ParaView

In situ Post-Processing and Visualization

Nathan Fabian David Thompson

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

Example, Part 2

Sandia National Laboratories

Coordinating with Simulation

- A simulation may already have IO layer
  - This will provide the mechanisms for output
  - Can take advantage for in situ
- A good time to initialize is when the simulation is setting up the file headers
- As each step is output to the file, update the in situ
- Call the finalization when the files are closed

Binding Fortran

- Everything will be a subroutine
  - "write-only"
- It is possible to pass by reference
  - From C: myFunc (int *num_particles)
  - Used as *num_particles
- Fortran 2003 lets you bind carefully using the interface statement
  - Bind(C,NAME="pvdotpython_init")
  - Integer(c_int), value :: num_part

Challenges of Fortran

- Upper/lower case in Fortran naming
- Leading underscore on function names
- Strings convention not standardized
  - Some null terminated
  - Length after char *
  - Length at end of arg list
- Complex numbers not native to C/C++
  - Different methods
  - Byte alignment especially problematic

InSituMacros.cmake

- Automatically creates linking files going from `dot.f90` to `vtkDotTask.cxx`
- To build the example, in ccmake set
  - DOT_IN_SITU=ON
  - DOT_IN_SITU_MODULES="pvDotPython"
  - pvDotPython_DIR="path_build_dir/insitutask"
- Creates `pvdotpython_m.f90` and a `in_situ_m.f90` that points to it
  - `pvdotpython_m.f90` links with `vtkDotTask.cxx`
  - `dot.f90` links with `in_situ_m.f90`

Sandia National Laboratories

Hardcode a pipeline

- This may be necessary if python is unavailable
- Does simplify the linking process a great deal
  - Except all the issues with Fortran ☺
- But it's not nearly as interesting
- Thus…

PVBatch Vs InSituPVBatch

- Main ()
  - Initialize MPI
  - Initialize script
  - Run script
  - Stop MPI

- Initialize ()
  - Initialize MPI
  - Initialize Script
- Update ()
  - Runs Script
- Exit ()
  - Stop MPI

Initialize InSituPVBatch

vtkDotTask.cxx

```
void pvdotpython_init(...
{
...
  pvDotBatch = vtkInsituPVBatch::New ();
  pvDotBatch->Initialize (initScript);
...
```

- Doesn't reinitialize MPI,
  but does setup internal VTK structures
  (for instance `vtkMultiProcessController::GetGlobalController`)
- `initScript` is `vtkStdString` so it can be read either from within the simulation's input or from some other source like a separate file

Sandia National Laboratories

Initialize Source

vtkDotTask.cxx

```
void pvdotpython_init(…, num_particles,
   double *pxyz, double *mass, double *pmom,
   double *pfrc)
{
...
pvDotSource->Initialize( num_particles,
   pxyz, mass, pmom, pfrc, …);
...
```

- Note, here passed in as pointers to do shallow copy
- May need to copy data later, during update
- Source is a singleton

Source Plugin Wrapping

DotSource.xml

```
<ServerManagerConfiguration>
  <ProxyGroup name="sources">
     <SourceProxy name="DotSource”
                  class="vtkDotSource">
     </SourceProxy>
  </ProxyGroup>
</ServerManagerConfiguration>
```

- A paraview plugin is also a dynamic library
  - Link that to the simulation
- This xml can also be used through the external module interface

Initialize Script Pipeline

insitu.py.in

```
import os
import sys
sys.path.append( '@pvDotPython_SOURCE_DIR@' )
sys.path.append( '@PARAVIEW_LIBRARY_DIRS@' )
sys.path.append( '@ParaView_DIR@/Utilities/
  VTKPythonWrapping' )
from paraview.simple import *
servermanager.LoadPlugin
  ("@pvDotPython_BINARY_DIR@/libpvDotPython.dylib")
source = servermanager.sources.DotSource ()
```

- Managed by cmake
  Sets the paths for module loading
- XML specifies servermanager.sources

Sandia National Laboratories

Initialize Script Pipeline

insitu.py.in

```
source = servermanager.sources.DotSource ()
glyph = Glyph()
glyph.Input = source
glyph.GlyphType = 'Sphere'

def update(process, cycle, time, dt):
```

- Note, glyph is built during initialization
  - Could be instead built during each update
  - For more complicated pipelines trades update speed for resident memory consumption
- Source is just a wrapper, better just to build it once

In Situ PVBatch Update

vtkDotTask.cxx

```
void pvdotpython_step(...
{
...
  pvDotBatch->Update (pvDotCallCount++, time, dt);
...
```

Sandia National Laboratories

Streaming-like update

Pseudo Code

```
void pvdotpython_step(...
{
   foreach (DataChunk D)
    {
    pvDotSource->SetChunk (D);
    pvDotBatch->Update (-1, time, dt);
    }
  pvDotBatch->Update (pvDotCallCount++, time, dt);
}
```

- Save image output, use vtkImageBlend
- Or build a custom image combiner

Sandia National Laboratories

Adding a Script Update

insitu.py.in

```
dprop = GetDisplayProperties (glyph)
dprop.ColorAttributeType = "POINT_DATA"
dprop.ColorArrayName = "Mass"
dprop.LookupTable = MakeBlueToRedLT (0, 1)
cam = GetActiveCamera()
cam.SetPosition( 20,20,20 )
cam.SetFocalPoint( 3, 3, 0 )
cam.SetViewUp( 0, 0, 1 )
SetViewProperties (UseLight = 1)
```

- Can also set CELL_DATA
- Array names may not match what shows in Paraview
  - Needs to match Source wrapper class

Sandia National Laboratories

Output During Update

insitu.py.in

```
SetViewProperties (ViewTime = time)
wri = XMLPolyDataWriter()
wri.FileName = 'pord%02f.vtp' % (time*10)
wri.Input = source
wri.UpdatePipeline(time = time)
WriteImage ("image_%(p)03d_%(c)06d.png" %
  {'p':process, 'c':cycle})
if (process != 0):
  os.remove ("image_%(p)03d_%(c)06d.png" %
    {'p':process, 'c':cycle})
```

- VERY IMPORTANT
  - `ViewTime = time` tells pipeline to update
  - Else writes the same image/output over and over

Sandia National Laboratories

Simulation Update Freq.

- If plugging into an existing IO layer
  - Facilities probably exist for output frequency
  - All the user input can be managed through a familiar interface
  - The in situ update can be called as infrequently as needed
- If not…

Simple Frequency Update

insitu.py.in

```
def update (process, cycle, time, dt):
  if ((cycle % 10) != 0):
    return
...
  # rest of update script
```

More Complex Version

insitu.py.in

```
# index 0 is the offset and index 1 is the delta
times = [ [0, 1], [5, 2], [10, 5]]

lastIndex = 0
lastTime = 0.0
def isTime (time):
  global lastIndex, lastTime
  for i in range(lastIndex,len(times)):
    if (time >= times[i][0]):
      index = I
  delTime = time - lastTime
  if (index != lastIndex) or (delTime >= times[index][1]):
    lastIndex = index
    lastTime = time
    return True
  else:
    return False

def update (process, cycle, time, dt):
  if (not isTime(time)):
    return
  # rest of update script
```

Sandia National Laboratories

Some discussion

- Analyzing the data
  - Are the particles moving much?
  - Ignore below a certain average velocity
- Getting creative
  - Interacting with specialized filters
  - Statistics filters finding outliers
    - Perhaps using the statistics tutorial from before the break.

Finalizing

vtkDotTask.cxx

```
void pvdotpython_fini(...
{
...
  pvDotSource->Delete ();
  pvDotBatch->Finalize ();
  pvDotBatch->Delete ();
   vtkDotSource::DestroySingleton ();
...
```

- Delete the source and batch
  - All objects from the script will go
  - But anything else outside won't
- Reminder: "VTK_DEBUG_LEAKS" option

Linking

vtkDotSource.cxx

```
vtkDotSource* vtkDotSource::Singleton = 0;
vtkDotSource* vtkDotSource::New()
{
  if ( vtkDotSource::Singleton )
  {
    return vtkDotSource::Singleton;
  }
...
```

- A singleton can then be linked both in the library and the python plugin
- Most dynamic loaders are capable of dealing with this

Necessary ParaView

CMakeLists.txt

```
vtkPVServerCommon, vtkPVPythonInterpretor,
 vtkPVPython, vtkParallel, vtkIO, vtkGraphics,
 vtkFiltering, vtkCommon, vtkzlib, vtksys
```

- Notice no QT
  - (paraview is built without GUI or client)
- Can potentially exclude other libraries
  - Depends on need
  - Suggest doing this toward the end when the calls are settled

Sandia National Laboratories

Demo

Sandia National Laboratories

HPC Platforms

- Many HPC platforms do not provide support for
  - Sockets
  - Threads
  - Dynamic libraries
  - X11 or hardware-accelerated OpenGL.
- This often requires cross-compiling
  - Beyond the scope of this tutorial, but see http://www.paraview.org/Wiki/Cross_compiling_ParaView3_and_VTK
  - ParaView mailing list

Sandia National Laboratories

Cross Compile/Static Builds

- Things to consider:
  - External module
    - Create “MySourceParaViewImport.cmake”

```
PARAVIEW_INCLUDE_WRAPPED_SOURCES
 (“${SRCS}”)
PARAVIEW_INCLUDE_SERVERMANAGER_SOURCES
 (“path_to/Plugin.xml”)
```

  - Static python with statically linked modules is possible.
  - Build image will be larger
    - Recent ParaView Cmake option:

```
PARAVIEW_MINIMAL_BUILD
```

Sandia National Laboratories

Conclusion

Sandia National Laboratories

Take Away

- Outlined method for adding Python scripting of ParaView pipelines to running simulations.
- Provided example code

  http://paraview.org/Wiki/IEEE_Vis09_ParaView_Tutorial
- Demonstrated scalability to large systems, but also
- Illustrated that the procedure is simple enough to be useful on smaller scales as well.

Questions?

Sandia National Laboratories