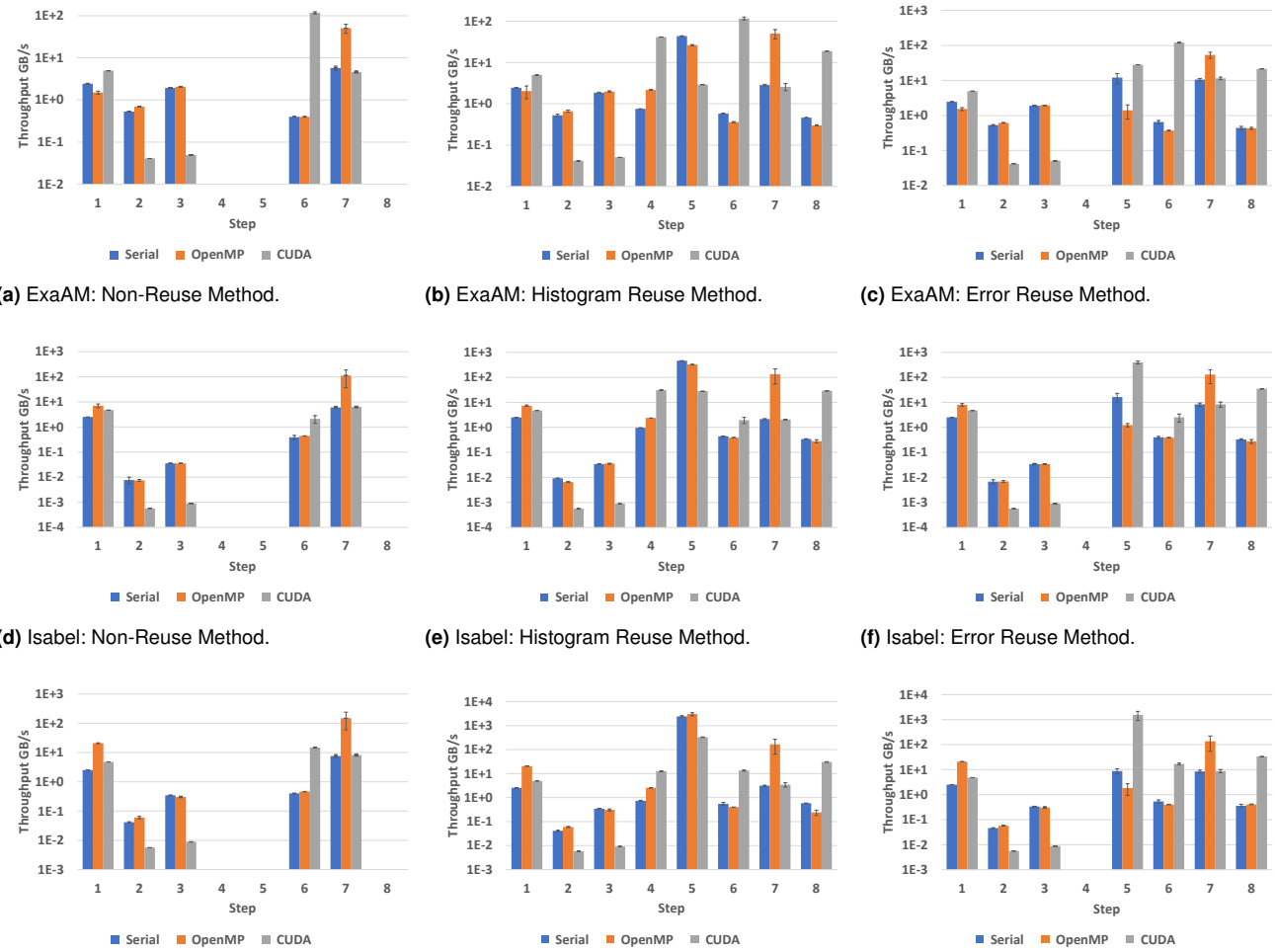
Using our three data sets, we evaluate the throughput of each of our eight sampling sub-process kernels. Figure 13 shows the average bandwidth of steps 1-8 for each dataset, sampling method, and optimization strategy, as calculated as the average of all time-steps in each series, using a 2% sample ratio.

Step 1: Data Histogram Creation Kernel

With the larger datasets, the accelerated CPU implementation of Step 1 has a 3× to 8× improvement over serial. However, with smaller datasets, such as ExaAM (0.8 MB per time-step), we see lower throughputs due to the OpenMP overhead not being outweighed by the data size. When using CUDA, we find consistent 2× improvements over serial, regardless of dataset. This consistency is due to having a constant number of threads in all cases. While both OpenMP and CUDA outperform serial in most cases, CUDA produces a consistent throughput improvement.

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 ran 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



(a) ExaAM: Non-Reuse Method. **(b)** ExaAM: Histogram Reuse Method. **(c)** ExaAM: Error Reuse Method.

(d) Isabel: Non-Reuse Method. **(e)** Isabel: Histogram Reuse Method. **(f)** Isabel: Error Reuse Method.

(g) Impact: Non-Reuse Method. **(h)** Impact: Histogram Reuse Method. **(i)** Impact: Error Reuse Method.

Figure 13. Average throughput per sub-process steps 1-8 per dataset and optimization technique (2% sample ratio).

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 throughputs 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 root-mean-squared error and utilization processes. The opposite is true for the Histogram-Reuse version, which sees lower CUDA throughputs due to the extra necessary computations and data accesses needed to compare the histograms of regions.

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 improvment 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. As we leave the sample gathering process as a sequential process for CUDA, due to the need for dynamically allocated arrays, we find similar performance levels for CUDA and serial. However, we use OpenMP to divide and conquer this step, leading to higher throughputs.

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.

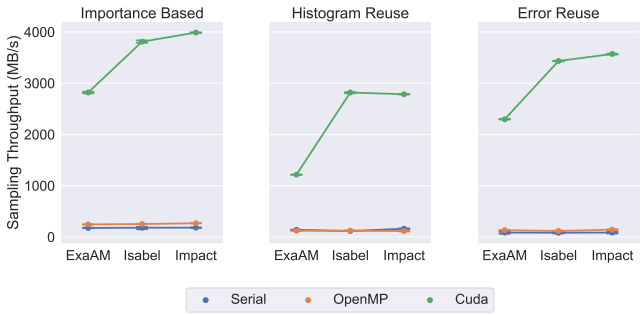


Figure 14. Average Total Sampling Process Bandwidth for each dataset and parallelization technique.

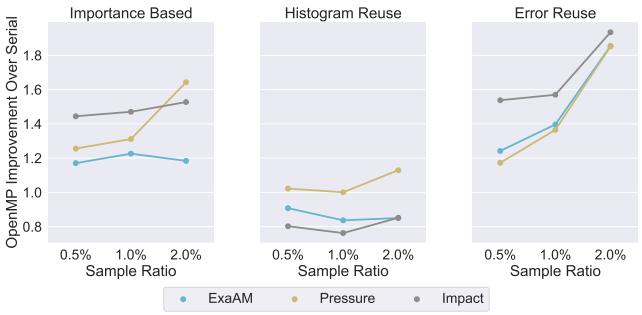


Figure 15. OpenMP Performance Improvement over Serial.

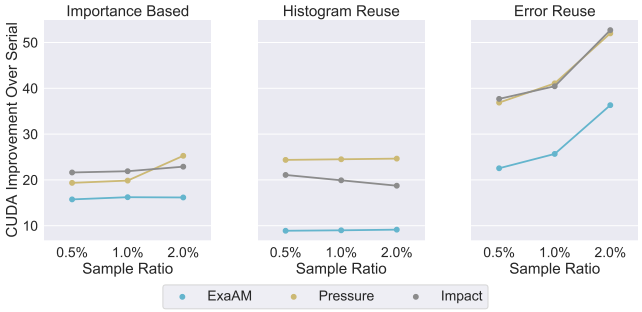


Figure 16. CUDA Performance Improvement over Serial.

5.4 *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 14 shows the average overall bandwidth for each dataset and parallelization technique, while Figures 15 and 16 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 14 we find CUDA throughputs range from 1211 to 3993 MB/s for all data sets while running these methods in serial only achieves between 63 to 196 MB/s. Additionally, previous implementations of Biswas et al.'s importance-based sampling method were only capable of achieving throughputs on the order of tens of megabytes (12; 29).

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 $(1.13\times)$ or lower than $(0.76\times)$ 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 $8.40\times$ to $24.65\times$ 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 ten 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 $175\times, 31\times, 92.69\times$ 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 $22.54\times$ to $52.70\times$ speedup over serial.

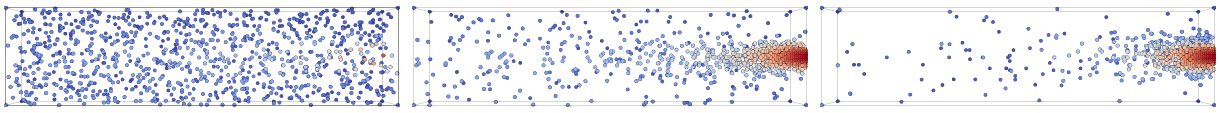
Overall, while the OpenMP implementation achieves higher throughput in some steps, our CUDA version achieves the greatest improvement over the serial version of each method.

6 Changing the Core Sampling Algorithm

The novelty of our spatial-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 in all methods and in HR and ER for gathering samples in regions that were not deemed to utilize previous samples.

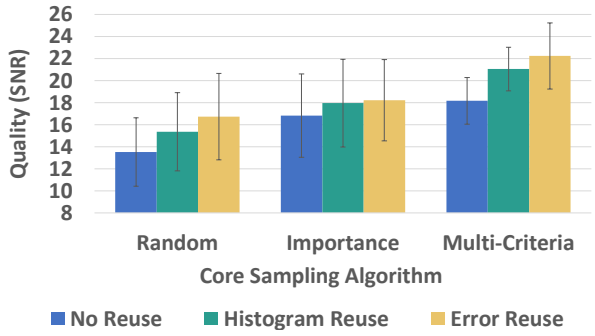
We have the ability to switch the CSA to any existing sampling algorithm, which allows our method to maintain relevance as the data reduction field grows. To show the variance and usefulness of this ability, we use our sampling algorithm configurations with three different cores: Simple Random, Importance-Based (12), and Multi-Criteria Importance-Based Sampling (29). We describe Importance-Based sampling in Section 3 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 (29).

To understand the effect the CSA has on our algorithm, we first show how they distribute their samples, as shown in Figure 17. 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

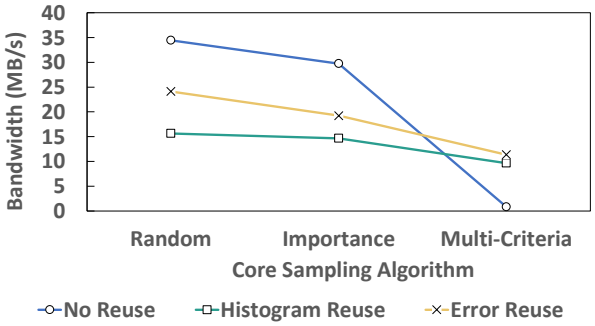


(a) Random Samples **(b)** Importance-Based Samples **(c)** Multi-Criteria Importance-Based Samples

Figure 17. Samples gathered by different Core Sampling Algorithms, Using ExaAM time-step 64 and a sample rate of 0.5%



(a) Quality.



(b) Bandwidth.

Figure 18. Quality and Bandwidth comparison with 1% sample rate of ExaAM data set with varying core sampling algorithm.

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 18a. Figure 18b 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 HR and ER 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 spatial-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.

7 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 paper, 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. Lastly, 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 spatial-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.

References

[1] Strand G. The cesm workflow re-engineering project. *AGUFM* 2015; 2015: IN11C–1791.

[2] Habib S, Morozov V, Frontiere N et al. Hacc: extreme scaling and performance across diverse architectures. In *SC'13: Proceedings of the International Conference on High*

Performance Computing, Networking, Storage and Analysis. IEEE, pp. 1–10.

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

[4] Tikhonova A, Correa CD and Ma KL. Explorable images for visualizing volume data. *PacificVis* 2010; 10: 177–184.

[5] Nouanesengsy B, Woodring J, Patchett J et al. 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.

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

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

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

[9] Childs H. Data exploration at the exascale. *Supercomputing frontiers and innovations* 2015; 2(3): 5–13.

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

[11] Biswas A, Dutta S, Lawrence E et al. Probabilistic data-driven sampling via multi-criteria importance analysis. *IEEE Transactions on Visualization and Computer Graphics* 2020; .

[12] Biswas A, Dutta S, Pulido J et al. In situ data-driven adaptive sampling for large-scale simulation data summarization. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization - ISAV '18*. ACM Press. ISBN 978-1-4503-6579-6, p. 13–18. DOI: 10.1145/3281464.3281467. URL http://dl.acm.org/citation.cfm?doid=3281464.3281467.

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

[14] Akiba H, Fout N and Ma KL. Simultaneous classification of time-varying volume data based on the time histogram. In *EuroVis*, volume 6. pp. 1–8.

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

[16] GPU NVTC. https://github.com/LLNL/zfp/tree/develop/src/cuda_zfp, 2019. [Online; accessed 23-April-2020].

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

[18] Fogal T, Schiewe A and Krüger J. 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.

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

[20] Belak J, Turner J and Team ET. Exaam: Additive manufacturing process modeling at the fidelity of the microstructure. *APS* 2019; 2019: C22–010.

[21] Jibben Z. truchas-pbf. https://gitlab.com/truchas/truchas-pbf/, 2020.

[22] Hurricane isabel simulation data. [Online]. This data was produced by the Weather Research and Forecast model, courtesy of NCAR, and the U.S. National Science Foundation. Available at http://vis.computer.org/vis2004contest/data.html.

[23] Patchett J and Gisler G. Deep water impact ensemble data set. Technical report, Los Alamos National Laboratory, 2017. URL https://datascience.dsscale.org/wp-content/uploads/2017/03/DeepWaterImpactEnsembleDataSet.pdf. LA-UR-17-21595.

[24] Hyndman RJ. The problem with sturges' rule for constructing histograms. *Monash University* 1995; : 1–2.

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

[26] Nvidia, CUDA. Curand library. In *NVIDIA Corporation*. 2010.

[27] N V T C GPU. https://www.nvidia.com/en-us/data-center/v100/, 2020. [Online; accessed 23-April-2021].

[28] Palmetto Cluster, Clemson University. http://citi.clemson.edu/palmetto/, 2021. [Online; accessed 24-June-2021].

[29] Biswas A, Dutta S, Lawrence E et al. Probabilistic data-driven sampling via multi-criteria importance analysis. *IEEE Transactions on Visualization and Computer Graphics* 2020; : 1–1DOI:10.1109/TVCG.2020.3006426.

Acknowledgements

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.