Mini Display Wall (Group 3)

Suprvisors:
Dr. Preeti Malakar
Dr. Soumya Dutta

Group Members:
Madhur Bansal(210572)
Sandeep Parmar(210922)
Paritosh Pankaj(210702)
Tanmay Purohit(211097)

1 Introduction and Motivation

In the realm of modern computing, handling vast datasets efficiently remains a pivotal challenge. Imagine unlocking the potential of numerous devices, each contributing to a collective task: enter the paradigm of dividing colossal data repositories into manageable chunks, distributing these segments across multiple client devices, and orchestrating their simultaneous visualization. This approach represents a groundbreaking leap in parallel computing, fostering scalability, accelerated processing, and resource optimization. By harnessing the combined computational prowess of diverse devices, this method not only expedites data analysis and visualization but also enables real-time insights and fault tolerance, laying the groundwork for swift decision-making and enhanced reliability. As industries grapple with the deluge of big data, this distributed processing and visualization model stands poised to revolutionize analytics, offering reduced latency, enhanced performance, and maximized resource utilization across diverse sectors, from finance and healthcare to research and beyond.

2 Problem Statement and Main Objectives

As a result of this, our group has come up with an approach of constructing a "mini display wall". This approach generates large simulation data on the server and tries to visualize different chunks of the data on different clients. We are going to construct and compare two approaches of building a "mini display wall". These two approaches are:

- In the first approach, we would be simulating the data using parallel processing using multiple processes on our server node and then the data

would be divided into certain parts (4 to be specific) and each part would be transmitted to different client nodes for visualisation. Since the data would be processed with different time steps, the visualisation would be real-time in this method.

- The second approach is to basically construct images in the server node itself after data generation and instead of dividing and sending the data, we partition the image and send the images to different client nodes for visualisation.

After implementing these two approaches, our next goal was to compare the performance of these approaches by altering the number of timesteps and size (resolution) of the generated data.

3 Technical Method

3.1 Data

In this whole project, we have worked with the weather simulation data. This data generates the values of a two dimensional time varying field. It updates the field using principles of advection and diffusion which involves differentials and dependence on previous timestep values. We could look at the below pseudocode of the data generation step.

Algorithm 1 Update Field with Advection and Diffusion

function UpdateField(field, $U0, V0, DT, DX, NX, NY, KX, KY$)
  temp_field $\leftarrow$ copy(field)
  for $i \leftarrow 0$ **to** $NX - 1$ **do**
    for $j \leftarrow 0$ **to** $NY - 1$ **do**
      $i_{\text{prev}} \leftarrow (i - \text{round}(U0 \times DT/DX) + NX) \mod NX$
      $j_{\text{prev}} \leftarrow (j - \text{round}(V0 \times DT/DX) + NY) \mod NY$
      temp_field$[i, j] \leftarrow$ field$[i_{\text{prev}}, j_{\text{prev}}]$
  for $i \leftarrow 0$ **to** $NX - 1$ **do**
    for $j \leftarrow 0$ **to** $NY - 1$ **do**
      $laplacian \leftarrow$ (field$[(i + 1) \mod NX, j]$ + field$[(i - 1 + NX) \mod NX, j]$
        +field$[i, (j + 1) \mod NY]$ + field$[i, (j - 1 + NY) \mod NY]$
        $-4 \times$ field$[i, j])/(DX \times DX)$
      temp_field$[i, j]$ += $(KX \times \text{laplacian} + KY \times \text{laplacian}) \times DT$
  field[:] $\leftarrow$ temp_field ▷ Update the original field
  return field ▷ Return the updated field

3.2 Libraries and utilities

For implementing the two stated approaches, we have used the following libraries and utilities: Python, Numpy, VTK, OpenCV and Socket

3.3 Approach 1

For implementing the first approach we were running a simulation code on python with a certain number of timesteps, every time step yielded a two dimensional array which represented weather simulation data. After the data was produced, the array was divided into certain parts and each part was transferred to different client nodes using TCP/IP sockets. On the client side, after receiving the designated array part, every client visualised the array part using the VTK library.

3.3.1 Server side implementation

Here is the pseudocode for the code of the server code:

Algorithm 2 Server-side Simulation

Initialize Server Socket
Listen for Connections
connected_clients $\leftarrow 0$
client_constants $\leftarrow \{\}$
while *connected_clients* $<$ total_clients **do**
    Receive *client_index* from connected client
    Send simulation constants $(NX, NY, \text{MAX\_TIMESTEPS})$ to client *client_index*
    connected_clients $+= 1$
Initialize Data Field based on received simulation constants
for *timestep* $\leftarrow 1$ **to** MAX_TIMESTEPS **do**
    Update the data field
    for *quadrant* $\leftarrow 1$ **to** 4 **do**
        Divide data field into quadrant *quadrant*
        Create thread for quadrant *quadrant*
        Send data for quadrant *quadrant* to corresponding client
    for *quadrant* $\leftarrow 1$ **to** 4 **do**
        Wait for thread corresponding to quadrant *quadrant* to finish
Close Server Socket

3.3.2 Client Side Implementation

Here is the pseudocode for client side functioning:

Algorithm 3 Client-side Simulation

Connect to Server
Receive constants NX, NY, and MAX_TIMESTEPS from the server
Create Initial Window and Setup Visualization
timestep $\leftarrow$ MAX_TIMESTEPS
while connected to server and timestep > 0 **do**
    Receive data from the server
    Update the visualization with the received data
    Render the window
    timestep $-= 1$
Close Connection to the Server

3.4 Approach 2

In the second approach, after creation of the array, we were converting the array to image format and saving images (not real time). After saving, we partitioned the images into a certain number of parts using openCV and transferred them to other clients for visualisation.

The client and server interaction using sockets is almost same in this approach as well. But here we are saving the images on the disk after generation and then partitioning them into smaller sub-images followed by transversal to different clients. So the image creation is the key. Here is the pseudocode for image generation:

Algorithm 4 Visualization and Distribution

Initialize the data field
Create a VTK window and setup visualization
for *timestamp* $\leftarrow 1$ **to** specified number of timestamps **do**
    Update the data field
    Render the data field
    Save the rendered image to a file
Split the saved images into four parts using OpenCV
for all *image_part* in *split_images* **do**
    Create folder corresponding to *image_part*
    Place *image_part* in the respective folder
Send the subpart-containing folders to the clients (client-server connection)

4 Performance Evaluation

Here we present the graphs and graphs of our experiments which basically comapre the streaming approach and the image transfer approach ang how the approaches scale as we vary the size of the data and the timesteps used for generation:

4.1 Varying the data size

Table 1: Performance Evaluation on varying the data sies

Data Resolution	**64*64**	**128*128**	**256*256**	**512*512**	**1024*1024**
Timestep	100	100	100	100	100
Data Size (mb)	1.6	6.55	26	104	419
Data Streaming Updates (s)	1.673	6.68	27.63	112.62	458.36
Data Streaming Sending (s)	0.703	2.24	8.28	40.08	149.92
Image Transfer Updates (s)	19.61	28.98	57.95	152.81	NA
Image transfer storing	108.98	108.98	108.98	108.98	NA

Here is the visualization of our approaches:

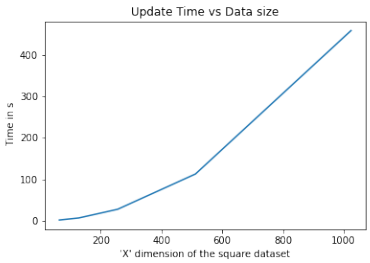


Figure 1: Approach one data generation time

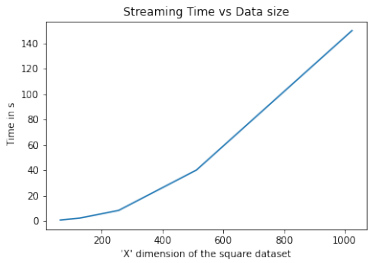


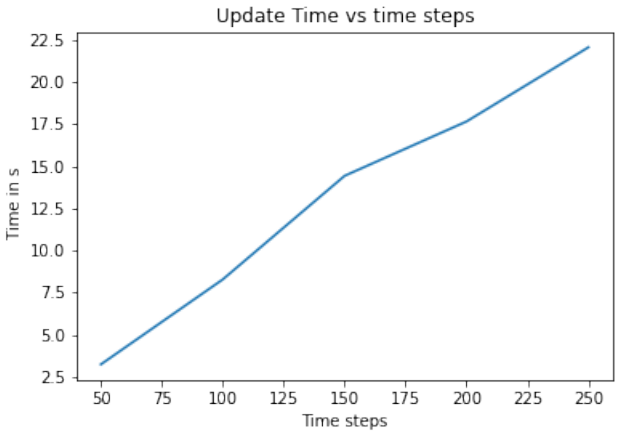
Figure 2: Approach 1 data sending time

4.2 Varying the timesteps

Table 2: Performance Evaluation on varying the timesteps

Data Resolution	**256*256**	**256*256**	**256*256**	**256*256**	**256*256**
Timestep	50	100	150	200	250
Data Size	13.1	26	39	52	65
Data Streaming Updates (s)	13.85	27.56	41.43	55.42	69.13
Sending (s)	3.23	8.28	14.44	17.67	22.08

Now we present a visualization of how the update time varies in seconds, as we gradually increase the number of timesteps from 50 to 250.



5 Challenges faced

5.1 Difficulty in VTK C++ Setup

Setting up VTK (Visualization Toolkit) in a C++ environment was complex due to various dependencies, configurations, and libraries required for VTK integration. It involved issues with library versions, linking, or system-specific configurations, making the setup process challenging and time-consuming.

5.2 Incomplete Data Reception in One recv() Call

When utilizing socket programming or network communication, the recv() function is commonly used to receive data from a sender. However, in certain scenarios, particularly when dealing with larger volumes of data or due to network limitations, we were not be able to retrieve the entire data in a single recv() call. This lead to partial data reception, requiring multiple recv() calls or buffer management to accumulate the complete dataset.

5.3 Network Traffic Impact on Streaming Time

Network traffic, influenced by factors like bandwidth limitations, congestion, or network latency, was significantly impacting data streaming performance. High network traffic or congestion lead to increased latency and slower data transmission rates, affecting the time taken to stream data between the server and client. This resulted in delays in receiving or transmitting data, ultimately impacting the overall streaming time for visualization or data transfer tasks. As a result, we averaged our observations over 5 runs to reduce uncertainity in our data.

5.4 Visit (Libsim Approach) Challenges

Visit, especially when using its libsim library, might present challenges related to its integration, configuration, or compatibility with the existing system setup. It is more complex to use in windows environment, people working in linux find it quite easy to integrate with their setup, hence we decided to go with VTK for visualization.

6 Conclusion and Future work

Concluding our work, we find that both the approaches have their own advantages. The first approach can be useful when we want to perform in-situ visualization of the approach, that is, visualize the data real time and the resolution of the images visualized will be better as compared to the other approach. The second approach, on the other hand, could be helpful in other approaches. We could say that if the size of the data is very large then the sending time would be very large and the visualization of the data live on the client side may not be very smooth.

We also present some directions of future work to our contributions:

- Data generation on server side can be done parallely using some distribute memory libraries like MPI.
- Visualisation of array can be improved
- We can extend our problem statement to 3 dimensional data.

7 Github link and References

Here is the github link for the report: **Group 3- Mini Display Wall**

References used in the project:

- **VTK resources**
- **TCP IP sockets**