CS677 : Assignment - 2

TOPICS IN LARGE DATA ANALYSIS AND VISUALIZATION

Parallel 3D Volume Rendering

GROUP 8	Divyanshu 241110023	Khushwant Kaswan 241110035	Senthil Ganesh P 241110089

Table of Contents

Brief Overview of our Task

We are given a 3D scalar dataset and we need to perform volume rendering by doing ray casting using front to back compositing method and merge parallely to form final image.

The domain will be decomposed in 3 dimensions. Each subdomain will be handled by 1 process to perform the ray casting.

x-direction will have PX processes, y will have PY and z will have PZ processes.

Code Sequence :

1. Dataset read by rank 0 and transformed either to 1000x1000x200 or 2000x2000x400 shape accordingly.
2. Rank 0 performs 3D Domain Decomposition on the dataset and sends subdomains to respective ranks.
3. RayCasting on the subdomains in parallel by each rank returning a subimage.
4. Parallel composition of subimages using Binary swap like technique to form the final image array.
5. Rank 0 generate the png file from the final image matrix and outputs all respective timings.

The output of our program is:

- Volume rendered image of the dataset
- Time taken for data decomposition and distribution
- Maximum time taken for computations across all ranks
- Maximum time taken for communication across all ranks
- Difference between max and min total execution time across all ranks
- Variance of the total execution time across all ranks
- Total time taken for code execution

We have used the difference between max and min total execution time across all ranks and Variance of the total execution time across all ranks as the means to show **load imbalance**.

Language Used : Python

Packages/libraries used : mpi4py, numpy, pillow, sys, math, numba

Numba is a just-in-time (JIT) compiler for Python that translates Python functions (mostly those that work on NumPy arrays) into optimized machine code at runtime, making them significantly faster. Using Numba, we can speed up certain parts of the code, especially those with heavy computations or nested loops.

For our code, Numba is useful in functions of ray_casting, interpolate_color, and interpolate_opacity. The @jit decorator in Numba is used to specify that the function should be JIT-compiled.

```
@jit(nopython=True)
```

nopython=True: This mode forces Numba to avoid using the Python interpreter inside the function, which allows for maximum speed gains.

The JIT-compiled ray_casting, interpolate_color, and interpolate_opacity functions resulted in a significant speedup in our code,

Implementation

Input Parameters

To handle input parameters as shown in test example, we have used Python **sys module**. If the number of inputs are less, we print and exit

```
    if len(sys.argv) < 8:
        if rank == 0:
            print("Usage: mpirun -np <num_procs> python file_name.py <dataset_name>
<PX> <PY> <PZ> <step_size> <opacity_tf> <color_tf>")
        sys.exit()

    dataset_name = sys.argv[1]
    PX = int(sys.argv[2])
    PY = int(sys.argv[3])
    PZ = int(sys.argv[4])
    step_size = float(sys.argv[5])
    opacity_tf_filename = sys.argv[6]
    color_tf_filename = sys.argv[7]
```

Shape of dataset

We are extracting the dataset shape from the dataset_name given as input and raise error when dataset name doesn't contain known dimensions.

```
def get_shape_from_dataset_name(dataset_name):
    if "1000x1000x200" in dataset_name:
        return (1000, 1000, 200)
    elif "2000x2000x400" in dataset_name:
        return (2000, 2000, 400)
    else:
        raise ValueError("Dataset name doesn't match known dimensions.")

shape = get_shape_from_dataset_name(dataset_name)
```

Reading Dataset and pad if necessary

Earlier we have used numpy library to read the .raw file and reshaped it to the shape we got earlier.

But now as we will be doing 3D decomposition, to handle uneven cases we have padded the dataset such that dataset shape ie 1000 or 2000 is divisible by PX or PY whatever necessary.

```
def load_data(filename, shape,px,py):
    data=np.memmap(filename, dtype=np.float32, mode='r', shape=shape, order='F')
    padded_data, original_shape = pad_data(data, px, py)
    return padded_data, original_shape
```

```
def pad_data(data, px, py):
    original_shape = data.shape
```

```
    # Calculate the target shape
    target_height = np.ceil(original_shape[0] / px).astype(int) * px
    target_width = np.ceil(original_shape[1] / py).astype(int) * py

    # Calculate padding widths
    pad_height = target_height - original_shape[0]
    pad_width = target_width - original_shape[1]

    # Pad heights and widths with (before, after) tuple
    pad_widths = [(0, pad_height), (0, pad_width), (0, 0)]  # No padding for z-axis

    # Pad the data
    padded_data = np.pad(data, pad_widths, mode='constant', constant_values=0)

    return padded_data, original_shape
```

This padding can be easily removed from final stiched image based on the original shape.

```
def unpad_image(final_image, original_shape):
    return final_image[:original_shape[0], :original_shape[1], :]
```

We took `order='F'` after checking the data point distribution in Paraview spreadsheet view.

Data Decomposition and Distribution

```
# Splitting data into subdomains based on PX, PY, PZ
if rank==0:
    x_slices = np.array_split(data, PX, axis=0)
    subdomains = [np.array_split(slice, PY, axis=1) for slice in x_slices]

        # Distributing subdomains to all ranks
    for i in range(PX):
        for j in range(PY):
            for k in range(PZ):
                dest_rank = i * PY * PZ + j * PZ + k
                to_send=subdomains[i][j][:, :, k::PZ]
                if dest_rank == 0:
                    subdomain = to_send  # Keep this subdomain for rank 0
                else:
                    comm.send(to_send, dest=dest_rank, tag=dest_rank)

        # Memory management
    del subdomains,x_slices,to_send,data
    gc.collect()

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

- First Level of Splitting (X-Axis): The data is first split into PX slices along the X-axis using np.array_split(data, PX, axis=0). This creates chunks that represent PX partitions in the first dimension.
- Second Level of Splitting (Y-Axis): Each X-axis slice is further split along the Y-axis into PY smaller subdomains.

- Third Level of Splitting (Z-Axis): Within each (i, j) subdomain, the data is partitioned along the Z-axis by taking slices in PZ parts using the expression [:, :, k::PZ]. This third level distributes chunks along the Z-axis.
- After data is split into smaller subdomains, each subdomain is assigned a unique rank identifier based on its (i, j, k) position within the (PX, PY, PZ) grid.
- Using comm.send(), subdomains are sent to ranks calculated as `dest_rank = i * PY * PZ + j * PZ + k`. This ensures each process gets a distinct subdomain according to the hierarchical grid.

Example

- Original data shape: (1000, 1000, 200)
- Grid configuration: (PX=2, PY=2, PZ=2), which means:
    - We split the data along the X-axis into 2 parts (PX=2).
    - Each X-axis slice is split along the Y-axis into 2 parts (PY=2).
    - Each resulting subdomain is further split along the Z-axis into 2 parts (PZ=2).
- First Level: Split along the X-axis (PX=2)
    - The data, initially (1000, 1000, 200), is split into 2 slices along the X-axis.
    - Each slice will have dimensions (500, 1000, 200).
- Second Level: Split each X-axis slice along the Y-axis (PY=2)
    - Each of the two X-axis slices of shape (500, 1000, 200) is further split along the Y-axis into 2 parts.
    - After this split, each subdomain has a shape of (500, 500, 200).
- Third Level: Split each subdomain along the Z-axis (PZ=2)
    - Each (500, 500, 200) subdomain is split along the Z-axis into 2 parts.
    - After this final split, each subdomain has dimensions (500, 500, 100).

Now each of 8 subdomains of shape (500, 500, 100) is assigned to a specific process rank, calculated based on the indices i, j, and k as follows:

$$dest\ rank = i \times (PY \times PZ) + j \times PZ + k$$

Memory Management and Garbage Collection

In order to ensure that there is no memory overflow issues, we have deleted the memory allocated to the dataset and other related variables which are not going to be used further after the domain distribution has finidhed and each rank got its subdomain.

```
del subdomains
gc.collect()
del x_slices
gc.collect()
del to_send
gc.collect()
del data
gc.collect()
```

Loading color and opacity transfer functions

For colour, we are creating an array with each element of form: **(data_value,r,g,b)**.

For opacity, we are creating an array with each element of form: **(data_value,opacity)**

```
def load_transfer_function(file_path, is_color=True):
    with open(file_path, 'r') as file:
        data = [float(val) for line in file for val in line.strip().replace(',',
'').split()]

    if is_color:
        return [(data[i], data[i + 1], data[i + 2], data[i + 3]) for i in range(0,
len(data), 4)]
    else:
        return [(data[i], data[i + 1]) for i in range(0, len(data), 2)]
```

Parallel Raycasting with Front-to-back compositing

- For each pixel position (x, y) in the subdomain, we are traversing through the z values incrementing it with step_size.
- We interploate the data value at (x,y,z+step_size) using values at data values index int(z) and int(z)+1 while being in depth bounds.
- Using this interploated value, we interploate color and opacity at that point.
- Use front to back compositing methods.
- For early ray termination, **we have chosen opacity threshold of 0.98.**
- Returns a rgba image of the subdomain.

```
def ray_casting(subdomain, opacity_points, color_points, step_size):
    height, width, depth = subdomain.shape
    img = np.zeros((height, width, 4))
    for y in range(width):
        for x in range(height):
            accumulated_color = np.zeros(3)
            accumulated_opacity = 0
            z = 0.0
            while z < depth:
                data_val = interpolate_value(subdomain, x, y, z)
                color = interpolate_color(data_val, color_points)
                opacity = np.array(interpolate_opacity(data_val, opacity_points))

                accumulated_color += (1 - accumulated_opacity) * color * opacity
                accumulated_opacity += (1 - accumulated_opacity) * opacity

                if accumulated_opacity >= 0.98:
                    break

                z += step_size

            img[x, y, :3] = accumulated_color
            img[x, y, 3] = accumulated_opacity

    return img
```

Parallel composition and Final Image

Each image returned by raycasting function is passed to `binary_swap_compositing(img, PX, PY, PZ, rank, comm)` function which will returning the final merged image and respective computation and communication timings.

Perform binary swap compositing on raycasted images along the pz depth axis for each (px, py) subdomain position.

For each rank we find out its px,py,pz position because we distrubted the subdomains according to the formula mentioned earlier.

Returns:

- final_image (numpy array): The final composited image by rank 0 (None on other ranks).
- total_communication_time (float): Total time spent in communication.
- total_computation_time (float): Total time spent in compositing computations.

Not shown the timing components in the code below

```
def binary_swap_compositing(img, PX, PY, PZ, rank, comm):
    num_ranks = comm.Get_size()
    sub_height, sub_width, _ = img.shape  # Dimensions of the subdomain image

    # Separate color and opacity channels
    local_composite_color = img[:, :, :3]
    local_composite_opacity = img[:, :, 3]

    # Determine the px, py, and pz coordinates for the current rank
    px = (rank // (PY * PZ)) % PX
    py = (rank // PZ) % PY
    pz = rank % PZ

    # Perform binary swap compositing only along the pz axis for the rank
    step = 1
    while step < PZ:
        partner_pz = pz ^ step  # XOR to find partner along the depth (pz) axis
        partner_rank = px * PY * PZ + py * PZ + partner_pz

        if partner_pz < PZ:
            if partner_rank > rank:
                # Sending data to partner rank
            else:
                # Receive data from partner rank
                # Perform the compositing operation
        step *= 2  # Move to the next depth level

    # Gather results to rank 0 for the (px, py) layer

    # Final image assembly on rank 0
    final_image = None
    if rank == 0:
        final_image = np.zeros((PX * sub_height, PY * sub_width, 3))
        total_opacity = np.zeros((PX * sub_height, PY * sub_width))
        for i in range(num_ranks):
            px = (i // (PY * PZ)) % PX
            py = (i // PZ) % PY
            pz = i % PZ

            # Place the composited subdomain in the final image grid for each (px, py)

        return final_image, total_communication_time, total_computation_time
```

```
    else:
        return final_image, total_communication_time, total_computation_time
```

- The function starts by extracting the color and opacity channels from the img array.
- Each rank identifies its position along the X, Y, and Z axes (px, py, pz) within the grid of processors (determined by PX, PY, PZ) thanks to the formulawise subdomain distribution.

```
    px = (rank // (PY * PZ)) % PX
    py = (rank // PZ) % PY
    pz = rank % PZ
```

- Binary Swap Compositing:
  - ## Binary Swap Logic
    - The compositing operation is performed along the Z-axis only, so ranks with the same px and py coordinates but different pz values will exchange data.
    - The binary swap is implemented by iterating over step, starting at 1 and doubling each time.
    - For each step, Each rank computes its "partner" in the Z-axis direction by XOR-ing pz with step.If partner_pz is a valid index within PZ, the partner rank is calculated and data is exchanged.
  - ## Data Exchange
    - Each rank either sends or receives data, depending on whether partner_rank<>rank.
    - The color and opacity arrays are either sent (if the rank is greater than its partner rank) or received.
    - Compositing Operation:
      - For the ranks that receive data, a compositing operation combines the received subdomain with the local one.
      - The local_composite_color is updated by adding the received color scaled by the remaining local opacity.
      - The local_composite_opacity is updated by adding the received opacity scaled by the remaining opacity, then clipped to stay within [0, 1].
  - ## Merging
    - After the compositing, each rank's (px, py) layer is gathered on rank 0. Rank 0 receives the merged color and opacity data for each rank's subdomain along pz.
    - Final Image Assembly (on Rank 0):
      - Rank 0 creates an empty final_image array and a total_opacity array to store the combined subdomain results.
      - Rank 0 places each rank's color and opacity data into the corresponding position in the final_image grid using the formula discussed above.
      - On Rank 0, final_image, total_communication_time, and total_computation_time are returned.
      - For other ranks, only the timing values are returned (with final_image as None), as only Rank 0 needs the final result.

Explanation of MPI.Wtime Usage

- **Data Loading and Distribution Time**: Rank 0 times the data loading and distribution to other ranks. Individual send_time is tracked per rank during domain decomposition.

- **Communication Times (Send and Receive)**: start_send_time and end_send_time record how long Rank 0 takes to send data to each rank. start_recv_time and end_recv_time record how long each rank takes to receive the data from Rank 0.
- **Computation Time for Ray Casting**: start_computation and end_computation encapsulate the ray-casting operation, giving ray_computation_time.
- **Communication and Computation Time while Compositing**: Tracks communication time for parallel merging (binary swap) along with the final gather operation.
- **Total Computation and Communication time by each rank**:

  Communication time for each rank : Send time + Recieve time during subdomain decomposition + Communication while parallel merging

  Computation Time for each rank : Ray casting time + computation while parallel merging

  Using MPI.reduce we have calculated the max computation and communication times across all ranks.
- **Total Execution Time**: Starts at the beginning just after `MPI.COMM_WORLD` to track the full runtime across all ranks. Used `comm.Barrier()` to ensure all ranks have finished their tasks. We have calculated the difference between max and min time taken by ranks for execution and Variance of Execution Time of all ranks for to check Load imbalancing.

Load imbalancing

To show load imbalancing if any we have calculated

- **Range of Execution Times:** By calculating the difference between the maximum and minimum execution times across all ranks (max_execution_time - min_execution_time), we can directly see if some ranks are taking much longer than others. A large range indicates significant load imbalance
- **Variance in Execution Times:** Variance provides a statistical measure of the dispersion of execution times among ranks. A high variance suggests that there is significant disparity in the workload distribution across ranks, which may contribute to inefficiency in parallel processing.
  - global_time_mean is calculated as the average of total_execution_time across ranks.
  - local_sq_diff and global_sq_diff allow calculation of the variance in execution times.

Interpretation of Results

- Large Execution Time Range (max_execution_time - min_execution_time): Indicates that some ranks are heavily loaded relative to others. A large difference shows high load imbalance.
- High Variance: A high variance in execution times indicates that the workload is unevenly distributed among the ranks. The larger the variance, the greater the disparity, signaling potential inefficiencies in load distribution.

Complete Terminal Output view

We have shown on the output screen :

- Which rank is handled by which system
- Domain decomposition and distribution finished status
- Time taken by each Rank to finish raycasting
- Time taken by each rank on computations
- Time taken by each rank on communication
- Time taken by each rank to finish execution
- Final image saved as: 2_2_8.png with dimensions (2000, 2000, 3)
- Time taken for data loading and distribution
- Max Computation time across all ranks
- Max Communication time across all ranks
- Time Difference between max and min total execution time across different ranks
- Variance among total execution time of all ranks
- Total execution time of code

One such output is shown below. We have only attached screenshot of main timings only as the output screen gets too big and cluttered.

```
khushwantk24@csews5:~/Downloads/A2$ mpirun --mca btl_tcp_if_include eno1 --hostfile
hostfile -np 32 --oversubscribe python3 A2.py Isabel_2000x2000x400_float32.raw 2 2 8
0.5 opacity_TF.txt color_TF.txt | tee out_32b.txt
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 6 running on csews5
Rank 7 running on csews5
Rank 8 running on csews1
Rank 9 running on csews1
Rank 10 running on csews1
Rank 11 running on csews1
Rank 12 running on csews1
Rank 13 running on csews1
Rank 14 running on csews1
Rank 15 running on csews1
Rank 16 running on csews6
Rank 17 running on csews6
Rank 18 running on csews6
Rank 19 running on csews6
Rank 20 running on csews6
Rank 21 running on csews6
Rank 22 running on csews6
Rank 23 running on csews6
Rank 24 running on csews9
Rank 25 running on csews9
Rank 26 running on csews9
Rank 27 running on csews9
Rank 28 running on csews9
Rank 29 running on csews9
Rank 30 running on csews9
Rank 31 running on csews9
```

```
Rank 2 finished raycasting in 15.4490 seconds.
Rank 2 has started merging...
Rank 1 finished raycasting in 17.8608 seconds.
Rank 1 has started merging...
Rank 3 finished raycasting in 16.1288 seconds.
Rank 3 has started merging...
Rank 7 finished raycasting in 14.9355 seconds.
Rank 7 has started merging...
Rank 5 finished raycasting in 26.7594 seconds.
Rank 5 has started merging...
Rank 8 finished raycasting in 22.4001 seconds.
Rank 8 has started merging...
Rank 6 finished raycasting in 26.8077 seconds.
Rank 6 has started merging...
Rank 10 finished raycasting in 20.5145 seconds.
Rank 10 has started merging...
Rank 12 finished raycasting in 17.1094 seconds.
Rank 12 has started merging...
Rank 4 finished raycasting in 33.3248 seconds.
Rank 4 has started merging...
Rank 9 finished raycasting in 28.2602 seconds.
Rank 9 has started merging...
Rank 11 finished raycasting in 24.4615 seconds.
Rank 11 has started merging...
Rank 15 finished raycasting in 15.4146 seconds.
Rank 15 has started merging...
Rank 16 finished raycasting in 16.8695 seconds.
Rank 16 has started merging...
Rank 19 finished raycasting in 16.9832 seconds.
Rank 19 has started merging...
Rank 14 finished raycasting in 29.7083 seconds.
Rank 14 has started merging...
Rank 13 finished raycasting in 32.4118 seconds.
Rank 13 has started merging...
Rank 17 finished raycasting in 22.6857 seconds.
Rank 17 has started merging...
Rank 18 finished raycasting in 22.9612 seconds.
Rank 18 has started merging...
Rank 22 finished raycasting in 13.8926 seconds.
Rank 22 has started merging...
Rank 23 finished raycasting in 13.5659 seconds.
Rank 23 has started merging...
Domain decomposition and distribution finished.
Rank 26 finished raycasting in 15.7722 seconds.
Rank 26 has started merging...
Rank 20 finished raycasting in 32.6110 seconds.
Rank 20 has started merging...
Rank 21 finished raycasting in 31.2851 seconds.
Rank 21 has started merging...
Rank 25 finished raycasting in 21.4418 seconds.
Rank 25 has started merging...
Rank 24 finished raycasting in 24.3196 seconds.
Rank 24 has started merging...
Rank 30 finished raycasting in 14.5074 seconds.
Rank 30 has started merging...
Rank 31 finished raycasting in 14.5712 seconds.
Rank 31 has started merging...
Rank 28 finished raycasting in 23.5446 seconds.
Rank 28 has started merging...
Rank 27 finished raycasting in 30.4808 seconds.
Rank 27 has started merging...
```

```
Rank 29 finished raycasting in 27.7883 seconds.
Rank 29 has started merging...
Rank 0 finished raycasting in 41.0795 seconds.
Rank 0 has started merging...
Time taken by rank 0 on computations : 41.1992 seconds
Time taken by rank 0 on communications : 78.1994 seconds
Time taken by rank 4 on computations : 33.4274 seconds
Time taken by rank 4 on communications : 120.1661 seconds
Time taken by rank 2 on computations : 15.5514 seconds
Time taken by rank 2 on communications : 138.0327 seconds
Time taken by rank 3 on computations : 16.2371 seconds
Time taken by rank 3 on communications : 137.3510 seconds
Time taken by rank 1 on computations : 17.9040 seconds
Time taken by rank 1 on communications : 135.6744 seconds
Time taken by rank 6 on computations : 26.9316 seconds
Time taken by rank 6 on communications : 126.6717 seconds
Time taken by rank 7 on computations : 15.0581 seconds
Time taken by rank 7 on communications : 138.5506 seconds
Time taken by rank 5 on computations : 26.9494 seconds
Time taken by rank 5 on communications : 126.6484 seconds
Time taken by rank 22 on computations : 13.9970 seconds
Time taken by rank 22 on communications : 140.6372 seconds
Time taken by rank 15 on computations : 15.5361 seconds
Time taken by rank 15 on communications : 138.6080 seconds
Time taken by rank 19 on computations : 17.0843 seconds
Time taken by rank 19 on communications : 137.3390 seconds
Time taken by rank 12 on computations : 17.1611 seconds
Time taken by rank 12 on communications : 136.7786 seconds
Time taken by rank 23 on computations : 13.6916 seconds
Time taken by rank 23 on communications : 141.0080 seconds
Time taken by rank 9 on computations : 28.3072 seconds
Time taken by rank 9 on communications : 125.4234 seconds
Time taken by rank 26 on computations : 15.8097 seconds
Time taken by rank 26 on communications : 139.1021 seconds
Time taken by rank 21 on computations : 31.4454 seconds
Time taken by rank 21 on communications : 123.1179 seconds
Time taken by rank 14 on computations : 29.8678 seconds
Time taken by rank 14 on communications : 124.2103 seconds
Time taken by rank 30 on computations : 14.6241 seconds
Time taken by rank 30 on communications : 140.5652 seconds
Time taken by rank 17 on computations : 22.7335 seconds
Time taken by rank 17 on communications : 131.5541 seconds
Time taken by rank 13 on computations : 32.5716 seconds
Time taken by rank 13 on communications : 121.4359 seconds
Time taken by rank 28 on computations : 23.6389 seconds
Time taken by rank 28 on communications : 131.4110 seconds
Time taken by rank 20 on computations : 32.7156 seconds
Time taken by rank 20 on communications : 121.7811 seconds
Time taken by rank 11 on computations : 24.5613 seconds
Time taken by rank 11 on communications : 129.3087 seconds
Time taken by rank 31 on computations : 14.6988 seconds
Time taken by rank 31 on communications : 140.5529 seconds
Time taken by rank 18 on computations : 23.0060 seconds
Time taken by rank 8 on computations : 22.4001 seconds
Time taken by rank 8 on communications : 131.2617 seconds
Time taken by rank 27 on computations : 30.5988 seconds
Time taken by rank 27 on communications : 124.3814 seconds
Time taken by rank 16 on computations : 16.8695 seconds
Time taken by rank 16 on communications : 137.3478 seconds
Time taken by rank 10 on computations : 20.5572 seconds
Time taken by rank 10 on communications : 133.2434 seconds
```

```
Time taken by rank 24 on computations : 24.3196 seconds
Time taken by rank 24 on communications : 130.4525 seconds
Time taken by rank 18 on communications : 131.3454 seconds
Time taken by rank 29 on computations : 27.9496 seconds
Time taken by rank 29 on communications : 127.1630 seconds
Time taken by rank 25 on computations : 21.4900 seconds
Time taken by rank 25 on communications : 133.3520 seconds

Final image saved as: 2_2_8.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 106.3183 seconds
Max Computation time across all ranks: 41.1992 seconds
Max Communication time across all ranks: 141.007997 seconds
Time taken by rank 5 to finish execution : 156.8186 seconds
Time taken by rank 4 to finish execution : 156.8187 seconds
Time taken by rank 2 to finish execution : 156.8186 seconds
Time taken by rank 3 to finish execution : 156.8186 seconds
Time taken by rank 1 to finish execution : 156.8186 seconds
Time taken by rank 6 to finish execution : 156.8186 seconds
Time taken by rank 7 to finish execution : 156.8185 seconds
Time taken by rank 0 to finish execution : 156.8187 seconds
Time taken by rank 19 to finish execution : 156.8203 seconds
Time taken by rank 13 to finish execution : 156.8178 seconds
Time taken by rank 15 to finish execution : 156.8178 seconds
Time taken by rank 11 to finish execution : 156.8178 seconds
Time taken by rank 17 to finish execution : 156.8204 seconds
Time taken by rank 8 to finish execution : 156.8178 seconds
Time taken by rank 23 to finish execution : 156.8203 seconds
Time taken by rank 9 to finish execution : 156.8178 seconds
Time taken by rank 22 to finish execution : 156.8203 seconds
Time taken by rank 14 to finish execution : 156.8178 seconds
Time taken by rank 21 to finish execution : 156.8204 seconds
Time taken by rank 10 to finish execution : 156.8178 seconds
Time taken by rank 16 to finish execution : 156.8204 seconds
Time taken by rank 12 to finish execution : 156.8178 seconds
Time taken by rank 18 to finish execution : 156.8204 seconds
Time taken by rank 20 to finish execution : 156.8204 seconds
Time taken by rank 27 to finish execution : 156.8215 seconds
Time taken by rank 25 to finish execution : 156.8216 seconds
Time taken by rank 29 to finish execution : 156.8215 seconds
Time taken by rank 24 to finish execution : 156.8216 seconds
Time taken by rank 26 to finish execution : 156.8216 seconds
Time taken by rank 28 to finish execution : 156.8216 seconds
Time taken by rank 31 to finish execution : 156.8216 seconds
Time taken by rank 30 to finish execution : 156.8216 seconds
Difference between max and min total execution time across different ranks : 0.0038
seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 156.8216 seconds
```

Results : Timings, Load imbalance, Scalability Plots

1000x1000x200 dataset timings and Load Imbalace

Test Case	np	Px_Py_Pz	Step size	Data loading and distribution (sec)	Max communication time (sec)	Max computation time (sec)	Total execution time of code (sec)	Range of Execution Times	Variance
1	8	2_2_2	0.5	4.61	5.44	7.95	12.06	0.0001	0.00000
	8	2_2_2	0.5	4.40	5.94	7.80	12.44	0.0001	0.00000
2	16	2_2_4	0.5	6.21	7.62	4.96	11.89	0.0003	0.00000
	16	2_2_4	0.5	6.29	7.83	5.56	12.03	0.0003	0.00000
3	32	2_2_8	0.5	14.19	11.01	7.10	14.19	0.0006	0.00000
	32	2_2_8	0.5	17.20	13.70	7.95	17.20	0.0006	0.00000

Range of Execution Times : difference between the maximum and minimum execution times across all ranks

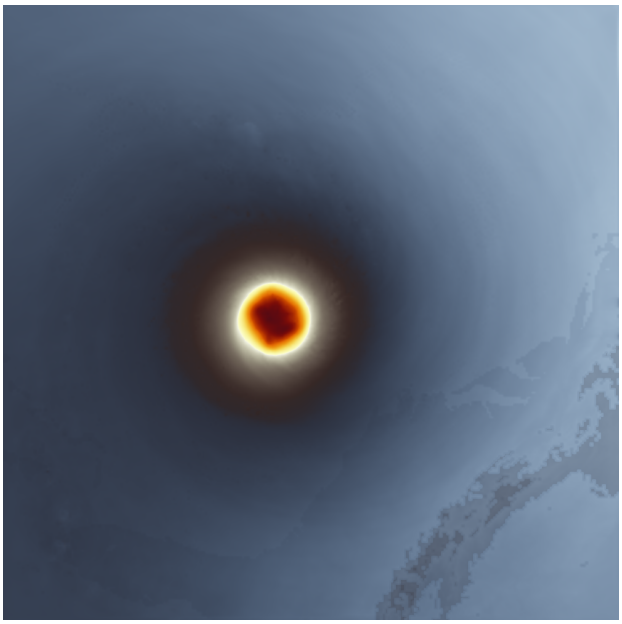
Variance : Variance in Execution Times (upto 5 decimal places)

No Load imbalance is observed.

1. **mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 8 --oversubscribe python3 Ultimate.py Isabel_1000x1000x200_float32.raw 2 2 2 0.5 opacity_TF.txt color_TF.txt**

```
Final image saved as: 2_2_2.png with dimensions (1000, 1000, 3)
Time taken for data loading and distribution: 4.6164 seconds
Max Computation time across all ranks: 7.9585 seconds
Max Communication time across all ranks: 5.441165 seconds
Difference between max and min total execution time across different ranks : 0.0001 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 12.0630 seconds
```

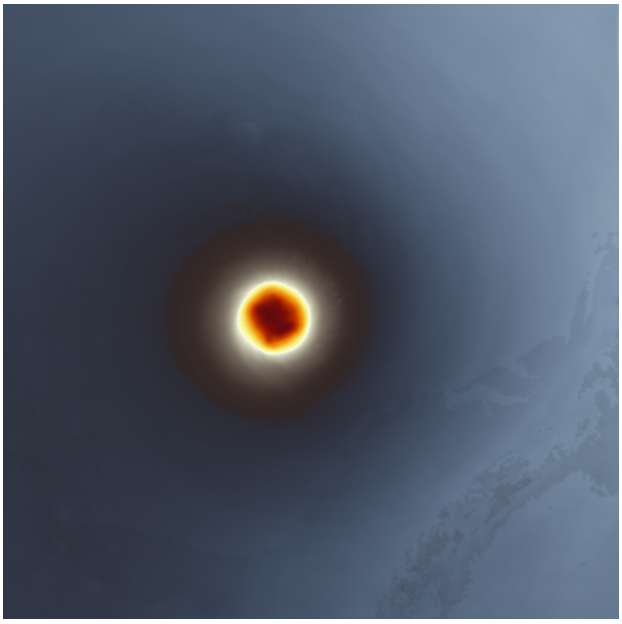
```
Final image saved as: 2_2_2.png with dimensions (1000, 1000, 3)
Time taken for data loading and distribution: 4.4012 seconds
Max Computation time across all ranks: 7.7992 seconds
Max Communication time across all ranks: 5.941221 seconds
Difference between max and min total execution time across different ranks : 0.0001 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 12.4460 seconds
```



1. **mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 16 --oversubscribe python3 Ultimate.py Isabel_1000x1000x200_float32.raw 2 2 4 0.5 opacity_TF.txt color_TF.txt**

```
Final image saved as: 2_2_4.png with dimensions (1000, 1000, 3)
Time taken for data loading and distribution: 6.2117 seconds
Max Computation time across all ranks: 4.9680 seconds
Max Communication time across all ranks: 7.621497 seconds
Difference between max and min total execution time across different ranks : 0.0003 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 11.8923 seconds
```

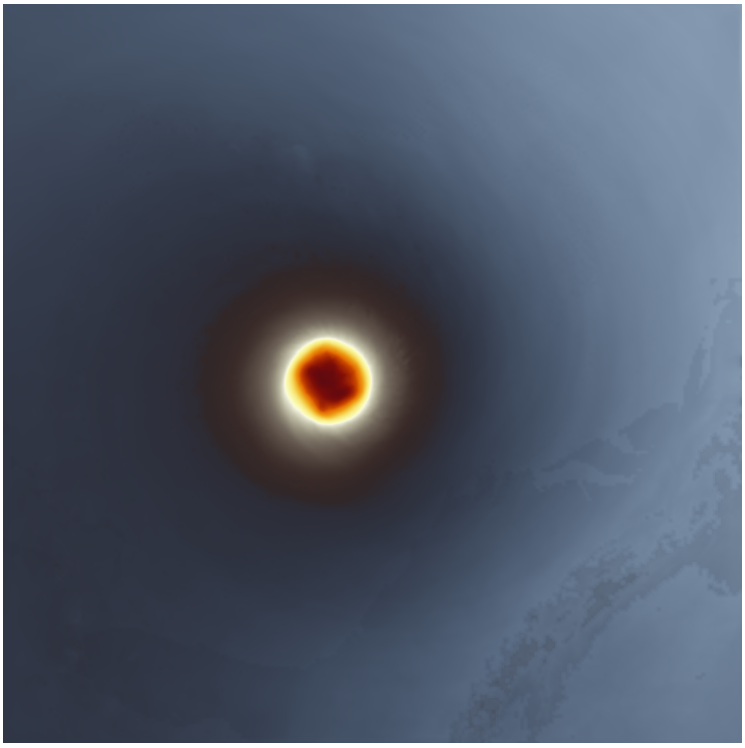
```
Final image saved as: 2_2_4.png with dimensions (1000, 1000, 3)
Time taken for data loading and distribution: 6.2963 seconds
Max Computation time across all ranks: 5.5654 seconds
Max Communication time across all ranks: 7.830195 seconds
Difference between max and min total execution time across different ranks : 0.0003 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 12.0321 seconds
```



3. **mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 32 --oversubscribe python3 Ultimate.py Isabel_1000x1000x200_float32.raw 2 2 8 0.5 opacity_TF.txt color_TF.txt**

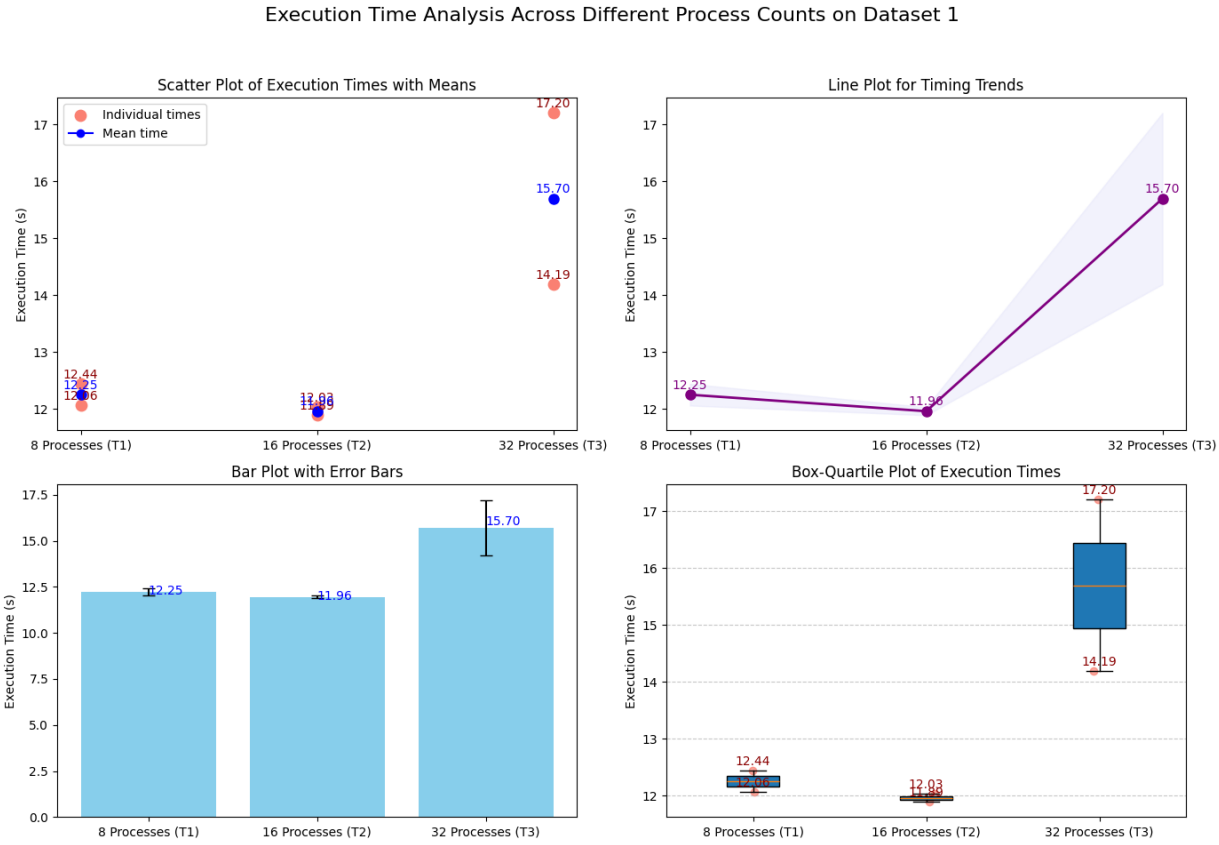
```
Final image saved as: 2_2_8.png with dimensions (1000, 1000, 3)
Time taken for data loading and distribution: 7.1510 seconds
Max Computation time across all ranks: 7.1016 seconds
Max Communication time across all ranks: 11.015396 seconds
Difference between max and min total execution time across different ranks : 0.0006 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 14.1910 seconds
```

```
Final image saved as: 2_2_8.png with dimensions (1000, 1000, 3)
Time taken for data loading and distribution: 6.9063 seconds
Max Computation time across all ranks: 7.9499 seconds
Max Communication time across all ranks: 13.707885 seconds
Difference between max and min total execution time across different ranks : 0.0006 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 17.2046 seconds
```

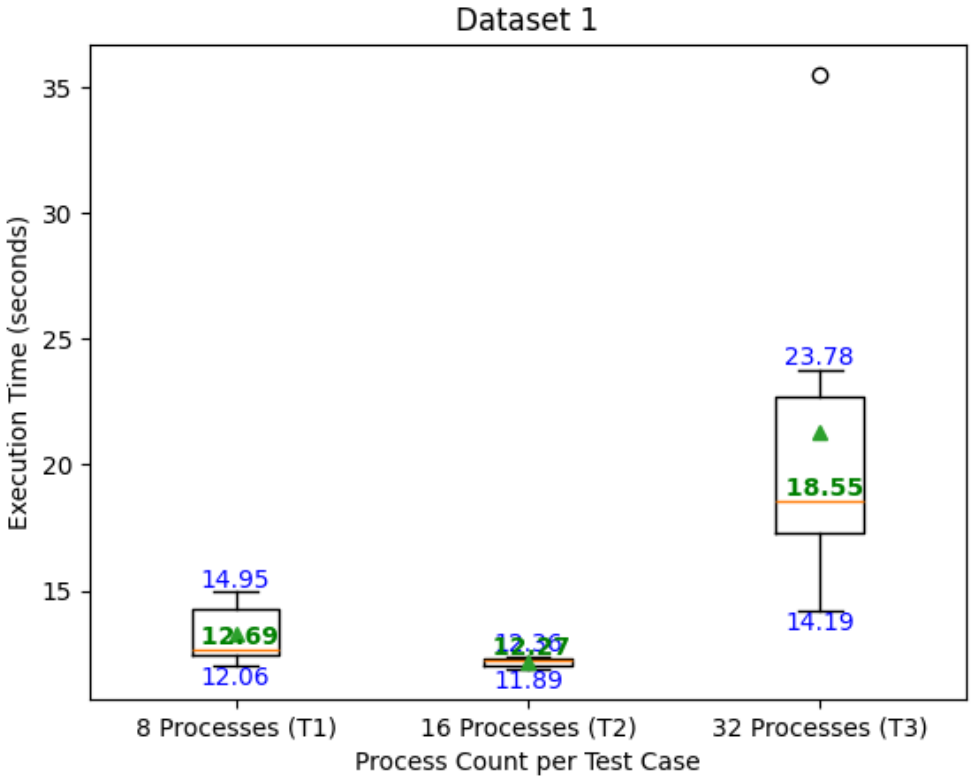


1000x1000x200 dataset plots

2 runs per test case :



5 runs per test case :



2000x2000x400 dataset timings and Load Imbalace

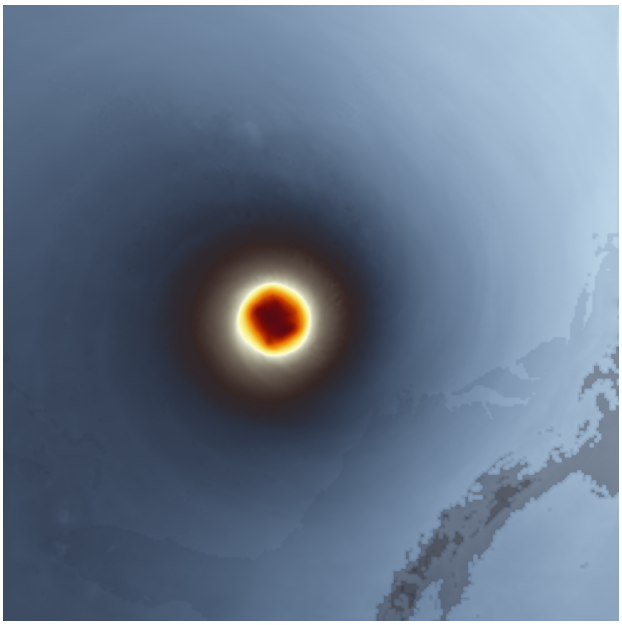
Test Case	np	Px_Py_Pz	Step size	Data loading and distribution (sec)	Max communication time (sec)	Max computation time (sec)	Total execution time of code (sec)	Range of Execution Times	Variance
1	8	2_2_2	0.5	63.90	72.44	39.47	100.96	0.0015	0.00000
	8	2_2_2	0.5	64.64	73.12	42.22	101.93	0.0014	0.00000
2	16	2_2_4	0.5	89.46	99.25	29.70	122.20	0.0021	0.00000
	16	2_2_4	0.5	69.84	79.36	33.55	102.53	0.0019	0.00000
3	32	2_2_8	0.5	72.65	106.62	40.48	122.89	0.0034	0.00000
	32	2_2_8	0.5	89.39	125.21	42.36	141.10	0.0038	0.00000

No Load imbalance is observed.

1. **mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 8 --oversubscribe python3 Ultimate.py Isabel_2000x2000x400_float32.raw 2 2 2 0.5 opacity_TF.txt color_TF.txt**

```
Final image saved as: 2_2_2.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 63.8929 seconds
Max Computation time across all ranks: 39.4700 seconds
Max Communication time across all ranks: 72.445496 seconds
Difference between max and min total execution time across different ranks : 0.0015 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 100.9609 seconds
```

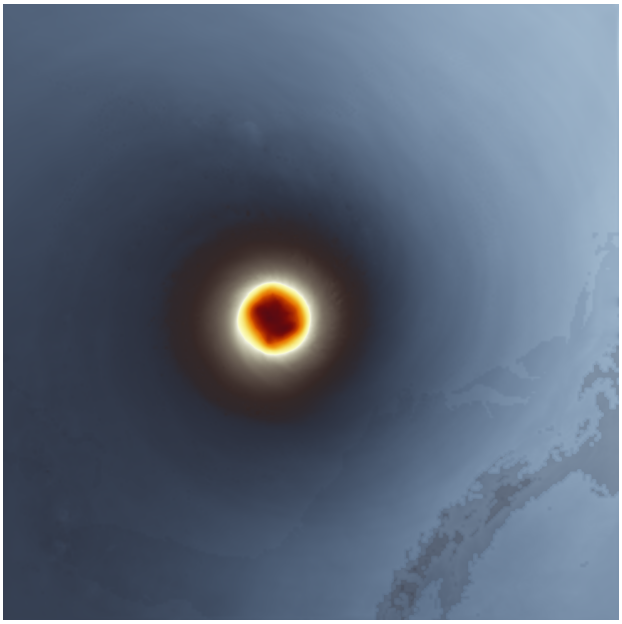
```
Final image saved as: 2_2_2.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 64.6475 seconds
Max Computation time across all ranks: 42.2275 seconds
Max Communication time across all ranks: 73.127900 seconds
Difference between max and min total execution time across different ranks : 0.0014 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 101.9341 seconds
```



2. **mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 16 --oversubscribe python3 Ultimate.py Isabel_2000x2000x400_float32.raw 2 2 4 0.5 opacity_TF.txt color_TF.txt**

```
Final image saved as: 2_2_4.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 89.4658 seconds
Max Computation time across all ranks: 29.7046 seconds
Max Communication time across all ranks: 99.250750 seconds
Difference between max and min total execution time across different ranks : 0.0021 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 122.2039 seconds
```

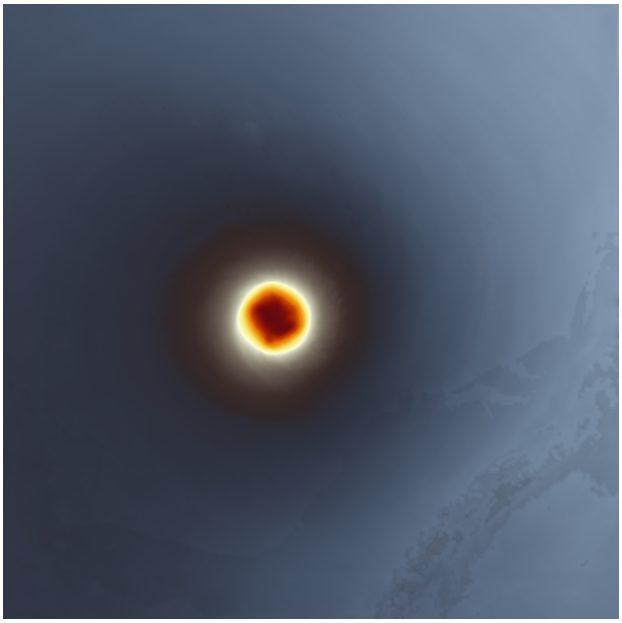
```
Final image saved as: 2_2_4.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 69.8442 seconds
Max Computation time across all ranks: 33.5511 seconds
Max Communication time across all ranks: 79.361238 seconds
Difference between max and min total execution time across different ranks : 0.0019 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 102.5372 seconds
```



3. **mpirun --mca btl_tcp_if_include eno1 --hostfile hostfile -np 32 --oversubscribe python3 Ultimate.py Isabel_2000x2000x400_float32.raw 2 2 8 0.5 opacity_TF.txt color_TF.txt**

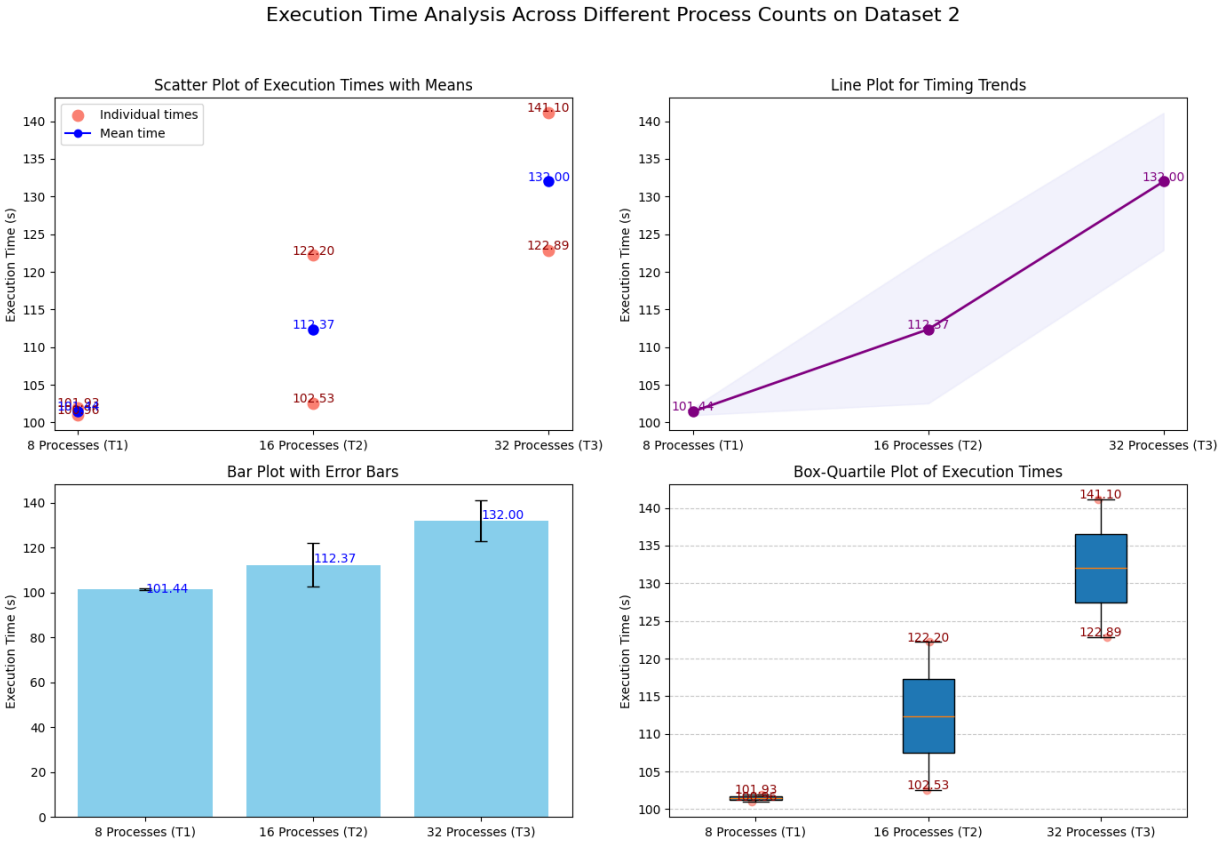
```
Final image saved as: 2_2_8.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 72.6517 seconds
Max Computation time across all ranks: 40.4819 seconds
Max Communication time across all ranks: 106.625851 seconds
Difference between max and min total execution time across different ranks : 0.0034 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 122.8988 seconds
```

```
Final image saved as: 2_2_8.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 89.3968 seconds
Max Computation time across all ranks: 42.3655 seconds
Max Communication time across all ranks: 125.218017 seconds
Difference between max and min total execution time across different ranks : 0.0038 seconds
Variance among total execution time of all ranks: 0.00000
Total execution time of code : 141.1018 seconds
```

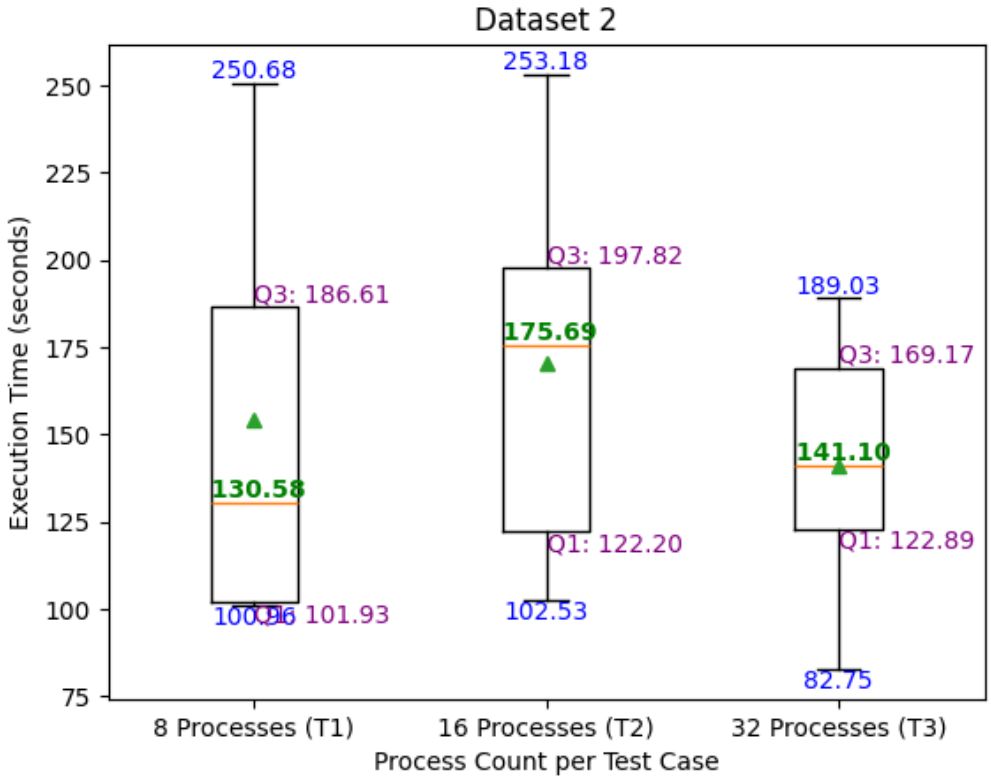


2000x2000x400 dataset plots

2 runs per test case :



5 runs per test case :



System setup to use hostfile

We have used mpi4py, numpy, numba and Pillow python libraries.

To install we have created a install_libraries.sh

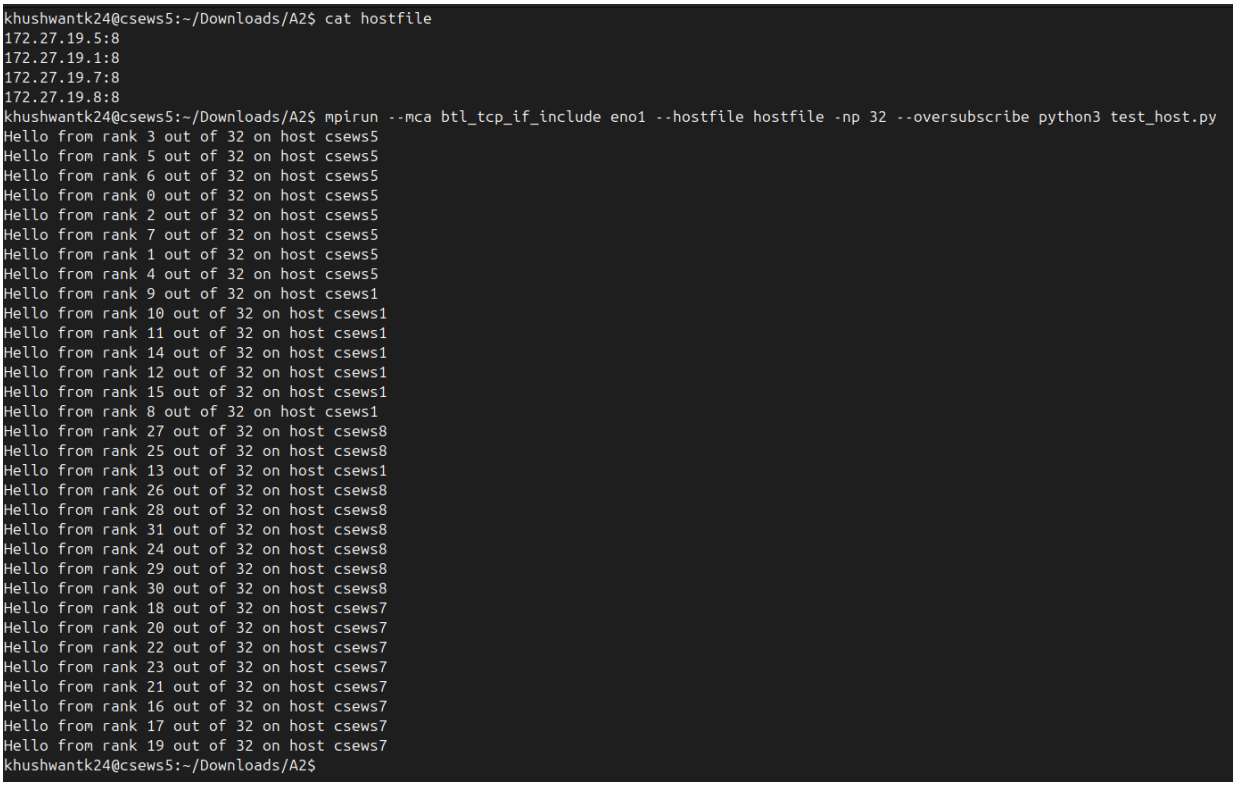
If that doesn't work :

```
pip install mpi4py
pip install numba numpy matplotlib
```

Hostfile

We have created a test_host.py

- Make sure all the systems mentioned in hostfile are ON/reachable.
- Setup passwordless ssh between the systems using ssh-keygen and ssh-copy-id.
- Do `ssh 172.27.19.xx` to each ip mentioned in hostfile.



Working with different hosts

Possible Caveats

Flipped and Rotated Output Image

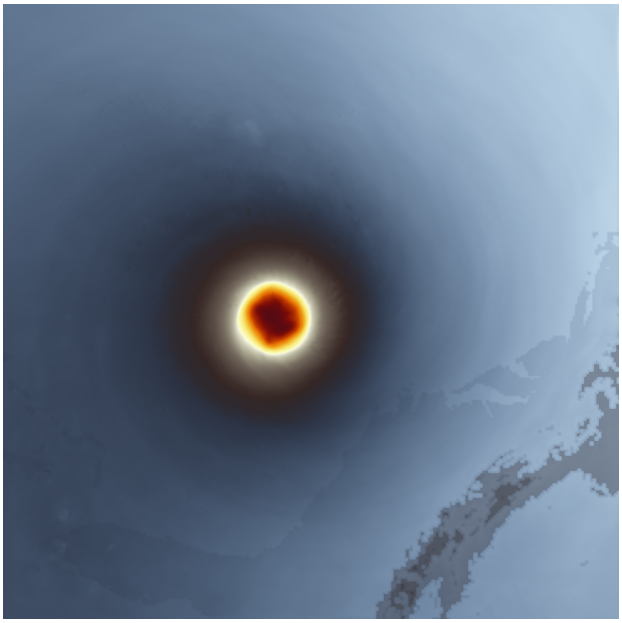
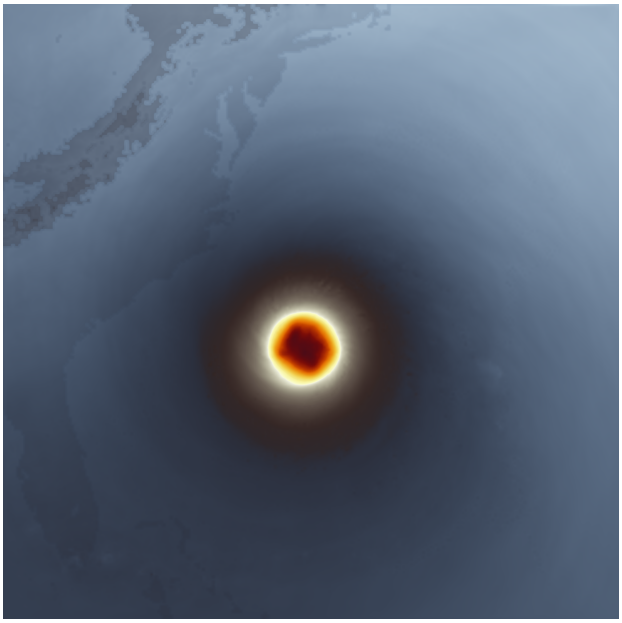


Image ParaView Gives after Clicking +Z (Looking down x axis from (0,0,1))



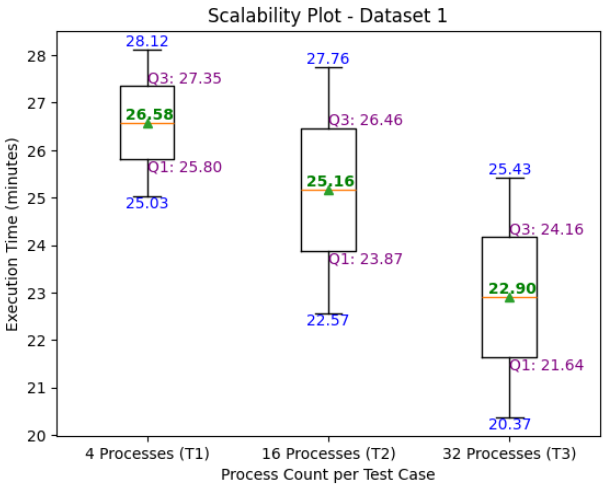
Our Full Extent Rendered Image

We have got the same output as ParaView by flipping and rotating our merged image.

```
flipped_img = np.flipud(final_image)
rotated_img = np.rot90(flipped_img)
plt.imsave(file_name,rotated_img)
```

Python numba library

We have used @jit decorator of numba library with heavy time consuming part of code ie RayCasting. The speedup achieved is very high. But the boxplot of timings achieved on the Python code without using numba shows little improvement in total execution time as process increases which is not the case while using numba.



Box Plot using python without numba-jit

Zero Variance

As we have reported 0 variance for all the test cases as we executed our test cases in early morning hours.

Its not always the case that we will get 0 variance, when we ran our code on our system, we got some value for variance. Even under high CPU usage situations, we got some variance values on csewsx systems.

```
Final image saved as: 2_2_8.png with dimensions (2000, 2000, 3)
Time taken for data loading and distribution: 91.2196 seconds
Max Computation time across all ranks: 44.0107 seconds
Max total communication time across all ranks: 128.405276 seconds
Difference between max and min total execution time across different ranks : 1.8854 seconds
Variance among total execution time of all ranks: 0.53419
Total execution time of code : 145.7558 seconds
khushwantk24@csews5:~/Downloads/A2$
```

Non zero Variance