Report

On

CS677 Assignment 2: Parallel 3D Volume Rendering

Group No: 9

Krishna Kumar Bais

(241110038)

Arnika Kaithwas

(241110011)

Pradeep Sahu

(241110050)

Department of Computer Science and Engineering

Indian Institute of Technology

Kanpur

1. Introduction

This report explains the design and results of a parallel 3D volume rendering system created using MPI (Message Passing Interface). The rendering method uses a ray-casting approach, with front-to-back compositing and an orthogonal projection along the XY plane. To handle large datasets efficiently, the program splits the input data into smaller parts, known as sub-volumes, and assigns each part to a different MPI process. Each process performs ray-casting on its specific sub-volume and builds partial images that contribute to the final result. Once all processes complete their calculations, they combine their partial images to form a single, complete representation of the 3D volume. The final output is saved as a PNG image. This parallel approach distributes the workload across multiple processes, making the rendering faster and suitable for complex 3D data in a parallel computing environment.

2. Implementation Details

2.1 Parallel 3D Volume Rendering Algorithm: A parallel 3D volume rendering algorithm efficiently visualizes 3D data by distributing the workload across multiple processors, allowing for faster processing, especially with large datasets.

- **Data Partitioning:** Split the 3D dataset into sub-volumes and assign each to a separate MPI process.
- **Ray-Casting:** Each process casts rays through its assigned sub-volume, accumulating color and opacity values for each voxel along the ray path, using **front-to-back** compositing.
- **Local Image Generation:** Each process generates a partial 2D image based on its sub-volume.
- **Compositing Partial Results**: Gather and blend partial images from all processors to form a consistent final image.
- **Final Output**: Save the fully composited image.

2.2 Performance Optimization

1. **Data Distribution:** Rank 0 reads the dataset and partitions it across X, Y, and Z dimensions, then sends each sub-volume to the respective process. Communication between processes is minimized by ensuring each process operates on its local sub-volume until the gathering stage.
2. **Parallel Composition:** The use of MPI's for collecting the sub-images is optimized to reduce communication bottlenecks.

3. **Scalability:** The program is designed to handle increasing process counts, demonstrating efficient load distribution and scalability.

3.Execution Setup

3.1 Dataset Description: There are two datasets which are used in this assignment. The first dataset used is “Isabel_1000x1000x200_float32.raw”, with following dimensions:

X: 1000

Y: 1000

Z: 200

The Second dataset used is very large compared to first dataset “Isabel_2000x2000x400_float32.raw”, with the following dimensions:

X: 2000

Y: 2000

Z: 400

3.2 Execution Commands: The following commands were executed on the csews cluster, using multiple processes as specified:

- **Test Case 1:** 8 Processes (2 x 2 x 2 decomposition)

```
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 8 --oversubscribe python3 code.py  Isabel_1000x1000x200_float32.raw 2 2 2 0.5 opacity_TF.txt color_TF.txt | tee
 output8.txt
```

- **2. Test Case 2:** 16 Processes (2 x 2 x 4 decomposition)

```
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 16 --oversubscribe python3 code.py  Isabel_1000x1000x200_float32.raw 2 2 4 0.5 opacity_TF.txt color_TF.txt | te
e output16.txt
```

- **Test Case 3:** 32 Processes (2 x 2 x 8 decomposition)

```
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 32 --oversubscribe python3 code.py  Isabel_1000x1000x200_float32.raw 2 2 8 0.5 opacity_TF.txt color_TF.txt | te
e output32.txt
```

These screenshots of commands are for the first dataset and for the second dataset the similar commands are used except the dataset file name.

For the second dataset, for the 32 processes the command used is shown below:

```
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 32 --oversubscribe python3 code.py  Isabel_2000x2000x400_float32.raw 2 2 8 0.5 opacity_TF.txt color_TF.txt | te
e LargeOutput32.txt
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
```

3.3 Host Configuration:

The hostfile is used by MPI to specify which machines (or nodes) will run the parallel processes and how many processes will be allocated to each machine. It typically contains a list of hostnames or IP addresses, along with the number of processes to run on each. The format we used:

172.27.19.5:8

172.27.19.1:8

172.27.19.6:8

172.27.19.9:8

This configuration allocates 8 processes to each of the four machines (172.27.19.5, 172.27.19.1, 172.27.19.6, 172.27.19.9), resulting in a total of 32 processes.

4.Code Description

4.1 Importing Libraries:

```
import warnings
warnings.simplefilter("ignore", UserWarning)
import socket
from mpi4py import MPI
import numpy as np
import sys
from PIL import Image
```

- **warnings**: Suppresses user warnings to avoid cluttering the output.
- **socket**: Retrieves the hostname of the machine running the process.
- **mpi4py**: Provides MPI functionalities for parallel processing.
- **numpy**: Facilitates numerical operations and array manipulations.
- **sys**: Handles command-line arguments.
- **PIL.Image**: Used for image creation and saving.

4.2 Linear Interpolation Function: The linear_interpolate function performs interpolation for color and opacity values based on a mapping table:

```
def linear_interpolate(value, table, is_color=False):
    """ Linearly interpolates a value based on a provided table of mapping
"""
    for i in range(len(table) - 1):
        x0, y0 = table[i]
        x1, y1 = table[i + 1]
        if x0 <= value <= x1:
            t = (value - x0) / (x1 - x0)
            return [(1 - t) * c0 + t * c1 for c0, c1 in zip(y0, y1)] if
is_color else (1 - t) * y0 + t * y1
    return [0, 0, 0] if is_color else 0
```

- It takes an input value, a table of mappings, and a boolean is_color.
- The function iterates through the mapping table to find the appropriate range for interpolation.
- It calculates an interpolation factor t*t* and returns the interpolated value or color.

4.3 Main Functionality

The `main` function orchestrates the overall volume rendering process:

```
def main():
    """ Main function for MPI-based volume rendering. """
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    num_procs = comm.Get_size()
    print(f"Rank {rank} running on {socket.gethostname()}")
```

- Initializes MPI and retrieves the rank (process ID) and total number of processes.
- Each process prints its rank and hostname for identification.

4.4 Argument Parsing and Validation:

```
if len(sys.argv) < 8:
    if rank == 0:
        print("Usage: mpirun -np <num_procs> python file.py <dataset_name>
<X_parts> <Y_parts> <Z_parts> <step_size> <opacity_tf> <color_tf>")
    sys.exit()

dataset_name, X_parts, Y_parts, Z_parts, step_size, opacity_file, color_fi
= sys.argv[1:8]
X_parts, Y_parts, Z_parts = int(X_parts), int(Y_parts), int(Z_parts)
step_size = float(step_size)
```

- Checks if sufficient command-line arguments are provided; otherwise, it prints usage instructions.
- Parses input arguments related to dataset name, dimensions for splitting the volume, step size for raycasting, and file names for opacity and color transfer functions.

4.5 Volume Dimensions Mapping:

```
volume_dims_map = {
    "1000x1000x200": (1000, 1000, 200),
    "2000x2000x400": (2000, 2000, 400),
}

for key in volume_dims_map:
    if key in dataset_name:
        volume_dims = volume_dims_map[key]
        break
```

- Defines a mapping of known dataset dimensions to their respective tuples.
- Determines the dimensions of the volume based on the dataset name.

4.6 Loading Transfer Functions:

The code loads opacity and color transfer functions from specified files:

```
opacity_map = []
with open(opacity_file, 'r') as file:
    values = [float(v) for line in file for v in line.replace(',',
'').strip().split()]
    opacity_map = [(values[i], values[i + 1]) for i in range(0, len(values),
2)]

color_map = []
with open(color_file, 'r') as file:
    values = [float(v) for line in file for v in line.replace(',',
'').strip().split()]
    color_map = [(values[i], (values[i + 1], values[i + 2], values[i + 3]))
for i in range(0, len(values), 4)]
```

- Reads opacity values as pairs from opacity_file.
- Reads color values as tuples from color_file, which includes RGB components.

4.7 Data Volume Loading and Distribution:

The code loads the data volume and prepares it for distribution among processes:

```
local_subvolume = None

if rank == 0:
    data_volume = np.fromfile(dataset_name,
dtype=np.float32).reshape(volume_dims, order='F')
    sub_volumes = [np.array_split(slice, Y_parts, axis=1) for slice in
np.array_split(data_volume, X_parts, axis=0)]
```

- Only rank 0 loads the entire data volume from the specified file.
- The data volume is split into sub-volumes based on specified dimensions.

4.8 Sending Sub-volumes to Processes:

```
send_start = MPI.Wtime()
for i in range(X_parts):
    for j in range(Y_parts):
        for k in range(Z_parts):
            target_rank = i * Y_parts * Z_parts + j * Z_parts + k
            portion = sub_volumes[i][j][:, :, k::Z_parts]
            if target_rank == 0:
                local_subvolume = portion
            else:
                comm.send(portion, dest=target_rank, tag=target_rank)
send_time = MPI.Wtime() - send_start
```

- Each sub-volume is sent to its corresponding target process using comm.send.
- Rank 0 directly assigns its portion to local_subvolume.

4.9 Receiving Sub-volumes:

```
else:
    local_subvolume = comm.recv(source=0, tag=rank)

if local_subvolume is None:
    print(f"Rank {rank} received no data.")
    sys.exit()
```

- Non-root ranks receive their assigned sub-volumes using comm.recv.
- If a rank does not receive any data successfully, it exits with an error message.

4.10 Rendering Process:

```
h, w, d = local_subvolume.shape
local_img = np.zeros((h, w, 3))
render_start = MPI.Wtime()

for col in range(w):
    for row in range(h):
        color_accum = np.zeros(3)
        opacity_accum = 0
        z_pos = 0.0

        while z_pos < d:
            z_int = int(z_pos)
            z_int_next = min(z_int + 1, d - 1)
            ratio = z_pos - z_int

            value = (1 - ratio) * local_subvolume[row, col, z_int] + ratio *
local_subvolume[row, col, z_int_next

            color = np.array(linear_interpolate(value, color_map,
is_color=True))
            opacity = linear_interpolate(value, opacity_map)

            color_accum += (1 - opacity_accum) * color * opacity
            opacity_accum += (1 - opacity_accum) * opacity

            if opacity_accum >= 0.98:
                break

            z_pos += step_size

        local_img[row, col, :] = color_accum
computation_time = MPI.Wtime() - render_start
```

- Initializes an image array (local_img) to store pixel colors.
- For each pixel in the sub-volume:
    - Performs raycasting by iterating through depth slices.
    - Interpolates intensity values and retrieves corresponding colors and opacities.
    - Accumulates colors based on their opacities until reaching a threshold.

4.11 Gathering Results:

After rendering is complete:

```
gather_start = MPI.Wtime()
gathered_imgs = comm.gather(local_img, root=0)
gather_time = MPI.Wtime() - gather_start

max_times = comm.reduce([computation_time, send_time, recv_time,
                         gather_time,
                         MPI.Wtime() - start_time_total], op=MPI.MAX)

if rank == 0:
    final_img_shape...
```

- Each process sends its rendered image back to rank 0 using comm.gather.
- The root process assembles all sub-images into a final composite image.

4.12 Final Image Assembly and Saving:

```
if rank == 0:
    final_img...

output_file = f"{X_parts}_{Y_parts}_{Z_parts}.png"
final_image...
Image.fromarray((final_image * 255).astype(np.uint8)).save(output_file)
print(f"Image saved as {output_file}")
```

- The final image is constructed from gathered images by accumulating colors while respecting transparency.

4.13 Execution Time Reporting

```
labels...
print("\nExecution Times (seconds):\n" + "\n".join([f"{lbl:<25} | {t:.4f}"
for lbl,t in zip(labels,max_times)]))
```

- Execution times for different parts of the process are printed out to provide performance insights.

5.Results and Analysis

5.1 Outputs For First Dataset:

Here are the output of final images and the Output of communication time and computation time and total execution time shown below:

- **Test Case 1:** 8 Processes (2 x 2 x 2 decomposition)

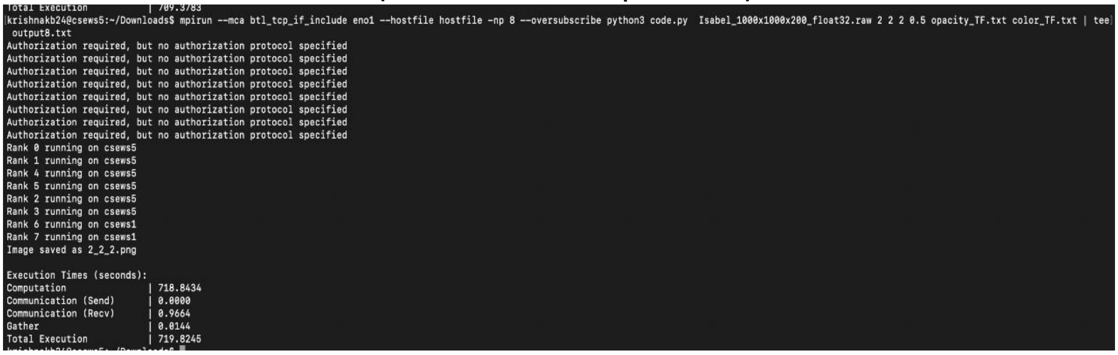


Figure 1: first_run

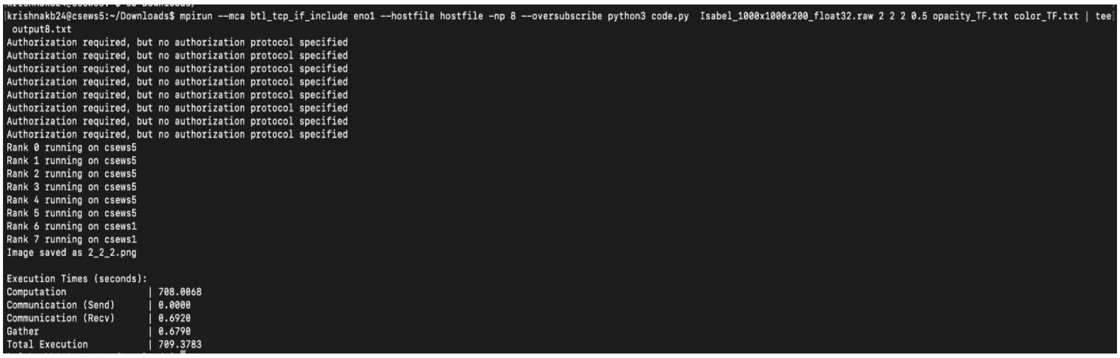


Figure 2: Second_Run

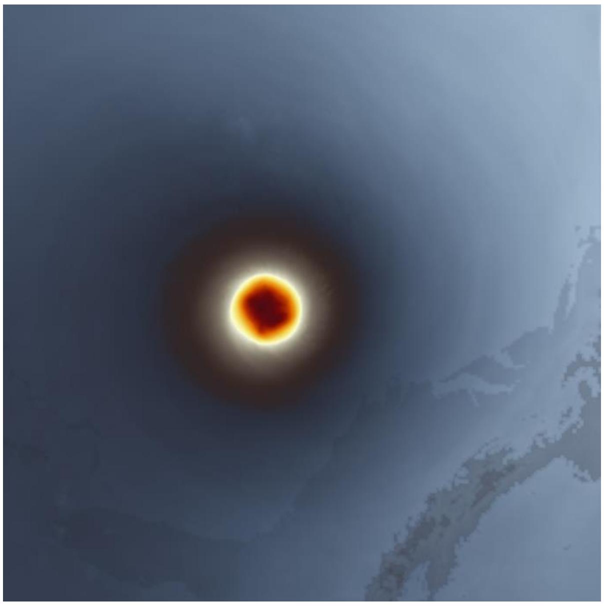
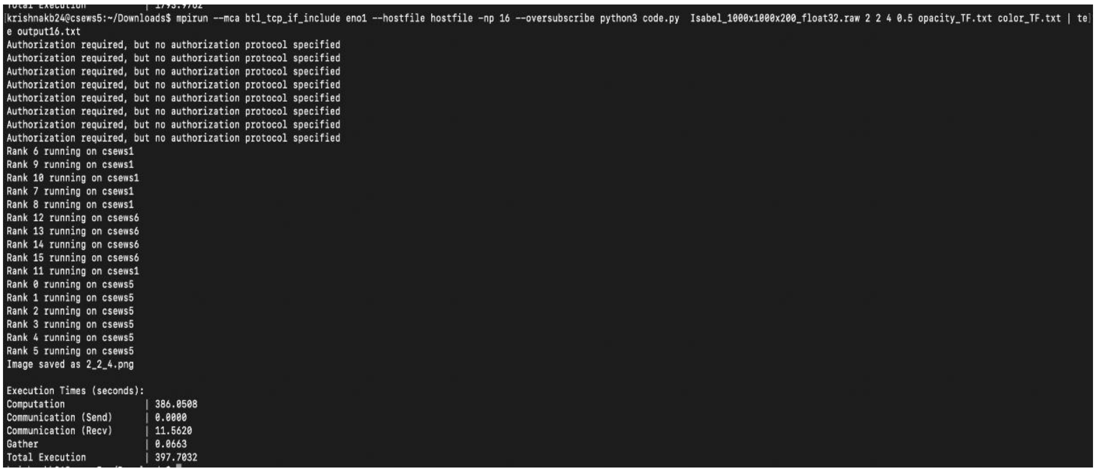


Figure 3: 2_2_2(Output_image)

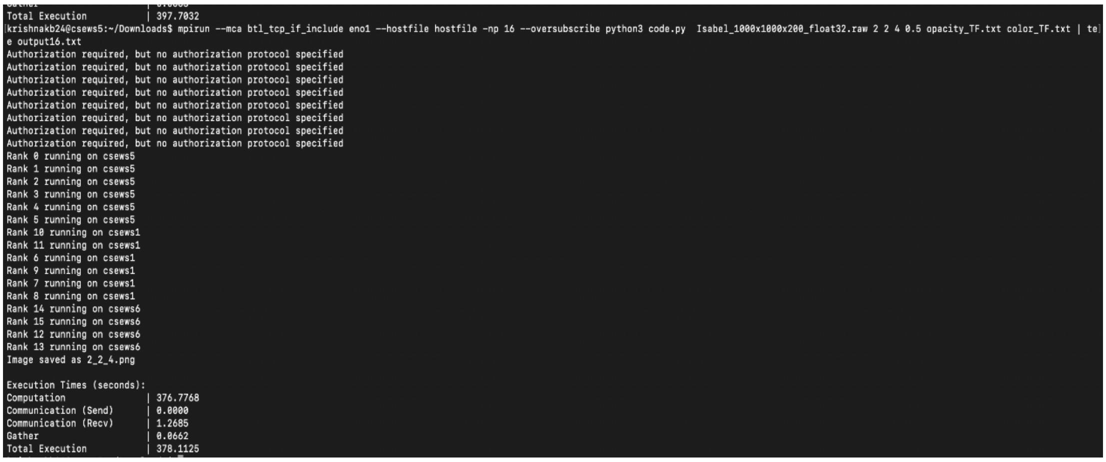
- **Test Case 2:** 16 Processes (2 x 2 x 4 decomposition)



```
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 16 --oversubscribe python3 code.py  Isabel_1000x1000x200_float32.raw 2 2 4 0.5 opacity_TF.txt color_TF.txt | te
e output16.txt
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Rank 6 running on csews1
Rank 9 running on csews1
Rank 10 running on csews1
Rank 7 running on csews1
Rank 8 running on csews1
Rank 12 running on csews6
Rank 13 running on csews6
Rank 14 running on csews6
Rank 15 running on csews6
Rank 11 running on csews1
Rank 0 running on csews5
Rank 1 running on csews5
Rank 2 running on csews5
Rank 3 running on csews5
Rank 4 running on csews5
Rank 5 running on csews5
Image saved as 2_2_4.png

Execution Times (seconds):
Computation              | 386.0508
Communication (Send)     | 0.0000
Communication (Recv)     | 11.5620
Gather                   | 0.0663
Total Execution          | 397.7032
```

Figure 4: first_Run



```
Total Execution          | 397.7032
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 16 --oversubscribe python3 code.py  Isabel_1000x1000x200_float32.raw 2 2 4 0.5 opacity_TF.txt color_TF.txt | te
e output16.txt
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Rank 0 running on csews5
Rank 1 running on csews5
Rank 2 running on csews5
Rank 3 running on csews5
Rank 4 running on csews5
Rank 5 running on csews5
Rank 10 running on csews1
Rank 11 running on csews1
Rank 6 running on csews1
Rank 9 running on csews1
Rank 7 running on csews1
Rank 8 running on csews1
Rank 14 running on csews6
Rank 15 running on csews6
Rank 12 running on csews6
Rank 13 running on csews6
Image saved as 2_2_4.png

Execution Times (seconds):
Computation              | 376.7768
Communication (Send)     | 0.0000
Communication (Recv)     | 1.2685
Gather                   | 0.0662
Total Execution          | 378.1125
```

Figure 5: Second_Run

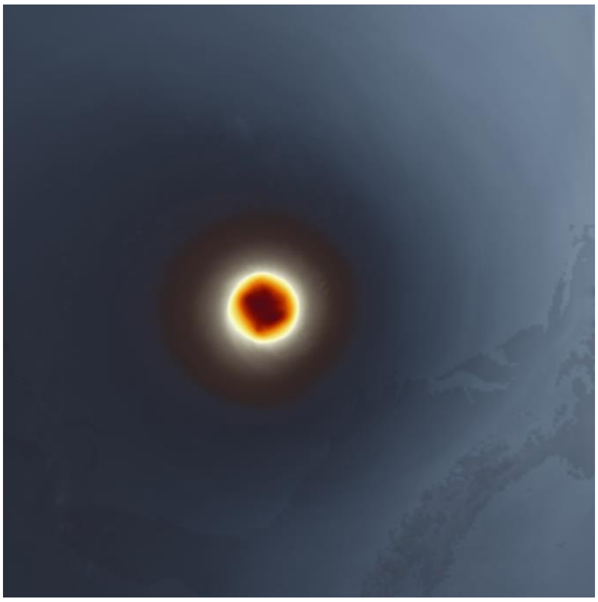


Figure 6: 2_2_4(Output_Image)

- **Test Case 3:** 32 Processes (2 x 2 x 8 decomposition)

```
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Rank 1 running on csews5
Rank 2 running on csews5
Rank 5 running on csews5
Rank 7 running on csews5
Rank 3 running on csews5
Rank 4 running on csews5
Rank 6 running on csews5
Rank 0 running on csews5
Rank 9 running on csews1
Rank 12 running on csews1
Rank 13 running on csews1
Rank 14 running on csews1
Rank 15 running on csews1
Rank 8 running on csews1
Rank 10 running on csews1
Rank 11 running on csews1
Rank 20 running on csews6
Rank 21 running on csews6
Rank 18 running on csews6
Rank 19 running on csews6
Rank 22 running on csews6
Rank 23 running on csews6
Rank 16 running on csews6
Rank 17 running on csews6
Rank 27 running on csews9
Rank 30 running on csews9
Rank 26 running on csews9
Rank 29 running on csews9
Rank 24 running on csews9
Rank 25 running on csews9
Rank 28 running on csews9
Rank 31 running on csews9
Image saved as 2_2_8.png

Execution Times (seconds):
Computation              | 414.5462
Communication (Send)     | 0.0000
Communication (Recv)     | 1.9755
Gather                   | 0.2371
Total Execution          | 416.7611
krishnakb24@csews5:~/Downloads$
```

Figure 7: First_Run

```
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Rank 6 running on csews5
Rank 7 running on csews5
Rank 0 running on csews5
Rank 1 running on csews5
Rank 2 running on csews5
Rank 3 running on csews5
Rank 4 running on csews5
Rank 5 running on csews5
Rank 8 running on csews1
Rank 14 running on csews1
Rank 27 running on csews9
Rank 30 running on csews9
Rank 10 running on csews1
Rank 11 running on csews1
Rank 13 running on csews1
Rank 15 running on csews1
Rank 12 running on csews1
Rank 9 running on csews1
Rank 25 running on csews9
Rank 28 running on csews9
Rank 29 running on csews9
Rank 31 running on csews9
Rank 24 running on csews9
Rank 26 running on csews9
Rank 20 running on csews6
Rank 22 running on csews6
Rank 23 running on csews6
Rank 21 running on csews6
Rank 16 running on csews6
Rank 17 running on csews6
Rank 18 running on csews6
Rank 19 running on csews6
Image saved as 2_2_8.png

Execution Times (seconds):
Computation              | 419.9810
Communication (Send)     | 0.0000
Communication (Recv)     | 1.7739
Gather                   | 0.1712
Total Execution          | 421.9266
krishnakb24@csews5:~/Downloads$
```

Figure 8: Second_Run

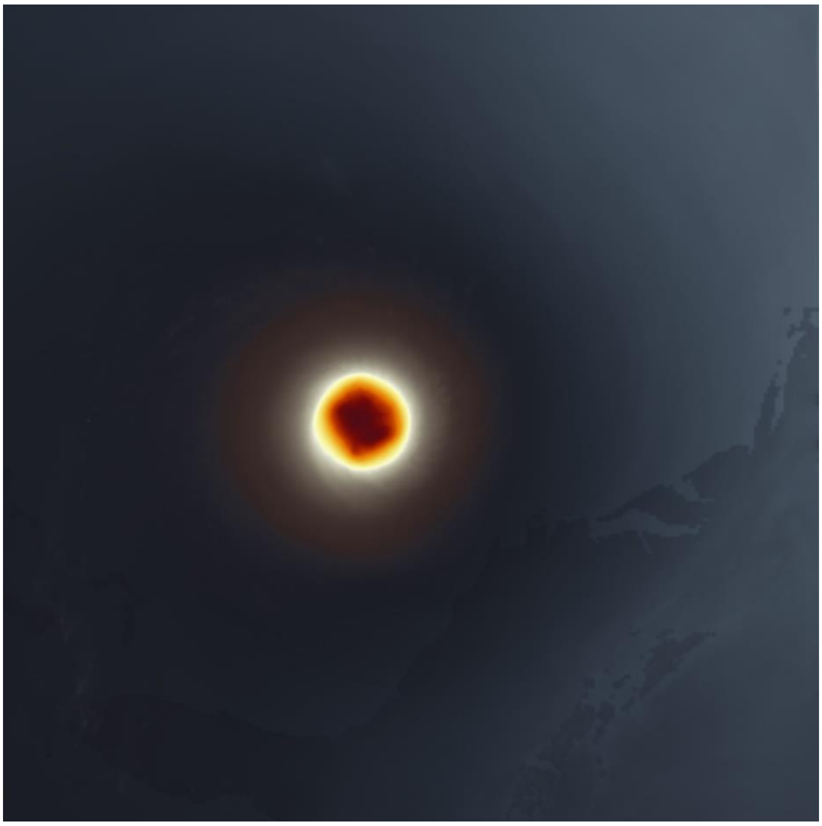


Figure 9: 2_2_8(Output_Image)

5.2 Timing Analysis (First Dataset)

1. Computation Time:

- This is the time each process spends on actual volume rendering computations. It involves performing raycasting on its assigned sub-volume, calculating color and opacity values through linear interpolation, and accumulating these values along each ray. The computation time is measured from the start to the end of rendering for each process.

2. Communication Time:

Communication time is the combination of send time and receive time:

- Send Time: The time taken by Rank 0 to distribute sub-volumes to each process.
- Receive Time: The time each non-zero rank spends receiving its assigned sub-volume from Rank 0.

3. Total Execution Time:

- This is the overall elapsed time from the start of the program until the final image is rendered and saved.

- Total execution time includes both computation and communication times, along with any additional I/O or setup time.
- The total execution time for the first dataset represents the combined time spent across all stages of the program (data loading, sub-volume distribution, rendering, gathering, and image saving).

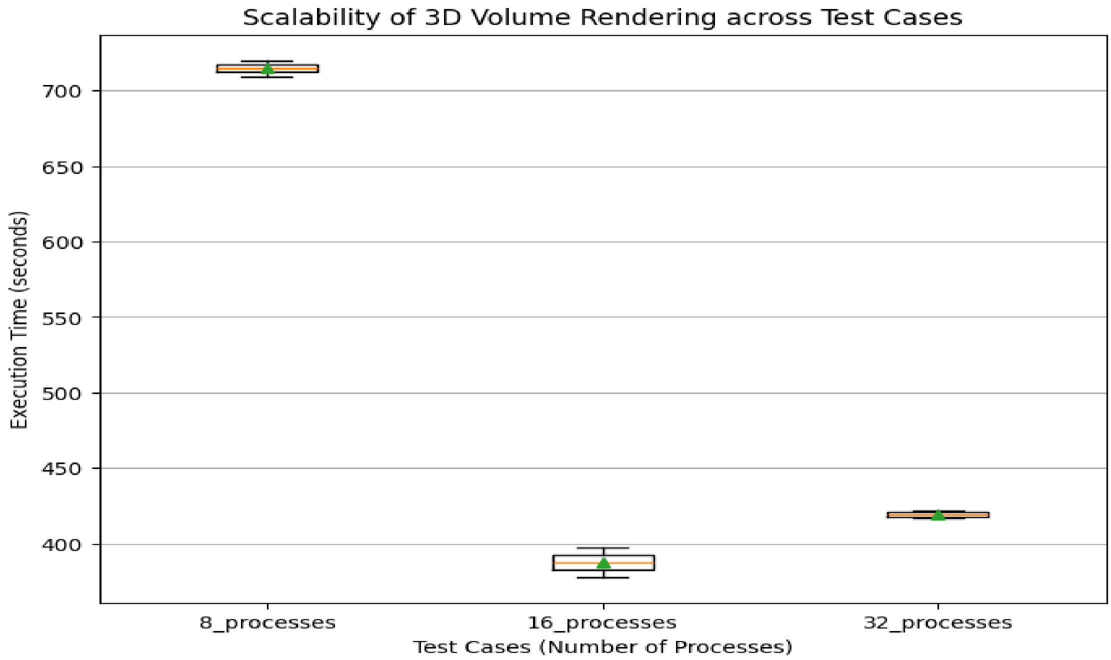
For each test case, we recorded computation, send, receive, and total execution times. Below is a table summarizing the maximum time taken by any process in each category across both runs for each test case.

Test Cases	Computation(s)	Send(s)	Receive(s)	Total Execution(s)
8 Processes	718.8434	0.0000	0.9664	719.8245
	708.0068	0.0000	0.6920	709.3783
16 Processes	386.0508	0.0000	11.5620	397.7032
	376.7768	0.0000	1.2685	378.1125
32 Process	414.5462	0.0000	1.9755	416.7611
	419.9810	0.0000	1.7739	421.9266

Table 1: Showing the time split (compute, communication, total)

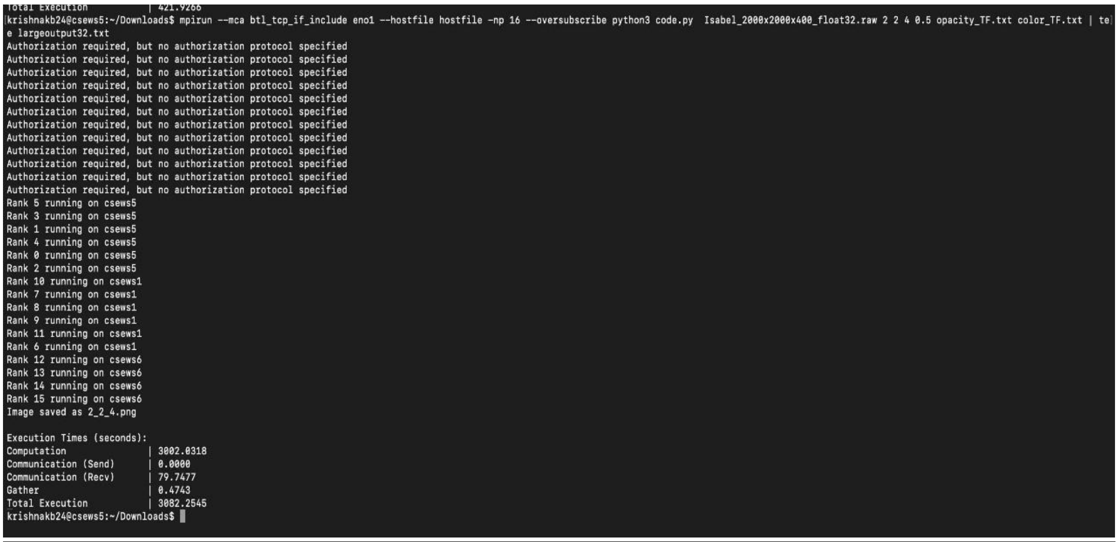
5.3 Scalability Plot:

The box plot below illustrates the scalability across the three test cases. For each configuration, the box represents the variation in total execution times observed across the two runs, showing how performance changes as the number of processes increases.



5.4 Outputs (For Second Dataset):

- **Test Case 2:** 16 Processes



```
Total Execution          | 421.9266
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 16 --oversubscribe python3 code.py  Isabel_2000x2000x400_float32.raw 2 2 4 0.5 opacity_TF.txt color_TF.txt | te
e largeoutput32.txt
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Rank 5 running on csews5
Rank 3 running on csews5
Rank 1 running on csews5
Rank 4 running on csews5
Rank 0 running on csews5
Rank 2 running on csews5
Rank 10 running on csews1
Rank 7 running on csews1
Rank 8 running on csews1
Rank 9 running on csews1
Rank 11 running on csews1
Rank 6 running on csews1
Rank 12 running on csews6
Rank 13 running on csews6
Rank 14 running on csews6
Rank 15 running on csews6
Image saved as 2_2_4.png

Execution Times (seconds):
Computation              | 3002.0318
Communication (Send)     | 0.0000
Communication (Recv)     | 79.7477
Gather                   | 0.4743
Total Execution          | 3082.2545
krishnakb24@csews5:~/Downloads$
```

Figure 10

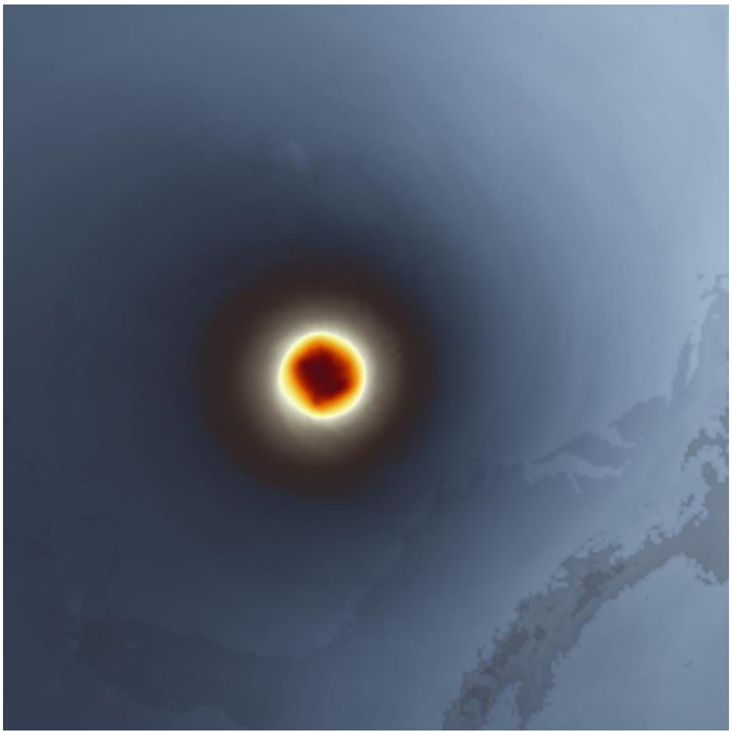


Figure 11: 2_2_4(Output_Image)

- **Test Case 3:** 32 Processes (2 x 2 x 8 Decomposition)

```
krishnakb24@csews5:~/Downloads$ mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 32 --oversubscribe python3 code.py  Isabel_2000x2000x400_float32.raw 2 2 8 0.5 opacity_TF.txt color_TF.txt | te
e LargeOutput32.txt
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Rank 14 running on csews1
Rank 8 running on csews1
Rank 9 running on csews1
Rank 10 running on csews1
Rank 12 running on csews1
Rank 15 running on csews1
Rank 11 running on csews1
Rank 13 running on csews1
Rank 27 running on csews9
Rank 28 running on csews9
Rank 26 running on csews9
Rank 31 running on csews9
Rank 24 running on csews9
Rank 29 running on csews9
Rank 25 running on csews9
Rank 30 running on csews9
Rank 0 running on csews5
Rank 2 running on csews5
Rank 5 running on csews5
Rank 7 running on csews5
Rank 6 running on csews5
Rank 1 running on csews5
Rank 3 running on csews5
Rank 4 running on csews5
Rank 18 running on csews6
Rank 16 running on csews6
Rank 17 running on csews6
Rank 20 running on csews6
Rank 21 running on csews6
Rank 19 running on csews6
Rank 22 running on csews6
Rank 23 running on csews6


Image saved as 2_2_8.png

Execution Times (seconds):
Computation              | 4080.8892
Communication (Send)     | 0.0000
Communication (Recv)     | 41.7697
Gather                   | 2.0162
Total Execution          | 4124.6757
```

Figure 12: Here the output log is so long so we have captured it in multiple screenshots.

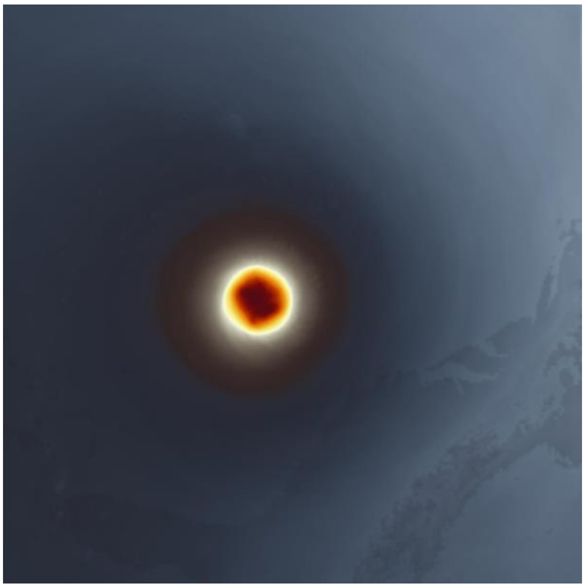


Figure 13: 2_2_8(Output_Image)

5.5 Timing Analysis(Large Dataset):

Test Cases	Computation	Send	Receive	Total Execution
32 Processes	4080.8892	0.0000	41.7697	4124.6757
16 Processes	3002.0318	0.0000	79.7477	3082.2545

6.Observations

6.1 For Isabel_1000x1000x200_float32.raw

6.1.1 Scalability:

- The execution time decreases as the number of processes increases from 8 to 16, indicating effective scalability. This is evident in the reduction of total execution time from approximately 719.82 seconds (8 processes) to 397.70 seconds (16 processes).
- However, the improvement in execution time is less substantial when moving from 16 to 32 processes. The total execution time only drops from 397.70 seconds (16 processes) to 416.76 seconds (32 processes) in one run and actually increases slightly in another. This diminishing return is likely due to increased communication overhead, especially during the gather phase, which affects performance at higher process counts.

6.1.2 Load Imbalance:

In the 32-process configuration, the computation times are not fully balanced across all processes. This imbalance likely happens because the 3D dataset doesn’t divide evenly across the processes. When the data dimensions don’t split perfectly, some processes end up with slightly larger sections (or "sub-volumes") to process. As a result, these processes take a bit longer to finish their work, causing small differences in computation times.

6.1.3 Communication Overhead:

As the number of processes increases, the time spent on communication also grows. Specifically, the time it takes to receive data varies between different configurations. For example, with 16 processes, the receive time is quite high, reaching up to 11.56 seconds, whereas with 32 processes, the time drops significantly to around 1.97 seconds. This increase in communication time as more processes are added suggests that collecting data becomes more complicated with a larger number of processes. This indicates that improving how communication is handled could help reduce delays, especially as the number of processes increases.

6.2 For Isabel_2000x2000x400_float32.raw

6.2.1 Scalability:

- Increasing the number of processes from 8 to 16 reduces the total execution time, showing good scalability.
- However, going from 16 to 32 processes doesn't help as much. In fact, the execution time sometimes increases slightly. This is likely due to added communication delays when using more processes.

6.2.2 Load Imbalance: With 32 processes, computation times vary slightly between processes because the 3D dataset doesn't split evenly. Some processes end up with larger sections to handle, causing a slight delay in finishing their tasks.

7.Conclusion

The volume rendering algorithm works well and performs efficiently when using a moderate number of processes. As you increase the number of processes, the execution time goes down, which is expected because more processes share the workload. However, when the number of processes gets very high, the time spent on communication between processes starts to reduce the benefits of adding more processes. To improve performance further, future updates could focus on making the compositing step more efficient and better balancing the workload across processes, reducing communication time and improving overall speed.