In Situ Visualization for Large-Scale Combustion Simulations

Research Paper Literature Review

Ankur Kumar, Jai Verma, Sahil Gala

Indian Institute of Technology Kanpur

CS677

Topics in Large Data Analysis and Visualization

Overview of the Presentation

1. **Introduction**
2. **Implementation Details**
3. **Case Study on Turbulent Combustion**
4. **Results**
5. **Summary**

The Power and Challenge of Supercomputers

- Supercomputers are vital for scientific breakthroughs
- Enables large-scale simulations like climate patterns and astrophysical events.
- However, cost of maintenance means limited access, with researchers competing for it.
- Researchers must maximize the efficiency and insights of each simulation.

Traditional Methods: Post-processing

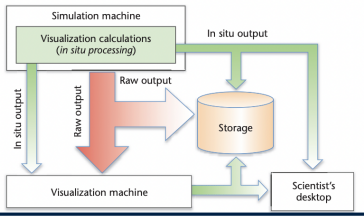
- **Post-processing:** Post Processing stored the vast raw simulation data for later analysis and visualization.
- **The Limitations:** As simulations became complex, storing and transferring TBs/PBs of data became a major challenge, often delaying insights by weeks.

Traditional Methods: Co-processing

- **Co-processing:** Here, visualization occurs on separate machine during the simulation, reducing the simulation data to smaller visualization data, decreasing storage load.
- **The Limitations:** Moving large datasets to visualization system remained an I/O bottleneck, especially when dealing with the scale of modern simulations.

What is In Situ Visualization?

- **Definition:** Generating visual representations of data directly on the same computational resources where the simulation is being run, without the need to transfer data to a separate visualization system.
- **Basic Concept:** We want to reduce the data size that needs to be transferred and stored by performing visualization tasks simultaneously with the simulation, providing real-time insights.
- **Key Features:** In situ visualization is tightly coupled with the simulation process, enabling direct interaction with the data as it is being generated.



Benefits of In Situ Visualization

- **Scalability:** Can handle the massive volumes of data generated by modern simulations, alleviating storage and transfer limitations.
- **Real-Time Insights:** Researchers can gain immediate feedback on the simulation's progress.
- **Data Reduction:** Replaces the need for storing large raw data by significantly smaller visualization data.
- **Data Transfer:** Replaces the need for storing large raw data by significantly smaller visualization data.
- **Lesser Compute Need:** An alternative that simplifies the calculations such that visualization accounts for only a small fraction of the simulation time

Comparison with Traditional Methods

- **Postprocessing:** In traditional postprocessing, data is stored after simulation and analyzed later. This method is time-consuming and requires large storage capacities.
- **Coprocessing:** Coprocessing involves visualizing data on a separate machine during the simulation, which reduces some of the data transfer issues but still requires significant resources for data movement.
- **In Situ Visualization:** In situ visualization eliminates the need for large-scale data transfer and storage by processing and visualizing data in real-time on the same machine, offering a highly scalable solution.

Challenges with In Situ Visualization

- The visualization code must **interact directly** with the simulation code.
- **Balancing the visualization workload** is more difficult because the visualization must comply and tightly couple with the simulation architecture.
- Visualization calculations **must not incur excessive costs**, with decoupled I/O delivering rendering results while simulation is running
- The scalability of in situ visualization becomes a concern,introducing significant bottlenecks (particularly in areas like image compositing).

- **System Architecture:** In situ visualization works alongside simulation codes, often sharing the same resources and to minimize data movement.
- **Key Components:** The main components include data processing modules, rendering engines, and communication protocols that allow the fluid interaction between the two processes.
- **Integration with Simulations:** Ensures that visualization is tightly coupled with simulation, eminimizing delays wherever possible.

- To handle the massive datasets generated by simulations, we rely on parallel rendering techniques for independent similar tasks.
- **Volume Rendering:** Generating visual representations of 3D data, typically by casting rays through the data and calculating how light interacts with each voxel.
- **Particle Rendering:** Used for simulations involving particles (here, combustion) where each particle's position and properties are rendered to visualize the simu.

- The simulation domain is divided into smaller regions, each assigned to a different processor, to handle the large data in parallel.
- Duplicate the data along the edges of the data regions **(two voxels wide)**, to achieve seamless rendering across data partition boundaries
- A processor responsible for a data region typically needs to communicate with its **26** neighboring processors.
- By avoiding diagonal communication, a processor only needs to communicate with **six** neighboring processors.

Particle Rendering

- Blend the Boundary points correctly along the boundaries when integrated with volume rendering.
- Then, we use the normal to calculate the lighting using the Phong model and the eye and light directions.
- Algorithm:
  1. Each processor renders in its data region.
  2. Exchanges information with particles along the neighbouring boundaries.
  3. Renders the particles along its boundary (2 voxel) on both sides
  4. We perform volume ray casting with depth lookup of the particle's RGBA values.

Caveat: Boundary Data

Domain Decomposition Challenges:

- In large-scale simulations, data must be replicated across domain boundaries to ensure smooth continuity.
- **Simulation Code:**
  - Requires four voxel-wide boundaries for accurate gradient and derivative calculations.
  - Uses a single buffer to store boundary data for one variable at a time to manage memory efficiently.
- **Volume Rendering:**
  - Needs only two voxel-wide boundaries for seamless visual output.
  - The simulation's buffer often lacks the specific variables required by the renderer, creating additional challenges.

Caveat: Boundary Data

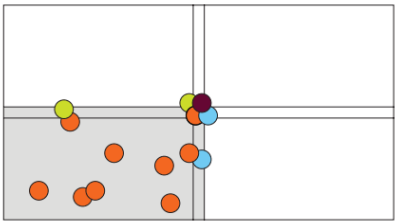
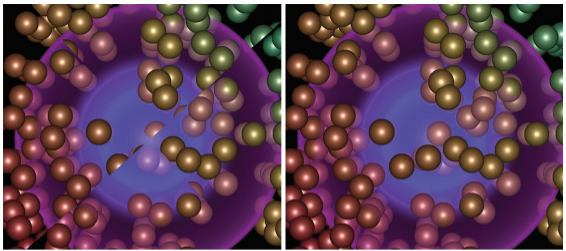


Figure. 2. Handling particles along neighboring data region boundaries. The processor assigned to the gray data region must exchange particles along the boundary with neighboring processors so that the particles can blend correctly when integrated with volume rendering. The particles from neighboring data regions are blue, green, and brown.



(a) (b)

Figure. 3. Integrating volume and particle rendering. The synthesized volume data set comprises multiple-layer spheres. (a) Not exchanging and handling the particles along the boundary results in incorrect rendering along the diagonals. (b) Exchanging those particles results in correct rendering.

Balancing Memory and Performance:

- Storing all boundary data on the simulation side would lead to excessive memory use and require complex code modifications.

Practical Solution:

- The visualization code performs an additional boundary exchange among processors to gather the necessary two-voxel-wide data for rendering.
- **Outcome:**
  - This adds some communication overhead, but it is manageable and essential for effective integration of the simulation and visualization processes.
  - Helps maintain a balance between memory efficiency and the need for accurate, real-time visualization.

- Partial images generated by different processors are combined to produce the final output.
- **Binary Swap Algorithm:** It involves pairing processors in a binary tree structure to exchange and combine image data. At any stage, a processor communicates only with its counterpart.
- Direct send requires n processors to exchange $O(n^2)$ messages, binary swap requires only $O(n \log n)$ messages
- **Challenges with Binary Swap:** The binary swap algorithm, requires the number of processors to be a power of 2.
- For $2^{k-1} < n < 2^k$, we send the image data from $n - 2^{k-1}$ processors to the remainder of 2^{k-1} processors. We then perform binary swap directly on the 2^{k-1} processors.

2-3 Swap Algorithm

- The 2-3 swap algorithm partitions processors into small groups of two or three for the compositing process, unlike traditional methods that pair processors.
- Within each group, a processor communicates only with the other processors in its group, which is typically no more than four processors per group. This localized communication reduces the complexity and overhead of data exchange.
- **Performance Improvements:** The 2-3 swap algorithm merges Direct Send's flexibility with binary swap's efficiency, scaling well to thousands of processors. It improves performance in large-scale simulations by reducing communication bottlenecks and enhancing parallel efficiency.

2-3 Swap Algorithm

Algorithm 1 ConstructCompositingTree ($node, n, d$)

1: **if** $d = 0$ **then**
2: $node_{list} \Leftarrow procid$ {In $node$, $node_{list}$ keeps the order of the indices of processors for image partition assignment; $procid$ is a global variable, initialized as 1.}
3: $procid \Leftarrow procid + 1$
4: **else**
5: $l \Leftarrow \lfloor n/2 \rfloor$
6: $r \Leftarrow \lceil n/2 \rceil$
7: **if** $r < 2^d$ **then**
8: create two children, namely, $node_l$ and $node_r$, for $node$
9: ConstructCompositingTree ($node_l, l, d-1$)
10: ConstructCompositingTree ($node_r, r, d-1$)
11: **else**
12: $l \Leftarrow \lfloor n/3 \rfloor$
13: $m \Leftarrow \lfloor n/3 \rfloor$
14: $r \Leftarrow \lceil n/3 \rceil$
15: create three children, namely, $node_l$, $node_m$, and $node_r$, for $node$
16: ConstructCompositingTree ($node_l, l, d-1$)
17: ConstructCompositingTree ($node_m, m, d-1$)
18: ConstructCompositingTree ($node_r, r, d-1$)
19: **end if**
20: **end if**

Algorithm 2 OrderAssignment ($node$)

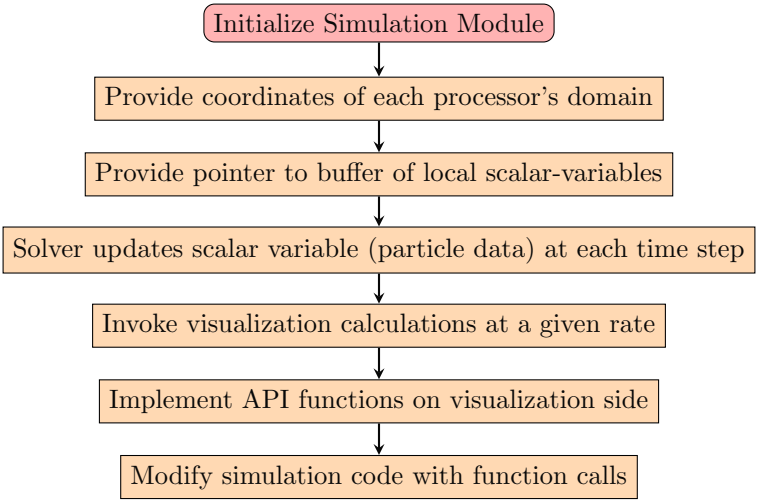
1: **if** NumberOfChildren ($node$) = 0 **then**
2: return
3: **else**
4: **if** NumberOfChildren ($node$) = 2 **then**
5: OrderAssignment ($node_l$)
6: OrderAssignment ($node_r$)
7: **if** SizeOfList ($node_r$) > SizeOfList ($node_l$) **then**
8: $node_{list} \Leftarrow$ merge the lists of $node_r$ and $node_l$ interleavingly
9: **else**
10: $node_{list} \Leftarrow$ merge the lists of $node_l$ and $node_r$ interleavingly
11: **end if**
12: **else**
13: OrderAssignment ($node_l$)
14: OrderAssignment ($node_m$)
15: OrderAssignment ($node_r$)
16: **if** SizeOfList ($node_r$) > SizeOfList ($node_l$) **then**
17: $node_{list} \Leftarrow$ merge the lists of $node_r$, $node_l$, and $node_m$ interleavingly
18: **else**
19: $node_{list} \Leftarrow$ merge the lists of $node_l$, $node_m$, and $node_r$ interleavingly
20: **end if**
21: **end if**
22: **end if**

- S3D, developed by Sandia National Laboratories, is a DNS code designed for high-fidelity simulations of turbulent combustion.
- S3D employs an eighth-order approximation for spatial derivatives and a high-order Runge-Kutta method to advance mass, momentum, energy, and species equations.
- It accurately resolves the Navier-Stokes equations coupled with detailed chemical kinetics, essential for accurate combustion modeling.

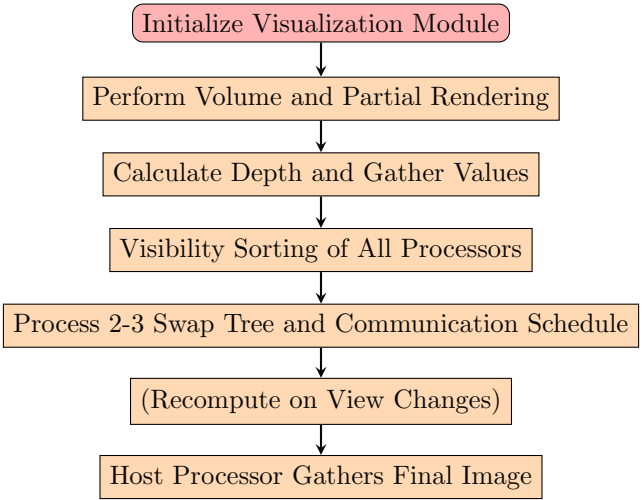
$$\frac{\partial \rho Y_\alpha}{\partial t} = -\frac{\partial \rho u_i Y_\alpha}{\partial x_i} - \frac{\partial \rho V_{\alpha i} Y_\alpha}{\partial x_i} + \omega_\alpha \tag{1}$$

- In situ visualization is integrated directly within the S3D code, ensuring minimal data movement and maintaining synchronization.

Integration with Simulation Side



Integration with Visualization Side



- **Simulation Environment**: Tested with a lifted-jet combustion simulation at JaguarPF, Sandia Lab and Cray XT5, Oak Ridge National Laboratory.
- **Core Configurations**:
  - Each processor core managed a $27 \times 40 \times 40$ region.
  - Tested with four core numbers: 240, 1,920, 6,480, and 15,360.
  - Core configurations were $15 \times 8 \times 2$, $30 \times 16 \times 4$, $45 \times 24 \times 6$, and $60 \times 32 \times 8$.
- **System Details**:
  - **Phase 2**: 240, 1,920, and 6,480 cores on XT5 with 20,928 AMD Opteron processors (2.3 GHz).
  - **Phase 4**: 15,360 cores on XT5 with 37,376 AMD Opteron processors (2.6 GHz).

- **Data Sizes and Timing**:
  - Volume data: Double floating-point (8 bytes); Particle data: Single floating-point (4 bytes).
  - Image resolutions tested: $2{,}048^2$, $1{,}024^2$, and 512^2.
  - Image compositing: 32-bit float precision for RGBA, depth channels.
- **Timing Measurements**:
  - Timing measured for one time step includes simulation, I/O, and visualization.
  - For 6,480 cores and $1{,}024^2$ image resolution: Visualization time = 6.92% of simulation time; I/O time exceeds four times the simulation time.
  - Visualization is performed every 10th time step.

Results

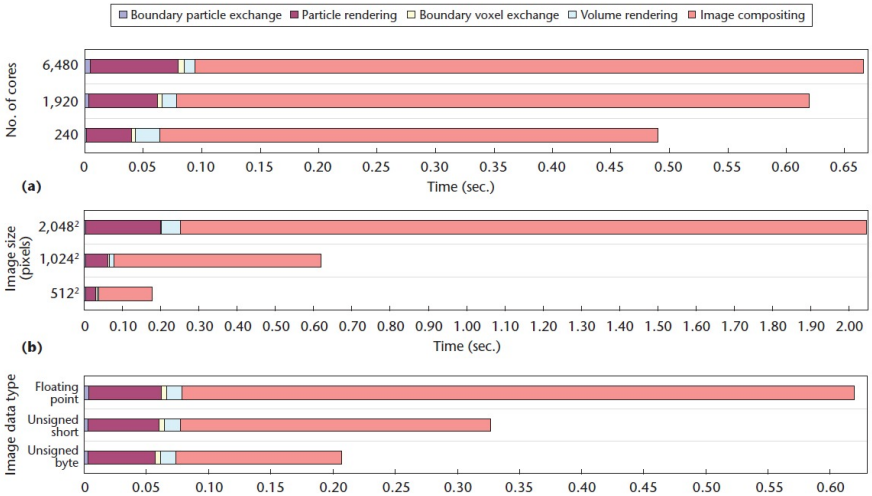
	No. of cores			
	240	**1,920**	**6,480**	**15,360**
Volume rendering				
Volume size	405 × 320 × 80	810 × 640 × 160	1,215 × 960 × 240	1,620 × 1,280 × 320
No. of variables	27	27	27	27
Data size (Gbytes)	2.1	16.7	56.3	133.5
Particle rendering				
No. of particles (millions)	0.8	5.2	17.4	41.1
No. of variables	118	118	118	118
Data size (Gbytes)	0.3	2.5	8.3	19.5
Simulation time (sec.)				
Total	8.1659	8.6680	9.6293	11.867
Reaction rate evaluation	3.8779	3.8924	3.8900	3.3164
Mixture average diffusion calculation	1.2728	1.2948	1.3610	1.3041
Derivative evaluation	0.9246	1.2115	1.6240	3.5597
Other temporal derivative calculation	1.7332	1.8995	2.3677	3.2143
Runge-Kutta integration	0.3472	0.3536	0.3566	0.4052
Tracer advection	0.0102	0.0162	0.0300	0.0668
I/O time (sec.)	8.2675 (101.24%)*	25.611 (295.47%)	42.660 (432.64%)	136.690 (1,151.90%)

Results

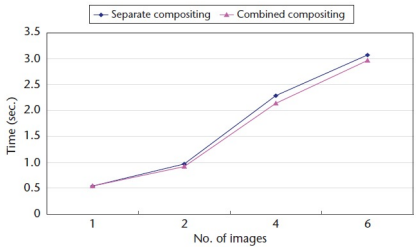
Visualization time with $2{,}048^2$ image resolution (sec.)				
Total	1.7699 (21.67%)	2.0459 (23.60%)	2.6689 (27.72%)	4.3595 (36.74%)
Boundary particle exchange	0.0012	0.0034	0.0041	0.0061
Particle rendering	0.1255	0.1953	0.2710	0.5314
Boundary voxel exchange	0.0023	0.0039	0.0051	0.0080
Volume rendering	0.0834	0.0499	0.0380	0.0230
Image compositing	1.5575	1.7934	2.3507	3.7910
Visualization time with $1{,}024^2$ image resolution (sec.)				
Total	0.4903 (6.00%)	0.6198 (7.15%)	0.6661 (6.92%)	1.0381 (8.75%)
Boundary particle exchange	0.0015	0.0035	0.0044	0.0063
Particle rendering	0.0387	0.0586	0.0759	0.1438
Boundary voxel exchange	0.0031	0.0042	0.0049	0.0081
Volume rendering	0.0212	0.0125	0.0094	0.0059
Image compositing	0.4258	0.5410	0.5715	0.8740
Visualization with 512^2 image resolution (sec.)				
Total	0.1304 (1.60%)	0.1782 (2.06%)	0.1978 (2.05%)	0.2879 (2.43%)
Boundary particle exchange	0.0012	0.0037	0.0046	0.0066
Particle rendering	0.0167	0.0260	0.0345	0.0649
Boundary voxel exchange	0.0023	0.0043	0.0050	0.0083
Volume rendering	0.0053	0.0031	0.0027	0.0015
Image compositing	0.1049	0.1411	0.1510	0.2066

- **Timing Breakdown**:
  - Image compositing, requires interprocessor communication, and takes most time.
  - Volume-rendering time decreases with increasing core count due to smaller screen projection per core.
  - Compositing time increases with more cores due to non-optimized 2-3 swap algo.
  - **C**ompositing time decreases with lower image resolution due to decreased work.
  - **C**ompositing time decreases with lower image data precision (8-bit, 16-bit, 32-bit) for RGBA channels.

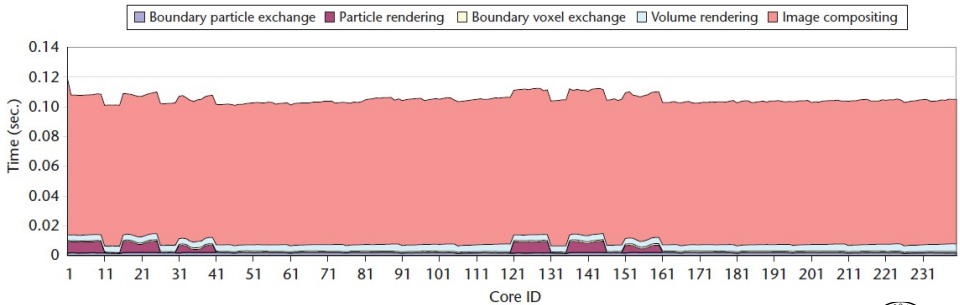
Results


Boundary particle exchange
Particle rendering
Boundary voxel exchange
Volume rendering
Image compositing
No. of cores
6,480
1,920
240
0 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65
Time (sec.)
(a)
Image size (pixels)
2,048²
1,024²
512²
0 0.10 0.20 0.30 0.40 0.50 0.60 0.70 0.80 0.90 1.00 1.10 1.20 1.30 1.40 1.50 1.60 1.70 1.80 1.90 2.00
Time (sec.)
(b)
Image data type
Floating point
Unsigned short
Unsigned byte
0 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60

- **Multivariate Data Visualization**:
  - **Separate-Compositing Mode**: Renders and composites one image at a time.
  - **Combined-Compositing Mode**: Renders and composites multiple images simultaneously, reducing the number of messages exchanged.
  - **Performance gain** from combined-compositing mode is marginal due to low message-passing-interface latency (approximately 1 ms).



- **Visualization Time per Core**:
  - Rendering time (boundary data exchange, particle and volume rendering) is uneven among cores due to domain decomposition and transfer functions.
  - Overall visualization time is balanced among cores because image-compositing time is well balanced.



- **Volume-Rendering Results**:
  - Figure below shows volume-rendering results for six selected variables.

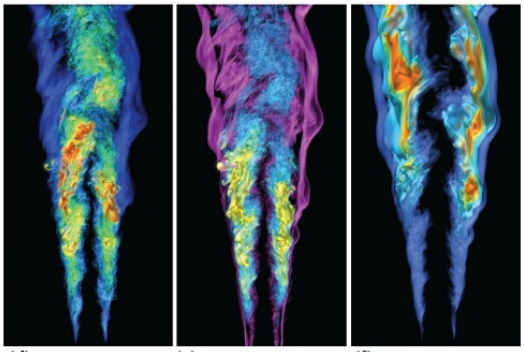
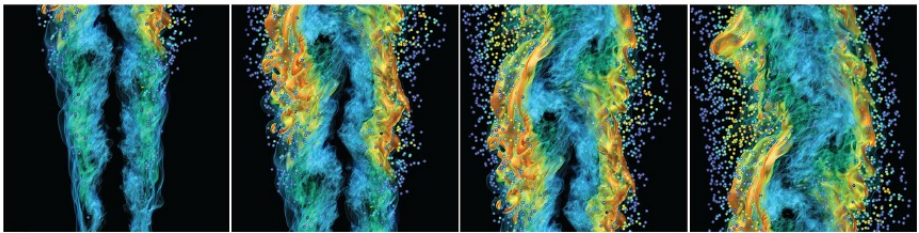


Figure 7. Volume rendering for six selected variables involved in the combustion process: (a) C_2H_4, (b) CH_2O, (c) CH_3, (d) H_2O_2, (e) HO_2, and (f) OH. All six images are generated under the combined-compositing mode in a single pass.

- **Volume-Rendering Results**:
  - Figure below provides detailed views of integrated volume and particle rendering for some variables.



(b)

Figure 8. Detailed views of the volume and particle data mix rendering. (a) The volume variable is CH_2O; the particle variable is HO_2. (b) The volume variable is CH_3; the particle variable is OH. Our high-resolution in situ visualization lets the scientists examine details and monitor the simulation on the fly.

To Summarize

- **Key Advantages:** Provides real-time insights, reduces data storage and transfer needs, and enhances scalability for large-scale simulations.
- **Challenges:** Involves managing computational load, achieving load balance, and optimizing image compositing.
- **Comparison with Other Methods:** Offers more benefits than postprocessing and coprocessing but demands careful integration and advanced computational resources.

Image Compositing Bottleneck

- Image compositing is a key bottleneck in in situ visualization, particularly as image resolution and processor count increase.
- **Potential Solutions:** If costs become too high, we can optimize the compositing algorithm by focusing on two aspects:
  1. **Eliminating Background Pixels:** Instead of using full-size partial images, we can reduce the data by removing background pixel data and focusing only on effective pixels for compositing.
  2. **Enhancing Parallelism:** By leveraging the 2-3 swap algorithm's local communication, cores can operate independently, allowing different levels of the compositing tree to execute tasks concurrently.

1. **Expanding Beyond Visualization:** Extend in situ techniques to include feature extraction, data compression, and statistical analysis, reducing storage and transfer needs.
2. **Real-Time Feature Extraction:** Conduct feature extraction and analysis in situ for real-time insights, enabling dynamic exploration and faster decision-making.
3. **Integrating Additional Tasks:** Incorporate more processing tasks into the in situ workflow to boost efficiency and provide a deeper understanding of simulation results.

Questions

Thank you!