In-Situ Data Analysis and Visualization: ParaView, Calalyst and VTK-m

GTC, San Jose, CA
March, 2015

Robert Maynard
Marcus D. Hanwell
Agenda

• Introduction to ParaView Catalyst
• Run example Catalyst Script
• Creating your own Catalyst Script
• Rendering grid and field information from live simulation
Going From Data to Visualization
The Visualization Pipeline

• A sequence of algorithms that operate on data objects to generate geometry
Data Types

- **Uniform Rectilinear** (vtkImageData)
- **Non-Uniform Rectilinear** (vtkRectilinearData)
- **Curvilinear** (vtkStructuredData)
- **Polygonal** (vtkPolyData)
- **Unstructured Grid** (vtkUnstructuredGrid)

**Multi-block**

- Hierarchical Adaptive Mesh Refinement (AMR)
- Hierarchical Uniform AMR
Why *In Situ*?

Need a supercomputer to analyze results from a hero run

2 orders of magnitude difference between each level
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
Quick and Easy Run-Time Checks

MPAS-O (LANL) simulation
CTH (Sandia) simulations comparing different workflows
Small Run-Time Overhead

- data generation
- write
- render
- cell to point
- contour

XRAGE (LANL) simulation
## Reduced File IO Costs

<table>
<thead>
<tr>
<th>Time of Processing</th>
<th>Type of File</th>
<th>Size per File</th>
<th>Size per 1000 time steps</th>
<th>Time per File to Write at Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post</td>
<td>Restart</td>
<td>1,300 MB</td>
<td>1,300,000 MB</td>
<td>1-20 seconds</td>
</tr>
<tr>
<td>Post</td>
<td>Ensight Dump</td>
<td>200 MB</td>
<td>200,000 MB</td>
<td>&gt; 10 seconds</td>
</tr>
<tr>
<td>In Situ</td>
<td>PNG</td>
<td>.25 MB</td>
<td>250 MB</td>
<td>&lt; 1 second</td>
</tr>
</tbody>
</table>

XRAGE (LANL) simulation
What is ParaView Catalyst?
ParaView Catalyst Architecture

ParaView Catalyst
Python Wrappings
ParaView Server
Parallel Abstractions and Controls
VTK
Core Visualization Algorithms
ParaView User Interface

Menu Bar
Toolbars
Pipeline Browser
Properties Panel
Advanced Toggle
3D View
Geometry Representations

Points
Wireframe
Surface
Surface with Edges
Volume

Toggle Color Legend
Reset Scalar Range
Mapped Variable
Vector Component
Representation
Edit Colors
Custom Scalar Range

Mapped Variable
Vector Component
Legend
Reset Scalar Range
Edit Colors
Mapped Variable
Vector Component
Legend
Reset Scalar Range
Edit Colors
Custom Scalar Range

Points
Wireframe
Surface
Surface with Edges
Volume
Running Simulation with Catalyst

Simulation → Disk Storage → Visualization

In Situ

Simulation → Disk Storage → Visualization

In Transit

Separate MPI
Running Simulation with Catalyst

Today we are going to be running *In Situ* and interactively with Catalyst.
# Create the reader and set the filename.
reader = sm.sources.Reader(FileNames=path)
view = sm.CreateRenderView()
repr = sm.CreateRepresentation(reader, view)
reader.UpdatePipeline()
dataInfo = reader.GetDataInformation()
pinfo = dataInfo.GetPointDataInformation()
arrayInfo = pinfo.GetArrayInformation("displacement9")

Augmented script in input deck.
GETTING READY TO RUN INTERACTIVELY
Two Ways for Live *In Situ* Analysis and Visualization

- **Without blocking**
  - Simulation proceeds while the user interacts with the data from a specific time step
  - “At scale” mode
- **With blocking** (*new in ParaView 4.2*)
  - Simulation is blocked while the user interacts with the data
  - “Debugging” mode
- Can switch between interactive and batch during run as well as disconnecting from the run
Getting Ready to Run Interactively

• From the command shell run
  – “~/insitu_demo/demo1.sh”

• This runs a fake simulation with a prebuilt catalyst pipeline
Run Interactively

Load ParaView and connect to the simulation
Run Interactively

Enable Extraction of Contour

Visualize the Contour
Run Interactively
CREATING CATALYST OUTPUT
Creating Catalyst Output

- ParaView GUI plugin to create Python scripts

- Developer generated “canned” scripts
  - See ParaView Catalyst User’s Guide
Create Python Scripts from ParaView

Load the Catalyst script generator GUI plugin
Create Python Scripts from ParaView

Load File

~/Demo2/filename_20_0.vti

Create Contour Filter
Create Python Scripts from ParaView

Using the Writers menu add a writer
Create Python Scripts from ParaView

1. Load the Catalyst script generator GUI plugin
2. Load File
3. Create Contour Filter
4. Using the Writers menu add a writer
Export Catalyst Live Script

Using the CoProcessing menu start the export process
Export Catalyst Live Script

1. Using the CoProcessing menu start the export process
2. Select ‘filename_20_0.pvti’ as the simulation inputs
3. Select Live Visualization
Other Catalyst Export Options

- Live visualization
- Output to Cinema
- Output Views
Other Catalyst Export Options

• Output Views
  – Image Type
  – File Name
  – Write Frequency
  – Magnification
  – Fit to Screen
  – %t for timestep
Other Catalyst Export Options

- Output to Cinema
  - Static vs Spherical Camera
GETTING READY TO RUN OUR SCRIPT
Getting Ready to Run Interactively

- run ~/insitu_demo/demo2.sh

We are running our premade python script, if you want to run the version you made change driver_path variable in demo2.sh
Run Interactively

Load ParaView and connect to the simulation
Run Interactively (Update with new Images)
Enable Extraction of Contour

Visualize the Contour
Developers Section Overview

• Most work is done in creating the adaptor between the simulation code and Catalyst
• Small footprint in main simulation code base
• Needs to be efficient both in memory and computationally
• Fortran, C++ and Python examples:
  – https://github.com/Kitware/ParaViewCatalystExampleCode
Interaction Overview

• Simulation has separate data structures from VTK data structures
• Use an adaptor to bridge the gap
  – Try to reuse existing memory
  – Also responsible for other interactions between simulation code and Catalyst
Developer Perspective

Simulation

- **INITIALIZE** (in pipelines)
- **COPROCESS** (in time, in grid, in fields)
- **FINALIZE**()

Adaptor

- **INITIALIZE**()
- **ADDPipeline** (in pipeline)
- **REQUESTDATADESCRIPTION** (in time, out fields)
- **COPROCESS** (in *vtkDataSet*)
- **FINALIZE**()
Online Help

• ParaView Catalyst User’s Guide:

• Email list:
  – paraview@paraview.org

• Doxygen:

• Sphinx:

• Websites:
  – http://www.paraview.org
  – http://www.paraview.org/in-situ/

• Examples:
  – https://github.com/Kitware/ParaViewCatalystExampleCode
VTK-m Project Goals

• A single place for the visualization community to collaborate, contribute, and leverage massively threaded algorithms.

• Reduce the challenges of writing highly concurrent algorithms by using data parallel algorithms.
VTK-m Architecture

- Combines strengths of multiple projects:
  - EAVL, Oak Ridge National Laboratory
  - DAX, Sandia National Laboratory
  - PISTON, Los Alamos National Laboratory
VTK-m Framework

Control Environment
- Grid Topology
- Array Handle
- Invoke

Device Adapter
- Allocate
- Transfer
- Schedule
- Sort
- ...

Execution Environment
- Cell Operations
- Field Operations
- Basic Math
- Make Cells

cont

exec

Worklet
VTK-m Arbitrary Composition

- VTK-m allows clients to access different memory layouts through the Array Handle and Dynamic Array Handle.
  - Allows for efficient in-situ integration
  - Allows for reduced data transfer
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>, FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT T operator() (T x) const {
        return vtkm::math::Sin(x);
    }
};
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>), FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT
    T operator()(T x) const {
        return vtkm::math::Sin(x);
    }
};
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>), FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT
    T operator()(T x) const {
        return vtkm::math::Sin(x);
    }
};
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>), FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT
    T operator()(T x) const {
        return vtkm::math::Sin(x);
    }
};
struct Sine: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<>), FieldOut<>);
  typedef _2 ExecutionSignature(_1);

  template<typename T>
  VTKM_EXEC_EXPORT T operator()(T x) const {
    return vtkm::math::Sin(x);
  }
};
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>, FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT T operator()(T x) const {
        return vtkm::math::Sin(x);
    }
};
struct Sine: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<>, FieldOut<>);
  typedef _2 ExecutionSignature(_1);

  template<typename T>
  VTKM_EXEC_EXPORT T operator()(T x) const {
    return vtkm::math::Sin(x);
  }
};
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>, FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT T operator()(T x) const {
        return vtkm::math::Sin(x);
    }
};

vtkm::cont::ArrayHandle<vtkm::Float32> inputHandle =
    vtkm::cont::make_ArrayHandle(input);
vtkm::cont::ArrayHandle<vtkm::Float32> sineResult;

vtkm::worklet::DispatcherMapField<Sine> dispatcher;
dispatcher.Invoke(inputHandle, sineResult);
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>, FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT T operator() (T x) const {
        return vtkm::math::Sin(x);
    }
};

vtkm::cont::ArrayHandle<vtkm::Float32> inputHandle =
    vtkm::cont::make_ArrayHandle(input);
vtkm::cont::ArrayHandle<vtkm::Float32> sineResult;

vtkm::worklet::DispatcherMapField<Sine> dispatcher;
dispatcher.Invoke(inputHandle, sineResult);
struct Sine : public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(FieldIn<>, FieldOut<>);
  typedef _2 ExecutionSignature(_1);

  template<typename T>
  VTKM_EXEC_EXPORT T operator()(T x) const {
    return vtkm::math::Sin(x);
  }
};

vtkm::cont::ArrayHandle<vtkm::Float32> inputHandle =
  vtkm::cont::make_ArrayHandle(input);
vtkm::cont::ArrayHandle<vtkm::Float32> sineResult;

vtkm::worklet::DispatcherMapField Sine dispatcher;
dispatcher.Invoke(inputHandle, sineResult);
struct Sine : public vtkm::worklet::WorkletMapField {

typedef void ControlSignature( FieldIn<>, FieldOut<>);

typedef _2 ExecutionSignature(_1);

template<typename T>
VTKM_EXEC_EXPORT
T operator()(T x) const {
    return vtkm::math::Sin(x);
}
};

vtkm::cont::ArrayHandle<vtkm::Float32> inputHandle =
    vtkm::cont::make_ArrayHandle(input);

vtkm::cont::ArrayHandle<vtkm::Float32> sineResult;

vtkm::worklet::DispatcherMapField<Sine> dispatcher;
dispenser.Invoke( inputHandle, sineResult );
struct Sine: public vtkm::worklet::WorkletMapField {
    typedef void ControlSignature(FieldIn<>, FieldOut<>);
    typedef _2 ExecutionSignature(_1);

    template<typename T>
    VTKM_EXEC_EXPORT
    T operator()(T x) const {
        return vtkm::math::Sin(x);
    }
};
Dispatcher Invoke Operations

- Convert polymorphic types to static types
- Check types
- Dispatcher-specific operations
  - Find domain length
  - Build index arrays
- Transport data from control to execution
- Run worklet invoke kernel
- Fetch thread-specific data
- Invoke worklet
- Push thread-specific data

```c
DispatcherMapField<Sine> dispatcher;
dispatcher.Invoke(inputHandle, sineResult);
```
struct ImagToPolar: public vtkm::worklet::WorkletMapField {
  typedef void ControlSignature(
    FieldIn<vtkm::TypeListTagScalar>,
    FieldIn<vtkm::TypeListTagScalar>,
    FieldOut<vtkm::TypeListTagScalar>,
    FieldOut<vtkm::TypeListTagScalar>);
  typedef void ExecutionSignature(_1, _2, _3, _4);

  template<typename RealType,
    typename ImaginaryType,
    typename MagnitudeType,
    typename PhaseType>
  VTKM_EXEC_EXPORT void operator()(RealType real,
    ImaginaryType imag,
    MagnitudeType &magnitude,
    PhaseType &phase) const {
    magnitude = vtkm::math::Sqrt(real*real + imag*imag);
    phase = vtkm::math::ATan2(imaginary, real);
  }
};
What We Have So Far

• Features
  – Statically and Dynamically Typed Arrays
  – Device Interface (Serial, Cuda, TBB under development)
  – Basic Worklet and Dispatcher
What We Have So Far

• Compiles with
  – gcc (4.8+), clang, msvc (2010+), icc, and pgi

• User Guide

• Ready for larger collaboration
Thank You!

Marcus D. Hanwell
- mhanwell@kitware.com
- @mhanwell
- +MarcusHanwell

Robert Maynard
- robert.maynard@kitware.com
- @robertjmaynard

Checkout out Kitware @ www.kitware.com and ParaView @ www.paraview.org

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!