PARALLEL PROBABILISTIC DATA SAMPLING ALGORITHM IMPLEMENTATION AND PERFORMING SCALING STUDY

CS677 PROJECT REPORT

Dasari Charithambika
210302
Statistics and Data Science
Department of Mathematics & Statistics
Indian Institute of Technology, Kanpur

Divya Gupta
210353
Statistics and Data Science
Department of Mathematics & Statistics
Indian Institute of Technology, Kanpur

Om Shivam Verma
210684
Statistics and Data Science
Department of Mathematics & Statistics
Indian Institute of Technology, Kanpur

Palak Mishra
210690
Statistics and Data Science
Department of Mathematics & Statistics
Indian Institute of Technology, Kanpur

Siddharth Pathak
211034
Statistics and Data Science
Department of Mathematics & Statistics
Indian Institute of Technology, Kanpur

May 1, 2024

1 Brief Background

Although supercomputers are becoming increasingly powerful, their components have thus far not scaled proportionately. Thus, in-situ analysis of large-scale simulations, where the analysis is performed while the simulation is producing the data, has become an important part of data analysis and the visualization pipeline in the past decade.

Compared to sophisticated data modeling approaches, *data sampling* can represent the complete data with much smaller memory and computation requirements for data reduction.

In this project, we implement a data-driven approach to identify the data values that are probabilistically more *important* or maybe more interesting to the domain expert for analysis. More *important* points are sampled more often, ensuring as many interesting features are sampled as possible. Various factors can decide this importance: sparsity of values, gradient values, or a mixture of both. The sampling method converts structured data into a point cloud.

Although this is a simple and powerful method for data reduction, for very large datasets, the computation power of a single supercomputer may not be enough to sample in a reasonable time. Moreover, for in-situ data reduction, we need an even faster way to sample and store the point cloud. So, there is a need to parallelize these sampling methods and study how efficiently they scale to multiple processes running in parallel.

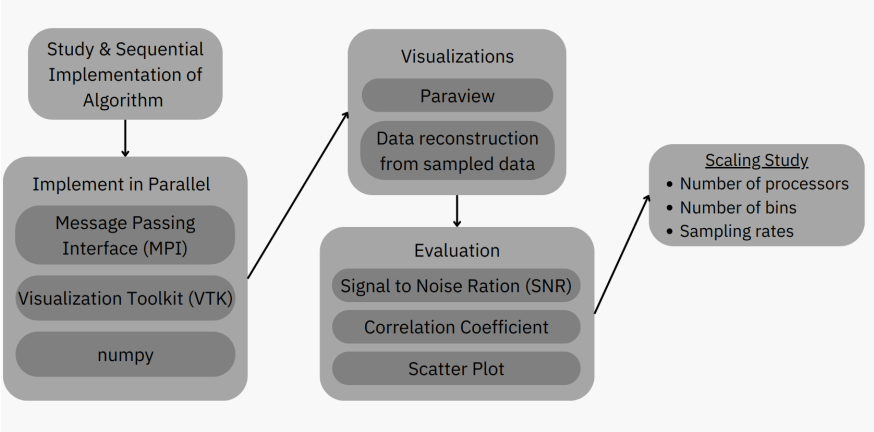
2 Problem Statement

We aimed to study and implement data sampling algorithms in a parallel processing manner using the Message Passing Interface (MPI). Points from regularly structured data are sampled to create a point cloud that retains as many important features and regions of interest as possible for effective analysis by domain experts.

The correctness of the implementation is validated by both visual and quantitative evaluations. This also tries to measure the degree of presence of features that may interest the domain experts. The data is also reconstructed to compare various features with the original data.

Finally, a scaling study of each method is conducted. This involves variations in the number of processors, sampling rates, and wherever applicable, the number of bins.

3 Methodology



1. Study & Sequential Implementation of Algorithm
2. **Implementation in Parallel**: We used VTK, numpy and MPI for storing, processing, and parallelly distributing workload among the nodes.
   - **Message Passing Interface (MPI)**: We are using MPI's wrapper for python, mpi4py. Various parallel operations are used: broadcast, scatter, gather, and reduce.
   - **Visualization Toolkit (VTK)**: VTK Image dataset is taken as the input and sampled points are saved as polydata
   - **numpy**: NumPy's n-dimensional arrays are very useful and efficient in handling and working with the 3D dataset. So, we can efficiently generate random values and points from the dataset.
3. **Reconstruction and Visualization** Visualisation is done by reconstructing sampled data via linear interpolation, then Volume-based and Isocontour-based visualization in Paraview and Matplotlib.
4. **Evaluation**: We have used two defined quality measures:
   - **Signal−to−Noise Ratio (SNR)**:
     $SNR = 20 * \log_{10}(\sigma_{raw}/\sigma_{noise})$
     σ_{raw}: standard deviation of the original data
     σ_{noise}: standard deviation of the error of the reconstruction
     As the reconstruction improves, σ_{noise} gets smaller while σ_{raw} remains constant and the SNR increases. Larger values indicate better reconstructions.
   - **Correlation Coefficient**: between the original and reconstructed datasets.
     – Plot correlation coefficient across different sampling methods for sampling percentage: 1%, 2%, 3%, 5%.
     – Scatter plot taking points from each dataset, original and reconstructed on X and Y
5. **Scaling Study**: Comparing no. of processors, bins, and sampling rates across time with implemented algorithms and plotting it

4 Datasets

- We are currently leveraging the **Isabel Cyclone** dataset's pressure values for the purpose of visualization and implementing parallelization techniques in our computational workflows.
- We are using the two different dimensions of the Isabel dataset.

- "Small" dimension: 250 x 250 x 50
- "Large" dimension: 750 x 750 x 150

5 Implementation

5.1 Simple Random Sampling

This method is one of the simplest forms of sampling methods. Every point is assumed to have identical importance and sampled accordingly. We take a sampling ratio η. At every point, we sample a random number from the uniform distribution, and if it is less than η it is stored, otherwise it is removed.

Parallel implementation involves subsequent Scatter and Gather commands to distribute data to processors, sample the data and get all the sampled points at a node to store into the VTK's Polydata format.

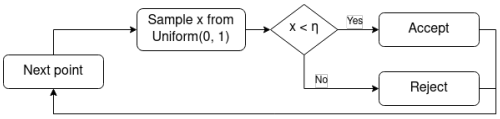


Figure 1: Simple random sampling algorithm

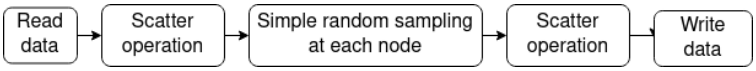


Figure 2: Simple random sampling in parallel

5.1.1 Reconstructing based Visualization

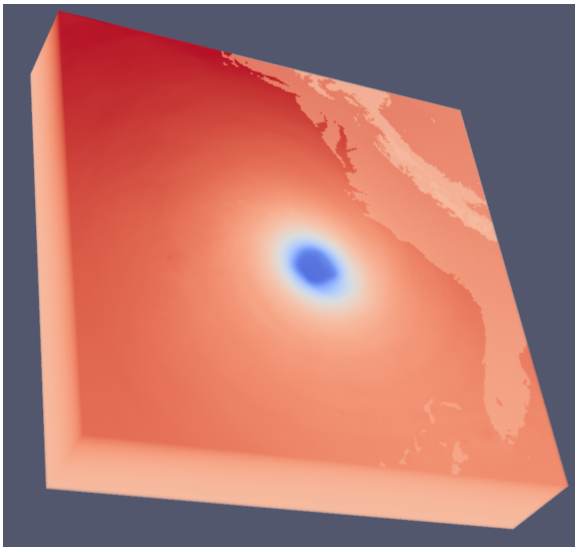


Figure 3: Original Datset

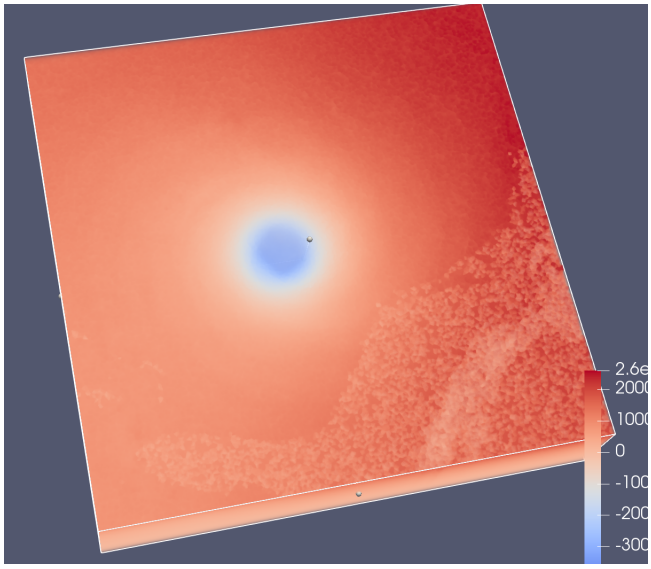


Figure 4: Reconstructed dataset of sampling percentage(1%)

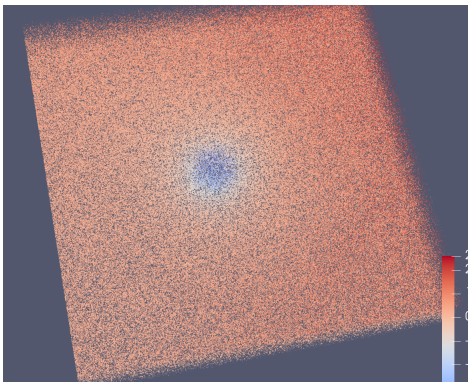


Figure 5: Sampled point data in paraview

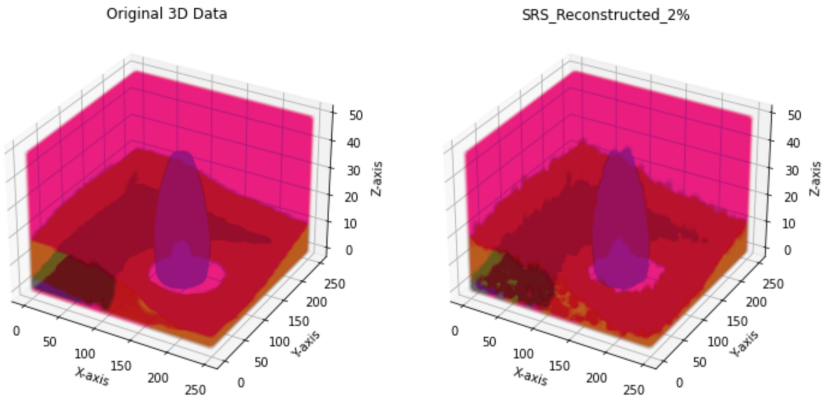


Figure 6: Visualization in matplotlib

5.1.2 Evaluation

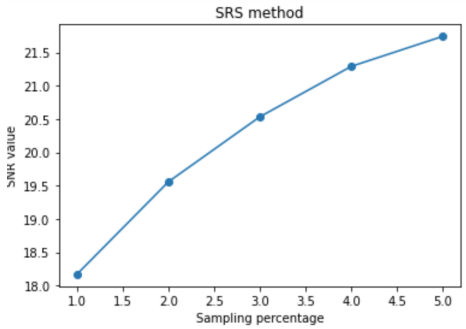


Figure 7: SNR-sampling plot

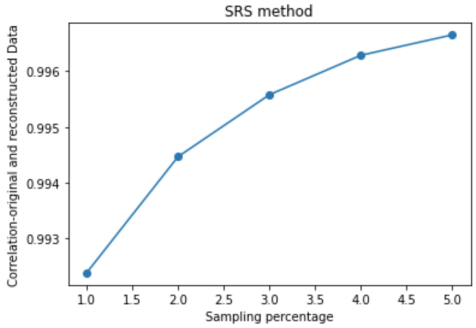


Figure 8: Correlation-sampling rate

5.1.3 Scaling

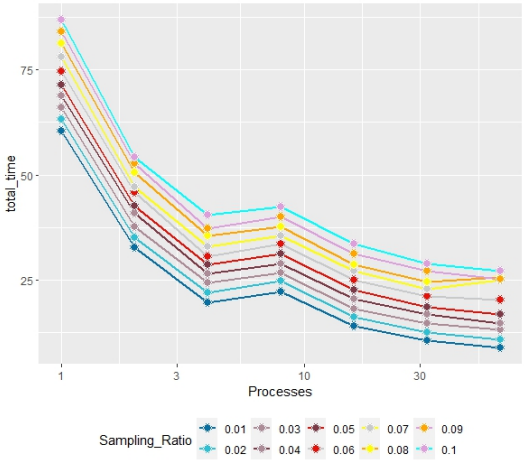


Figure 9: Time v/s number of processors

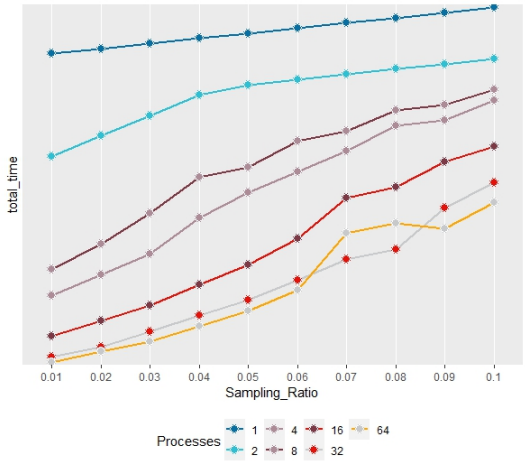


Figure 10: Time v/s sampling rates

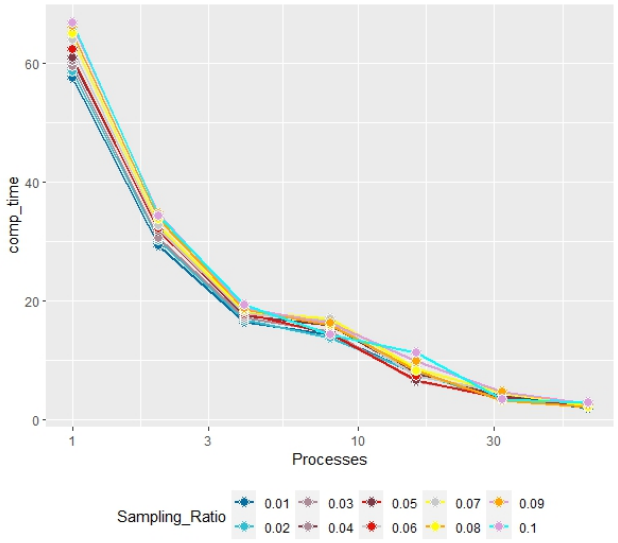


Figure 11: Computation Time v/s number of processors

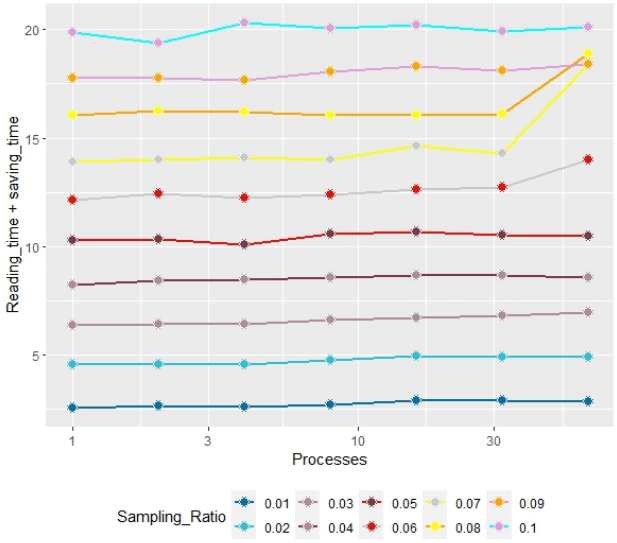


Figure 12: Reading & Saving Time v/s Processes

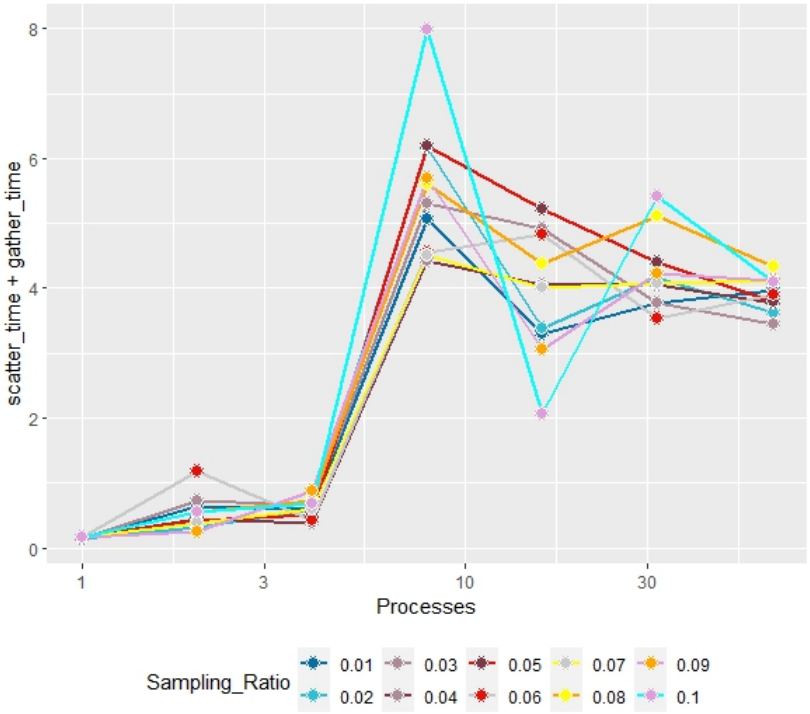


Figure 13: Scatter & Gather Time v/s Processes

5.2 Value-Based Importance Sampling

Instead of giving equal weightage to all the points, we give an importance value to each. The importance of the data points is determined by an importance function (I_f). The data points whose field values are highly likely in the dataset are assigned low priority. To obtain a uniform sample from the data, our approach resembles the rejection sampling algorithm and is also closely related to the importance of sampling. For achieving value-based importance sampling begins by creating the histogram of the input dataset. We sort the histogram bins according to their counts from smallest

to highest. After creating the target histogram counts, we perform a bin-wise division of the count values between the original and target histogram. This will give us our important function. Using this, we perform the data sampling.

Compared to random sampling which would distribute the samples in a space-filling manner throughout the space, value-based importance sampling provides more samples from the feature region of the dataset.

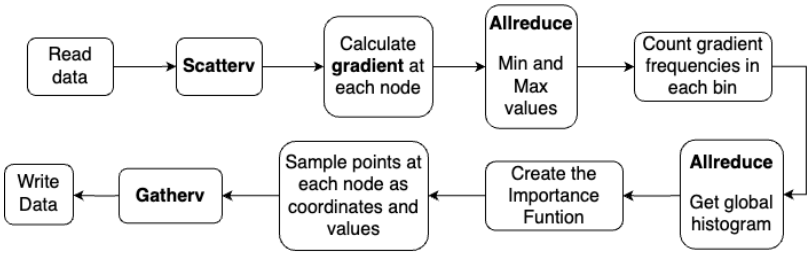


Figure: Value-based Importance Sampling Algorithm

5.2.1 Reconstructing based Visualization

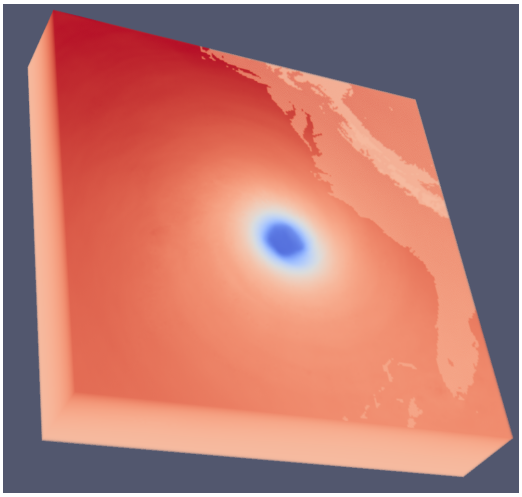


Figure 14: Original dataset

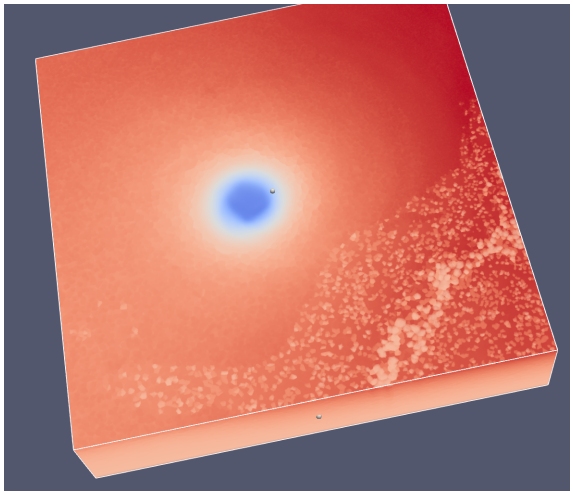


Figure 15: Reconstructed dataset of sampling percentage(1%)

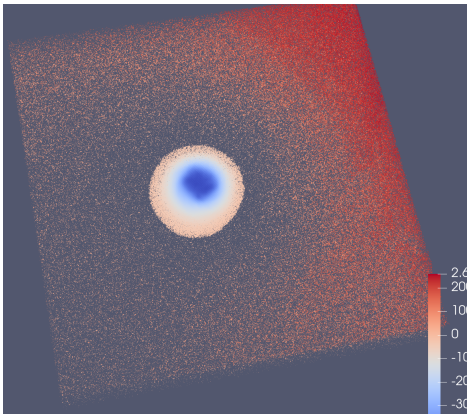


Figure 16: Sampled point data in paraview

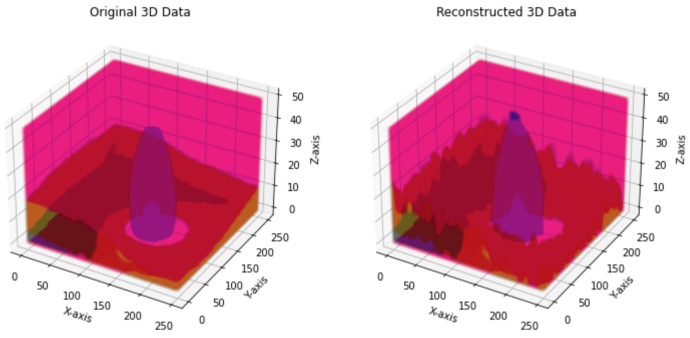


Figure 17: Visualization in matplotlib

5.2.2 Evaluation

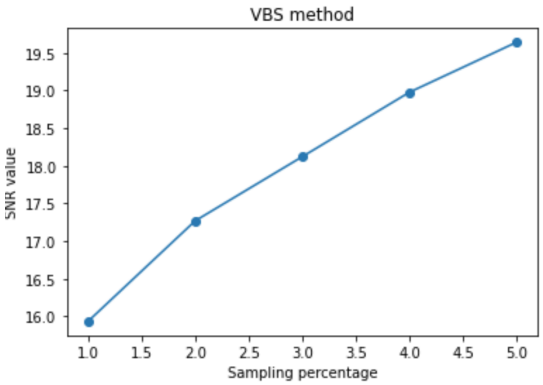


Figure 18: SNR-sampling plot

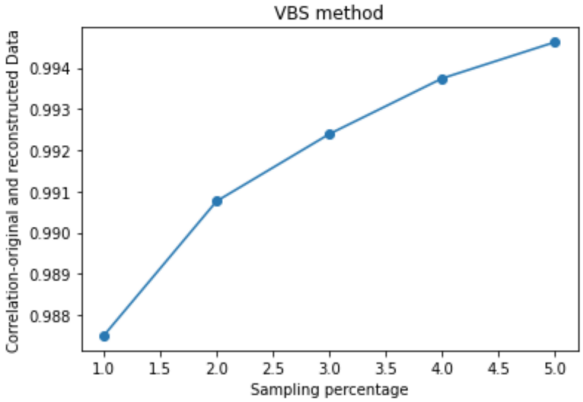


Figure 19: Correlation-sampling rate

5.2.3 Scaling

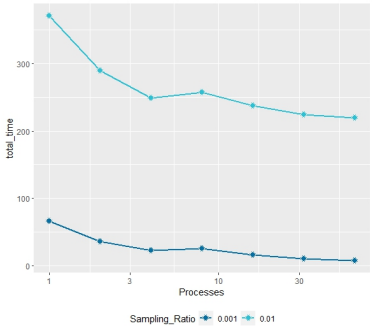


Figure 20: Time v/s number of processors

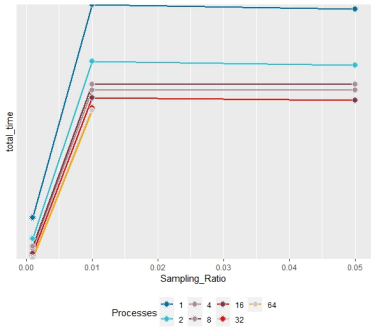


Figure 21: Time v/s sampling rates

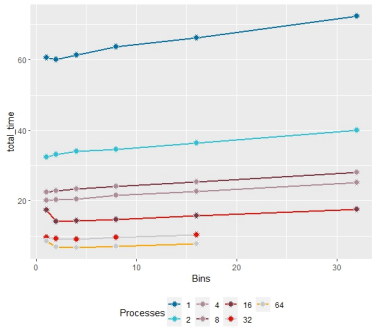


Figure 22: Time v/s number of bins

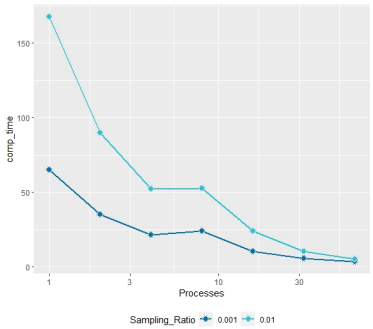


Figure 23: Computation Time v/s number of processors

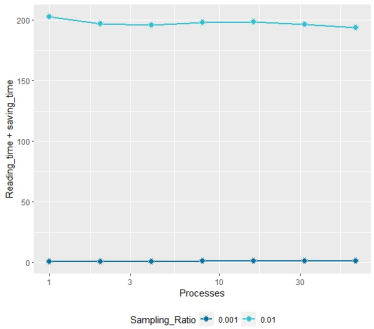


Figure 24: Read & Save Time v/s Processes

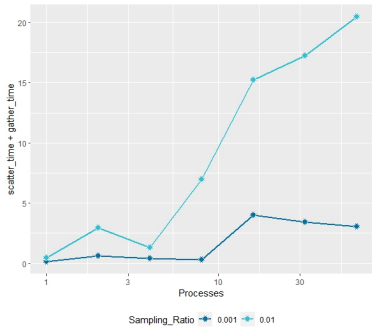


Figure 25: Scatter & Gather Time v/s Processes

5.3 Smoothness-Based Sampling

A key consideration when examining a data point in a scientific dataset is its local homogeneity or smoothness. When it comes to data sampling, a local region that experiences sudden changes in field values needs more representative samples in order to do reconstruction or local property prediction. In other words, a larger probability will be applied to

the points in this zone. High gradient zones are very important from the perspective of data reconstruction because reconstruction is much more difficult in those areas. Our importance function I_F is therefore directly proportional to the gradient for gradient-based sampling. We build a gradient histogram with a user-specified bin number. Assigning the samples from the bin with the greatest count is where we begin.

When analyzing a data point in a scientific dataset, an important aspect is its local smoothness or homogeneity. In the context of data sampling, if field values change abruptly in a local region, then this region requires more representative samples to perform reconstruction or local property prediction. That is, the points in this region will be probabilistically assigned higher importance. From the point of view of data reconstruction, high-gradient regions are of high importance because reconstruction is much harder in those regions. Thus, for gradient-based sampling, our importance function I_F is directly proportionate to the gradient. We proceed by creating a histogram of gradients with a user-given bin number. We start to assign the samples from the bin with the highest count. The process is repeated until all the samples are picked. To determine the bin-wise acceptance probability, the resultant histogram can now be bin-wise divided with the original gradient histogram. We are aware of which data point has the highest probability of being accepted since we know which bin each data point falls into based on the gradient.

Given the same storage constraint, this gradient-based sampling does not have the space-filling property compared to random sampling and value-based sampling.

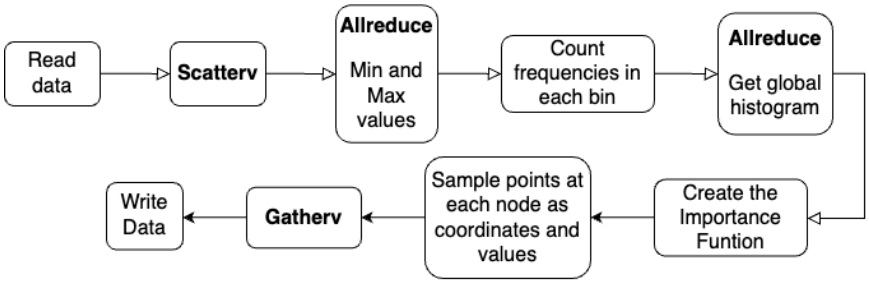


Figure: Smoothness-based Sampling Algorithm

5.3.1 Reconstructing based Visualization

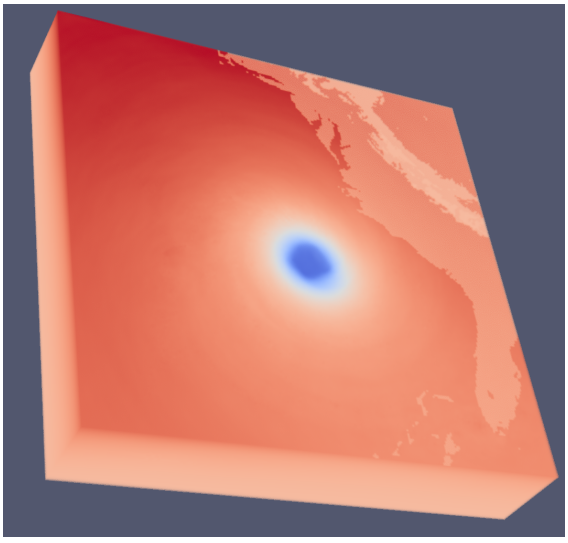


Figure 26: Original dataset

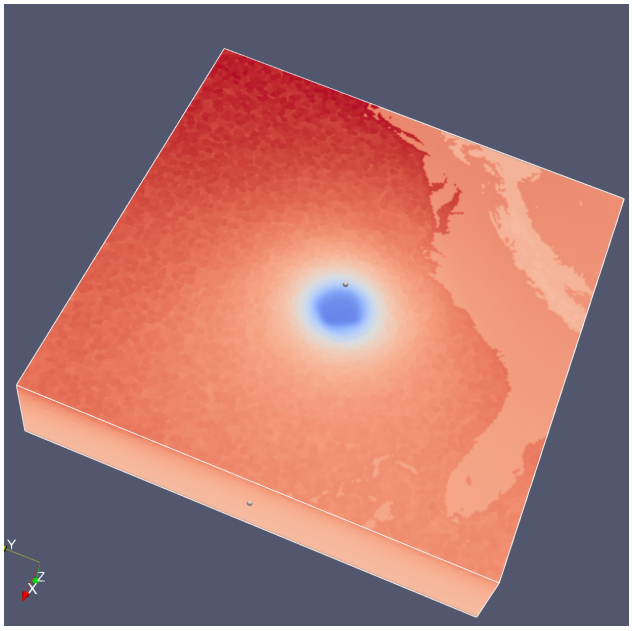


Figure 27: Reconstructed dataset of sampling percentage (1%)

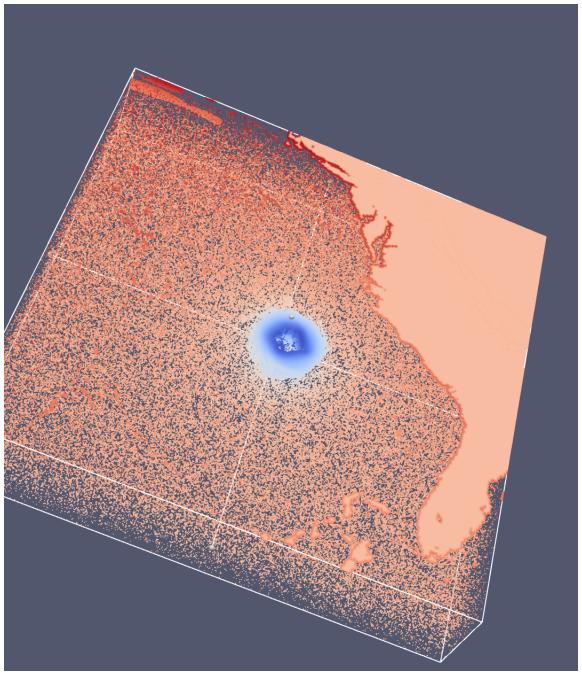


Figure 28: Sampled point data in paraview

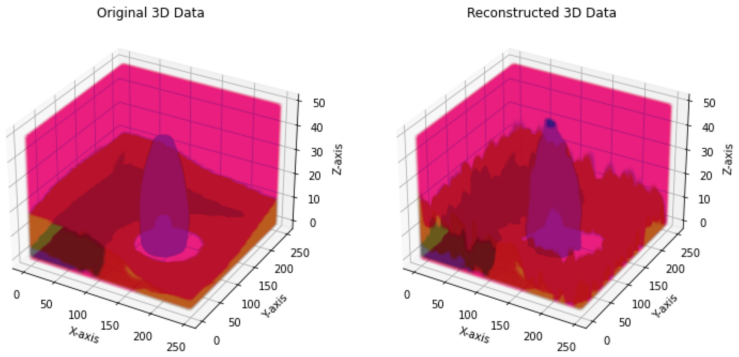


Figure 29: Visualization in matplotlib

5.3.2 Evaluation

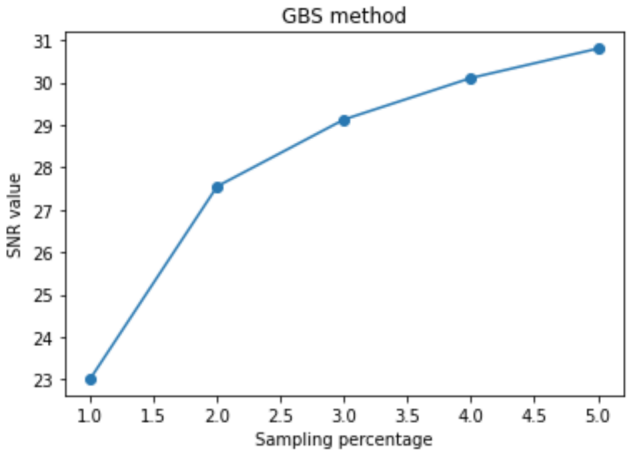


Figure 30: SNR-sampling plot

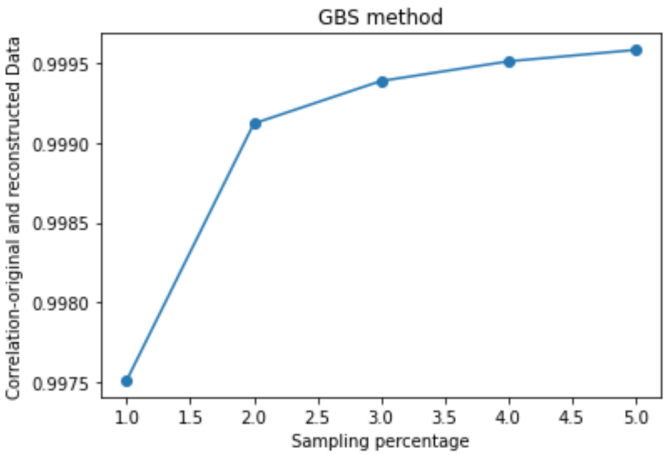


Figure 31: Correlation-sampling rate

5.3.3 Scaling

Below we have attached graphs for the scaling study of gradient-based sampling.

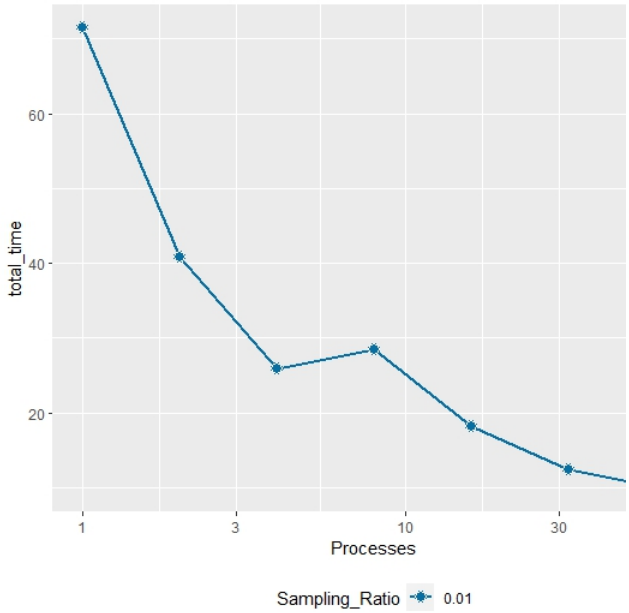


Figure 32: Time v/s number of processes

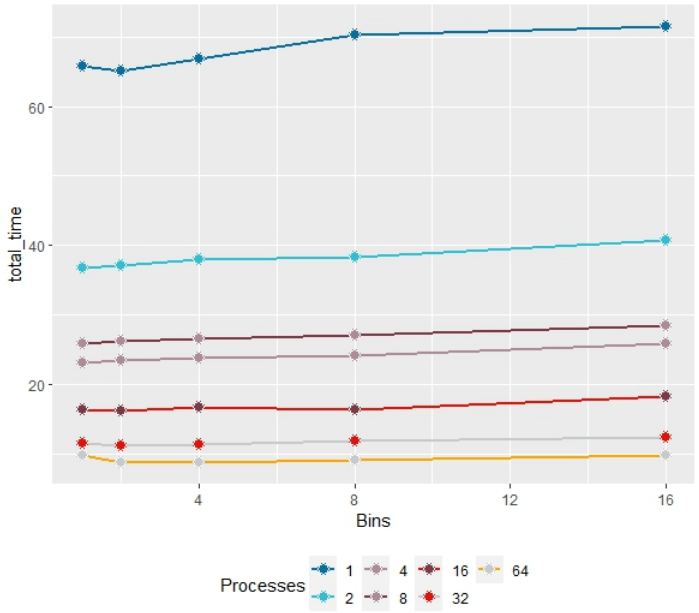


Figure 33: Time v/s bins

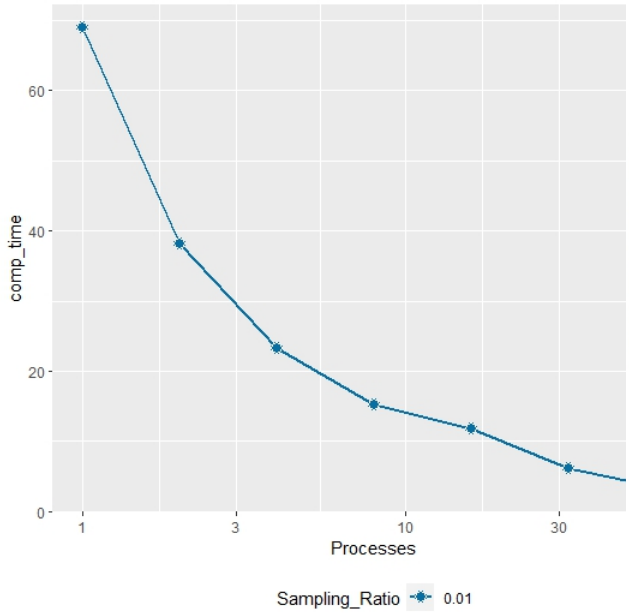


Figure 34: Computation Time v/s number of processes

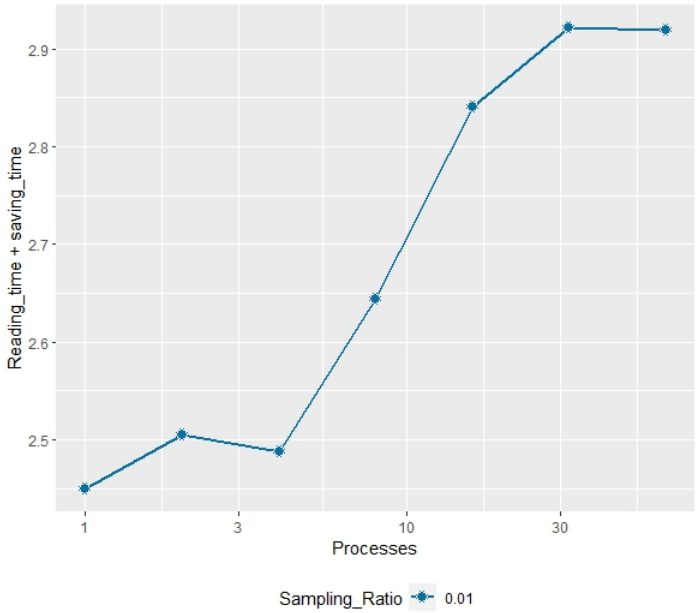


Figure 35: Read & Scatter Time v/s Processes

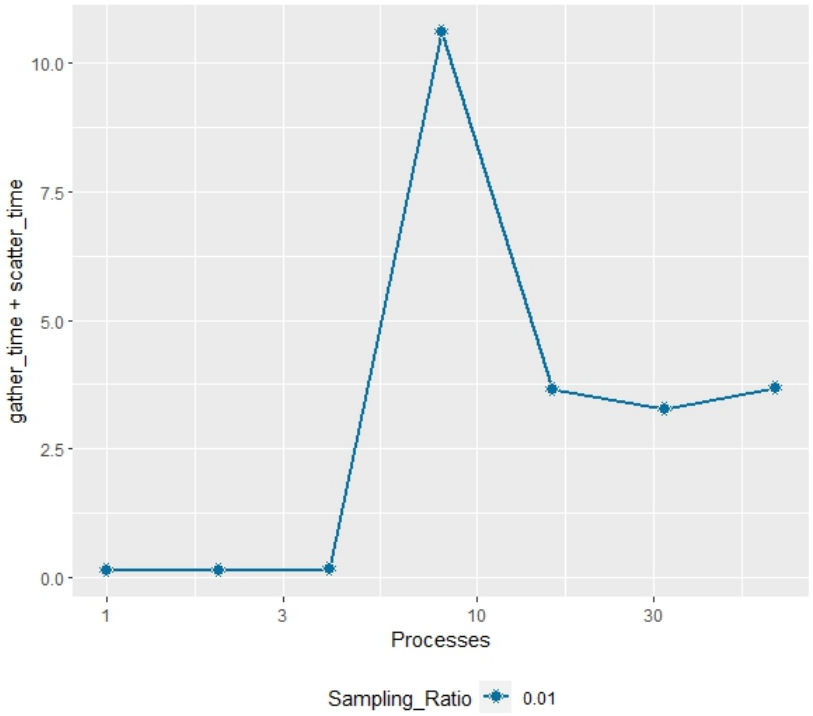
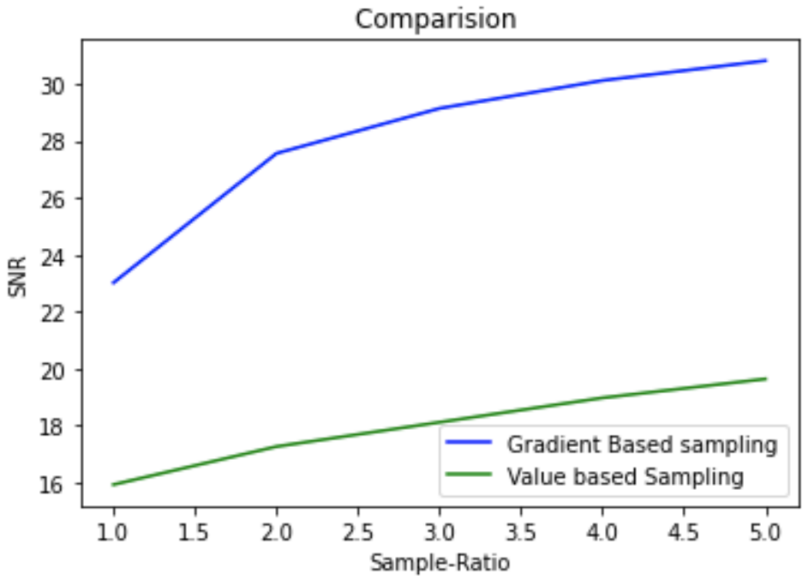


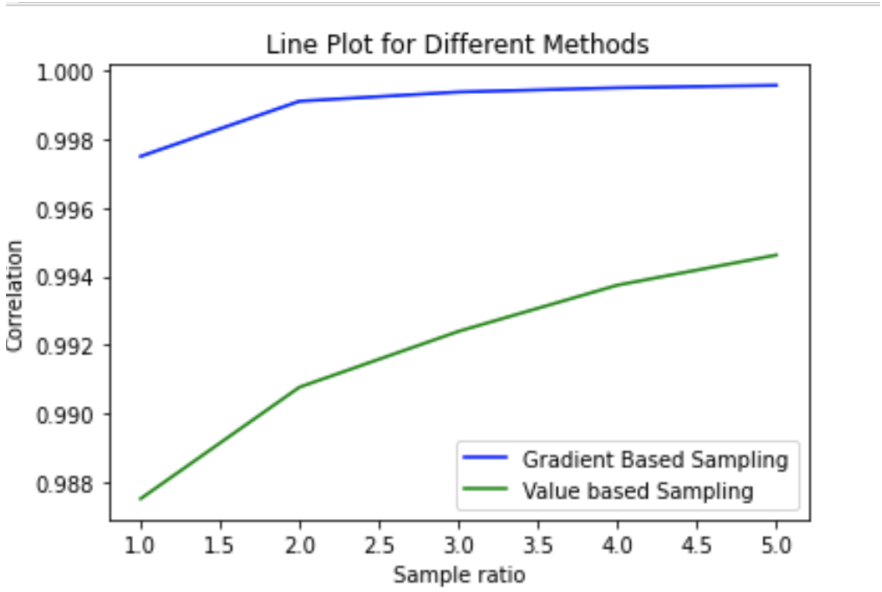
Figure 36: Scatter & Gather Time v/s number of processes

6 Performance Evaluation

6.1 SNR Plot



6.2 Correlation coefficient plot



7 Bottlenecks

Our implementation faces the following bottlenecks:

1. The input file is read from a single node despite the disk space being common for all the processes. This requires a Scatter operation to distribute the data to all the processors.
2. All the sampled points are gathered into a single node before being written to a polydata file. This again becomes a bottleneck where only one processor is working to save the file to the disk.

8 Challenges Faced

1. Found that the data block boundaries interfered with gradient-based histogram creation in the smoothness-based sampling. This caused data block boundaries to be sampled instead of useful boundaries in the data.
2. Care had to be taken about how *numpy* stores the arrays in memory since MPI moves the memory blocks without caring for how numpy stores its array. We used the row-major order to store the numpy arrays, across all processes.

9 Conclusions

- We have implemented sampling techniques.
  1. Simple random sampling
  2. Value-based importance sampling
  3. Smoothness-based Importance Sampling
- We have utilized mpi4py for parallelization, employed to distribute the computational workload across multiple nodes or processors
- We have used linear interpolation-based reconstruction-based visualization.
- We have used these evaluation metrics to compare the existing methods to our method.
  1. Signal to Noise Ratio (SNR)
  2. Correlation Coefficient
  3. Scatter Plot

- Implementation Tools
  1. Employed a combination of programming languages and libraries suitable for implementing the sampling techniques, mpi4py, and visualization algorithms
  2. Ensured compatibility and optimization for parallel computing using mpi4py
- Objective
  1. The overarching goal was to enhance the efficiency and accuracy of data sampling and visualization through the integration of diverse sampling techniques and parallelization strategies.

10 Future Work

- In the future, we hope to use in situ data sampling in the future to cut down on the time and resources required for transmitting, storing, and analyzing big datasets.
- Additionally, we would like to compute gradients on data block boundaries using ghost cells.
- Furthermore, we prefer to utilize file formats that facilitate I/O parallel file reading.

Student's Name	Work
Dasari Charithambika	Reconstruction, Evaluation, Report, PPT
Divya Gupta	Reconstruction, Evaluation,Report, PPT
Om Shivam Verma	Parallelisation using MPI, Report, PPT
Palak Mishra	Implementation of Sampling Algorithms, Report, PPT
Siddharth Pathak	Scaling study, Report, PPT

11 References

- https://ieeexplore.ieee.org/document/9130956t
- https://mpi4py.readthedocs.io/
- https://examples.vtk.org/site/Python/