



# In Situ Analysis and Visualization with Ascent and ParaView Catalyst

## [Tutorial Introduction]

SC23 Tutorial  
Monday November 13th, 2023

Cyrus Harrison, Lawrence Livermore National Laboratory (LLNL)

Jean M. Favre, Swiss National supercomputing Centre (CSCS)

Corey Wetterer-Nelson, Kitware Inc.

Nicole Marsaglia, Lawrence Livermore National Laboratory (LLNL)





# In Situ Analysis and Visualization with Ascent and ParaView Catalyst

SC23 Tutorial

Monday November 13th, 2023

Cyrus Harrison (LLNL)

Jean Favre (CSCS)

Corey Wetterer-Nelson (Kitware)

Nicole Marsaglia (LLNL)



# Welcome! Today you will learn:

- About in situ scientific visualization paradigms and use cases
- How to use ***Conduit***, a shared interface for describing in-memory mesh-based data
- How to use two in situ HPC scientific visualization tools:

***Ascent, ParaView Catalyst 2***



**CONDUIT**

<http://software.llnl.gov/conduit>



<http://ascent-dav.org>



<https://catalyst-in-situ.readthedocs.io/en/latest/>

# Tutorial Outline

---

- **Introduction** [Lecture, 30 minutes]
  - Scientific Visualization and In Situ Processing Concepts
  - Ascent and Catalyst Project Overviews
- **Hands-on Cloud Environment Setup** [Hands-on, 10 minutes]
- **Using Conduit to Describe Mesh Data** [Hands-on, 50 minutes]
- ***Break*** [30 minutes]
- **Learning Ascent** [Hands-on, 40 minutes]
- **Learning Catalyst** [Hands-on, 40 minutes]
- **Closing Remarks and Questions** [10 minutes]

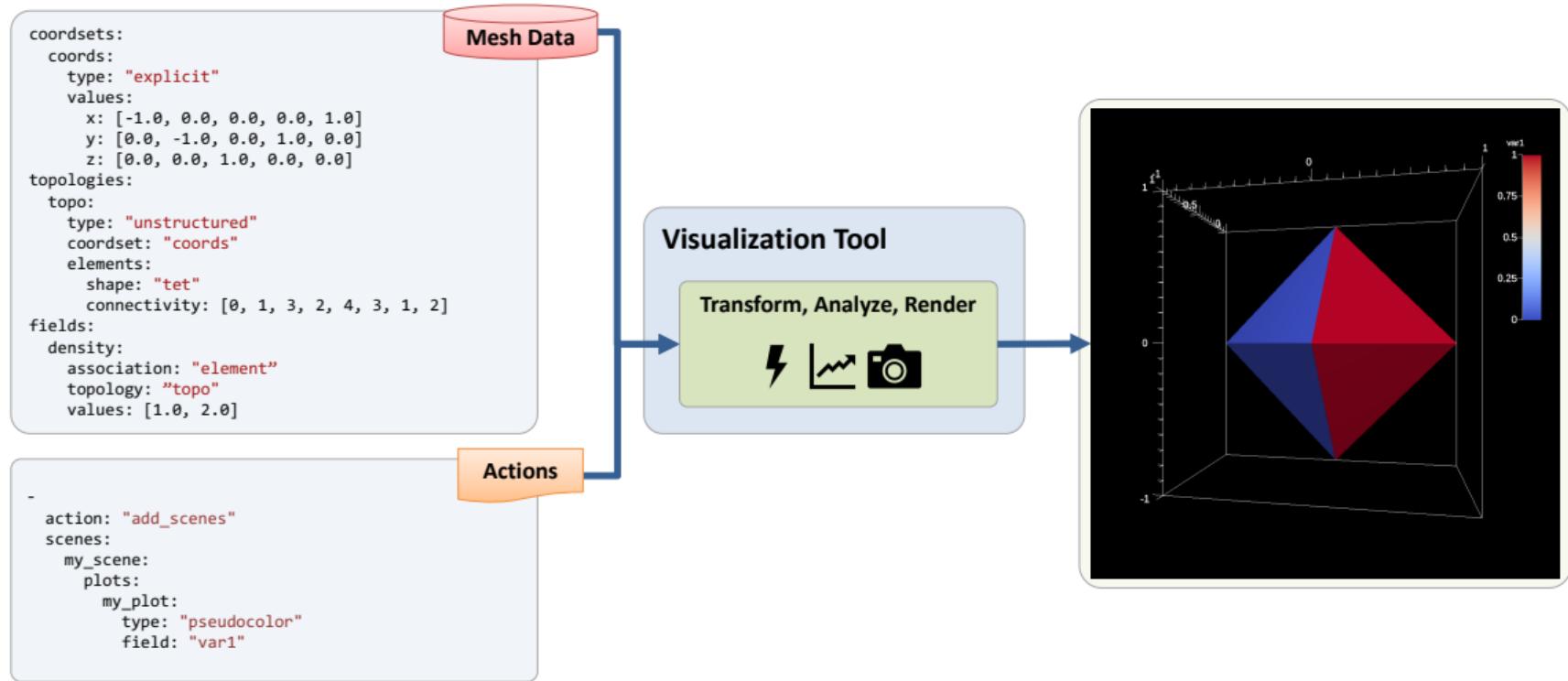


---

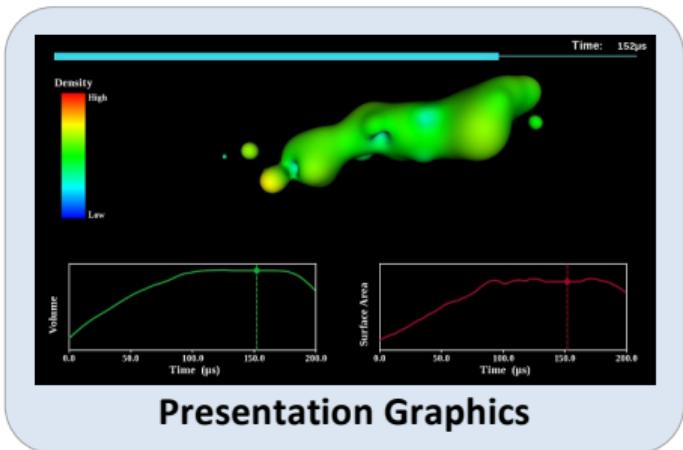
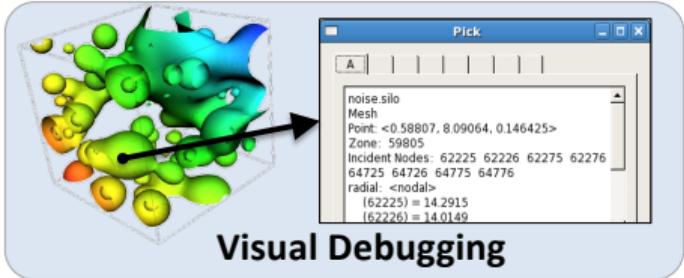
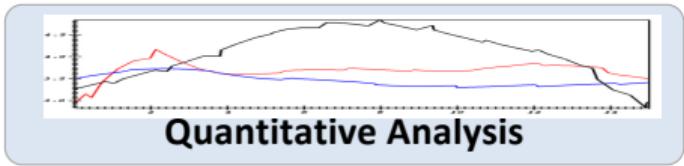
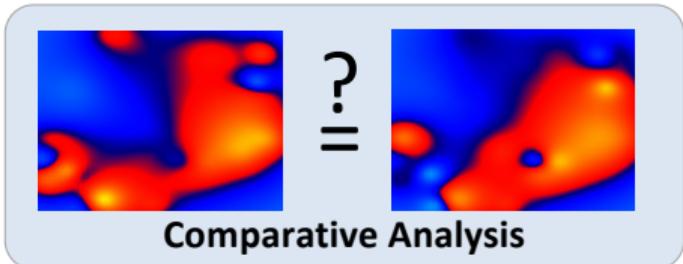
# Introduction:

## Scientific visualization and In Situ Processing Concepts

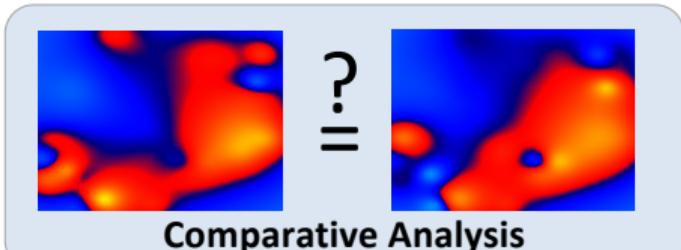
# Scientific visualization tools transform, analyze, and render mesh-based data from HPC simulations



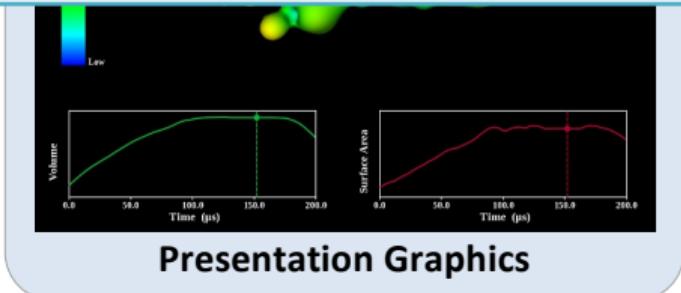
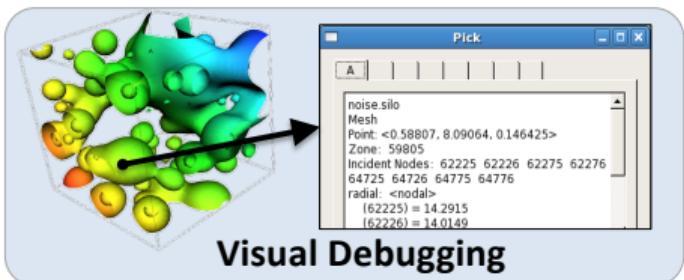
# Scientific visualization tools support a wide range of use cases



# Scientific visualization tools support a wide range of use cases



These tools are used daily by scientists to digest and understand HPC simulation results



# Scientific visualization tools are used both *post hoc* and *in situ*



## *Post Hoc*

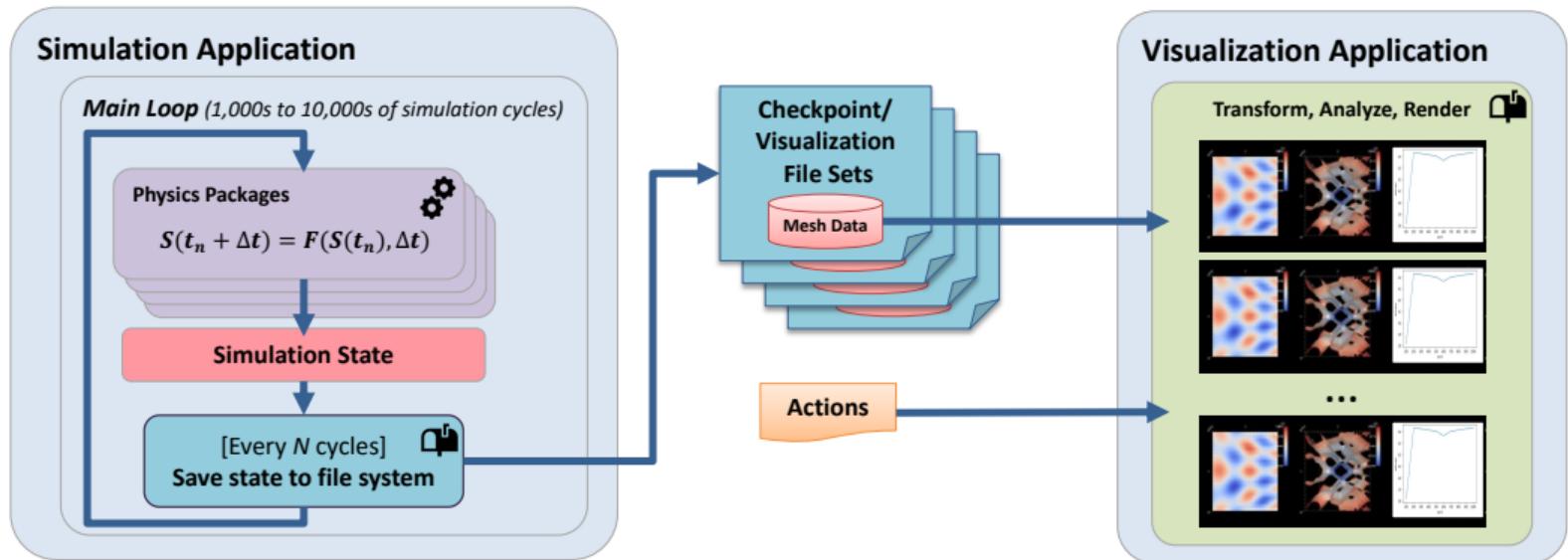
Simulation data is processed after the simulation is run using distinct compute resources.



## *In Situ*

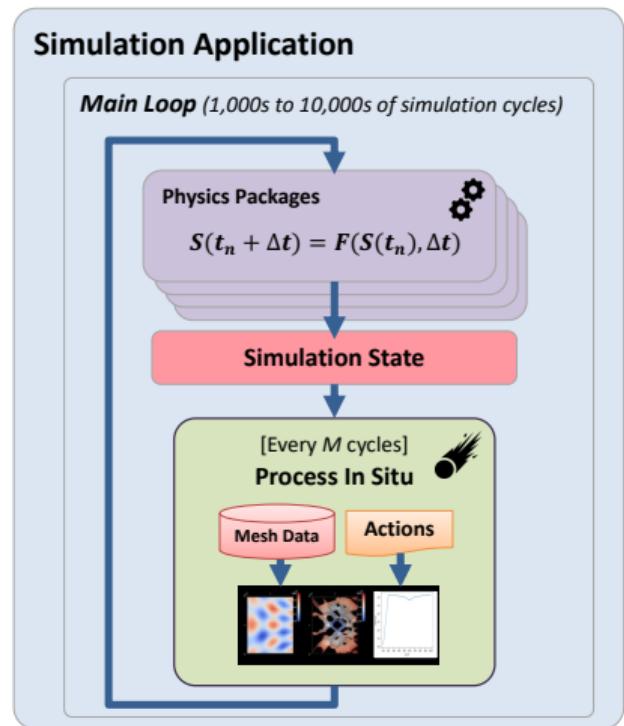
Simulation data is processed while it is generated, sharing compute resources with the simulation application.

# *Post Hoc* visualization is the most widely used paradigm to process simulation results



# *In situ* processing expands the data we can access

- In situ tools couple visualization and analysis routines with the simulation application (avoiding file system I/O)
- Pros:
  - No or greatly reduced I/O vs post hoc processing
  - Can access all simulation data
  - Computational power is readily available
  - Results are ready after simulation completes
- Cons:
  - More difficult when lacking a priori knowledge of what to visualize/analyze
  - Increasing complexity
  - Constraints (memory, network)

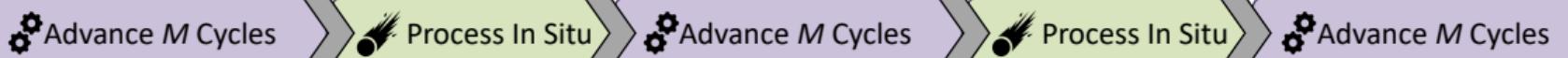


# HPC Compute vs I/O speed ratios can favor in situ processing

Simulation Run Timeline for Post Hoc Processing



Simulation Run Timeline for In Memory Processing

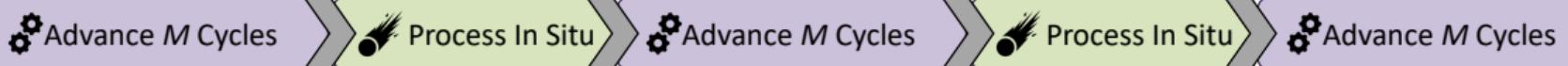


# In transit is a flavor of in situ processing that can use additional resources to improve runtime

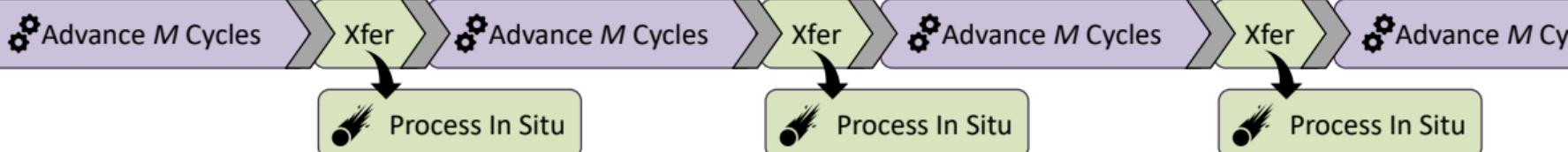
Simulation Run Timeline for Post Hoc Processing



Simulation Run Timeline for In Memory Processing

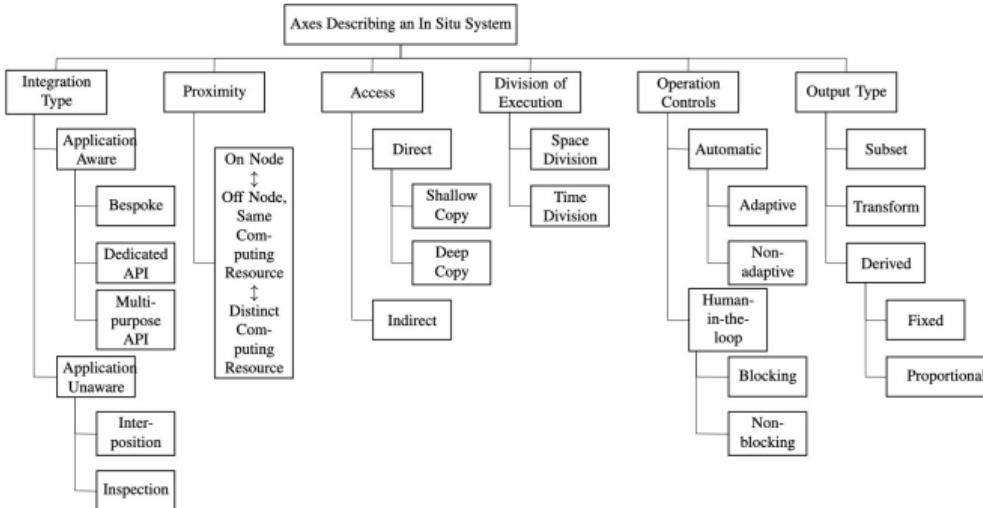


Simulation Run Timeline for In Transit Processing



# There are many considerations and flavors of in situ processing

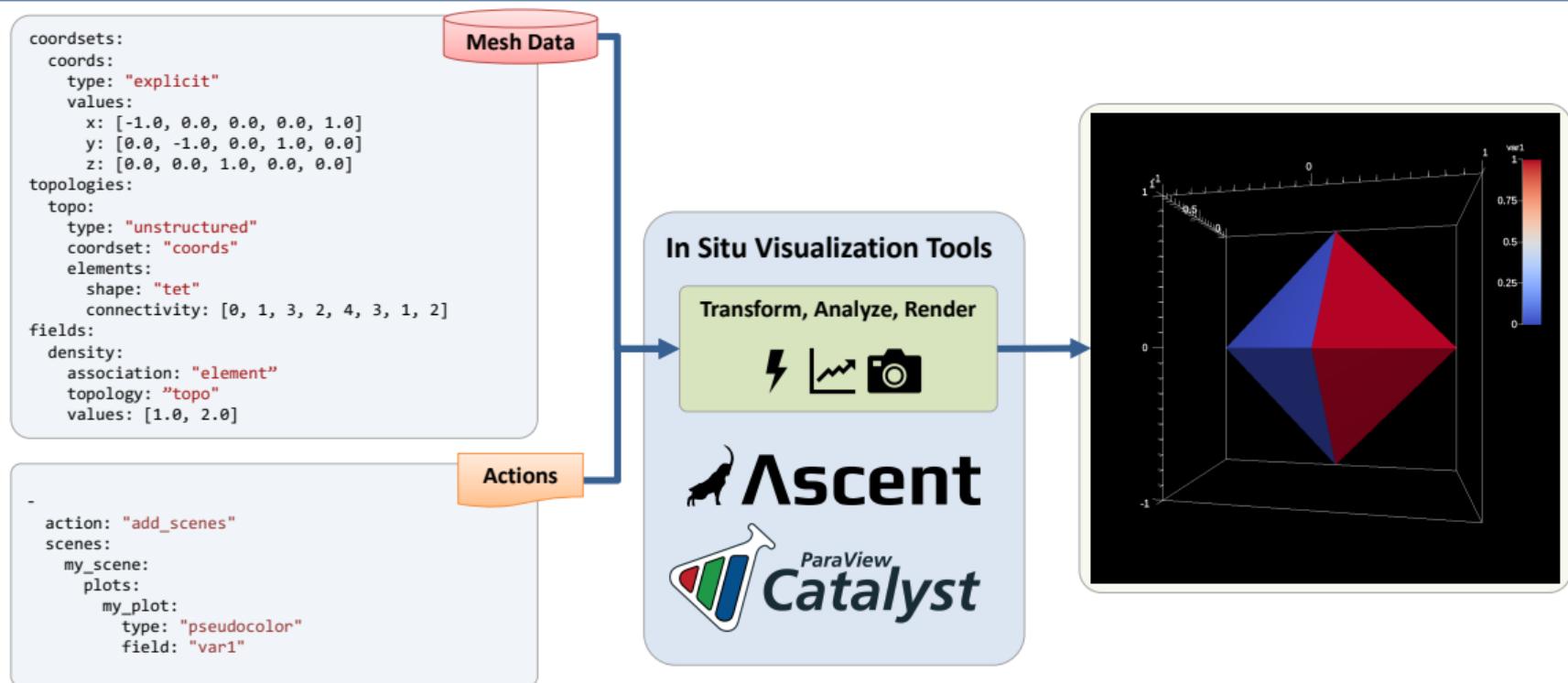
**Question:** How deep does the rabbit hole go?



**Answer:** “A Terminology for In Situ Visualization and Analysis Systems”, H. Childs, et al.

<https://cdux.cs.uoregon.edu/pubs/ChildsIJHPCA.pdf>

# We need to pass simulation mesh data and user actions to our in situ visualization tools



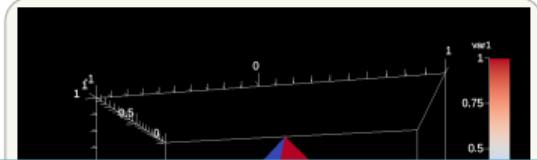
# We need to pass simulation mesh data and user actions to our in situ visualization tools

```
coordsets:  
  coords:  
    type: "explicit"  
    values:  
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]  
      y: [0.0, -1.0, 0.0, 1.0, 0.0]  
      z: [0.0, 0.0, 1.0, 0.0, 0.0]  
topologies:  
  topo:  
    type: "unstructured"  
    coordset: "coords"
```



Mesh Data

In Situ Visualization Tools

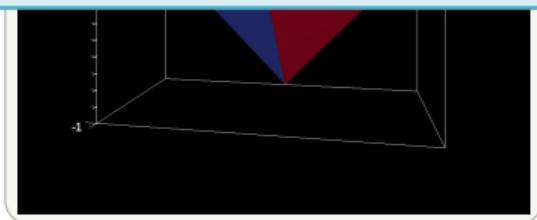


**Question 1:** How do we pass simulation meshes to these tools?

```
topology: topo  
values: [1.0, 2.0]
```

Actions

```
action: "add_scenes"  
scenes:  
  my_scene:  
    plots:  
      my_plot:  
        type: "pseudocolor"  
        field: "var1"
```



## HPC simulation applications implement and leverage a wide range of mesh data structures and APIs

- A variety of simulation codes leverage their own bespoke in-memory mesh data models.
- Other tools leverage a range of mesh-focused toolkits, frameworks, and APIs including: VTK, VTK-m, MFEM, SAMRAI, AMReX, (and many more ...)
- A wide set of powerful analysis tools are mesh agnostic (NumPy, PyTorch, etc) and recasting mesh data into these tools is a challenge
- *A single full-fledged API will never cover all use cases across the ecosystem*

## HPC simulation applications implement and leverage a wide range of mesh data structures and APIs

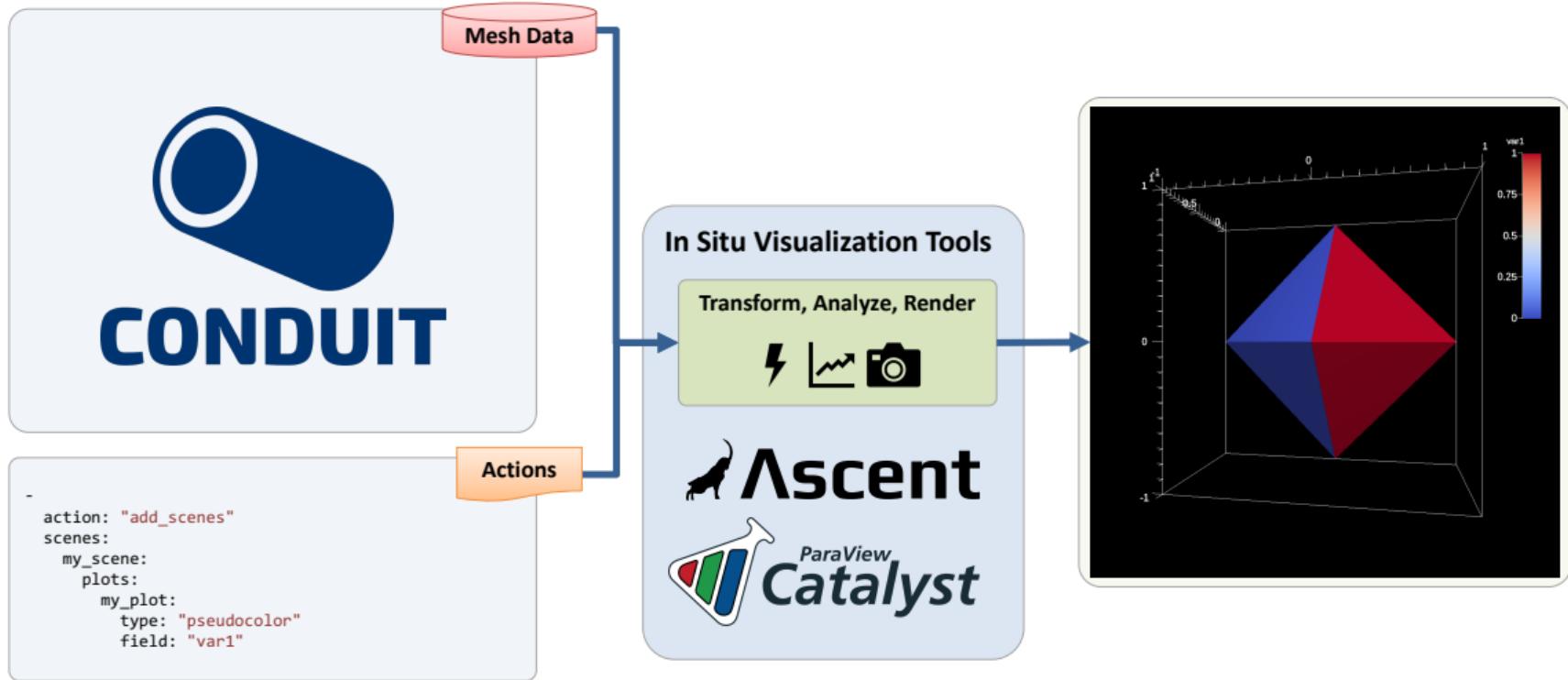
- A variety of simulation codes leverage their own bespoke in-memory mesh data models.
- Other tools leverage a range of mesh-focused toolkits, frameworks, and APIs

***Conduit Mesh Blueprint provides a strategy to describe and adapt mesh data between a wide range of APIs***

and recasting mesh data into these tools is a challenge

- *A single full-fledged API will never cover all use cases across the ecosystem*

# Both Ascent and Catalyst use *Conduit* as a shared interface to describe and accept simulation mesh data



# Conduit provides intuitive APIs for in-memory data description and exchange

- **Provides an intuitive API for in-memory data description**

- Enables *human-friendly* hierarchical data organization
  - Can describe in-memory arrays without copying
  - Provides C++, C, Python, and Fortran APIs

- **Provides common conventions for exchanging complex data**

- Shared conventions for passing complex data (e.g. *Simulation Meshes*) enable modular interfaces across software libraries and simulation applications

- **Provides easy to use I/O interfaces for moving and storing data**

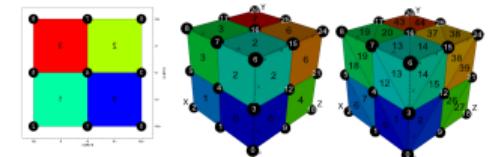
- Enables use cases like binary checkpoint restart
  - Supports moving complex data with MPI (serialization)



CONDUIT



Hierarchical in-memory data description



Conventions for sharing in-memory mesh data

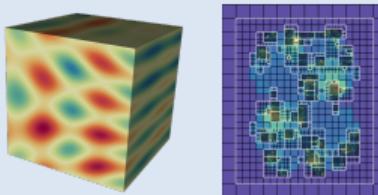
<http://software.llnl.gov/conduit>  
<http://github.com/llnl/conduit>

Website and GitHub Repo

# The Conduit Blueprint library provides tools to share common flavors of data with Conduit

## Blueprint

Supports shared higher-level conventions for using Conduit to represent data



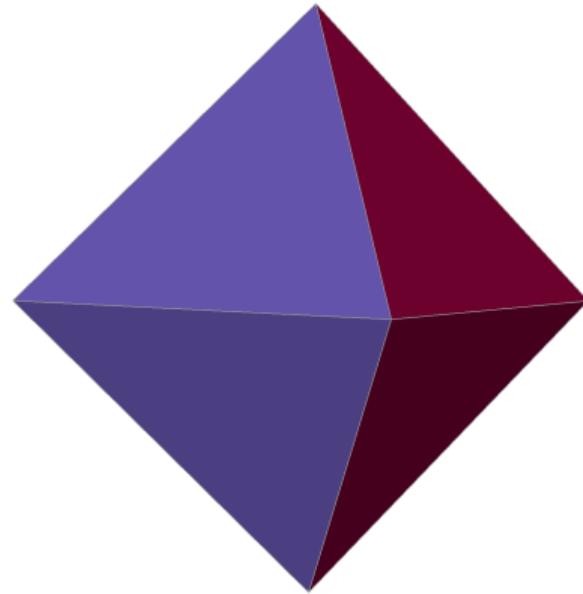
- Computational Meshes
- Multi-component Arrays
- One-to-many Relations
- Example Meshes
- Mesh Transforms



# We will share several examples of Conduit “Blueprint” meshes in this tutorial

```
coordsets:  
  coords:  
    type: "explicit"  
    values:  
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]  
      y: [0.0, -1.0, 0.0, 1.0, 0.0]  
      z: [0.0, 0.0, 1.0, 0.0, 0.0]  
topologies:  
  topo:  
    type: "unstructured"  
    coordset: "coords"  
    elements:  
      shape: "tet"  
      connectivity: [0, 1, 3, 2, 4, 3, 1, 2]  
fields:  
  density:  
    association: "element"  
    topology: "topo"  
    values: [1.0, 2.0]
```

Example YAML Output



An unstructured tet mesh

# We need to pass simulation mesh data and user actions to our in situ visualization tools

```
coordsets:  
  coords:  
    type: "explicit"  
    ...
```



**Question 1:** How do we pass simulation meshes to these tools?

```
coordset: "coords"  
elements:  
  shape: "tet"
```

IN SITU VISUALIZATION TOOLS

**Answer:** Ascent and Catalyst both use



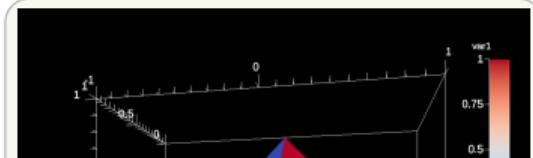
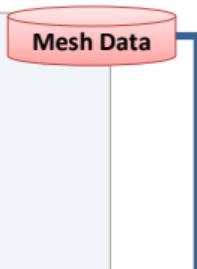
```
action: "add_scenes"  
scenes:  
  my_scene:  
    plots:  
      my_plot:  
        type: "pseudocolor"  
        field: "var1"
```

ACTIONS



# We need to pass simulation mesh data and user actions to our in situ visualization tools

```
coordsets:  
  coords:  
    type: "explicit"  
    values:  
      x: [-1.0, 0.0, 0.0, 0.0, 1.0]  
      y: [0.0, -1.0, 0.0, 1.0, 0.0]  
      z: [0.0, 0.0, 1.0, 0.0, 0.0]  
topologies:  
  topo:  
    type: "unstructured"  
    coordset: "coords"
```

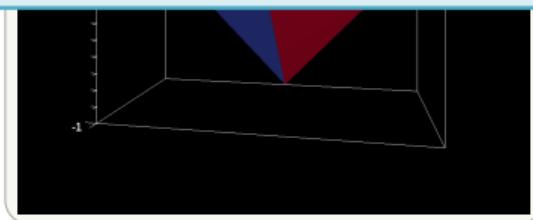


**Question 2:** How do we pass user actions to these tools?

```
topology: topo  
values: [1.0, 2.0]
```



```
action: "add_scenes"  
scenes:  
  my_scene:  
    plots:  
      my_plot:  
        type: "pseudocolor"  
        field: "var1"
```



# We need to pass simulation mesh data and user actions to our in situ visualization tools

```
coordsets:  
  coords:  
    type: "explicit"
```



**Question 2:** How do we pass user actions to these tools?

```
coordset: "coords"  
elements:  
  shape: "tet"
```

In Situ Visualization Tools

**Answer:** Ascent and Catalyst have their own APIs and multiple ways to leverage them (via C++, Fortran, Python, YAML, etc).

You will learn about both APIs in hands-on sessions.

```
my_plot:  
  type: "pseudocolor"  
  field: "var1"
```



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.  
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.





# In Situ Analysis and Visualization with Ascent and ParaView Catalyst

[Ascent Project Overview]

SC23 Tutorial  
Monday November 13th, 2023

Cyrus Harrison, Lawrence Livermore National Laboratory (LLNL)

Nicole Marsaglia, Lawrence Livermore National Laboratory (LLNL)





# Ascent: Flyweight In Situ Visualization and Analysis for HPC Simulations

SC23 Tutorial

Monday November 13th, 2023

Cyrus Harrison (LLNL)  
Nicole Marsaglia (LLNL)



# Ascent is an easy-to-use flyweight in situ visualization and analysis library for HPC simulations

- **Easy to use in-memory visualization and analysis**

- Use cases: *Making Pictures*, *Transforming Data*, and *Capturing Data*
  - Young effort, yet already supports most common visualization operations
  - Provides a simple infrastructure to integrate custom analysis
  - Provides C++, C, Python, and Fortran APIs



Visualizations created using Ascent

- **Uses a flyweight design targeted at next-generation HPC platforms**

- Efficient distributed-memory (MPI) and many-core (CUDA, HIP, OpenMP) execution
    - Demonstrated scaling: In situ filtering and ray tracing across **16,384 GPUs** on LLNL's Sierra Cluster
  - Has lower memory requirements than current tools
  - Requires less dependencies than current tools (ex: no OpenGL)
    - Builds with  Spack <https://spack.io/>



Extracts supported by Ascent

<http://ascent-dav.org>

<https://github.com/Alpine-DAV/ascent>

Website and GitHub Repo

# Acknowledgements

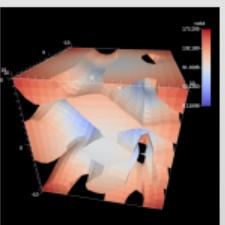
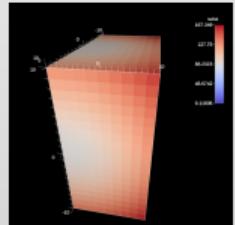
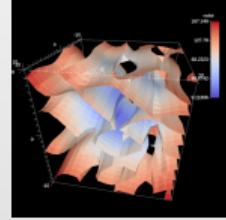
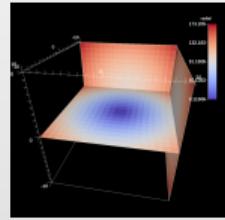
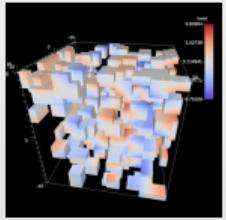
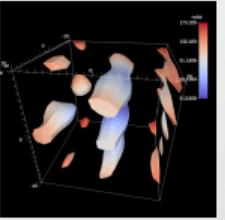
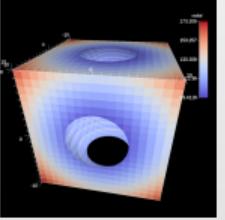
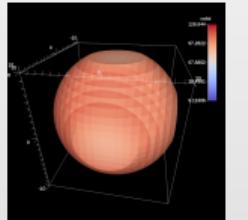


This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.

Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

# Ascent supports common visualization use cases



Clips

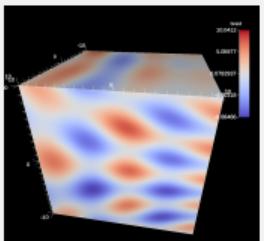
Iso-Volume

Threshold

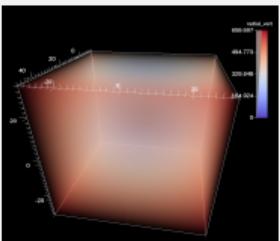
Slice

Contour

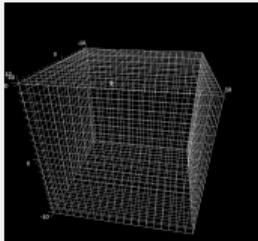
## Rendering



Pseudocolor



Volume



Mesh

[powered by]



mfem

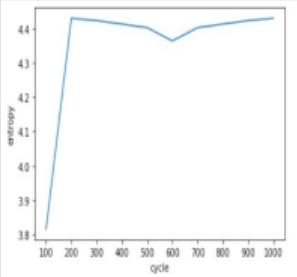


Devil Ray

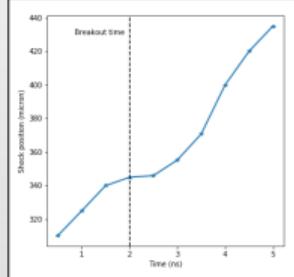
# Ascent supports common analysis use cases

```
expression: |  
    du = gradient(field('velocity', 'u'))  
    dv = gradient(field('velocity', 'v'))  
    dw = gradient(field('velocity', 'w'))  
    w_x = dw.y - dv.z  
    w_y = dw.z - dv.x  
    w_z = dw.x - dv.y  
    vector(w_x,w_y,w_z)  
name: vorticity
```

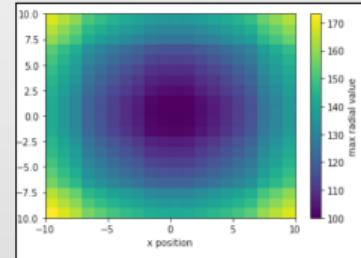
Derived Fields



Time Histories



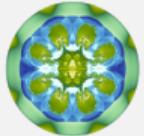
Lineouts and Spatial Binning



```
condition:  
    entropy - history(entropy,  
        relative_index = 1) > 0.5
```

Triggers

Extracts



Scalar Images



HDF5 Files



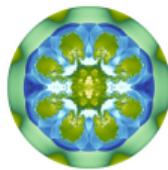
Cinema  
Databases



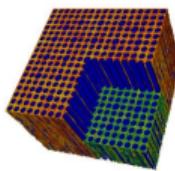
# We are working to integrate and deploy Ascent with HPC simulation codes (ECP and beyond)



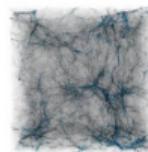
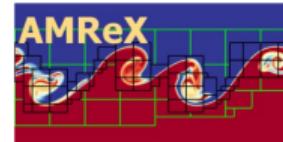
**CEED**  
EXASCALE DISCRETIZATIONS



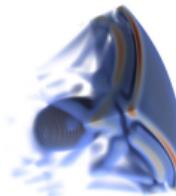
MARBL



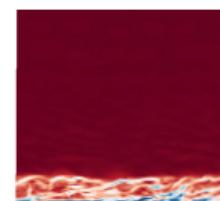
NekRS



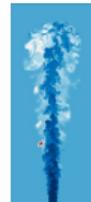
Nyx



WarpX



AMRWind

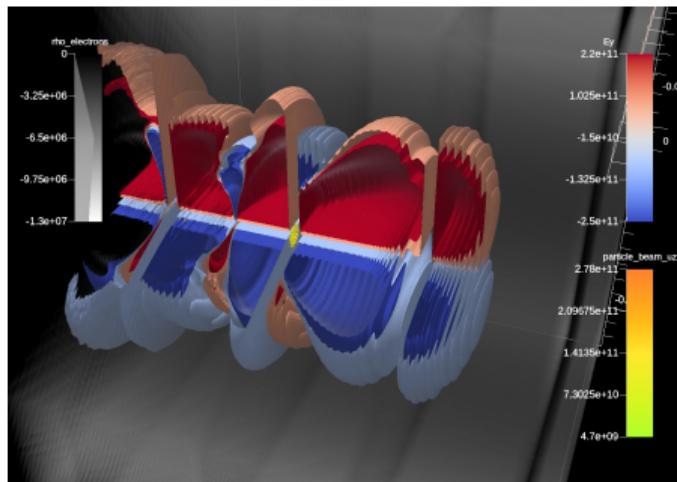


Pele

The Ascent logo features a small silhouette of a dog's head on the left, followed by the word "Ascent" in a large, bold, sans-serif font.

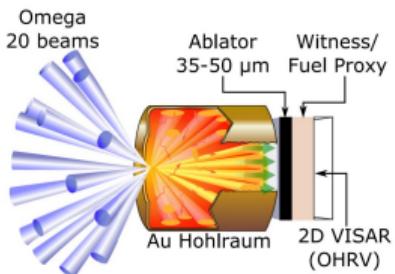
# Science Enabling Results: WarpX Simulations on ORNL Frontier (2023)

- The **578.8 million** element simulation ran across **552 GPUs** on **69 Nodes**
- The simulation application used **HIP** to run on the GPUs and passed device pointers to Ascent, providing **zero-copy** in situ processing of the time-varying data
- Ascent leveraged **RAJA** to create derived fields and **VTK-m** to run visualization algorithms on the GPUs

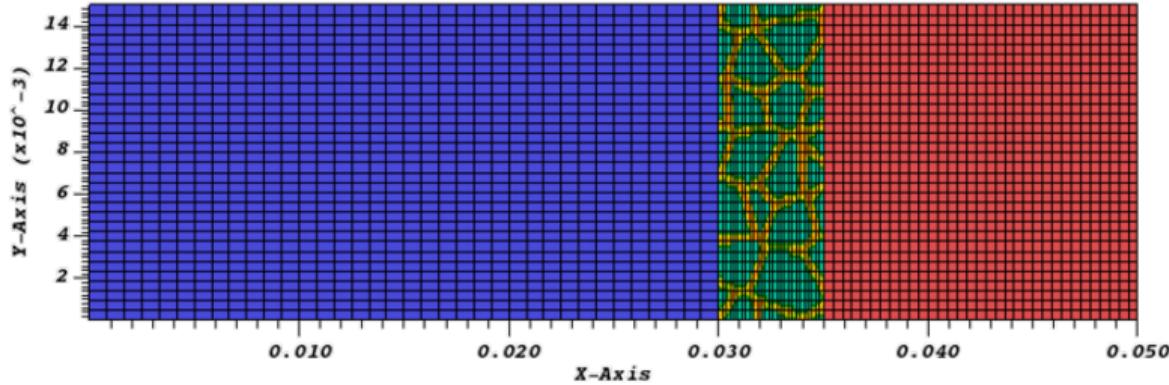


Visualization of a staged laser-wakefield accelerator simulation. Shown is the strong traversal focusing fields (red-blue) in the first plasma stage (gray) and injected into this structure is an electron beam (orange-green) that is accelerated to the right to high energies.

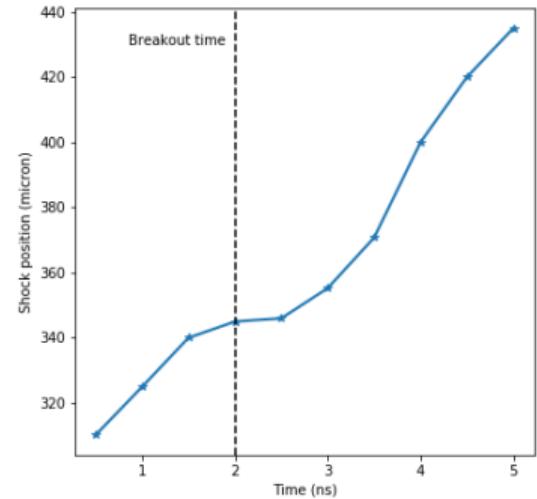
# Science Enabling Results: Shock Front Tracking (VISAR)



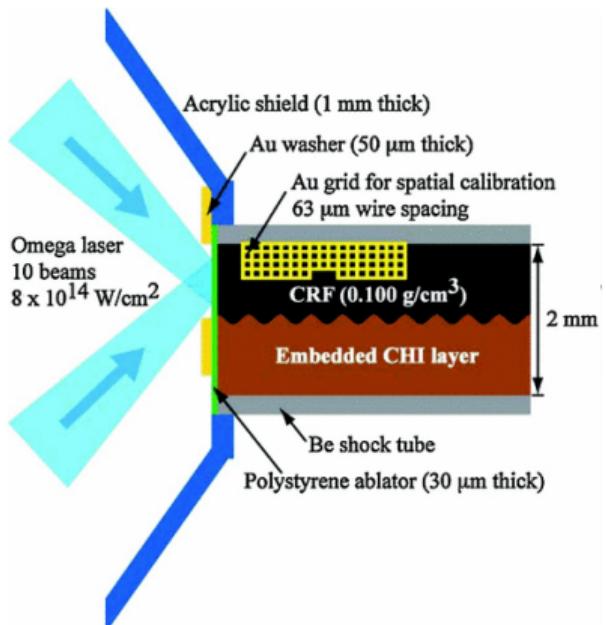
Velocity interferometer system for any reflector (VISAR)



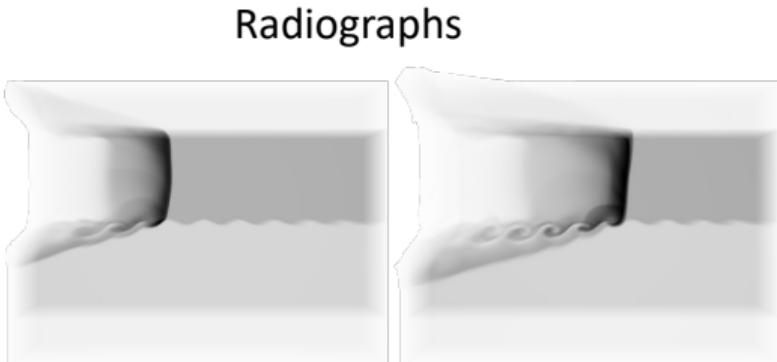
Shock position tracked  
in Ascent



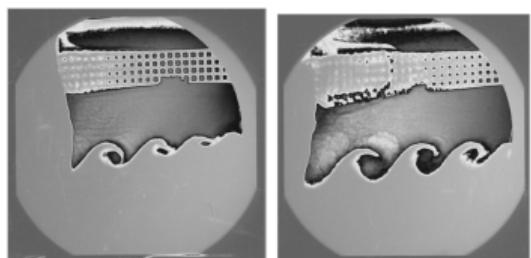
# Science Enabling Results: Simulation Validation



Simulated



Experimental



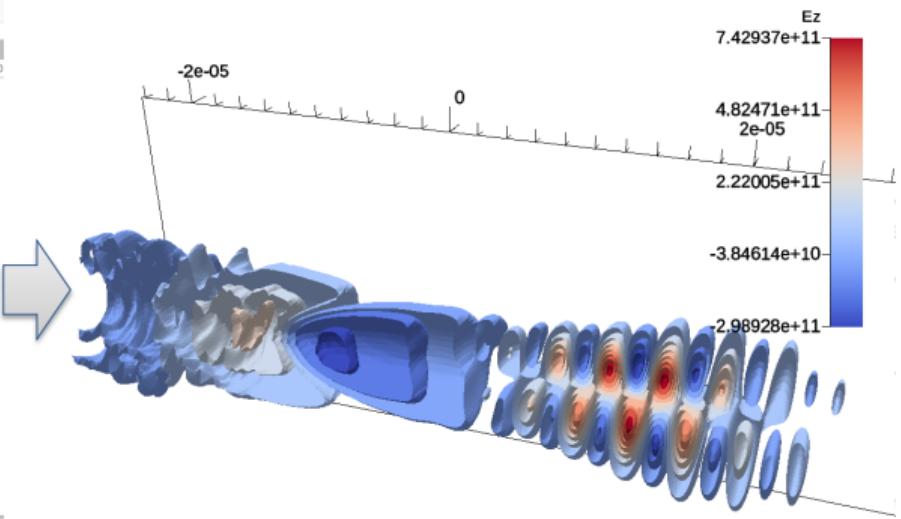
# Science Enabling Results: WarpX Workflow Tools (Jupyter Lab)

The screenshot shows the Jupyter Lab interface. On the left, there is a file browser with a tree view of files and a list view below it. The list view shows files like `ascent_replay.py`, `ascent_replay.yaml`, and `replay_actions.yaml`. On the right, there is a code editor window titled "ascent\_replay.warpx.pyw" containing Python code. A large blue arrow points from the Jupyter Lab interface towards the resulting visualization.

```
# this block are data pipelines
# each entry in pipelines: executes a series of functions from top to bottom,
# results of prior functions can be used in later calls of the same pipeline
#
# action: "add_pipelines"
pipelines:
  slice_field:
    #1:
      type: "slice"
      params:
        topology: tspo
        point: (x: 0.0, y: 0.0, z: 0.0)
        normal: (x: 1.0, y: 1.0, z: 0.0)

  sampled_particles:
    #1:
      type: histrosampling
      params:
        fields: particle_electrons_u2
        bins: 64
        sample_rate: 0.00
    #2:
      type: "clip"
      params:
        topology: particle_electrons # particle data
        Multi_plane:
          point1: (x: 0.0, y: 0.0, z: 0.0)
          point2: (x: 0.0, y: 0.0, z: 0.0)
```

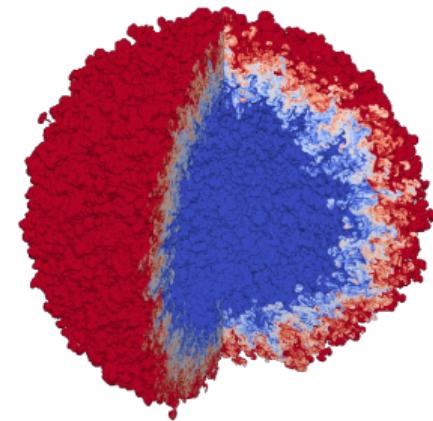
Jupyter Lab Interface



Resulting Image

# Science Enabling Results: Rendering At Scale (2018)

- The **97.8 billion** element simulation ran across **16,384 GPUs** on **4,096 Nodes**
- The simulation application used **CUDA** via **RAJA** to run on the GPUs
- Time-varying evolution of the mixing was visualized in-situ using **Ascent**, also leveraging 16,384 GPUs
- Ascent leveraged **VTK-m** to run visualization algorithms on the GPUs



Visualization of an idealized Inertial Confinement Fusion (ICF) simulation of Rayleigh-Taylor instability with two fluids mixing in a spherical geometry.

# Ascent Project Resources and Contacts

---

## Ascent Resources:

- Github: <https://github.com/alpine-dav/ascent>
- Docs: <http://ascent-dav.org/>
- Tutorial Landing Page: <https://www.ascent-dav.org/tutorial/>

## Contact Info:

Cyrus Harrison: [cyrush@llnl.gov](mailto:cyrush@llnl.gov)

Nicole Marsaglia: [marsaglia1@llnl.gov](mailto:marsaglia1@llnl.gov)

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.  
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.





# In Situ Analysis and Visualization with Ascent and ParaView Catalyst

[Catalyst Project Overview]

SC23 Tutorial  
Monday November 13th, 2023

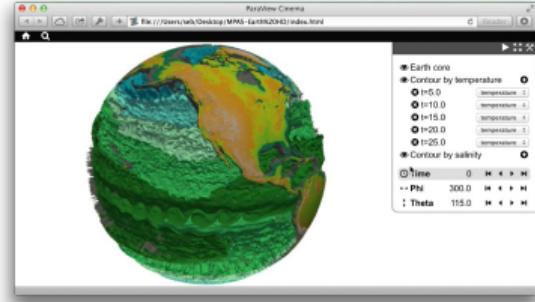
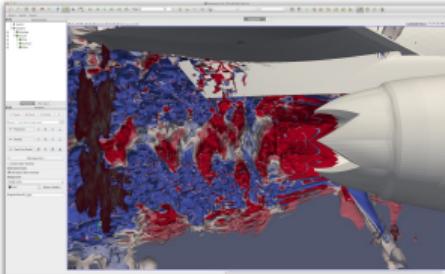
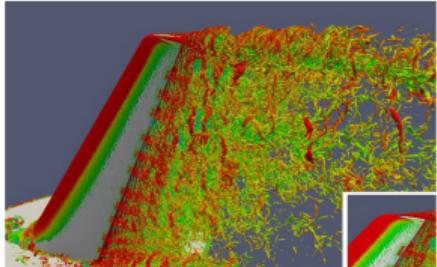
Corey Wetterer-Nelson, Kitware Inc.





# Catalyst

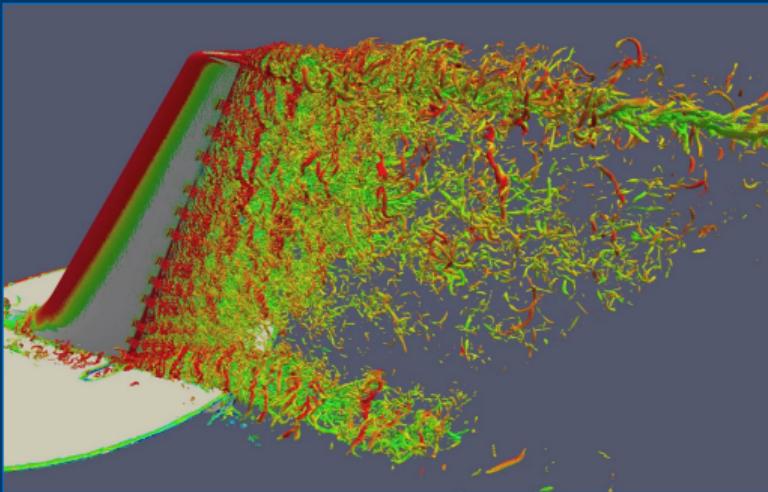
ParaView Catalyst is an **in situ library**, with an adaptable application programming interface (API), that orchestrates the delicate alliance between simulation and analysis and/or visualization tasks.



# ParaView Catalyst background

- Initial work started in 2008 from an Army SBIR
- Sep 2010 First ParaView release with CoProcessing (renamed to Catalyst in 2012)
- Sep 2014 Catalyst Live released
- Sep 2014 ParaView Cinema released
- Scaled to 1Mi MPI ranks on ALCF's Mira BG/Q
- SC16 visualization showcase winner generated animation using Catalyst
- Based on the ParaView libraries – full access to ParaView capabilities
- Batch and interactive in situ analysis and visualization
- In transit workflows done with standalone ParaView and ADIOS

>1M MPI ranks on Mira@ANL



# Simulation, Adaptor and Catalyst Interactions



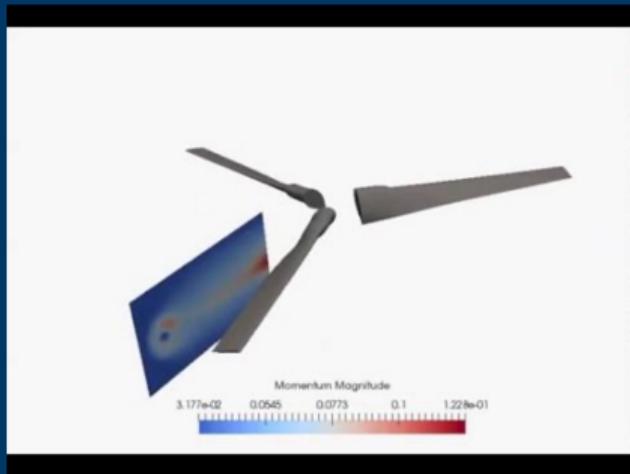
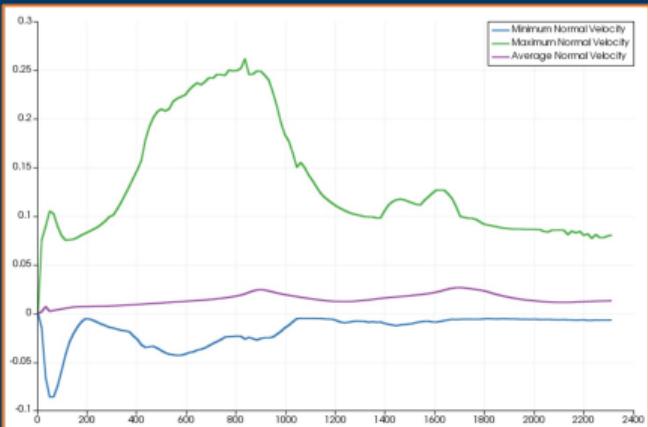
- Typically 3 calls between simulation code and adaptor
  - **catalyst\_initialize()**
    - MPI communicator (optional)
    - Add analysis scripts
  - **catalyst\_execute()**
    - Does the work
  - **catalyst\_finalize()**
- Information provided by solver to adaptor
  - Time, time step, force output
  - Grids and fields
- **Information provided by adaptor**
  - Pipelines to execute
  - Time, time step, force output
  - Grid and fields when needed
  - MPI communicator
- **Information provided by Catalyst**
  - If co-processing needs to be done
  - What grids and fields are needed
- **User data can be shared both ways**

# Performing In Situ Efficiently with Catalyst

- Small run-time overhead
  - Small initialization and finalization times
  - Scalable analysis and visualization algorithms
  - Reduced amount of IO (possibly at expense of more complex IO patterns)
- Small code footprint
  - Typically ~3 calls from simulation code to Catalyst (initialize, execute, finalize)
- Efficiency in computing and memory-management
  - On demand compute
  - Request only needed information
  - Flexible ways to represent different pieces of information
  - Zero-copy use of simulation data structures

# Configurability

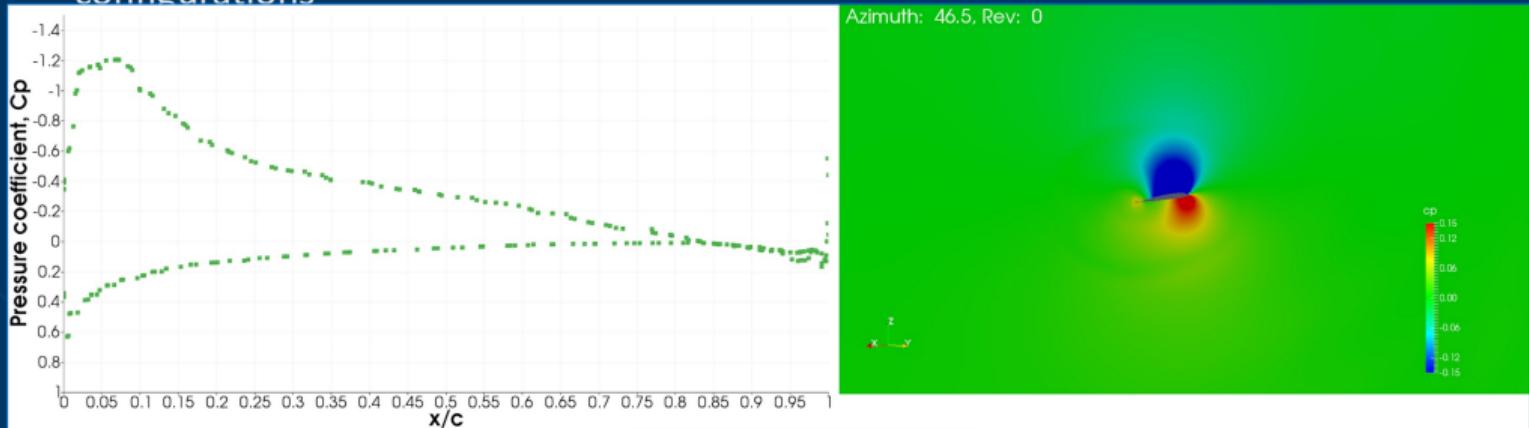
- Allow simulation specific output customization
- Edit Python scripts for Catalyst
  - Can change in situ output without any recompilation



Results courtesy Rajneesh Singh & Ben Jimenez (ARL)

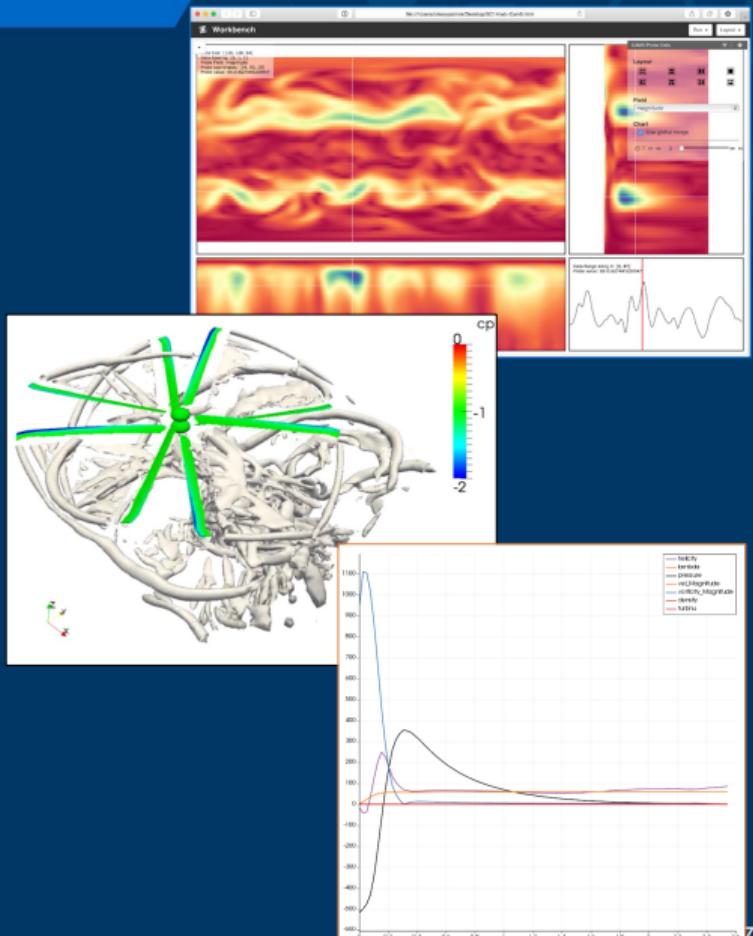
# Combined Viz & Analysis

- Constructing relatively complex in situ pipelines requires both domain and viz experts
  - Increased interactions
  - Viz experts can help guide work early on
- Conceptually complex in situ output can be done through parameterized input configurations

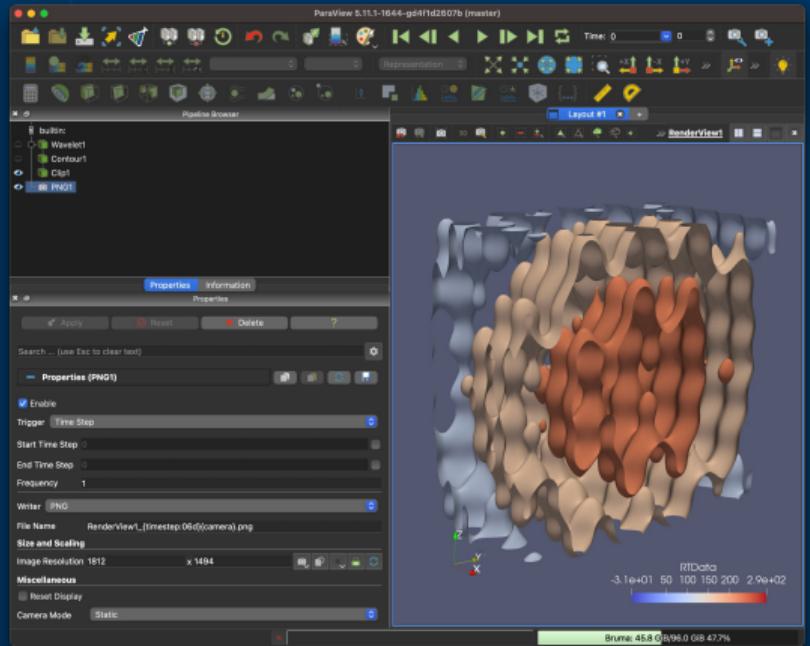


## Multiple Output Types

- Images
    - Size is independent of grid size
    - Explorable through Cinema output ([cinemascience.org](http://cinemascience.org))
    - Charts
  - Data extracts
    - Large collection of file formats supported including VTK, ADIOS, HDF5 (in progress)
  - ASCII output
    - Single quantities (e.g. field values at a point)
    - Statistics
    - Custom (e.g. value and location of highest temperature value)



# Automated Pipeline Generation in ParaView



```
Downloads -- vi powerful-catalyst-script.py -- 117x69

# script-version: 2.0
# Catalyst state generated using paraview version 5.11.1-1644-gd4ff1d2807b
import paraview
import paraview.servermanager
paraview.servermanager.PARAVIEW_COMPATIBILITY_MAJOR = 5
paraview.servermanager.PARAVIEW_COMPATIBILITY_MINOR = 11

### Import the single module from the paraview
from paraview.simple import *
### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# -----
# setup views used in the visualization
# ----

# Create a new 'Render View'
renderView = CreateView('RenderView')
renderView.ViewSize = [1220, 140]
renderView.CameraType = 'Perspective 3D Actor'
renderView.CenterOfRotation = [59.0, 59.0, 59.0]
renderView.StereoType = 'Crystal Eyes'
renderView.CameraPosition = [246.5970004649193, -122.31188380372246, 141.2557230493831]
renderView.CameraFocalPoint = [105.055437998544, 37.88612567534465, 51.4857330851351]
renderView.CameraviewUp = [-1.279855262148546, 0.17643366851954914, 0.9447927790862187]
renderView.CameraFocalDisk = 1.0
renderView.CameraParallelScale = 46.0224237844385
renderView.Legendary = 1
renderView.LegendGridActor = 'Legend Grid Actor'

SetActiveView(None)

# -----
# setup view layouts
# ----

# create new layout object 'Layout #1'
layout1 = CreateLayout(name='Layout #1')
layout1.AssignView(0, renderView)
layout1.GetSize([1220, 140])

# restore active view
SetActiveView(renderView)
# -----
# setup the data processing pipelines
# ----

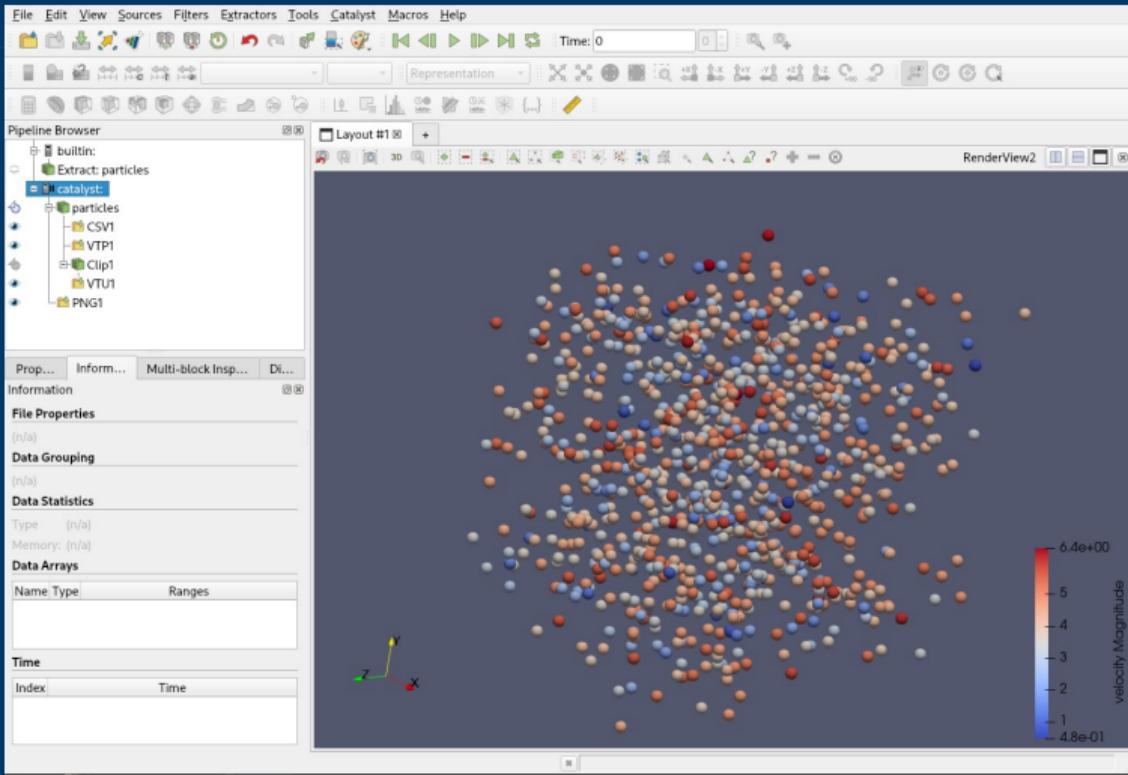
# create a new 'Wavelet'
wavelet1 = Wavelet(registrationName='Wavelet')
wavelet1.WholeExtent = [0, 100, 0, 100, 0, 100]
wavelet1.Center = [50.0, 50.0, 50.0]

# create a new 'Contour'
contour1 = Contour(registrationName='Contour', Input=wavelet1)
contour1.UpdatePipeline()
contour1.Isosurfaces = [15.95202699914508, 32.921148906816408, 46.95467948913074, 129.95826999145588, 222.9617484937, 744.285, 965.952099914508]
contour1.PointMergeMethod = 'Uniform Clipping'

# create a new 'Clip'
clip1 = Clip(registrationName='Clip1', Input=contour1)
clip1.ClipType = 'Plane'
clip1.ClipType.TransformType = 'Plane'
clip1.ClipType.TransformType = 'Plane'
clip1.Scalars = ['RTData', 'RTData']
clip1.Value = 191.45997819628989

# init the 'Plane' selected for 'ClipType'
```

# ParaView Enables Live Visualization



# Catalyst Runtime Selectable Backends

- **catalyst-stub**: no external dependencies, suitable for building
- **catalyst-paraview**: full in situ access to ParaView and VTK
- **catalyst-adios**: in transit workflows for asynchronous interactive analysis and visualization
- **catalyst-ascent** (you read that right!): Catalyst passthrough to Ascent platform
- Debug, replay, etc.
- your implementation!



# In Situ Analysis and Visualization with Ascent and ParaView Catalyst

## [Conduit Mesh Representation Introduction and Hands-on]

SC23 Tutorial  
Monday November 13th, 2023

Cyrus Harrison, Lawrence Livermore National Laboratory (LLNL)

Jean M. Favre, Swiss National supercomputing Centre (CSCS)



# A heat-diffusion mini-app to demonstrate in-situ visualization with Ascent and Catalyst

Jean M. Favre

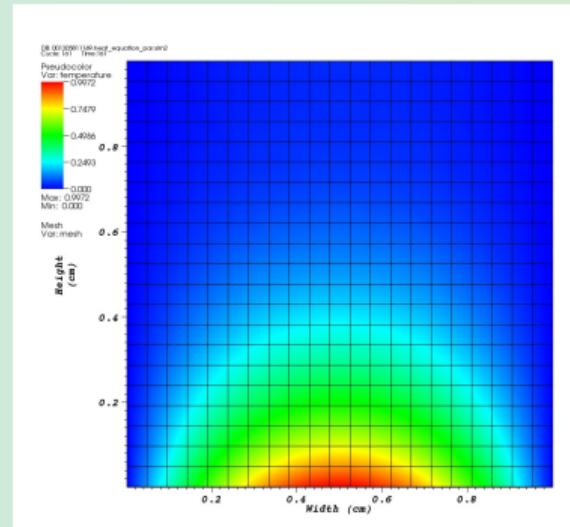
Swiss National supercomputing Centre (CSCS)

# Motivations

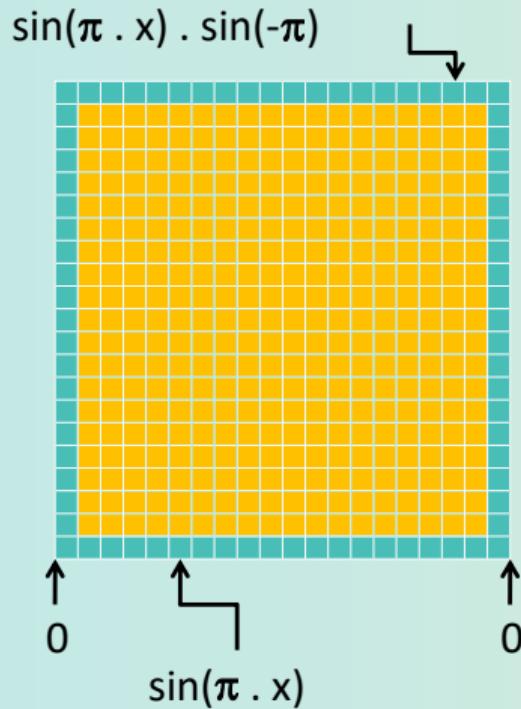
- A simple demonstrator for in-situ couplings based on Conduit + {Ascent, Catalyst}
  - in C++ and Python
  - with MPI (optional)
  - with four different grid types (uniform, rectilinear, structured and unstructured)

# Assumptions

- This 2D heat-diffusion solver relies on a very simple 5-point stencil to update the nodes of a mesh.
- No focus on computational performance
- A very simple partitioning grid for parallel runs
- The underlying grid points form a uniform, Cartesian grid, defined solely with an **origin**, a fixed **spacing** between points (in X and Y), and a grid **resolution**
- For didactic purposes only, the same grid can also be viewed as:
  - a rectilinear grid (special case with equally spaced grid points along both axis)
  - A structured grid (special case with the points' coordinates explicitly given)
  - An unstructured grid of quadrilateral cells (with an explicit connectivity list)
- The physical global domain is [0.,0.]->[1.,1.]



# The domain fixed boundary conditions



Update grid with solver



Fixed boundary conditions

Laplace equation:  $\Delta u = 0$



# Generic run-time options

```
usage: heat_diffusion_initu_*.py [-h] [-t TIMESTEPS] [--res RES] [-m {uniform,rectilinear,structured,unstructured} [-n] [-v]
```

**-h, --help** show this help message and exit

**-t TIMESTEPS, --timesteps TIMESTEPS**

number of timesteps to run the miniapp (default: 1000)

**--res RES** resolution in each coordinate direction (default: 64)

**-m {uniform,rectilinear,structured,unstructured}, --mesh {uniform,rectilinear,structured,unstructured}**

mesh type (default: uniform)

**-n, --noinsitu** toggle the use of the in-situ vis coupling

**-v, --verbose** toggle printing of the conduit nodes

# API-specific run-time options

## With Ascent

**-f FREQUENCY, --frequency FREQUENCY**

How often should the Ascent script be executed

**-d DIR, --dir DIR** path to a directory where to dump the Blueprint output (final step)

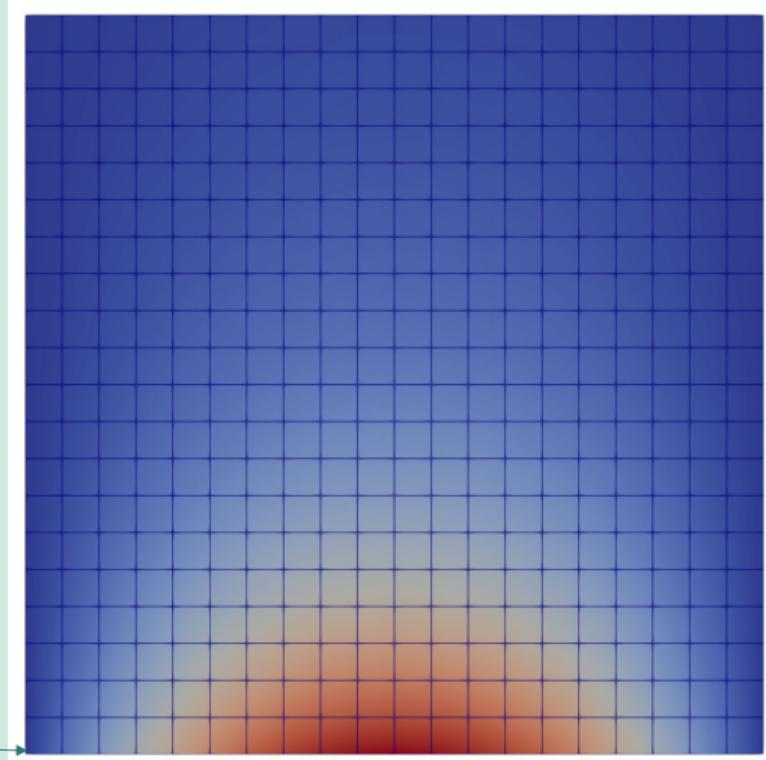
## With Catalyst

**-s SCRIPT, --script SCRIPT**

path to the Catalyst script to use for in situ processing.

# The default case: a uniform grid

```
coordsets:  
coords:  
  type: "uniform"  
dims:  
  i: 21  
  j: 21  
origin:  
  x: 0.0  
  y: 0.0  
spacing:  
  dx: 0.05  
  dy: 0.05  
topologies:  
mesh:  
  type: "uniform"  
  coordset: "coords"  
fields:  
temperature:  
  association: "vertex"  
  topology: "mesh"  
Values: [0.0, 0.156, 0.309, ..., 0.006, 5.29e-18]
```



# The rectilinear grid

coordsets:

coords:

  type: "rectilinear"

  values:

    x: [0.0, 0.05, 0.1, ..., 0.95, 1.0]

    y: [0.0, 0.05, 0.1, ..., 0.95, 1.0]

topologies:

mesh:

  type: "rectilinear "

  coordset: "coords"

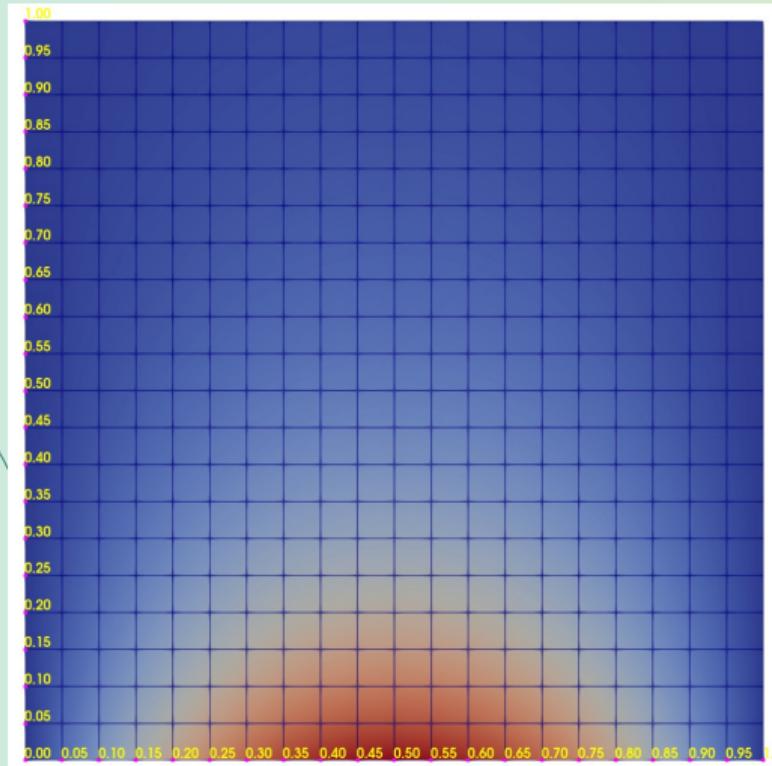
fields:

temperature:

  association: "vertex"

  topology: "mesh"

  values: [0.0, 0.156, 0.309, ..., 0.006,  
5.29e-18]



# The structured grid

coordsets:

coords:

type: "explicit"

values:

x: [0.0, 0.05, 0.1, ..., 0.95, 1.0]

y: [0.0, 0.0, 0.0, ..., 1.0, 1.0]

topologies:

mesh:

type: "structured"

coordset: "coords"

elements:

dims:

i: 20

j: 20 }

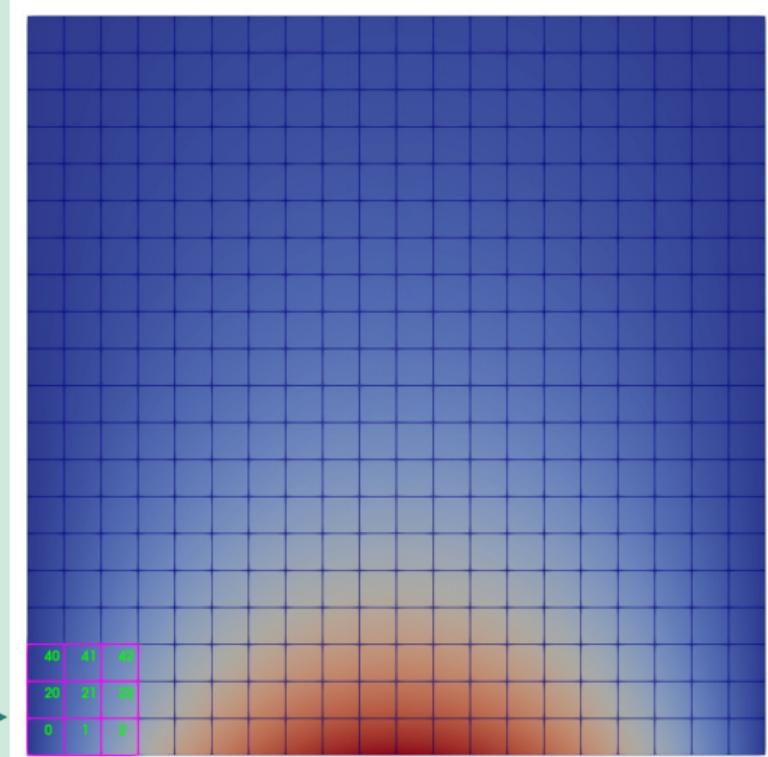
fields:

temperature:

association: "vertex"

topology: "mesh"

values: [0.0, 0.149, 0.294, ..., 0.006, 5.29e-18]



# The unstructured grid

coordsets:

coords:

type: "explicit"

values:

x: [0.0, 0.05, 0.1, ..., 0.95, 1.0]

y: [0.0, 0.0, 0.0, ..., 1.0, 1.0]

topologies:

mesh:

type: "unstructured"

coordset: "coords"

elements:

shape: "quad"

connectivity: [0, 21, 22, 1,..., 440, 419]

fields:

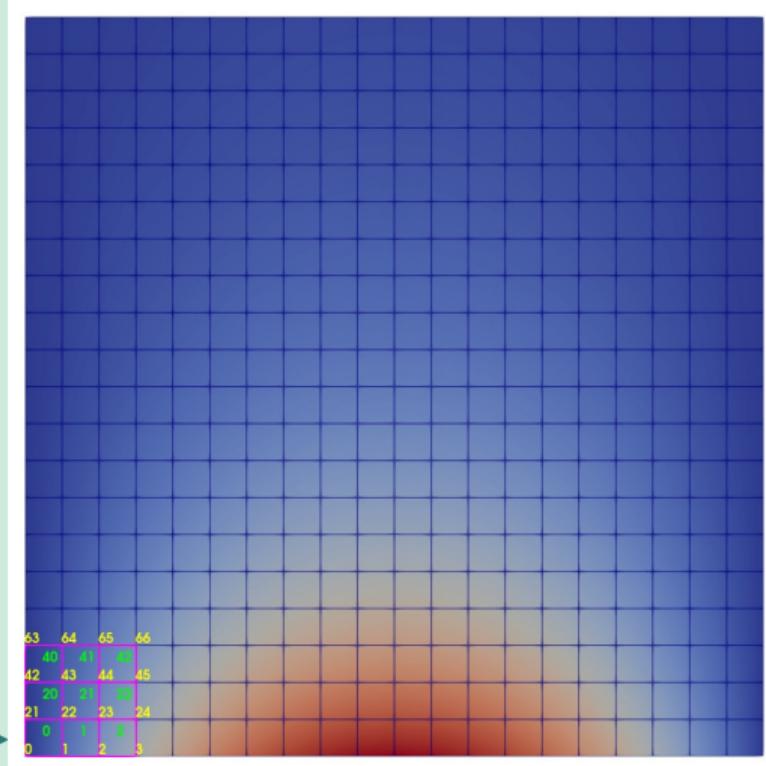
temperature:

association: "vertex"

topology: "mesh"

values: [0.0, 0.156, 0.309, ..., 0.006,

5.29e-18]



# Actions need to be defined

## Ascent

```
actions = conduit.Node()
add_act = actions.append()
add_act["action"] = "add_scenes"

# declare a scene (s1) to render the dataset
scenes = add_act["scenes"]
scenes["s1/plots/p1/type"] = "pseudocolor"
scenes["s1/plots/p1/field"] = "temperature"
# add a second plot to draw the grid lines
scenes["s1/plots/p2/type"] = "mesh"
```

## ParaView Catalyst (can be written by the ParaView client)

```
renderView1 = GetRenderView()
reader = TrivialProducer(registrationName='grid')

rep = Show(reader, renderView1)
rep.Representation = 'Outline'
ColorBy(rep, ['POINTS', 'temperature'])
temperatureLUT = GetColorTransferFunction('temperature')
temperatureLUT.RescaleTransferFunction(0.0, 1.0)

contour1 = Contour(Input=reader)
contour1.ContourBy = ['POINTS', 'temperature']
contour1.ComputeScalars = 1
contour1.Isosurfaces = [i*0.1 for i in range(11)]

png1 = CreateExtractor('PNG', renderView1)
png1.Trigger = 'TimeStep'
png1.Trigger.Frequency = 100
png1.Writer.FileName = 'view-{timestep:06d}{camera}.png'
```

# Going parallel

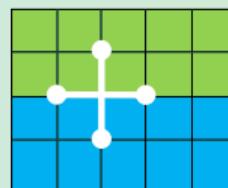
MPI-BASED EXECUTION

# The Python example uses a 1D grid partitioning

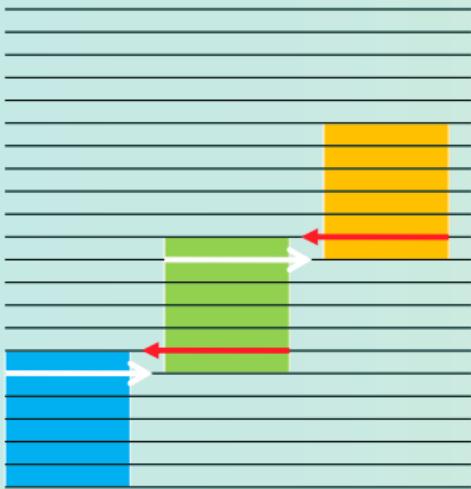
- The grid is partitioned along the Y direction



- Boundary conditions are set
- A single line of ghost-nodes insure that the 5-point stencil is continuous across MPI task boundaries



# Ghost data exchange for N MPI tasks



Overlap Send and Receive

Proc. 0 does not receive data from “below”

Proc. (N-1) does not send data “above”



# Conduit nodes: What's new in parallel?

coordsets:

coords:

type: "uniform"

dims:

i: 21

j: ... => new for each MPI task

origin:

x: 0.0

y: ... => new for each MPI task

coordsets:

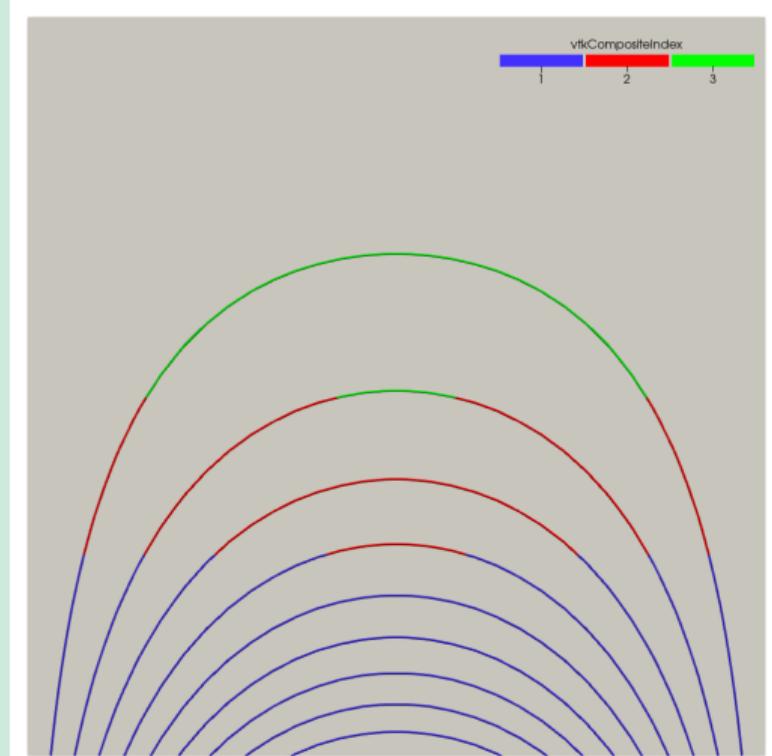
coords:

type: "rectilinear"

values:

x: [0.0, 0.05, 0.1, ..., 0.95, 1.0]

y: ... => new for each MPI task



Iso-contour lines, colored by their process-id



# Conduit nodes:

## What's new in parallel?

```
coordsets:  
coords:  
  type: "explicit"  
  values:  
    x: [0.0, 0.05, 0.1, ..., 0.95, 1.0]  
    y: ... => new for each MPI task  
topologies:  
mesh:  
  type: "structured"  
  coordset: "coords"  
elements:  
dims:  
i: 20  
j: ... => new for each MPI task
```

```
coordsets:  
coords:  
  type: "explicit"  
  values:  
    x: [0.0, 0.05, 0.1, ..., 0.95, 1.0]  
    y: ... => new for each MPI task topologies  
mesh:  
  type: "unstructured"  
  coordset: "coords"  
elements:  
  shape: "quad"  
  connectivity: ... => new for each MPI task
```

N.B. the connectivity list can use local point ids  
(starting at 0)

# The Blueprint output

- It is common for Blueprint data files to represent meshes that have been partitioned and must later be treated as a whole.
- Blueprint **root files** contain an index that facilitates reading in many individual Blueprint files => metadata about the overall contents of individual files
- The **root file** is a hierarchical index dataset created with Conduit that has been saved to a file using Relay
- [https://llnl-conduit.readthedocs.io/en/latest/blueprint\\_mesh.html#mesh-index-protocol](https://llnl-conduit.readthedocs.io/en/latest/blueprint_mesh.html#mesh-index-protocol)
  
- Can be read by VisIt out-of-the-box
- Currently developing a reader for ParaView (WIP)

# The Blueprint output

```
>>> import conduit  
  
>>> import conduit.relay.io  
  
>>> import numpy as np  
  
>>> mesh = conduit.Node()  
  
>>> conduit.relay.io.load(mesh,  
"/dev/shm/mesh.cycle_001000.root", "hdf5")  
  
>>> mesh
```

```
blueprint_index:  
mesh:  
state:  
    number_of_domains: 4  
    partition_pattern:  
        "mesh.cycle_001000/domain_{domain:06d}.hdf5:/"  
    partition_map:  
        file: [0, 1, 2, 3]  
        domain: [0, 1, 2, 3]  
coordsets:  
coords:  
    type: "uniform"  
    coord_system:  
        type: "cartesian"  
    path: "mesh/coordsets/coords"  
topologies:  
mesh:  
    type: "uniform"  
    coordset: "coords"  
    path: "mesh/topologies/mesh"  
fields:  
temperature:  
    number_of_components: 1  
    topology: "mesh"  
    association: "vertex"
```

# Demonstrations / exercises

```
python3 heat_diffusion_insitu_parallel_Ascent.py --help
```

```
python3 heat_diffusion_insitu_parallel_Ascent.py -v --mesh=uniform --res=64
```

```
mpiexec -n 4 python3 heat_diffusion_insitu_parallel_Ascent.py -v --mesh=uniform --res=64
```

```
python3 heat_diffusion_insitu_parallel_Catalyst.py --help
```

```
python3 heat_diffusion_insitu_parallel_Catalyst.py -v --mesh=uniform --res=64
```

```
mpiexec -n 4 python3 heat_diffusion_insitu_parallel_Catalyst.py -v --script=pvPythonScript.py
```



# In Situ Analysis and Visualization with Ascent and ParaView Catalyst

[Ascent Hands-on]

SC23 Tutorial  
Monday November 13th, 2023

Cyrus Harrison, Lawrence Livermore National Laboratory (LLNL)

Nicole Marsaglia, Lawrence Livermore National Laboratory (LLNL)





# Ascent Hands-on Session

SC23 Tutorial

Monday November 13th, 2023

Cyrus Harrison (LLNL)  
Nicole Marsaglia (LLNL)



# Today we will teach you about Ascent's API and capabilities

---

## You will learn:

- How to use Conduit, the foundation of Ascent's API
- How to get your simulation data into Ascent
- How to tell Ascent what pictures to render and what analysis to execute

# Ascent tutorial examples are outlined in our documentation and included ready to run in Ascent installs

The screenshot shows a web browser displaying the Ascent documentation site. The header features the Ascent logo with a dog icon and the word "Ascent". Below the logo is a "latest" button and a search bar labeled "Search docs". A sidebar on the left contains links to "Quick Start", "Ascent User Documentation", "Developer Documentation", and a "Tutorial" section which is expanded to show "Tutorial Setup", "Introduction to Ascent", and "CloverLeaf3D Ascent Demos". The main content area has a breadcrumb navigation "Docs » Tutorial" and a "Edit on GitHub" link. The title "Tutorial" is displayed, followed by a paragraph about the tutorial's purpose and a bulleted list of topics:

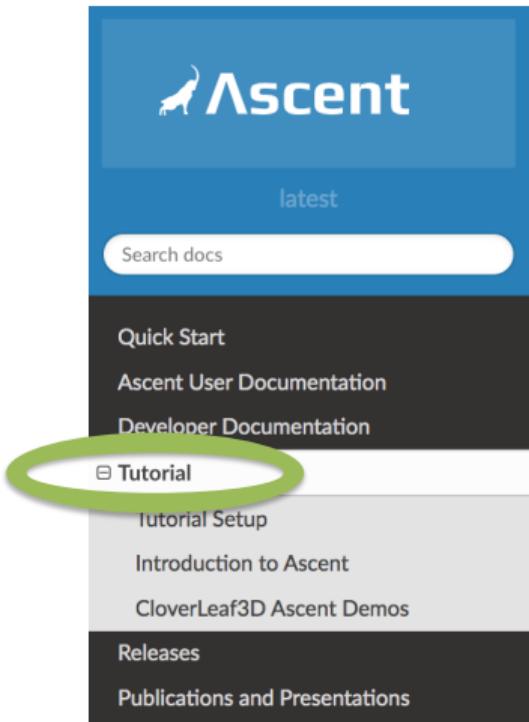
- Formating mesh data for Ascent
- Using Conduit and Ascent's Conduit-based API
- Using and combining Ascent's core building blocks: Scenes, Pipelines, Extracts, Queries, and Triggers
- Using Ascent with the Cloverleaf3D example integration

Below the list, a paragraph explains that Ascent installs include examples for this tutorial, located in the "examples/ascent/tutorial/ascent\_intro/" directory and "examples/ascent/tutorial/cloverleaf\_demos/".

<http://ascent-dav.org>

# Ascent tutorial examples are outlined in our documentation and included ready to run in Ascent installs

- <http://ascent-dav.org>
- Click on “Tutorial”



# Ascent's interface provides five top-level functions

- **open() / close()**

- Initialize and finalize an Ascent instance

- **publish()**

- Pass your simulation data to Ascent

- **execute()**

- Tell Ascent what to do

- **info()**

- Ask for details about Ascent's last operation

```
//  
// Run Ascent  
//  
  
Ascent ascent;  
ascent.open();  
  
ascent.publish(data);  
ascent.execute(actions);  
ascent.info(details);  
  
ascent.close();
```

The *publish()*, *execute()*, and *info()* methods take Conduit trees as an argument.  
What is a Conduit tree?

# Conduit provides intuitive APIs for in-memory data description and exchange

- **Provides an intuitive API for in-memory data description**

- Enables *human-friendly* hierarchical data organization
  - Can describe in-memory arrays without copying
  - Provides C++, C, Python, and Fortran APIs

- **Provides common conventions for exchanging complex data**

- Shared conventions for passing complex data (e.g. *Simulation Meshes*) enable modular interfaces across software libraries and simulation applications

- **Provides easy to use I/O interfaces for moving and storing data**

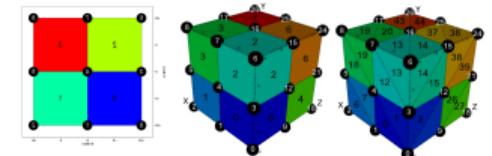
- Enables use cases like binary checkpoint restart
  - Supports moving complex data with MPI (serialization)



CONDUIT



Hierarchical in-memory data description



Conventions for sharing in-memory mesh data

<http://software.llnl.gov/conduit>  
<http://github.com/llnl/conduit>

Website and GitHub Repo

# Ascent uses Conduit to provide a flexible and extendable API

- Conduit underpins Ascent's support for C++, C, Python, and Fortran interfaces
- Conduit also enables using YAML to specify Ascent actions
- Conduit's zero-copy features help couple existing simulation data structures
- Conduit Blueprint provides a standard for how to present simulation meshes

Learning Ascent equates to learning how to construct and pass Conduit trees that encode your data and your expectations.

## To start, let's look at the Ascent "First Light" Example in C++

- [https://ascent.readthedocs.io/en/latest/Tutorial\\_Intro\\_First\\_Light.html](https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html)

```
#include <iostream>

#include "ascent.hpp"
#include "conduit_blueprint.hpp"

using namespace ascent;
using namespace conduit;

int main(int argc, char **argv)
{
    // Echo info about how Ascent was configured
    std::cout << ascent::about() << std::endl;

    // Create conduit node with an example mesh using
    // conduit blueprint's braid function
    // ref: https://llnl-conduit.readthedocs.io/en/latest/blueprint\_mesh.html#braid

    // Things to explore:
    //   changing the mesh resolution

    Node mesh;
    conduit::blueprint::mesh::examples::braid("hexs",
                                              50,
                                              50,
                                              50,
                                              mesh);
```

Instrument your “main” loop or similar function  
with access to evolving simulation state

This code generates an example mesh

# To start, let's look at the Ascent “First Light” Example in C++

- [https://ascent.readthedocs.io/en/latest/Tutorial\\_Intro\\_First\\_Light.html](https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html)

```
// create an Ascent instance  
Ascent a;  
  
// open ascent  
a.open();  
  
// publish mesh data to ascent  
a.publish(mesh);
```

Create an Ascent instance and set it up

Now Ascent has access to our mesh data

# To start, let's look at the Ascent "First Light" Example in C++

- [https://ascent.readthedocs.io/en/latest/Tutorial\\_Intro\\_First\\_Light.html](https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html)

```
//  
// Ascent's interface accepts "actions"  
// that tell Ascent what to execute  
//  
Node actions;  
Node &add_act = actions.append();  
add_act["action"] = "add_scenes";  
  
// Create an action that tells Ascent to:  
// add a scene (s1) with one plot (p1)  
// that will render a pseudocolor of  
// the mesh field 'braid'  
Node & scenes = add_act["scenes"];  
  
// things to explore:  
// changing plot type (mesh)  
// changing field name (for this dataset: radial)  
scenes["s1/plots/p1/type"] = "pseudocolor";  
scenes["s1/plots/p1/field"] = "braid";  
// set the output file name (ascent will add ".png")  
scenes["s1/image_name"] = "out_first_light_render_3d";
```

Create a tree that describes the actions we want Ascent to do

```
// view our full actions tree  
std::cout << actions.to_yaml() << std::endl;
```

```
-  
  action: "add_scenes"  
  scenes:  
    s1:  
      plots:  
        p1:  
          type: "pseudocolor"  
          field: "braid"  
          image_name: "out_first_light_render_3d"
```

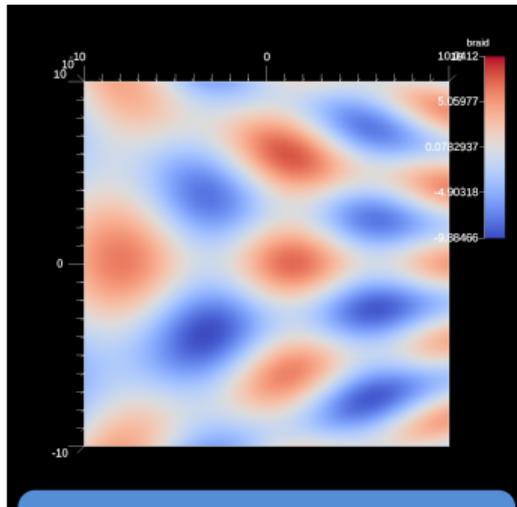
Equivalent YAML Description

# To start, let's look at the Ascent "First Light" Example in C++

- [https://ascent.readthedocs.io/en/latest/Tutorial\\_Intro\\_First\\_Light.html](https://ascent.readthedocs.io/en/latest/Tutorial_Intro_First_Light.html)

```
// execute the actions  
a.execute(actions);
```

Tell Ascent to execute these actions



Rendered Result!

# Ascent's interface provides five composable building blocks

**Scenes**

(Render Pictures)

**Pipelines**

(Transform Data)

**Extracts**

(Capture Data)

**Queries**

(Ask Questions)

**Triggers**

(Adapt Actions)

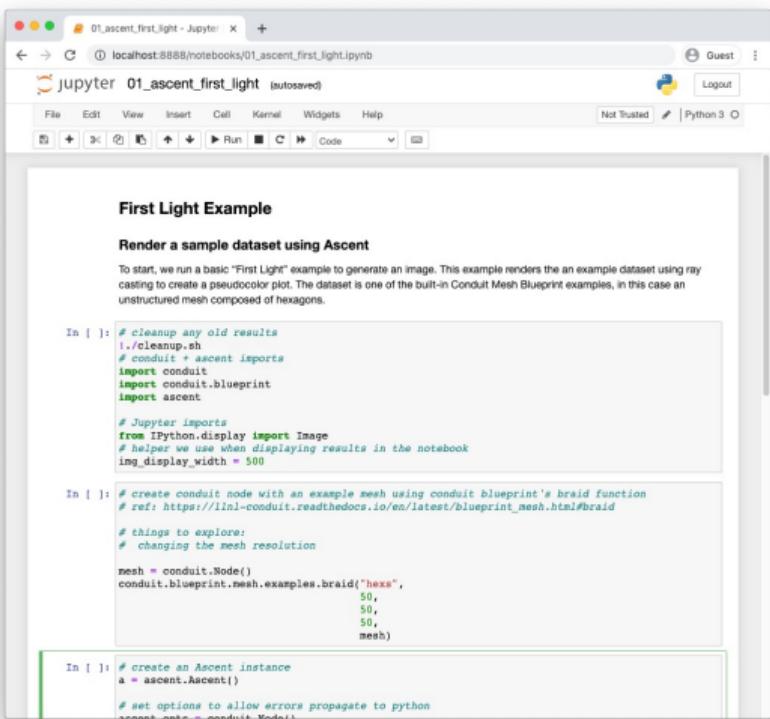
The tutorial provides examples for all of these.

# For the remainder of the tutorial, we will run the Ascent Tutorial examples using Jupyter Notebooks

## NOTE:

- VPNs or firewalls may block access to general AWS IP addresses and ports
- You may need to disconnect from VPN or request a firewall exemption
- LLNL attendees, you can use the EOR process:

<https://cspservices.llnl.gov/eor/>



The screenshot shows a Jupyter Notebook window titled "01\_ascent\_first\_light - Jupyter". The notebook contains three code cells. The first cell runs a script to clean up old files and import Conduit and Ascent modules. The second cell creates a mesh using Conduit's braid function. The third cell creates an Ascent instance and sets options to propagate errors to Python.

```
# cleanup any old results
!./cleanup.sh
# conduit + ascent imports
import conduit
import conduit.blueprint
import ascent

# Jupyter imports
from IPython.display import Image
# helper we use when displaying results in the notebook
img_display_width = 500

# create conduit node with an example mesh using conduit blueprint's braid function
# ref: https://llnl-conduit.readthedocs.io/en/latest/blueprint_mesh.html#braid

# things to explore:
#   changing the mesh resolution

mesh = conduit.Node()
conduit.blueprint.mesh.examples.braid("hexes",
                                       50,
                                       50,
                                       50,
                                       mesh)

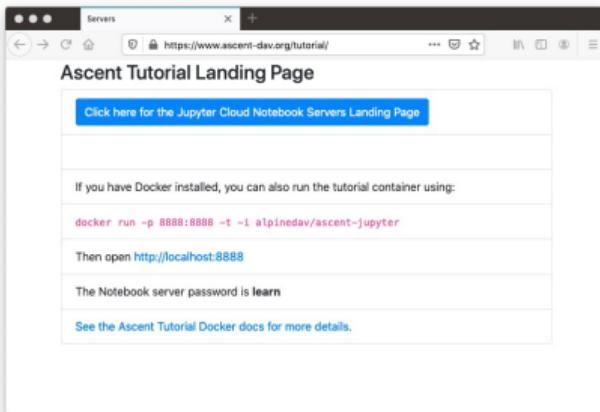
# create an Ascent instance
a = ascent.Ascent()

# set options to allow errors propagate to python
ascent_opts = conduit.Node()
```

You can run our tutorial examples using cloud hosted Jupyter Lab servers

Start here:

<https://www.ascent-dav.org/tutorial/>



# Thanks!

---

## Ascent Resources:

- Github: <https://github.com/alpine-dav/ascent>
- Docs: <http://ascent-dav.org/>
- Tutorial Landing Page: <https://www.ascent-dav.org/tutorial/>

## Contact Info:

Cyrus Harrison: [cyrush@llnl.gov](mailto:cyrush@llnl.gov)

Nicole Marsaglia: [marsaglia1@llnl.gov](mailto:marsaglia1@llnl.gov)

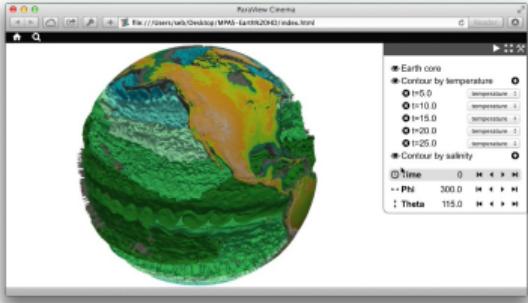
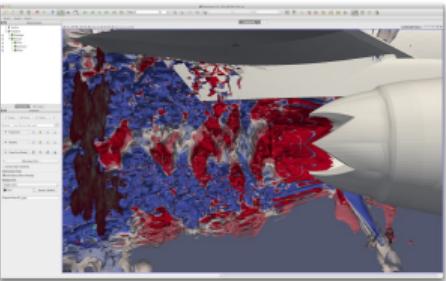
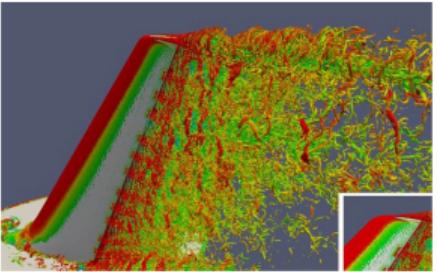
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.  
Lawrence Livermore National Security, LLC

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.





# Catalyst



# Anatomy of a Catalyst Adaptor

<https://gitlab.kitware.com/paraview/paraview/-/tree/master/Examples/Catalyst2>

- Image Data
- Polygonal
- Polyhedra
- Multimesh
- Multi Channel

# Anatomy of a Catalyst Adaptor

## Initialize

```
void Initialize(int argc, char* argv[])
{
    conduit_cpp::Node node;
    for (int cc = 1; cc < argc; ++cc)
    {
        node["catalyst/scripts/script" + std::to_string(cc - 1)].set_string(argv[cc]);
    }
    node["catalyst_load/implementation"] = "paraview";
    node["catalyst_load/search_paths/paraview"] = PARAVIEW_IMPL_DIR;
    catalyst_status err = catalyst_initialize(conduit_cpp::c_node(&node));
    if (err != catalyst_status_ok)
    {
        std::cerr << "Failed to initialize Catalyst: " << err << std::endl;
    }
}
```

# Anatomy of a Catalyst Adaptor

## Execute

```
void Execute(int cycle, double time, Grid& grid, Attributes& attrs)
{
    conduit_cpp::Node exec_params;

    // add time/cycle information
    auto state = exec_params["catalyst/state"];
    state["timestep"].set(cycle);
    state["time"].set(time);
    state["multiblock"].set(1);
```

# Anatomy of a Catalyst Adaptor

## Execute

```
// Add channels.
// We only have 1 channel here. Let's name it 'grid'.
auto channel = exec_params["catalyst/channels/grid"];

// Since this example is using Conduit Mesh Blueprint to define the mesh,
// we set the channel's type to "mesh".
channel["type"].set("mesh");

// now create the mesh.
auto mesh = channel["data"];

// start with coordsets (of course, the sequence is not important, just make
// it easier to think in this order).
mesh["coordsets/coords/type"].set("explicit");

mesh["coordsets/coords/values/x"].set_external(
    grid.GetPointsArray(), grid.GetNumberOfPoints(), /*offset=*/0, /*stride=*/3 * sizeof(double));
mesh["coordsets/coords/values/y"].set_external(grid.GetPointsArray(), grid.GetNumberOfPoints(),
    /*offset=*/sizeof(double), /*stride=*/3 * sizeof(double));
mesh["coordsets/coords/values/z"].set_external(grid.GetPointsArray(), grid.GetNumberOfPoints(),
    /*offset=*/2 * sizeof(double), /*stride=*/3 * sizeof(double));

// Next, add topology
mesh["topologies/mesh/type"].set("unstructured");
mesh["topologies/mesh/coordset"].set("coords");
mesh["topologies/mesh/elements/shape"].set("hex");
mesh["topologies/mesh/elements/connectivity"].set_external(
    grid.GetCellPoints(0), grid.GetNumberOfCells() * 8);
```

# Anatomy of a Catalyst Adaptor

## Execute

```
// Finally, add fields.
auto fields = mesh["fields"];
fields["velocity/association"].set("vertex");
fields["velocity/topology"].set("mesh");
fields["velocity/volume_dependent"].set("false");

// velocity is stored in non-interlaced form (unlike points).
fields["velocity/values/x"].set_external(
    attrbs.GetVelocityArray(), grid.GetNumberOfPoints(), /*offset=*/0);
fields["velocity/values/y"].set_external(attrbs.GetVelocityArray(), grid.GetNumberOfPoints(),
    /*offset=*/grid.GetNumberOfPoints() * sizeof(double));
fields["velocity/values/z"].set_external(attrbs.GetVelocityArray(), grid.GetNumberOfPoints(),
    /*offset=*/grid.GetNumberOfPoints() * sizeof(double) * 2);

// pressure is cell-data.
fields["pressure/association"].set("element");
fields["pressure/topology"].set("mesh");
fields["pressure/volume_dependent"].set("false");
fields["pressure/values"].set_external(attrbs.GetPressureArray(), grid.GetNumberOfCells());

catalyst_status err = catalyst_execute(conduit_cpp::c_node(&exec_params));
if (err != catalyst_status_ok)
{
    std::cerr << "Failed to execute Catalyst: " << err << std::endl;
}
```

# Anatomy of a Catalyst Adaptor

## Finalize

```
void Finalize()
{
    conduit_cpp::Node node;
    catalyst_status err = catalyst_finalize(conduit_cpp::c_node(&node));
    if (err != catalyst_status_ok)
    {
        std::cerr << "Failed to finalize Catalyst: " << err << std::endl;
    }
}
```

# Customizing Catalyst Pipelines

- Build example with Catalyst enabled
- Run example with sample pipeline script
- Modify the script to manipulate the pipeline
- Rerun the example with your new script

# ParaView Show & Tell

Follow along!

<https://www.paraview.org/download/>

