Streaming Data Visualization

Ajitesh Shree (210079), Kumar Kanishk Singh (210544),
Kundan Kumar (210547), Aayushman Gupta (210021), Vala Yash (211142)

November 24, 2023

1 Brief Background

Streaming Data Visualization

Scientists and researchers across diverse fields rely on visualization techniques to analyze simulation data. However, this endeavor often proves cumbersome due to the resource-intensive nature of simulations, demanding high computational power typically found in supercomputer. Unfortunately, access to supercomputers and other resources is not consistently available. To address this challenge, there emerged a pressing need for a solution that allows simulations to run on computers with high computational power while enabling scientists and researchers to visualize the results conveniently on any computer, irrespective of its computational capabilities. This project aims to bridge this gap by developing an efficient streaming data visualization system that accommodates large-scale simulations, providing a flexible and accessible platform for data analysis and interpretation.

2 Problem Statement

- There is a simulation running on a cluster
- This cluster will generate data for the simulation
- This data will be streamed to another cluster for visualization

3 Technical Method

In this project, we utilized a weather simulation to generate 3D data with dimensions representing time steps, height, and width (grid size).The simulation, designed for atmospheric advection and diffusion modeling, outputs data to a NetCDF file named 'output.nc', a common format for scientific data storage. Instead of writing this data on disk, we shall stream the data from memory from one cluster to another cluster for visualisation.
The subsequent sections of the report will outline our approach to efficiently stream this data and implement visualization on a different cluster.

3.1 Data Streaming

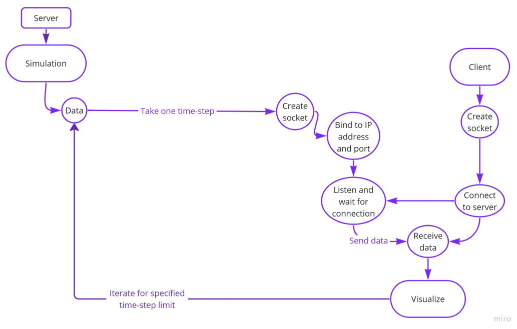
Our primary objective is to efficiently transfer simulation data from the cluster to the client, enabling real-time 3D visualization. As the simulation progresses, data is generated. Our approach involves streaming and visualizing this dynamic data to the client via TCP/IP for each time step. Additionally, we approached it in another manner where the data is generated for the entire specified time step limit and then streamed to the client through TCP/IP transmission for visualization. We termed the first approach "Multiple Socket Visualization" and the second "Single Socket Visualization."

- **TCP/IP Setup:**

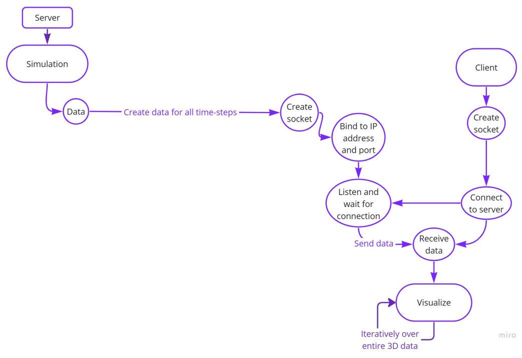
  To establish a connection between the simulation and the client, we leverage TCP/IP communication fundamentals. This process initiates with the creation of a socket object using the Python socket module. The socket is then bound to a specific IP address, entering a listening mode to await incoming connections. The client connects to this IP address, establishing a communication link.

- **Data Transmission Process:**

  In the Multiple Socket Visualization approach, after establishing the connection, the simulation transmits the data of a single time step to the client. This data is encapsulated in a NumPy array, subsequently converted into a byte format for efficient transmission over the TCP/IP connection. The connection is then closed, ensuring optimal resource utilization.



(a) Multiple Socket Visualization


Server
Simulation
Data
Create data for all time-steps
Create socket
Bind to IP address and port
Client
Create socket
Listen and wait for connection
Connect to server
Send data
Receive data
Visualize
Iteratively over entire 3D data
miro

(b) Single Socket Visualization

Figure 1

In the Single Socket Visualization approach, rather than sending the data for one time step, the entire 3D array is constructed first and then converted into a byte format for efficient transmission over the TCP/IP connection to the client processes.

- **Client-Side Processing:**

  On the client side, the received byte data undergoes processing to reconstruct the original NumPy array. This reconstruction is based on the known array size, which is transmitted to the client beforehand using a TCP/IP connection. This facilitates accurate interpretation of the simulation data. After successful reception and processing, the client closes the connection, completing the data transfer cycle for one iteration.

  In the case of the "Single Socket Visualization" variation, the client doesn't receive data for one time step in each iteration. Instead, it receives the entire dataset for all time steps as a 3D array before closing the connection.

3.2 Visualization

- **Dynamic Rendering Window Creation:**

  In the "Multiple Socket Visualization" approach, the visualization process dynamically generates a rendering window for each time step, seamlessly incorporating the most recently received simulation data. This window persists until the visualization concludes for the current time step, offering a continuous and seamless experience. Upon closing the window, the system effortlessly transitions to the subsequent time step, iteratively progressing through the simulation data.

  Contrastingly, in the second approach, the visualization process generates only one window for the entire 3D dataset and updates the image for each time step on that screen instead of creating a new one for each iteration.

- **User-Friendly Interaction Mechanism:**

  An intuitive and user-friendly interaction mechanism has been implemented to enhance the overall user experience. Users can smoothly advance to the next time step by simply pressing the 'Enter' key, ensuring straightforward and precise navigation through the visualization process.

  However, due to the consideration of large time steps in our calculations, pressing 'Enter' after each step became a tedious task. Consequently, we opted to eliminate this requirement and instead print a statement after the completion of each time step and after the entire process in both approaches.

- **Pseudocolor Enhancement for Interactivity:**

  The rendering window is enriched with pseudocoloring to augment interactivity during the visualization. Pseudocoloring involves assigning colors to specific data ranges, making the window more interactive and providing a visually informative representation of the simulation data.

- **Scalar Range Definition:**

  The scalar range for pseudocolor mapping is dynamically defined based on the received simulation data. The minimum and maximum values within the data are determined to establish an optimal range for effective pseudocolor representation in the visualization.

- **Renderer and Render Window Creation:**

VTK renderer and render window instances are dynamically created to facilitate the visualization process. The renderer is responsible for rendering the scene, while the render window provides the canvas for the visualization to unfold.

- **User Interaction Setup:**

  A VTK render window interactor is configured to handle user interactions during the visualization. This includes setting up the size of the render window and defining the background color for an optimal viewing experience.

- **Color Transfer Function Design:**

  A color transfer function is defined to customize the color map based on the scalar range. This function assigns colors to specific data values, enhancing the visual representation of the simulation data.

- **Structured Grid Creation:**

  A VTK structured grid is created with dimensions corresponding to the shape of the received data. This grid serves as the foundational structure for representing the simulation data in the visualization.

- **Point and Scalar Data Configuration:**

  Points are generated within the grid to represent spatial coordinates, and a VTK array is configured to store scalar data. The 2D simulation data is flattened into a 1D array for seamless integration into the VTK framework.

  In the case of the "Single Socket Visualization," the received 3D data is iterated over, and the 2D elements are processed using the same method mentioned in the previous paragraph.

- **Mapper and Actor Initialization:**

  A VTK mapper is initialized to map the data onto the grid, and an actor is configured to represent the visual entity within the scene. The actor's properties, such as color and opacity, are set to achieve the desired visual appearance.

- **Scene Rendering and Interaction Launch:**

  The configured actor is added to the renderer, and the scene is rendered in the VTK render window. The render window interactor is then initiated to allow user interaction during the visualization. The user is prompted to press 'Enter' to advance to the next time step, ensuring a controlled and interactive visualization experience. (opted against this possibility in our final code.)

- **Communication with Server and User Prompt:**

  After completing the visualization for the current time step, a response is sent to the server, indicating readiness for the next time step. The user is prompted to press 'Enter' to continue to the next time step, maintaining synchronization between the client and server components of the system. (opted against this possibility in our final code.)

4 Bottlenecks

- **Limited Visualization Capacity:**

  A significant constraint arises from our ability to visualize data, which is capped at the maximum size of the array on a given computer. This limitation restricts our capacity to handle larger datasets, especially when dealing with extensive simulation outputs. Additionally, the visualization process is bound by the maximum memory limit of the visualization node, further constraining the scale of data that can be effectively processed.

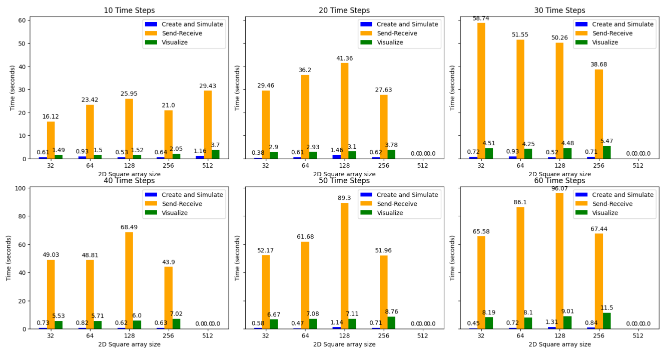
- **Sequential Visualization Process:**

  The current implementation lacks parallelization in the visualization phase. By not harnessing the power of parallel processing, we miss out on significant opportunities to expedite the visualization process. As a consequence, the time required for visualization remains suboptimal, and the potential benefits of parallelizing visualization processes across multiple nodes are not fully realized.

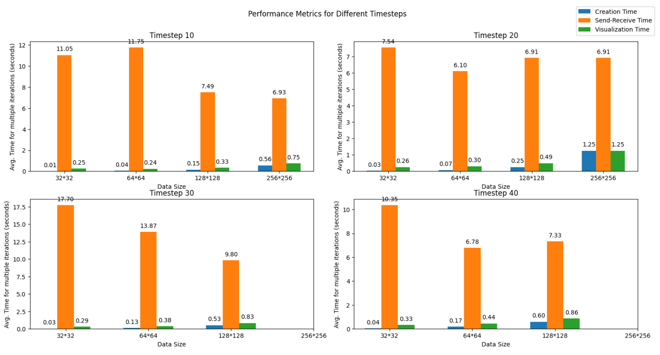
5 Result and Analysis

- Performance was evaluated based on the total time taken to create the data on the server side, send and receive the data on the server and client sides, respectively, and finally visualize the data on the client side.

- Figure 2(a) shows time performance for Multiple Socket approach for time-steps 10, 20, 30, 40, 50, 60, for grid sizes (32*32), (64*64), (128*128), (256*256), (512*512), a single iteration for each timestep.
- Figure 2(b) shows time performance for Single Socket approach for time-steps 10, 20, 30, 40, for grid sizes (32*32), (64*64), (128*128), (256*256), multiple iterations for each timestep.
- As shown in graphs below, the most time is consumed in sending and receiving data for both approaches.
- In Single Socket approach, we observed that the time for sending and receiving data is significantly lowered in comparison to Multiple Socket, contributing to using single socket connection for receiving the entire data, instead of using multiple connections for receiving data for each timestep in the Multiple Socket approach.
- We observed that the connection was being abruptly refused as we increased timestep more that 10 for grid size (512*512) in the Multiple Socket approach .Thus, the 0 value in Figure2(a).
- We also observed that the connection was being abruptly refused as we increased timestep more that 20 for grid size (256*256) in the Single Socket approach .Thus, the no value in Figure2(b).
- The previous two observations lead us to infer that adopting the Single Socket approach improves time efficiency but comes at the expense of scalability in handling large datasets.



(a) Multiple Socket Visualization Result



(b) Single Socket Visualization Result

Figure 2

6 Challenges we faced

- **In Situ Visualization Complexity:**
  - The initial intention was to implement in situ visualization using tools like VisIt LibSim and ParaView Catalyst. However, these tools present a substantial learning curve and complexity, especially for real-time visualization directly from simulation data arrays. Time constraints necessitated an alternative approach, leading us to opt for the Visualization Toolkit (VTK).
  - A critical consideration was that tools like VisIt and ParaView visualize from data files and not directly from the data array. This misalignment posed a challenge to our objective of seamless in situ visualization. The pragmatic decision to utilize VTK, known for its adaptability and efficiency, allowed us to address this specific constraint effectively.
- **Limitations of VTK for Complex Visualization:**

  While leveraging VTK for visualization, we confronted the constraint that complex visualizations, particularly those involving intricate data manipulations, were not feasible within the chosen framework. The need for simplicity in our approach constrained us to implement straightforward visualizations.
- **Challenges with Interactivity in VTK:**

  Creating a highly interactive model proved challenging due to limitations in the framework of VTK. The framework's inherent constraints impacted our ability to achieve a level of interactivity that aligns with the project's goals. Despite its strengths in certain aspects, VTK's framework posed limitations for developing highly interactive visualization models.

7 Future Work

- **Integration of Advanced Visualization Tools:**

  A potential avenue for future work involves integrating advanced visualization tools such as VisIt LibSim and ParaView Catalyst into our existing framework. These tools offer sophisticated capabilities for in situ visualization, enabling more complex and insightful representations of simulation data. Leveraging these tools could open up avenues for enhanced data exploration and interpretation.

- **Exploration of Parallel Visualization Techniques:**

  Given the parallel nature of our simulation and visualization processes (M:N), exploring and implementing advanced parallel visualization techniques can significantly enhance the overall system's efficiency. This exploration may involve optimizing the parallelization of visualization tasks across multiple nodes, ensuring a seamless and expedited visualization experience.

- **Real-time Interaction and Feedback Mechanisms:**

  Implementing real-time interaction features and feedback mechanisms can enhance the user experience and facilitate more interactive exploration of simulation results. This includes features such as dynamically adjusting visualization parameters, exploring different slices of the data, and receiving instant feedback on simulation progress.

- **Scalability Testing and Optimization:**

  Conducting rigorous scalability testing on our system and identifying potential scalability bottlenecks can pave the way for further optimizations. This involves assessing the system's performance under varying data sizes, simulation complexities, and cluster configurations to ensure robust scalability in different environments.

8 Work Distribution

- **Data Streaming:** Ajitesh Shree, Kumar Kanishk Singh, Aayushman Gupta
- **Visualization:** Kundan Kumar, Vala Yash
- **Results and Analysis:** Ajitesh Shree, Kumar Kanishk Singh, Aayushman Gupta

9 Conclusion

In a successful endeavor, we've managed to stream data from the simulation cluster and visualized it effectively on the visualization cluster While our visualizations maintained a simplicity and lacked some interactivity, our approach demonstrated good efficiency This method aims to contribute to the reduction of overall simulation and visualization times, and also makes simulations more widely accessible to scientists, even in locations with limited computational resources It's a step towards practical and accessible scientific exploration