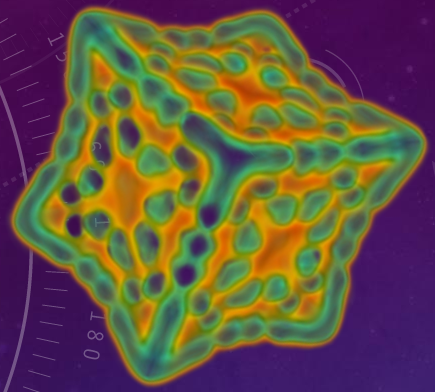
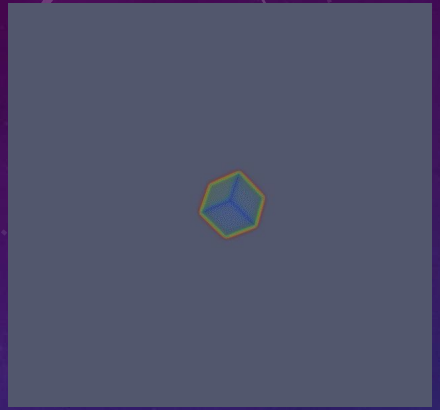
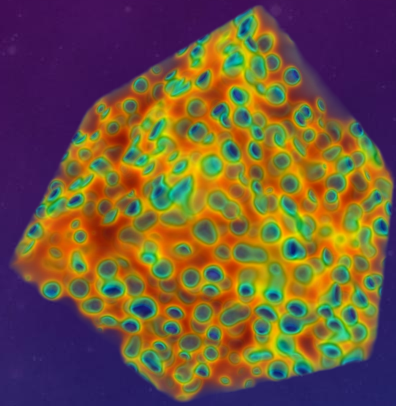
جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

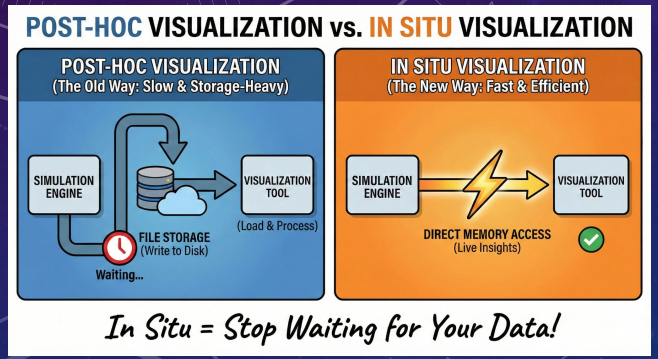






Scientific Visualization 210

Introduction to In Situ Visualization



KAUST Visualization Core Lab

Dr. James Kress

April 7, 2026

Initial Setup: Docker

Install Docker and pull the docker image now so that it completes by the time we need it:

1. Install Docker following these instructions:

   https://github.com/jameskress/Visualization_Vignettes/tree/master/Miniapps/gray-scott#running-with-docker

2. Get the latest image:

   docker pull ghcr.io/jameskress/visualization_vignettes:latest

Introducing the KVL

Who we are and what services we offer

The KVL Team

Dr. Sohaib Ghani
(LEAD STAFF SCIENTIST)

- VISUAL ANALYTICS
- INFORMATION VIS
- STATISTICAL ANALYSIS

Dr. James Kress
HPC SCIVIS

- VISUALIZATION SOFTWARE
- HPC INSITU VISUALIZATION
- DISTRIBUTED VISUALIZATION

Dr. Ronell Sicat
SCIVIS, AR/VR

- SCIENTIFIC VISUALIZATION
- SEGMENTATION & 3D ANALYSIS
- AR/VR DEVELOPMENT

Dr. Didier Barradas
Data Scientist

- DATA SCIENCE
- MACHINE LEARNING
- DEEP LEARNING

Dr. Abdelghafour Halimi
Data Scientist

- DATA SCIENCE
- MACHINE LEARNING
- DEEP LEARNING

KVL Core Expertise

Core Services

Training and Workshops

- Scientific Visualization Workshop Series
- Data Science Workshop Series
- AI Tools and Techniques Workshop Series

Data Visualization & Data Science Workflows

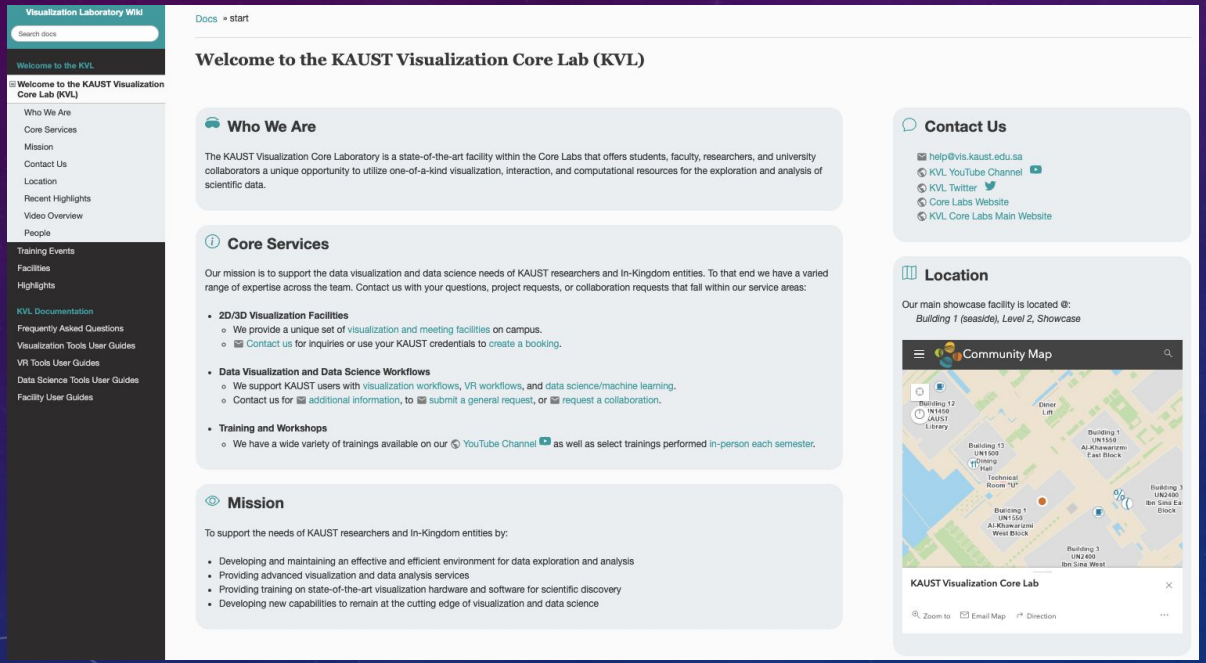
- HPC Visualization
- Image Segmentation & 3D Analysis
- AR/VR
- Large Scale AI & Data Science

Facilities Support

- AR/VR Hub
- Tiled Display Walls

KVL Wiki

--- Find Highlights, Training Info, and More ---



https://wiki.vis.kaust.edu.sa

KVL Training Events

--- Available In-Person and Online ---

- Scientific Visualization Workshop Series
  - ParaView, VisIt, Avizo/Amira
- Data Science Workshop Series
  - Shell, Conda, Python, Git, and more
- Hands-on AI Tools and Techniques Workshop Series
  - Intro to Machine Learning/Deep Learning, Visualization for Data Science

https://www.youtube.com/@kaustvislab

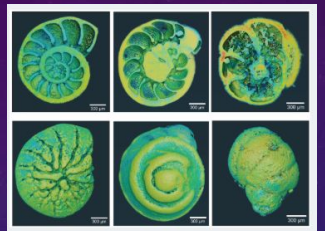
Data Visualization and Science Workflows

KVL Collaborates in areas such as:

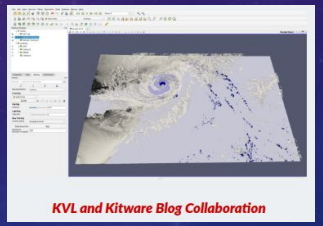
- HPC Visualization
- Scientific and Information Visualization
- AR/VR
- Image Segmentation and 3D Analysis
- Large-scale AI and Data Science

Send us an email if you have a collaborative project that can benefit from our expertise:

help@vis.kaust.edu.sa



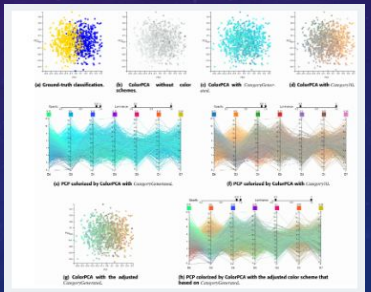
Micro-CT and Holographic Visualization of Late Cretaceous Benthic Foraminifera in Saudi Arabia



KVL and Kitware Blog Collaboration

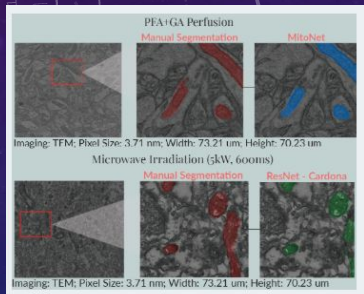
KVL Releases Open Source Software to Visualize Supercomputing Simulations

Mobile AR Visualization of Giant Red Sea Coral

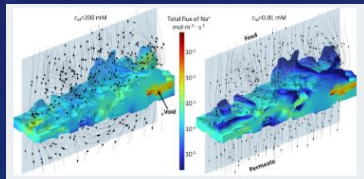


ColorPCA: Scalable Colored Dimensionality Reduction for Unlabeled High-Dimensional Data

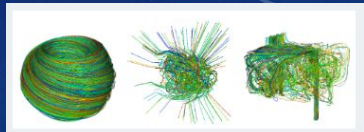
KVL Co-Organized Visualization Workshop



Segmentation and 3D Reconstruction of Glycogen and Mitochondria in Microwave-fixed Brain EM



3D Modeling of Composite Membranes from TEM



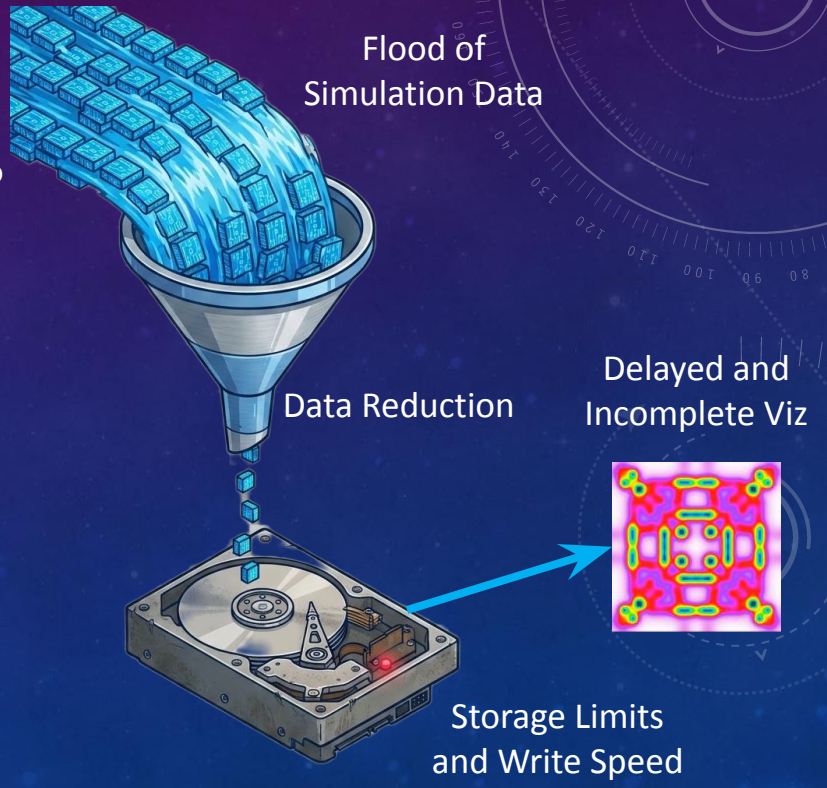
Analyzing Particle Advection Performance

In Situ Introduction

Why we move analysis to the data

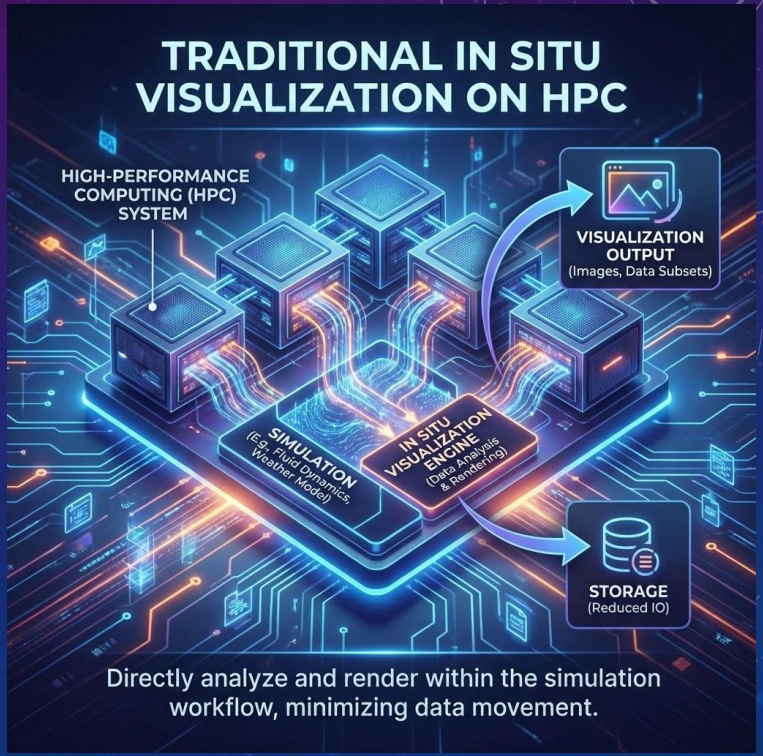
The Post-Hoc Bottleneck

- The I/O Wall
  - Writing to disk is the slowest part of the simulation.
  - As compute scales faster than storage, the time required to write data increases.
- The Data Compromise
  - To prevent the simulation from stalling, we save data less often.
  - Result: Large gaps in the timeline (Missing Data).
- 3. The Insight Lag
  - Analysis cannot begin until a sufficient number of checkpoints have been written.
  - Result: Errors or discoveries are not found quickly (Time-to-Insight).



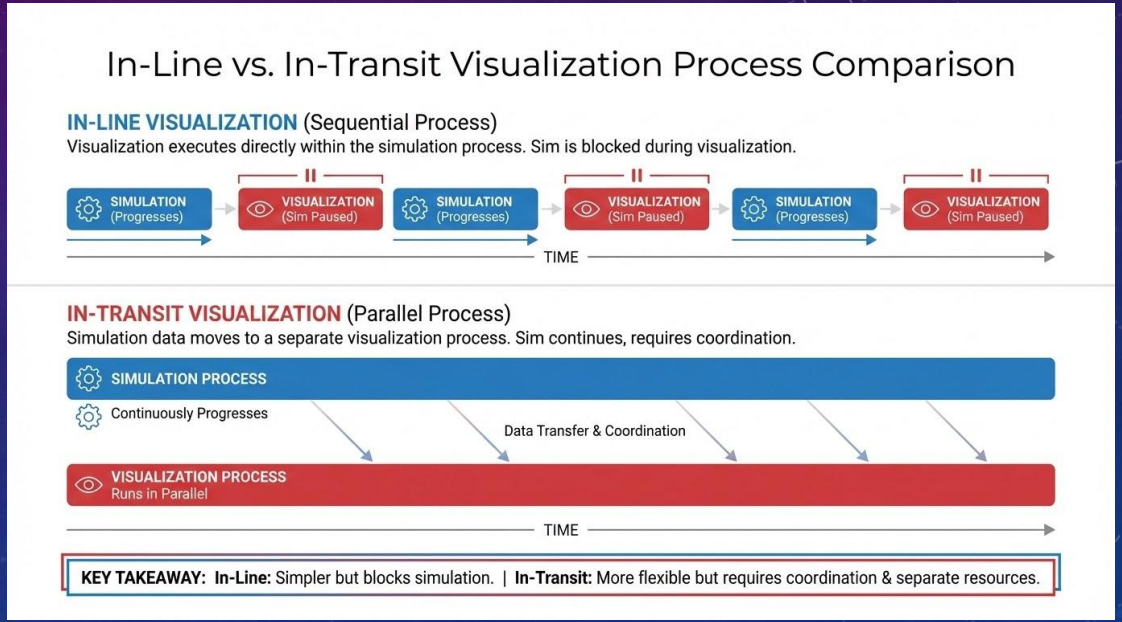
The In Situ Solution

- What is in situ?
  - Produce visualization & analysis during an active simulation.
  - Multiple ways in situ can be accomplished (in line, in transit, hybrid).
- My application already does this, right?
  - Some applications do have visualizations built-in.
- However,…
  - The goal of in situ visualization is to make visualization more adaptive, resilient, and familiar than coding every possible visualization into a simulation, which is brittle and error prone.
  - In situ is especially relevant to HPC applications.



Two Main Types of In Situ: In line and In Transit

- In Line executes visualization directly within the simulation process
  - In Line is simpler but can negatively impact simulation performance.
- In Transit moves simulation data to a separate visualization process
  - In Transit offers more flexibility and better resource separation

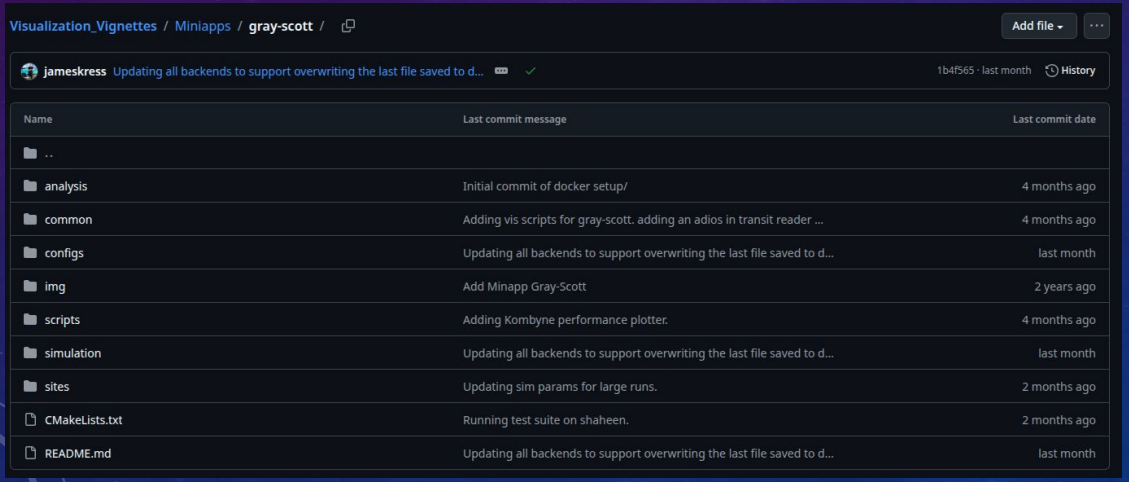


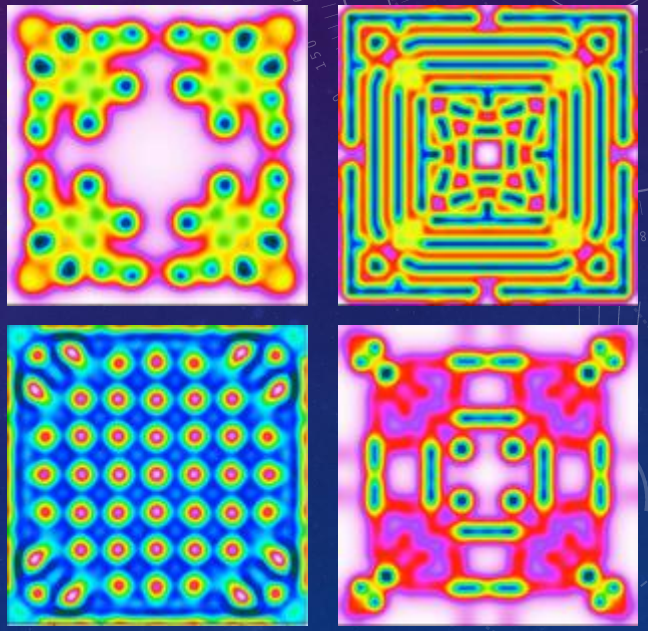
Introduction to Gray-Scott

A 3D Reaction-Diffusion model

- Simulates the chemical reaction between the chemicals ***U*** and ***V***

This miniapp is provided as part of the Visualization Vignettes repo that I have created, and includes example run scripts and configurations.





Gray-Scott Capabilities

Native In Situ Integrations (Multi-Backend) This miniapp is heavily instrumented, allowing you to switch between different visualization and I/O approaches without changing the source code:

- **Catalyst:** For interactive in situ visualization with ParaView.
- **Ascent:** For flyweight, many-core (GPU) native analysis.
- **ADIOS2:** For high-performance data transport (In Transit) and checkpointing.
- **Kombyne** is a commercial, supported in situ visualization tool developed by Intelligent Light.
- **Standard I/O:** capable of writing standard parallel VTK files (.pvti) for traditional post-processing.

Runtime Configuration

- **JSON-Based Settings:** All simulation parameters (reaction rates, grid dimensions, number of steps) are controlled via a file.
- **Backend Toggling:** You can enable or disable specific in situ backends (e.g., turn off File I/O and turn on Catalyst) just by editing the config file or run script.

HPC Scalability

- **Massively Parallel:** Written in C++ and MPI, it automatically decomposes the 3D domain across available processors.
- **From Laptop to Supercomputer:** It is designed to scale from a single core on your laptop to **tens of thousands of cores** on systems like Shaheen III.

Takeaway: We use Gray-Scott because it behaves like a real HPC application—it generates massive data and scales indefinitely—making it the perfect testbed for in situ tools.

Setting up Gray-Scott with Docker

1. Install Docker following these instructions:

   https://github.com/jameskress/Visualization_Vignettes/tree/master/Miniapps/gray-scott#running-with-docker

2. Get the latest image:

   ```
   docker pull ghcr.io/jameskress/visualization_vignettes:latest
   ```

3. Create a local data directory, e.g.:

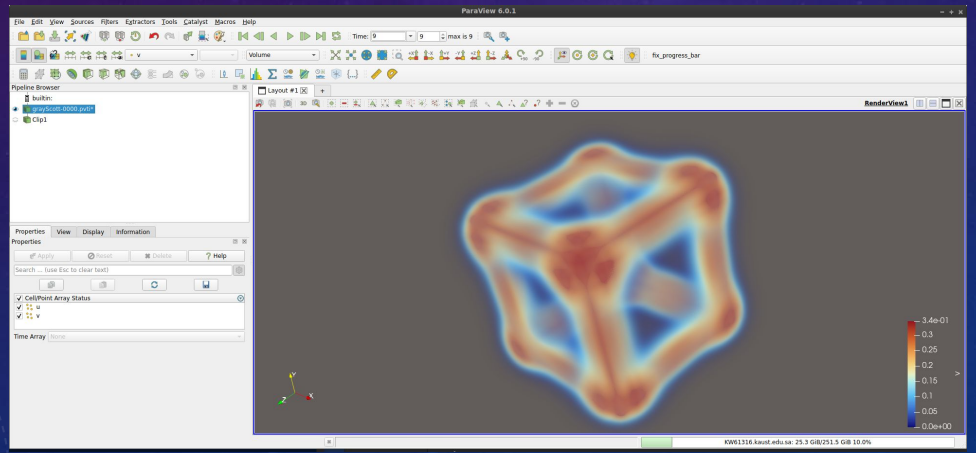
   ```
   mkdir data
   ```

4. Run docker container:

   https://github.com/jameskress/Visualization_Vignettes/tree/master/Miniapps/gray-scott#step-22-run-simulations-and-access-files

The “Baseline” — Traditional VTK Workflow

```
# Navigate to the shared data directory
cd /app/data
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-vtk-pvti.json
# On your local computer, open ParaView and visualize files in your `data` dir
```



```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.01,
  "k": 0.05,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 100,
  "noise": 0.0000001,
  "output_file_name": "grayScott-%04ts.vti",
  "output_type": "pvti",
  "catalyst_script_path": "",
  "catalyst_lib_path": "",
  "checkpoint": false,
  "checkpoint_freq": 0,
  "checkpoint_output": "",
  "restart": false,
  "restart_input": "",
  "adios_config": "",
  "adios_span": false,
  "adios_memory_selection": false,
  "mesh_type": "",
  "kombynelite_script_path": "",
  "burn_in_steps": 0,
  "overwrite_last_step": false
}
```

Hands–On Session 1

In Line In Situ: Catalyst & Ascent

What is Catalyst2?

Catalyst is a standalone API specification. It defines a standard "contract" that simulations use to announce their data to the outside world.

How It Works:

- Conduit Blueprints:
  - The simulation describes its data (mesh, fields, topologies) using Conduit, a simplified JSON-like schema, vs. VTK objects.
- Zero-Copy:
  - The API simply points to your simulation's existing memory. It does not copy or convert data itself.
- Backend Agnostic:
  - The API doesn't know what will process the data. It just says, "Here is the data."

Why Separate the API?

It provides ABI (Application Binary Interface) stability. You can update the visualization backend (e.g., install a newer version of ParaView) without needing to recompile your simulation.

Note: While the API supports various backends, its primary and most robust implementation is ParaView.

What is ParaView Catalyst?

ParaView Catalyst is the concrete implementation we use to connect the Catalyst API to the ParaView visualization engine.

The Workflow:

- Translation: It accepts the Conduit Blueprint from the API and instantly wraps it as VTK objects (in memory).
- Execution: It executes standard ParaView Python scripts to filter, render, and analyze that data.
- Output: It generates images, geometry files, or livestreams the data to a GUI.

Why We Use It:

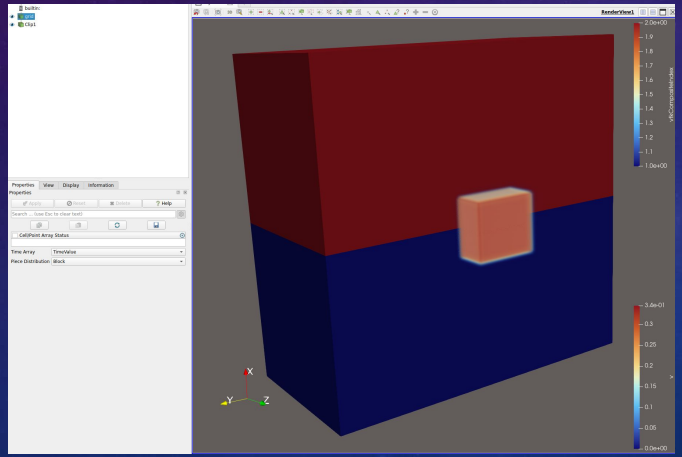
- Interactive Python Pipelines: You can use the exact same Python scripts from the ParaView GUI to drive your in situ analysis.
- Live Steering: It supports connecting a remote ParaView GUI to the running simulation to visualize results and modify parameters in real-time.

Takeaway:

In this workshop, we use the Catalyst API to expose the data, and ParaView Catalyst to visualize it.

Run Gray-Scott Using Catalyst File Writer

```
# Navigate to the shared data directory
cd /app/data
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-catalyst-file-io.json
# On your local computer, open ParaView and visualize the '.vtpd' files
```



```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.01,
  "k": 0.05,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 100,
  "noise": 0.0000001,
  "output_file_name": "grayScott-%04ts.vtpd",
  "output_type": "catalyst_io",
  "catalyst_script_path": "",
  "catalyst_lib_path": "/opt/paraview/lib/catalyst",
  "checkpoint": false,
  "checkpoint_freq": 0,
  "checkpoint_output": "",
  "restart": false,
  "restart_input": "",
  "adios_config": "",
  "adios_span": false,
  "adios_memory_selection": false,
  "mesh_type": "",
  "kombynelite_script_path": "",
  "burn_in_steps": 0,
  "overwrite_last_step": false
}
```

Run Gray-Scott Using Catalyst In Situ

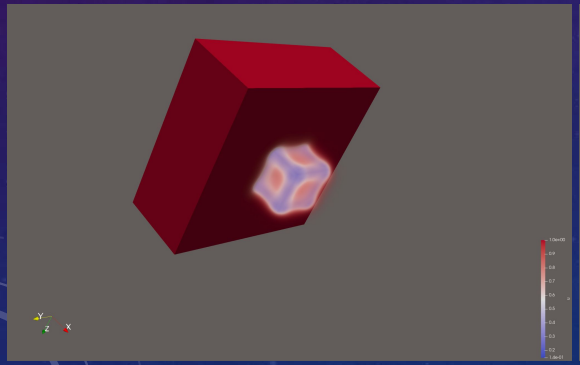
```
# Navigate to the shared data directory
cd /app/data
# Open and view the Catalyst script
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-catalyst-insitu.json
# On your local computer, open the images in the ‘datasets’ directory
```


```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.01,
  "k": 0.05,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 100,
  "noise": 0.0000001,
  "output_file_name": "",
  "output_type": "catalyst_insitu",
  "catalyst_script_path": "catalyst-extract-jpg.py",
  "catalyst_lib_path": "/opt/paraview/lib/catalyst",
  "checkpoint": false,
  "checkpoint_freq": 0,
  "checkpoint_output": "",
  "restart": false,
  "restart_input": "",
  "adios_config": "",
  "adios_span": false,
  "adios_memory_selection": false,
  "mesh_type": "",
  "kombynelite_script_path": "",
  "burn_in_steps": 0,
  "overwrite_last_step": false
}
```

Run Gray-Scott Using Catalyst In Situ

Modifying the Catalyst script

```
# Navigate to the shared data directory
cd /app/data
# Open and view the Catalyst script, update the settings file as shown
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-catalyst-insitu.json
# On your local computer, open the images in the ‘datasets’ directory
```



```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.01,
  "k": 0.05,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 100,
  "noise": 0.0000001,
  "output_file_name": "",
  "output_type": "catalyst_insitu",
  "catalyst_script_path": "catalyst-multi-pipeline.py",
  "catalyst_lib_path": "/opt/paraview/lib/catalyst",
  "checkpoint": false,
  "checkpoint_freq": 0,
  "checkpoint_output": "",
  "restart": false,
  "restart_input": "",
  "adios_config": "",
  "adios_span": false,
  "adios_memory_selection": false,
  "mesh_type": "",
  "kombynelite_script_path": "",
  "burn_in_steps": 0,
  "overwrite_last_step": false
}
```

Creating a Catalyst Script

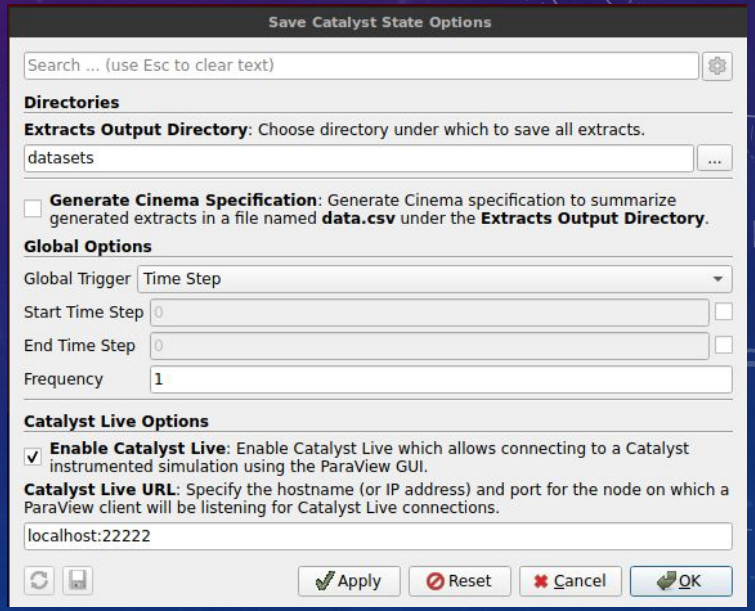
Navigate to the shared data directory

Open one of the Gray-Scott data sets in ParaView

Create an interesting pipeline

Add an extractor (png, data, etc)

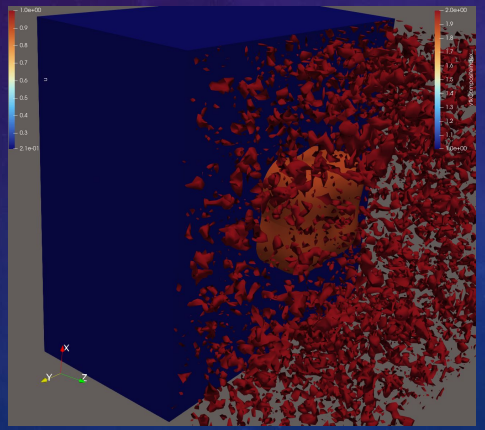
File->Save Catalyst State



Run Gray-Scott Using Catalyst In Situ

Using Your Catalyst script

```
# Navigate to the shared data directory
cd /app/data
# Update the settings file as shown
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-catalyst-insitu.json
# On your local computer, open the images in the ‘datasets’ directory
```



```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.01,
  "k": 0.05,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 100,
  "noise": 0.0000001,
  "output_file_name": "",
  "output_type": "catalyst_insitu",
  "catalyst_script_path": "catalyst-custom.py",
  "catalyst_lib_path": "/opt/paraview/lib/catalyst",
  "checkpoint": false,
  "checkpoint_freq": 0,
  "checkpoint_output": "",
  "restart": false,
  "restart_input": "",
  "adios_config": "",
  "adios_span": false,
  "adios_memory_selection": false,
  "mesh_type": "",
  "kombynelite_script_path": "",
  "burn_in_steps": 0,
  "overwrite_last_step": false
}
```

Run Gray-Scott Using Catalyst In Situ

Using Your Catalyst script and connect live to ParaView ~ Setup

Step 1: Prepare ParaView (Host Machine)

1. Launch ParaView.
2. Click **Catalyst → Connect**.
3. Accept default port 22222 and click **OK**.
    - *ParaView is now listening…*
4. *Click **Catalyst → Pause Simulation***
    - *Or we will miss it since the sim runs fast*

Step 2: Identify the Host IP: The container needs to know where to send the data.

- **Linux:** Use the Docker Bridge IP (usually `172.17.0.1`).
- **Windows / Mac:** Use the alias `host.docker.internal`.
- **Alternative:** Find your physical LAN IP (`ipconfig` or `ip addr`).

Step 3: Update the Catalyst Script: Using the code to the right

Step 3: Setup the Docker Terminal: Set the environment:

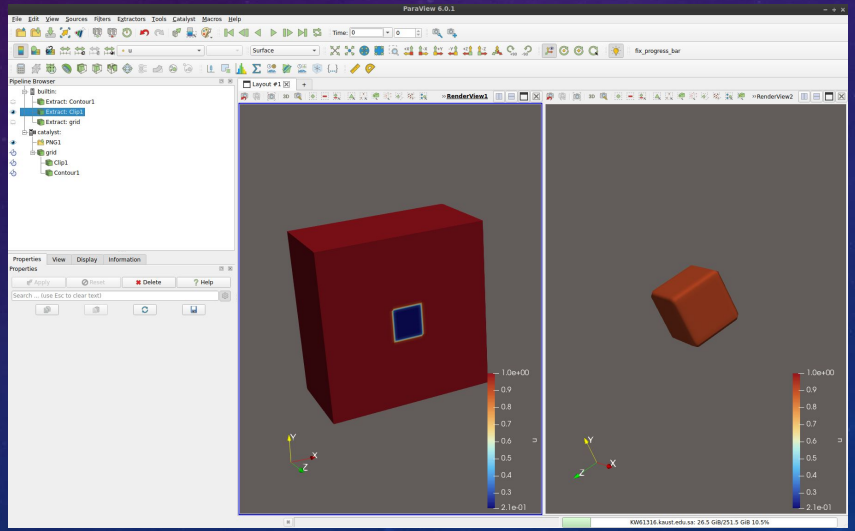
```
# Point to your Host IP (Choose the one from Step 2)
export CATALYST_CLIENT=172.17.0.1
```

```
import os

# Dynamically determine client connection
clientport = 22222
clienthost = "localhost"

# Check if an external IP is provided (e.g., from Docker)
if "CATALYST_CLIENT" in os.environ:
    clienthost = os.environ["CATALYST_CLIENT"]

# Apply the connection URL
options.CatalystLiveURL = str(clienthost) + ":" + str(clientport)
```

Run Gray-Scott Using Catalyst In Situ

Using Your Catalyst script and connect live to ParaView ~ Running

```
# Navigate to the shared data directory
cd /app/data
# Update the settings file as shown
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-catalyst-insitu.json
# Check the connection in ParaView
```



```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.01,
  "k": 0.05,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 100,
  "noise": 0.0000001,
  "output_file_name": "",
  "output_type": "catalyst_insitu",
  "catalyst_script_path": "catalyst-custom.py",
  "catalyst_lib_path": "/opt/paraview/lib/catalyst",
  "checkpoint": false,
  "checkpoint_freq": 0,
  "checkpoint_output": "",
  "restart": false,
  "restart_input": "",
  "adios_config": "",
  "adios_span": false,
  "adios_memory_selection": false,
  "mesh_type": "",
  "kombynelite_script_path": "",
  "burn_in_steps": 0,
  "overwrite_last_step": false
}
```

What is Ascent?

Ascent is an in situ visualization and analysis infrastructure designed specifically for exascale and many-core (GPU) architectures.

Key Strengths (The "Lean" Approach):

- Lightweight Footprint:
  - Designed to use minimal memory and CPU resources, leaving the maximum amount of power available for your simulation.
- Born for GPUs:
  - Built on top of VTK-m, allowing it to run visualization algorithms natively on GPUs (CUDA, HIP) and multi-core CPUs without moving data.
- Declarative Actions:
  - You control the visualization using a simple YAML or JSON file (e.g., "extract isosurface," "save image"). No complex coding required.

Integration:

Like Catalyst2, Ascent uses Conduit to describe data.

Takeaway:

Ascent is the "high-performance, low-overhead" option—ideal for massive simulations

Run Gray-Scott Using Ascent Data Extracts

```
# Navigate to the shared data directory

cd /app/data

# Open and view the Ascent scripts (settings, options, pipeline)
```

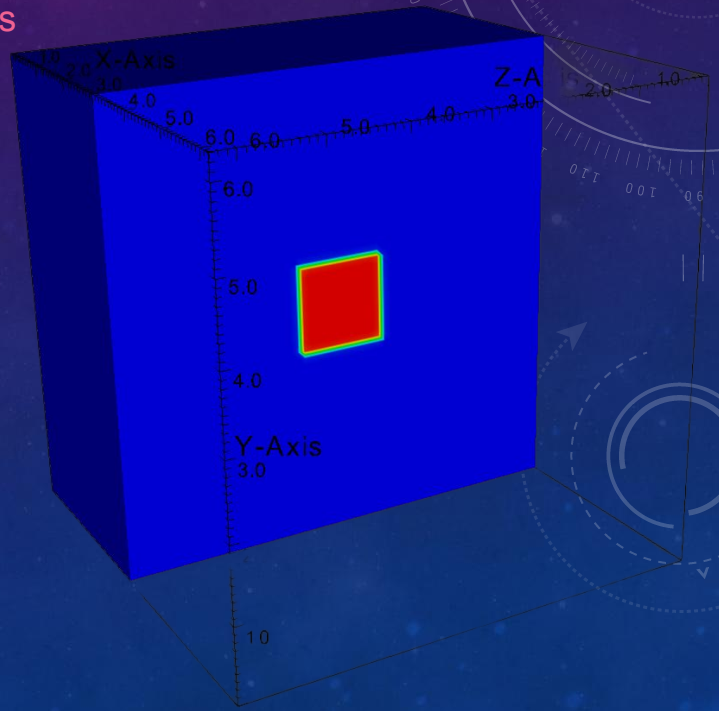
```
runtime:
  type: "ascent" # "ascent" "flow" "empty"
  vtkm:
    backend: "openmp"
actions_file: "ascent-save-data.yaml"
messages: "quiet" #"verbose" "quiet" # print extra ascent related messages
timings: "false" #"true" "false" # print a timing file per rank for filters
```

```
- action: "add_extracts"
  extracts:
    contour_data_extract:
      type: "relay"
      params:
        path: "ascent_data"
        protocol: "blueprint/mesh/hdf5"
```

```
{
  "L": 64,
  "Du": 0.2,
  "Dv": 0.1,
  "F": 0.01,
  "k": 0.05,
  "dt": 2.0,
  "plotgap": 10,
  "steps": 100,
  "noise": 0.0000001,
  "output_file_name": "grayScott-%04ts.vti",
  "output_type": "ascent",
  "catalyst_script_path": "",
  "catalyst_lib_path": "",
  "checkpoint": false,
  "checkpoint_freq": 0,
  "checkpoint_output": "",
  "restart": false,
  "restart_input": "",
  "adios_config": "",
  "adios_span": false,
  "adios_memory_selection": false,
  "mesh_type": "",
  "kombynelite_script_path": "",
  "burn_in_steps": 0,
  "overwrite_last_step": false
}
```

Run Gray-Scott Using Ascent Data Extracts

```
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-ascent.json
# On your local computer, open the ascent '*.root' files in VisIt
```



Run Gray-Scott Using Ascent Complex Pipeline

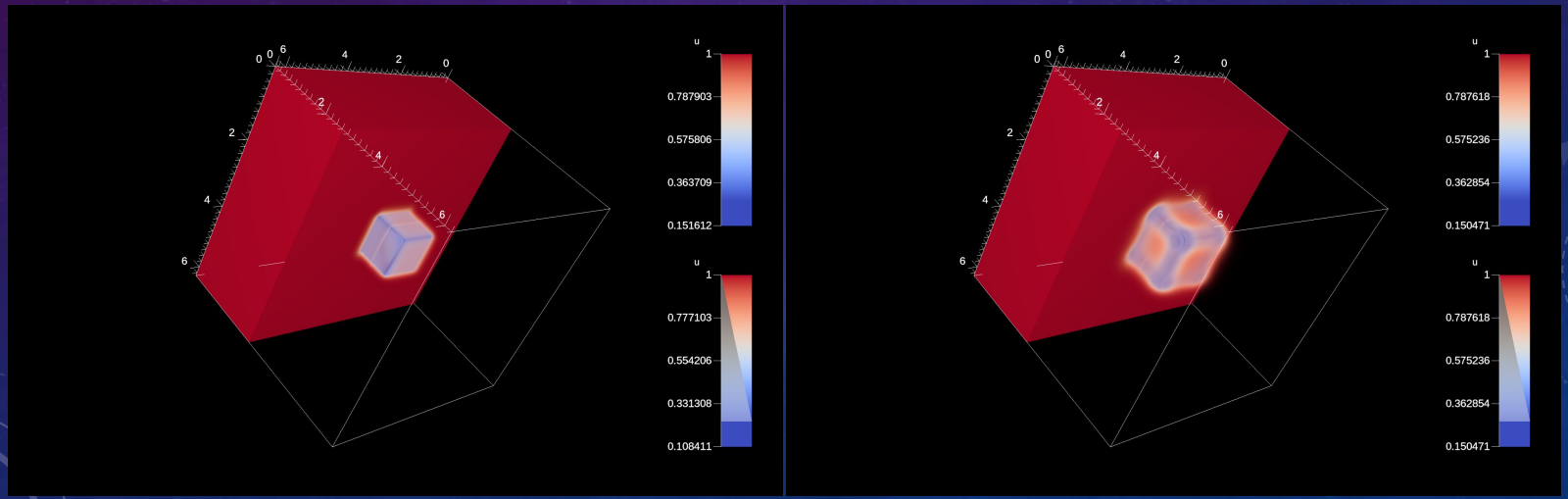
```
# Navigate to the shared data directory

cd /app/data

# Open and view the Ascent scripts (settings, options, pipeline)
```

```
runtime:
  type: "ascent" # "ascent" "flow" "empty"
  vtkm:
    backend: "openmp"
actions_file: "ascent-multi-pipeline.yaml"
messages: "quiet" #"verbose" "quiet" # print extra ascent related messages
timings: "false" #"true" "false" # print a timing file per rank for filters
```

```
# Action 1: Define a reusable pipeline of filters.
- action: "add_pipelines"
  pipelines:
    clip_pipeline:
      f1:
        type: "clip"
        params:
          invert: true
          plane:
            point:
              x: 2.62658
              y: 3.25000
              z: 3.25000
            normal:
              x: 1.0
              y: 0.0
              z: 0.0

# Action 2: Render a scene that uses the pipeline.
- action: "add_scenes"
  scenes:
    multi_view_scene:
      plots:
        # PLOT 1: Volume rendering of the full, original dataset.
        full_volume_plot:
          type: "volume"
          field: "u"
          color_table:
            control_points:
              - { type: "rgb", position: 0.145, color: [0.231, 0.298, 0.753] }
              - { type: "rgb", position: 0.572, color: [0.865, 0.865, 0.865] }
              - { type: "rgb", position: 1.0,   color: [0.706, 0.016, 0.149] }
              - { type: "alpha", position: 0.145, alpha: 1.0 }
              - { type: "alpha", position: 1.0,   alpha: 0.0 }

        # PLOT 2: Surface rendering of the clipped data.
        clipped_surface_plot:
          type: "pseudocolor"
          field: "u"
          pipeline: "clip_pipeline"
          color_table:
            control_points:
              - { type: "rgb", position: 0.145, color: [0.231, 0.298, 0.753] }
              - { type: "rgb", position: 0.572, color: [0.865, 0.865, 0.865] }
              - { type: "rgb", position: 1.0,   color: [0.706, 0.016, 0.149] }
              - { type: "alpha", position: 0.0, alpha: 1.0 }
              - { type: "alpha", position: 1.0, alpha: 1.0 }

      renders:
        render1:
          image_width: 2113
          image_height: 1338
          annotations: "on"
          image_prefix: "ascent_multi_render_%06d.png"

          camera:
            zoom: 2.0
            position: [13.662, 19.169, -7.296]
            look_at:  [3.25, 3.25, 3.25]
            up:       [-0.622, -0.107, -0.776]
```

Run Gray-Scott Using Ascent Complex Pipeline

```
# Run the simulation using the local executable and settings files
mpirun -np 2 ./gray-scott --settings-file=./settings-ascent.json
# On your local computer, open the ascent ‘png’ files
```



Hands–On Session 2

In Transit In Situ Using ADIOS2: Catalyst and Ascent

What is ADIOS2?

ADIOS2 (Adaptable Input Output System) is a high-performance framework primarily designed for extreme-scale I/O and data movement.

Move then Visualize

The Role of ADIOS2 is to act as the high-performance transport bridge between the simulation and visualization.

- On Simulation Nodes:
  - The code simply "writes" data to ADIOS2. It creates no isosurfaces and renders no images.
- On Visualization Nodes:
  - A separate process (running Catalyst or Ascent) "reads" the stream from ADIOS2 and performs the visualization.

Why Do It This Way?

- Safety:
  - If the visualization crashes or runs out of memory, it does not kill your long-running simulation.
- Asynchronous:
  - The simulation hands off the data and immediately continues to the next time step. It does not wait for the visualization to finish.
- Hardware Matching:
  - You can run the simulation on CPU nodes and the visualization on GPU nodes, optimizing resources for each task.

Takeaway:

We use ADIOS2 to move the data out of the simulation so we can analyze it somewhere else in real-time.

Run Gray-Scott Using Using ADIOS

Creating a BP file

```
# Navigate to the shared data directory
cd /app/data
# View the settings file
# Run the simulation using the local executable and settings files
mpirun -np 4 ./gray-scott --settings-file=./settings-adios-memselect.json
# On your local computer:
# Inspect with bpls:
bpls -al gs-adios-memselect.bp

# Visualize in ParaView:
# Open your .bp file in ParaView (e.g., gs-adios-memselect.bp).
# In the "Open File" dialog, make sure to select the ADIOS2FidesReader or FidesReader depending on ParaView version.
# You will initially see the data with visible gaps between the blocks from each MPI rank.
# To create a seamless image, select the dataset in the Pipeline Browser and apply the filter:
  Filters → Data Analysis → Stitch Image Data With Ghosts
# The output of the Stitch filter will be a single, continuous grid ready for further visualization.
```

```
{
    "L": 64,
    "Du": 0.2,
    "Dv": 0.1,
    "F": 0.01,
    "k": 0.05,
    "dt": 2.0,
    "plotgap": 10,
    "steps": 100,
    "noise": 0.0000001,
    "output_type": "adios",
    "output_file_name": "gs-adios-memselect.bp",
    "catalyst_script_path": "",
    "catalyst_lib_path": "",
    "checkpoint": false,
    "checkpoint_freq": 0,
    "checkpoint_output": "ckpt.bp",
    "restart": false,
    "restart_input": "ckpt.bp",
    "adios_config": "adios2.xml",
    "adios_span": false,
    "adios_memory_selection": true,
    "mesh_type": "",
    "kombynelite_script_path": "",
    "burn_in_steps": 0,
    "overwrite_last_step": false
}
```

Run Gray-Scott Using Using ADIOS

Post-Hoc Reader with Catalyst

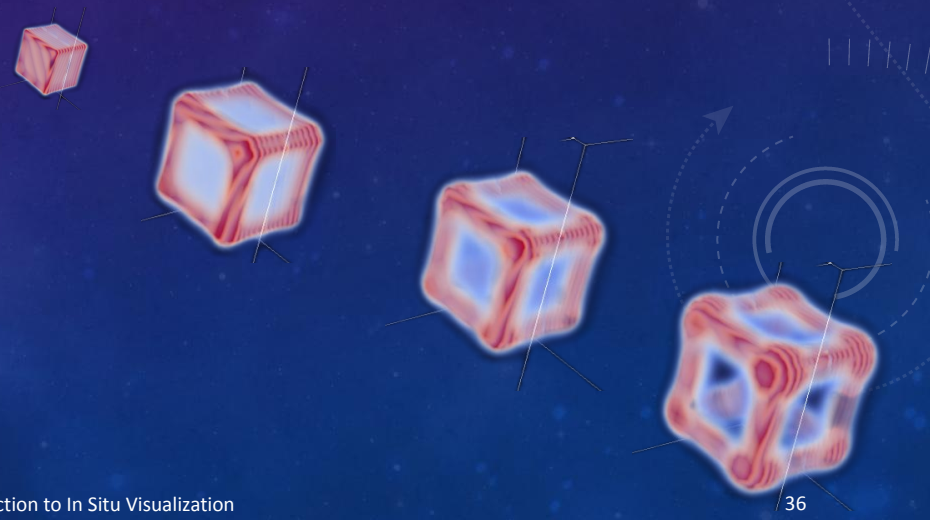
```
# Navigate to the shared data directory
cd /app/data
# View the settings file
# Run the reader using the local executable and settings files to save vtk files
mpirun -n 4 ./analysis-reader \
    --file gs-adios-memselect.bp \
    --block-mode repartition \
    --settings settings-catalyst-file-io.json
# This produces the 'vtpd' files just as in line mode does
# Run the reader using the local executable and settings files to save images
mpirun -n 4 ./analysis-reader \
    --file gs-adios-memselect.bp \
    --block-mode repartition \
    --settings settings-catalyst-insitu.json
```

Run Gray-Scott Using Using ADIOS

Post-Hoc Reader with Ascent

```
# Navigate to the shared data directory

cd /app/data

# View the settings file

# Run the reader using the local executable and settings files

mpirun -n 2 ./analysis-reader \
    --file gs-adios-memselect.bp \
    --block-mode repartition \
    --settings settings-ascent.json

# What do you notice if you change the number of processes up or down?
```

```
runtime:
  type: "ascent" # "ascent" "flow" "empty"
  vtkm:
    backend: "openmp"
actions_file: "ascent-extract-png.yaml"
messages: "quiet" #"verbose" "quiet" # print extra ascent related messages
timings: "false" #"true" "false" # print a timing file per rank for filters
```



Run Gray-Scott Using Using ADIOS

In Transit Reader with Catalyst

```
# Navigate to the shared data directory
cd /app/data
# View the ADIOS settings file and modify
# Launch simulation in the background
mpirun -n 16 ./gray-scott \
      --settings-file=./settings-adios-memselect.json &

# Launch the reader to connect to the stream
mpirun -n 2 ./analysis-reader --file=gs-adios-memselect.bp \
      --settings=settings-catalyst-insitu.json \
      --engine=SST \
      --block-mode repartition
```

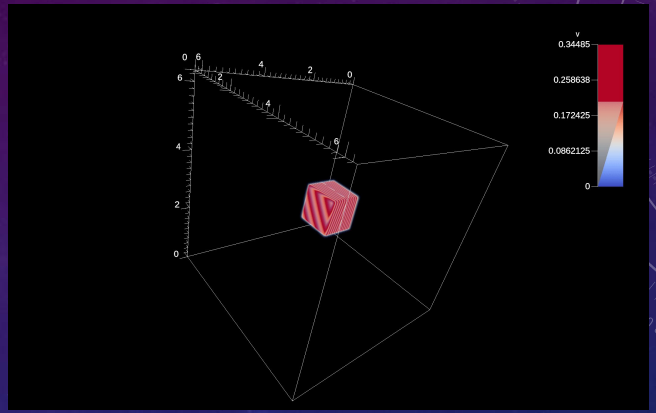
adios2.xml

```
<engine type="SST">
    <parameter key="verbose" value="0"/>
    <!-- SST engine parameters -->
    <parameter key="RendezvousReaderCount" value="1"/>
    <parameter key="QueueLimit" value="5"/>
    <parameter key="QueueFullPolicy" value="Block"/>
    <parameter key="DataTransport" value="evpath"/>
    <!-- BP5/SST engine parameters -->
    <parameter key="OpenTimeoutSecs" value="60.0"/>
    <!-- BP5 engine parameters -->
    <parameter key="NumAggregators" value="1"/>
    <parameter key="AsyncWrite" value="false"/>
    <!-- BP5 reader parameter -->
    <!--parameter key="SelectSteps" value="0:n:10"/-->
  </engine>
</io>
```

Run Gray-Scott Using Using ADIOS

In Transit Reader with Ascent

```
# Navigate to the shared data directory
cd /app/data
# View the ADIOS settings file and modify
# Launch simulation in the background
mpirun -n 16 ./gray-scott \
      --settings-file=./settings-adios-memselect.json &

# Launch the reader to connect to the stream
mpirun -n 2 ./analysis-reader --file=gs-adios-memselect.bp \
      --settings=settings-ascent.json \
      --engine=SST \
      --block-mode repartition
```



adios2.xml

```
<engine type="SST">
      <parameter key="verbose" value="0"/>
      <!-- SST engine parameters -->
      <parameter key="RendezvousReaderCount" value="1"/>
      <parameter key="QueueLimit" value="5"/>
      <parameter key="QueueFullPolicy" value="Block"/>
      <parameter key="DataTransport" value="evpath"/>
      <!-- BP5/SST engine parameters -->
      <parameter key="OpenTimeoutSecs" value="60.0"/>
      <!-- BP5 engine parameters -->
      <parameter key="NumAggregators" value="1"/>
      <parameter key="AsyncWrite" value="false"/>
      <!-- BP5 reader parameter -->
      <!--parameter key="SelectSteps" value="0:n:10"/-->
    </engine>
  </io>
```

Wrap Up

Final Thoughts and Q/A

Scaling Up to Ibex, Shaheen, or Your Own Cluster

Gray-Scott can be built and run on Ibex, Shaheen, or your own cluster

- It is necessary to build it on the bare metal using the native compilers for best performance
- We have run it on 50% of Shaheen already, with plans for larger scales soon

Catalyst

- Has been scaled to at least 1 million cores
  - “The feasibility of extreme-scale in situ processing"” &
    “In situ visualization of large-scale turbulence simulations in Nek5000 with ParaView Catalyst”

Ascent

- Has been scaled to full scale of Frontier Supercomputer
  - “The ECP ALPINE project: In situ and post hoc visualization infrastructure and analysis capabilities for exascale”

Wrap Up

Today we covered the basics in in situ visualization:

- In Line Visualization
  - Catalyst is an API for live visualization, allowing you to use existing ParaView Python scripts.
  - Ascent is a flyweight tool designed for GPUs, offering high-performance, low-overhead analysis.
- In Transit Visualization
  - ADIOS2 enables in-transit visualization by safely and asynchronously moving data to a separate process.

Takeaways

- In Line visualization is the easiest to implement, but can be problematic at scale depending on the visualization operation.
- In Transit visualization has the benefit of data reorganization but comes with synchronization and additional resource overheads.
- In situ visualization has the potential to save you time, resources and headaches, and is ready for production scale codes.

Thanks!

Contacts:

james.kress@kaust.edu.sa

help@vis.kaust.edu