Traditional/Post Hoc Analysis
Traditional/Post Hoc Analysis

- sometime later…
Traditional/Post Hoc Analysis
Traditional/Post Hoc Analysis
In Situ Analysis
Small Run-Time Overhead

- Small initialization and finalization times
- Scalable analysis and visualization algorithms
- Reduced amount of IO
  - More complex IO patterns

XRAGE (LANL) simulation
Access to More Data

CTH (Sandia) simulation with roughly equal data stored at simulation time

Reflections and shadows added in post-processing for both examples
Better Insight

Full dump every 400 time steps versus *in situ* every 25 time steps

Animation courtesy Sean Ziegeler (PETTT/Engility)
Quick and Easy Run-Time Checks

Expected wind stress field at the surface of the ocean

Wind stress in new run, quick glance indicates using wrong wind stress

MPAS-O (LANL) simulation
Going to Exascale

- Highly parallel algorithms (if possible)
- Avoid file IO issues
- Freeze Python
- Process 0 reads Python script and broadcasts to other processes
- Don’t generate .pyc files
- Static libraries
- Simulation appropriate levels of parallelism
ParaView Architecture

Client (qt gui & python)

Create Pipeline();
Add Reader();
Add Filter();
Add Renderer();

Render();
Save DataExtract();

TCP

ParaView Server

Read File
Create VTK DataStructures

Filters
Arbitrarily process

Display | Write Files

MPI
Catalyst Architecture

Simulation Code

- Function calls
  - Create Pipeline()
  - Define Data()
  - Add Filter()
  - Add Renderer()
  - CoProcess()
  - Give New Data()
  - Render()
  - Save DataExtract()

Catalyst

- Take Data ()
  - Create VTK DataStructures
- Filters Arbitrarily process
- Display | Write Files

MPI
Motivation Continued

Bespoke -
Why not just hardcode plot routines?
Motivation Continued

Why not just hardcode plot routines?

Why not code in assembler?

Take advantage of ParaView - an extremely capable, flexible, scalable library with tens of thousands of developer hours behind it.
Why not use an extremely capable, flexible, scalable library with tens of thousands of developer hours behind it? and millions of lines of code? 

millions of lines of code?
memory overhead?
unfamiliar syntax?
Since 2009 we’ve been steering ParaView

- Millions of Lines of Code?
  - VTK 6.0 Modularity: Catalyst Editions (even custom)
  - Boiled down to three entry points to add to simulation

- Memory Overhead?
  - Editions again
  - Zero copy arrays

- Unfamiliar Syntax?
  - Documentation
  - User Level Controls: choose nitty gritty details or high level record/play
    - Can code at VTK level (c++ or Python)
    - Can record python scripts in GUI and run them
    - Can encapsulate as Domain Specific Commands (Sparta) in Input Deck
All of ParaView in my Sim?! 🙄

- The entire reason for VTK 6.0
- a.k.a. modularization
  - 19 kits -> 160 modules (== libs)
  - Remove unused code by deleting directories
    - 100MB->5MB
  - Add code by dropping in directories
  - Dependency scripts traverse includes

- At Catalyst level - called “Editions”

Base
Base + Essentials
Base + Essentials + Render
Base + Essentials + Render + Python

Custom
Maker scripts that build source tree
See 4.1 in Catalyst Users Guide
Efficient Memory-Wise

• Try to use simulation data structure memories (read only)
  – Catalyst support structure-of-arrays and array-of-structures memory layouts
  – Pipeline architecture ensures data isn’t modified by Catalyst
• Only create objects that are needed when they’re needed
• Small library size
  – Catalyst editions – versions without linking to unneeded parts of VTK/ParaView, Python and/or rendering components
• UH3D using Catalyst editions
  – 400 MB extra memory footprint with full ParaView
  – 40 MB extra memory footprint with Catalyst edition with rendering support

https://blog.kitware.com/paraview-catalyst-editions-what-are-they/
https://blog.kitware.com/why-is-paraview-using-all-that-memory/
Catalyst Architecture

Simulation Code

Function calls

Create Pipeline()
Define Data()
Add Filter()
Add Renderer()

CoProcess()
Give New Data()
Render()
Save DataExtract ()

Take Data ()
Create VTK DataStructures

Filters Arbitrarily process

Display | Write Files

MPI
How to make an Adaptor

1. Link to Catalyst Library
2. Insert three calls into simulation
   – `vtkCPProcessor::Initialize()`
   – `vtkCPProcessor::CoProcess()`
   – `vtkCPProcessor::Finalize()`
3. Translate Simulation Data to VTK Data Structures
   – `vtkCPInputDataDescription::SetGrid(vtkDataObject *)`
4. Define Pipeline/Results to gather
   Use generic option and let simulation user do this
API for the Developer

- **vtkCPDataDescription**
  - A means to transfer information from sim to ParaView (->) and back (<-)
  - -> named map of Grids for which new data is ready
    - “name” : vtkCPInputDataDescription
  - -> Time
  - <- Is Data necessary at this time?

- **vtkCPInputDataDescription (CPIDD)**
  - A container for a grid
  - Adaptor is responsible for populating CPIDD’s vtkDataObject
Efficient Compute-Wise

- Catalyst called every time step
- Negligible compute time if no output is requested
- Only create VTK objects if there is *in situ* work

Simulation ➔ CoProcess(grid, fields, time, timeStep) ➔ Adaptor ➔ Catalyst

RequestedDataDescription(description)

hasWork

opt hasWork

CreateGridAndFields()

CoProcess(description)
Populating vtkDataObjects

- See Section 3.2 of the Catalyst User Guide v 2.0
  - Data Structures
  - Geometry, Connectivity, Values
  - Arrays

```cpp
vtkFloatArray* arr = vtkFloatArray::New();
arr->SetName("an array");
float* values = new float[300];
arr->SetArray(values, 300, 0, vtkDoubleArray::VTK_DATA_ARRAY_DELETE);
arr->SetNumberOfComponents(3);
```

- ZeroCopy Arrays/Grids
  - http://www.vtk.org/Wiki/VTK/InSituDataStructures
  - vtkMappedDataArray (old)
  - vtkGenericDataArray
  - vtkSOADaDataArray
API for the Developer continued

• vtkCPPipeline
  – a ParaView pipeline with arbitrary contents

  – vtkCPPythonScriptPipeline
    • Takes in a python script that defines the pipeline

  – Or derive your own subclass and manually create VTK/PV filters
API for the Developer continued

- **vtkCPProcessor** - Manages the whole server
  - Sim runs visualization through CPProcessor calls
  - Has 0 or more **vtkCPDataDescriptions**
  - Has 0 or more **vtkCPPipelines**
  - **Initialize()**
    - Create Pipeline(s)
  - **CoProcess()**
    - Use CPDataDescriptions to ask if Pipelines need to run
    - If yes:
      - Populate their CPIID’s with new data
      - Call Update (actually CoProcess) on Pipeline
  - **Finalize()**
Small Code Footprint

- Typically 3 calls between simulation code and adaptor
  - Initialize()
    - MPI communicator (optional)
    - Add analysis scripts
  - CoProcess()
    - Does the work (potentially)
  - Finalize()
- Information provided by solver to adaptor
  - Time, time step, (optional) force output
  - Grids and fields
Simulation

Catalyst

Augmented script in input deck

Polygonal Output with Field Data

Rendered Images

User Workflow

Script Export

Statistics

Series Data
Creating a Catalyst Python Script

• Open a representative data set in ParaView GUI
• Setup pipeline
• Define the outputs
  – Load in the Catalyst Script Generator Plugin
  – Add writers to Pipeline - Data Extracts
  – (within Export Script Dialog) Export Views - Rendered Images
• Save Script
Create Pipeline 1 - load plugin

Only if ParaView < 5.5

<table>
<thead>
<tr>
<th>Name</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonOrthogonalSource</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>SciberQuestToolKit</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>QuadView</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>pvNektarReader</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>GMVReader</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>AnalyzeNIFTIIO</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>HSPartReader</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>EyesDomeLightingView</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>MobileRemoteControl</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>UncertaintyRendering</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>PacMan</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>CatalystScriptGeneratorPlugin</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>/media/ssd/.../BUILDS/ParaView/.../plugins</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>Version</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>Location</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>Required Plugins</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>Status</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>Auto Load</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>Moments</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>StreamingParticles</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>ArrowGlyph</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>SurfaceLC</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>SierraPlotTools</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>RGBZView</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>SLACTools</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>PointSprite_Plugin</td>
<td>Not Loaded</td>
</tr>
<tr>
<td>vtkPVInitializerPlugin</td>
<td>Loaded</td>
</tr>
</tbody>
</table>
Plugin adds two new menus

Two new menu items specific to creating Catalyst Python scripts

< PV 5.5

CoProcessing
Export State

Writers

VTK ADIOS Writer
CSV Table Writer
Parallel Hierarchical Box Data Writer
Parallel MultiBlockDataSet Writer
Parallel Image Data Writer
Parallel PolyData Writer
Parallel Rectilinear Grid Writer
Parallel Structured Grid Writer
Parallel UnstructuredGrid Writer

> PV 5.5

Data Extract Writers

Connect...
Pause Simulation
Continue
Set Breakpoint
Remove Breakpoint
Generate Script

VTK ADIOS Writer
CSV Table Writer
ExodusII Writer
Parallel Hierarchical Box Data Writer
Parallel MultiBlockDataSet Writer
Parallel Image Data Writer
Parallel PolyData Writer
Parallel Rectilinear Grid Writer
Parallel Structured Grid Writer
Parallel UnstructuredGrid Writer
Create Pipeline

Load “representative” data set

- **File**→**Open…**
  - Shortcut
  - Choose `/home/catalystuser/filename_4.pvtu`
Define Writers

- Only valid writers available in Writers menu
- Parameters:
  - File Name – %t gets replaced with time step
  - Write Frequency
Export the Pipeline (Script)

< PV 5.5

CoProcessing
Export State

> PV 5.5

Export Co-Processing State

This wizard will guide you through the steps required to export the current visualization state as a python script that can be run in the co-processing component of ParaView. Make sure to add appropriate writers for the desired pipelines to be used in the Writers menu.
Configure Renders

- Click “Output rendering components i.e. views”
- Choose which Windows to export: Previous View, Next View
- Set Render Frequency

In PV >= 5.6 will UI will be streamlined:
Save Script

- Click on **Finish**
- Save scripts as **dataextracts.py**
LANL’s ParaView 5.6 refactoring
About the Catalyst Script

• Generated script will look something like this
• Pass script (or scripts) as an argument to simulation run.
  – Sim will run, periodically produce extracts and images
• Need to change vis?
  – Regenerate script and rerun
  – (or just edit it)
Some other ways to define pipeline

Lower level:
By hand at VTK/PV level

- Derive directly from vtkCPPipeline
  Instead of vtkCPPythonScriptPipeline
- No change to DataDescription and CoProcessor use model
- Create VTK or PV pipeline manually

Higher Level:
By hand in simulation input deck

```
CATALYST BLOCK

1 begin results output
2 database type = catalyst
3
4 begin catalyst
5 #save jpg images instead of png's
6 image format = jpg
7
8 #make them HD (720p) resolution
9 image size = 1280 720
10 end catalyst
11 end results output

Listing 5. Imageset Command Example

1 begin results output
2 database type = catalyst
3
4 begin catalyst
5 #slice plane through the origin normal to the X axis
6 slice = center 0 0 0 normal 1 0 0
7 end catalyst
8 end results output

Listing 6. Slice Shortcut Command Example
```
Honorable Mention : Live

- Live

Configuration

Select state configuration options.

- Live Visualization ✓
- Output rendering components i.e. views 
- Output to Cinema 

- Get references to simulation’s pipeline(s)
- See data change as it evolves
- Optionally transfer designated data for further processing
- Optionally control settings of filters
- Optionally set breakpoints in simulation
Honorable Mention

- Cinema -
  - Image based parameter exploration
  - Tell sim what vis parameters you want to inspect
  - In situ save image for all Combinations
  - [http://cinemascience.org](http://cinemascience.org)
  - [http://cinemaviewer.org](http://cinemaviewer.org)
Thank You!

Further information: andy.bauer@kitware.com