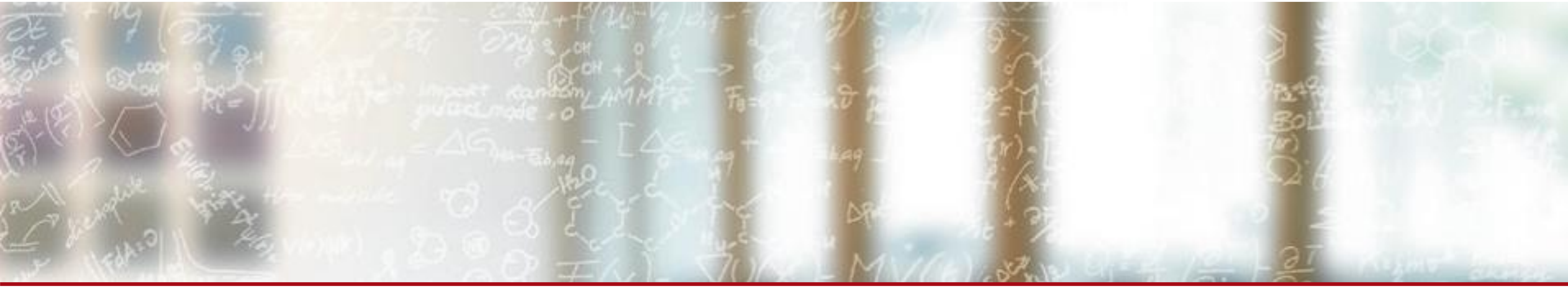


ParaView in a Jupyter notebook

Dr. Jean M. Favre, CSCS

April 26, 2021

Foreword

- ParaView is a very mature 3D parallel visualization ecosystem, in use at CSCS for many years.
- Users usually create a client-server connection from their remote desktop to a set of compute nodes on Piz Daint.
- ParaView uses an efficient and productive interface via Python scripts:
  - The client will read Python commands and the execution takes place [in parallel], on the server side
- A jupyter notebook can execute, stand-alone, or connected to a ParaView parallel server.

Overlook

Analyze data in a familiar, python-driven environment and create 3D interactive visualizations.

No need for a desktop ParaView client, and the [*sometimes complicated*] connection process in client-server mode.

Access to a GPU if you do not have a powerful desktop.

Outline

- Hello sphere ParaView program
- Hello sphere ParaView program + ipywidgets

- Hello sphere ParaView parallel program
- Local notebook connected to remote ParaView session on Piz Daint

- Numpy-to-ParaView
- SMP Parallelism
- MPI Parallelism

Pre-requisites if you use the Hybrid or EGL partition

Edit your $HOME/.jupyterhub.env

module load PyExtensions h5py/2.10.0-CrayGNU-20.11-python3-serial

module load ParaView

See the presentation by Tim Robinson@cscs for all generic details.

Resources on Piz Daint

- /users/jfavre/Projects/Visualization-training/ParaView

Hello_Sphere-ParaView.0.ipynb

- Standard ParaView Python initialization
- Standard pipeline
  - ParaView Source
  - ParaView Representation
  - Render

\+

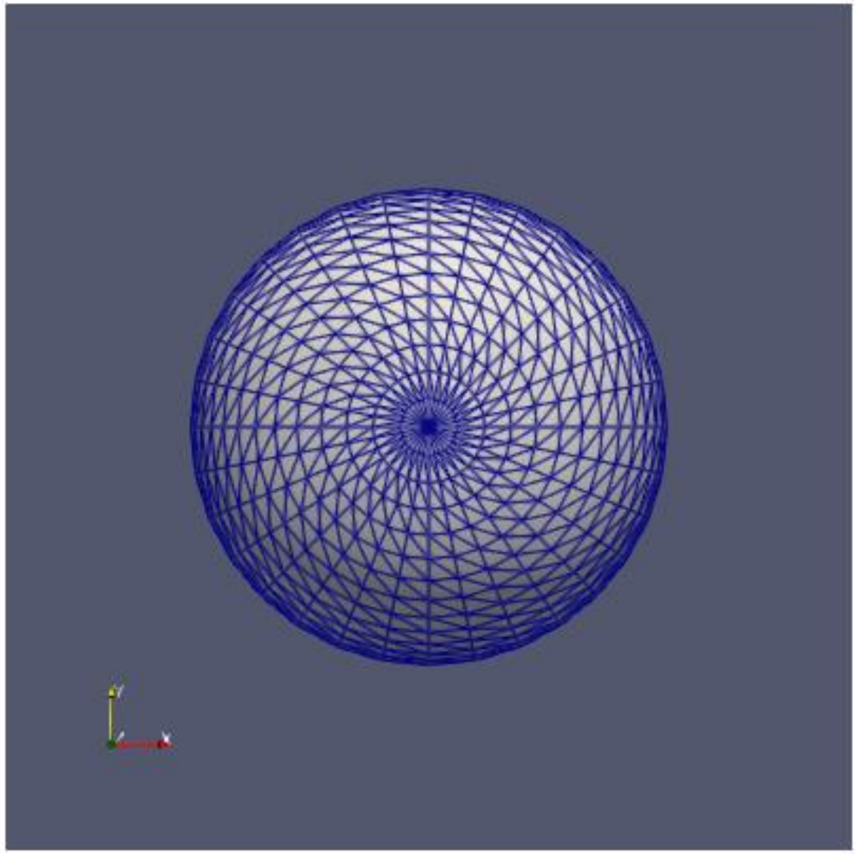
- PVDisplay widget (contributed by NVIDIA)

ParaView Hello Sphere Test

[1]:
```
from paraview.simple import *
```

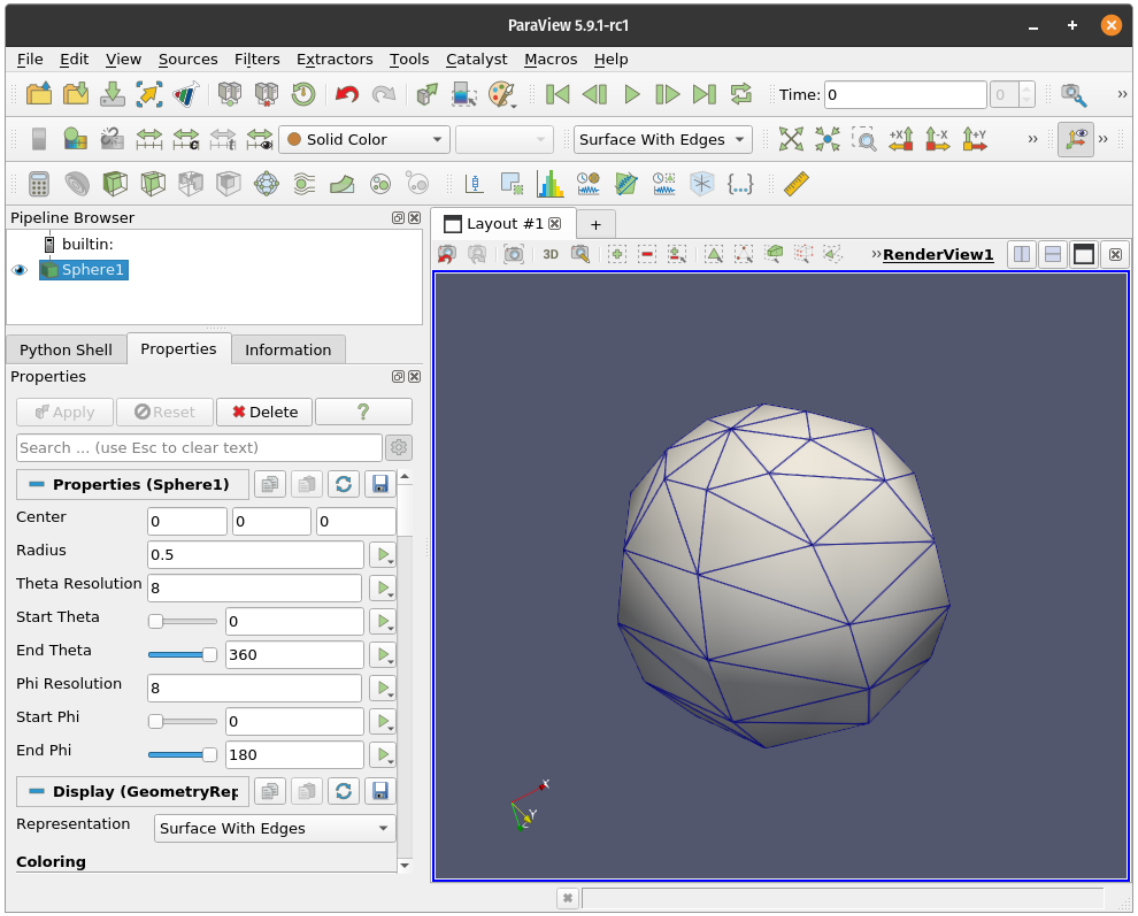
[2]:
```
sphere = Sphere(ThetaResolution=32, PhiResolution=32)

rep = Show()
rep.Representation = "Surface With Edges"
```

[3]:
```
from ipyparaview.widgets import PVDisplay
disp = PVDisplay(GetActiveView())
w = display(disp)
```



From your desktop to Jupyter

- The desktop ParaView app can give you a Python script with a new View, and all objects and their properties…
- You can bootstrap the process of writing a notebook with most of the Python code saved by the ParaView desktop app.
- The View will need a special handling.
- Live demonstration…



Hello World (Sphere) augmented with ipywidgets

sphere.ListProperties()

Attach PhiResolution and ThetaResolution to an IntSlider

```
['Center',
'EndPhi',
'EndTheta',
'PhiResolution',
'PointData',
'Radius',
'StartPhi',
'StartTheta',
'ThetaResolution‘]
```

Hello World (Sphere) augmented with ipywidgets

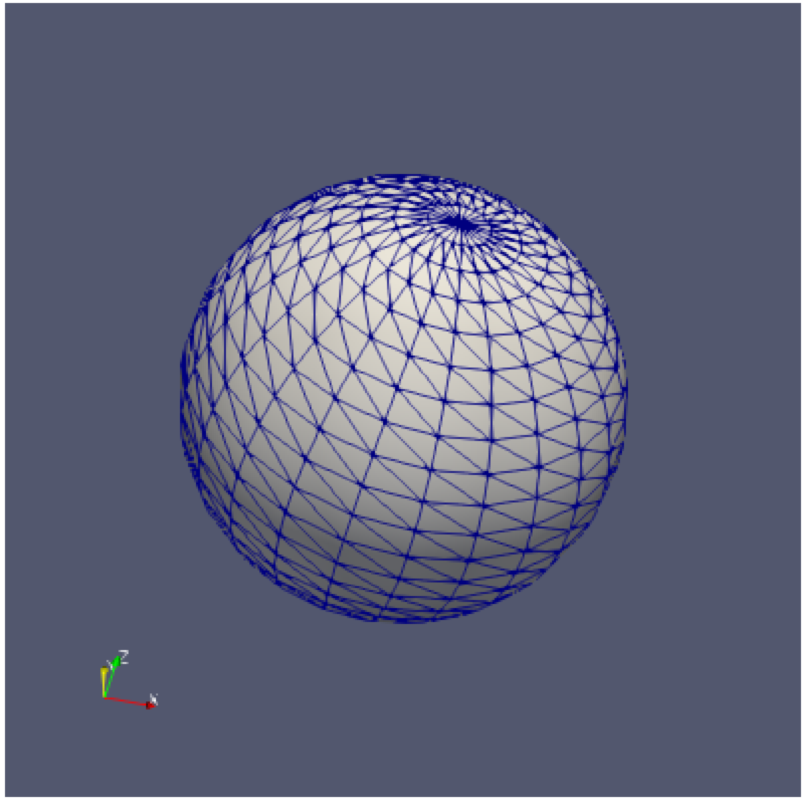
sphere.ListProperties()

Attach PhiResolution and ThetaResolution to an IntSlider

```
from ipywidgets import interact, IntSlider

# automatically triggers a pipeline update, and a render event
def Sphere_resolution(res):
    sphere.ThetaResolution = sphere.PhiResolution = res
    sphere.UpdatePipeline()

i = interact(Sphere_resolution,
             res=IntSlider(min=3, max=48, step=1, value=12)
            )
```

```
['Center',
'EndPhi',
'EndTheta',
'PhiResolution',
'PointData',
'Radius',
'StartPhi',
'StartTheta',
'ThetaResolution‘]
```

Hello_Sphere-ParaView.1.ipynb



[6]:
```python
# Interact from ipywidgets gives us a simple way to interactively control values
# with a callback function
from ipywidgets import interact, IntSlider

# set the Theta and Phi resolution and trigger a pipeline update
def Sphere_resolution(res):
    sphere.ThetaResolution = sphere.PhiResolution = res
    sphere.UpdatePipeline()

i = interact(Sphere_resolution, res=IntSlider(min=3, max=48, step=1, value=12))
```

res 24

Caveat

The standard SaveScreenshot() no longer works

Nick Leaf of NVIDIA has been very helpful in debugging the issue and for the moment, we have agreed to use the following code instead.

```
def SaveImage(filename):
  from vtk import vtkPNGWriter
  img_writer = vtkPNGWriter()
  img_writer.SetInputConnection(disp.w2i.GetOutputPort())
  img_writer.SetFileName(filename)
  img_writer.Write()

SaveImage("/users/jfavre/screenshot.png")
```

Data interfaces. How to read your favorite data arrays

From arrays to VTK grid objects

/users/jfavre/Projects/ParaView/Python/vtkGridConstructors.py …

is an early attempt at ingesting different types of [numpy] arrays and creating the proper grid structures from the VTK world.

Please try it, report issues, contribute…

Example:

pvTestGridConstructors.ipynb

Molecular Data Animation. Hello_Molecule.ipynb

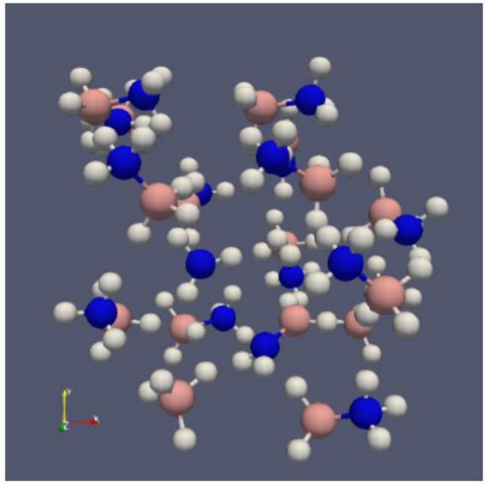
[1]:
```
from paraview.simple import *
```

[2]:
```
molecule1 = XYZReader(FileName='/users/jfavre/Projects/Rizzi/release_H2_ex.xyz')
nb_of_timesteps = len(molecule1.TimestepValues)
print("Molecule trajectories with ", nb_of_timesteps, " steps")
```

```
Molecule trajectories with  1500  steps
```

[3]:
```
computeMoleculeBonds1 = ComputeMoleculeBonds(Input=molecule1)
computeMoleculeBonds1.UpdatePipeline()
computeMoleculeBonds1Display = Show(computeMoleculeBonds1, GetActiveView())
```

[4]:
```
from ipyparaview.widgets import PVDisplay
disp = PVDisplay(GetActiveView())
w = display(disp)
```

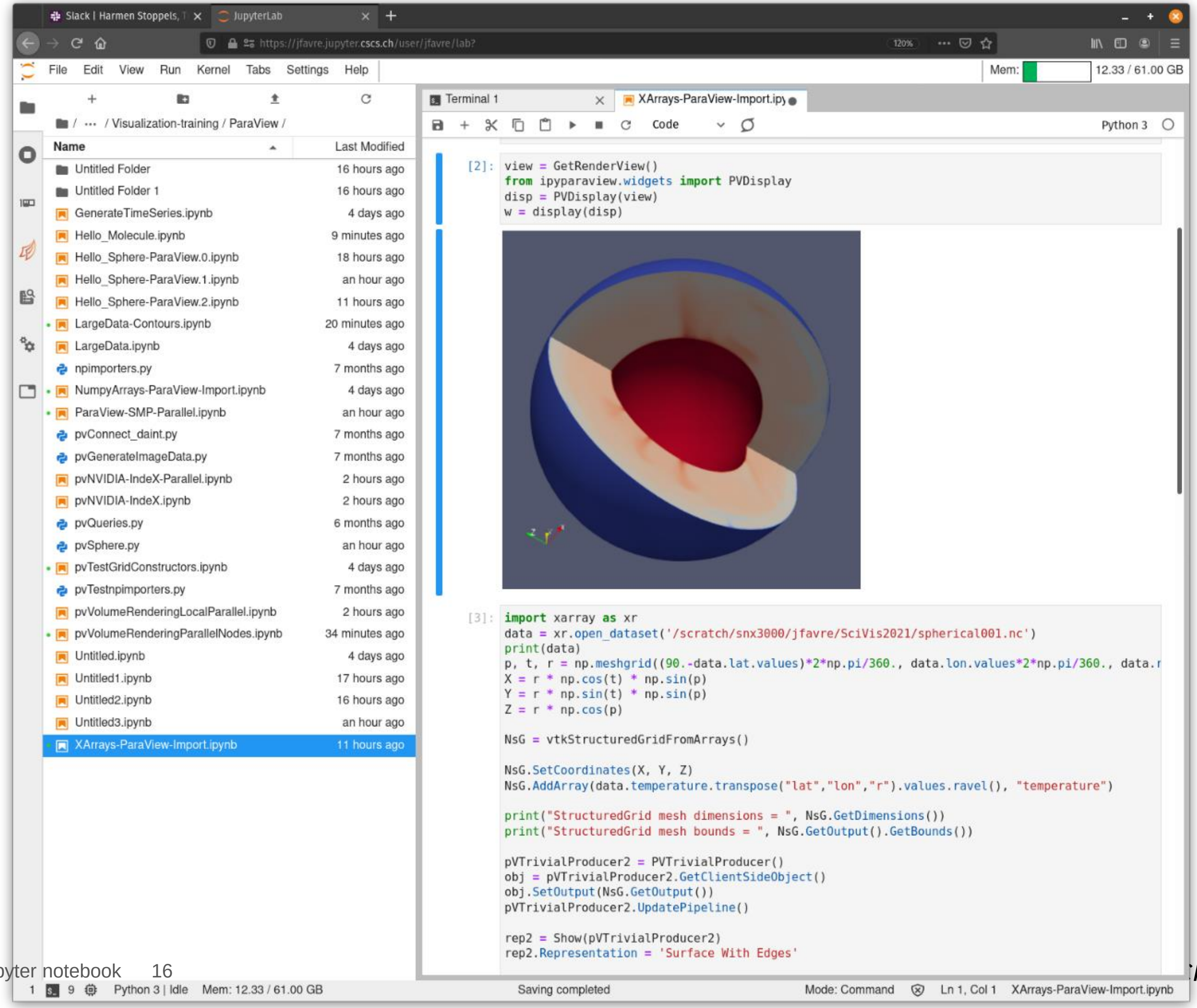


[5]:
```
# Interact from ipywidgets gives us a simple way to interactively control values
# with a callback function
from ipywidgets import interact, IntSlider

# set a new time-step
def time_slider(t):
    GetActiveView().ViewTime = t

i = interact(time_slider, t=IntSlider(min=0, max=nb_of_timesteps-1, step=1, value=0))
```

t 568

Example with xarray

- Xarrays-ParaView-Import.ipynb



Parallel visualization scenarios

Classic console output for client-server connection

Accepting connection(s): rancate:1100

#SBATCH --job-name=pvserver

#SBATCH --nodes=1

#SBATCH --ntasks-per-node=8

#SBATCH --ntasks=8

#SBATCH --time=00:20:00

#SBATCH --partition=debug

#SBATCH –account=csstaff

#SBATCH --constraint=gpu

srun -n 8 -N 1 --cpu_bind=sockets pvserver -rc -ch=daint103.cscs.ch -sp=1100

Submitted batch job 123456789

pvVolumeRenderingLocalParallel.ipynb (on-the-node parallelism)

This notebook can connect to a parallel set of ParaView servers running on the allocated compute node. It creates a synthetic data source (a sphere), and creates a polygonal display of it. Then, it creates a ParaView display widget showing the primary render view. The notebook further demonstrates how we may use interaction widgets (sliders), to change the resolution of the sphere.

[1]:
```
from paraview.simple import *
from paraview.modules.vtkRemotingCore import vtkProcessModule
```

[2]:
```
# to run in parallel on-the-allocated node, one would issue an
# srun command at the terminal:
# module load ParaView
# srun -n 8 `which pvserver`
# followed by a Connect() command

Connect("localhost")
```

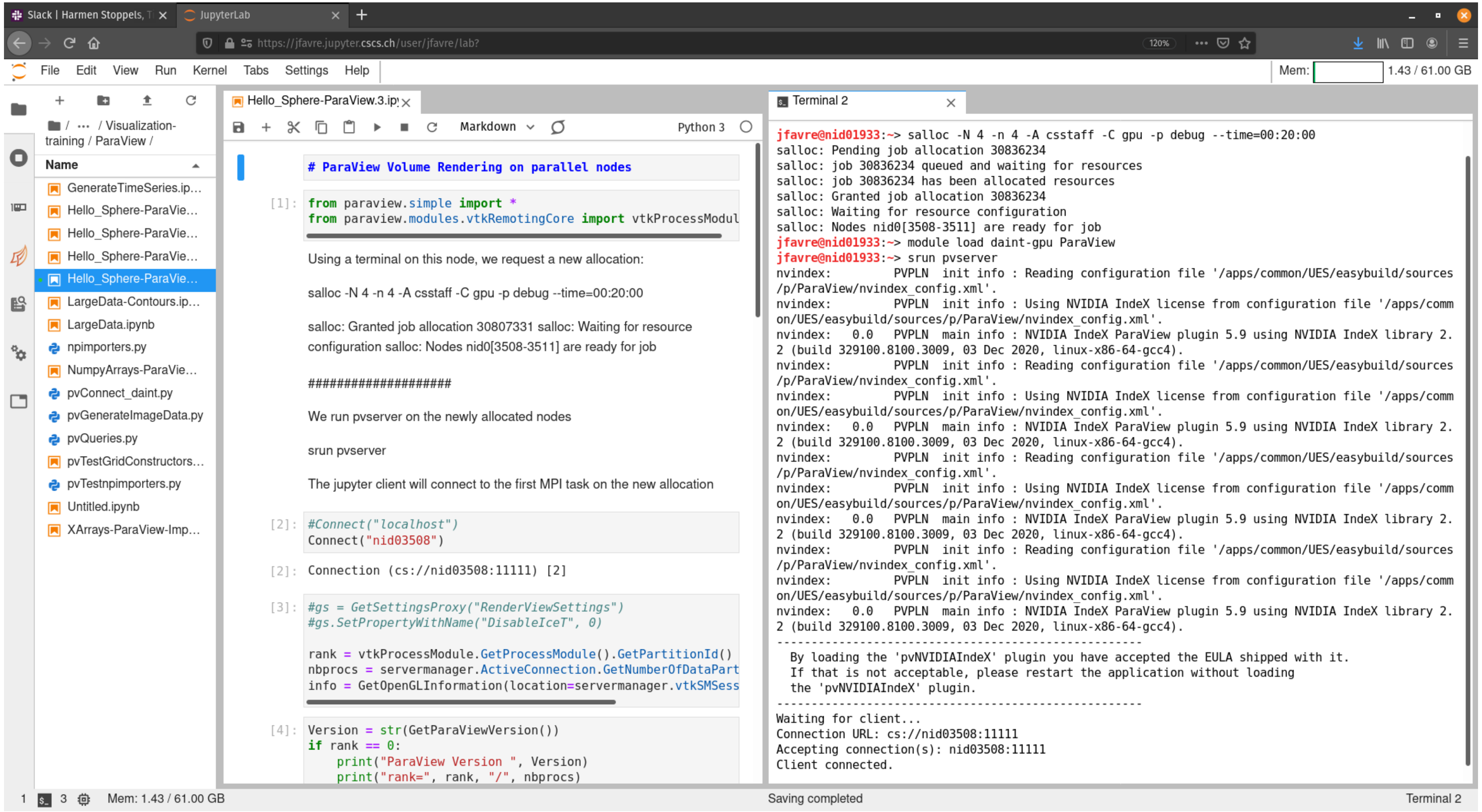
[2]:
```
Connection (cs://localhost:11111) [2]
```

[3]:
```
rank = vtkProcessModule.GetProcessModule().GetPartitionId()
nbprocs = servermanager.ActiveConnection.GetNumberOfDataPartitions()
info = GetOpenGLInformation(location=servermanager.vtkSMSession.RENDER_
print("nbprocs= ",nbprocs)
```

```
nbprocs=  8
```

```
jfavre@nid03173:~> module avail ParaView


ParaView/5.9.0-CrayGNU-20.11-EGL-python3(default)
jfavre@nid03173:~> module load ParaView
jfavre@nid03173:~>
jfavre@nid03173:~> srun -n 8 pvserver

Waiting for client...
Connection URL: cs://nid03173:11111
Accepting connection(s): nid03173:11111
Client connected.
```

pvVolumeRenderingParallelNodes.ipynb (with extra-node parallelism)



Local jupyter lab (on your desktop) + parallel pv server on Piz Daint

Local Jupyter Lab notebook

*from paraview.simple import **

ReverseConnect("1100")

N.B. The client is put in wait mode with the call above, **before** issuing the srun command on compute node(s)

- get your userid on Piz Daint (mine is 1100)
- Replace the call Connect(“localhost”) by a ReverseConnect(port)
- Use id as port number
- *ReverseConnect("1100")*

Local jupyter lab (on your desktop) + parallel pv server on Piz Daint

Local Jupyter Lab notebook

*from paraview.simple import **

ReverseConnect("1100")

Terminal window

- open an ssh tunnel on port 1100.
- select one login node. Here we use daint101.cscs.ch

ssh -l jfavre -R 1100:localhost:1100 daint101.cscs.ch

module load daint-gpu

module load ParaView

*srun -C gpu –A csstaff -p debug -t 00:10:00 -n 8 -N 1 *

pvserver -rc -ch=daint101.cscs.ch -sp=1100

There is more than MPI-based parallelism

- SMP parallelism

- Quite a few accelerated filter work in parallel, in a transparent fashion (desktop ParaView)

Raytracing.ipynb

csce RayTracing Last Checkpoint: 2 minutes ago (unsaved changes)

Logout Control Panel

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

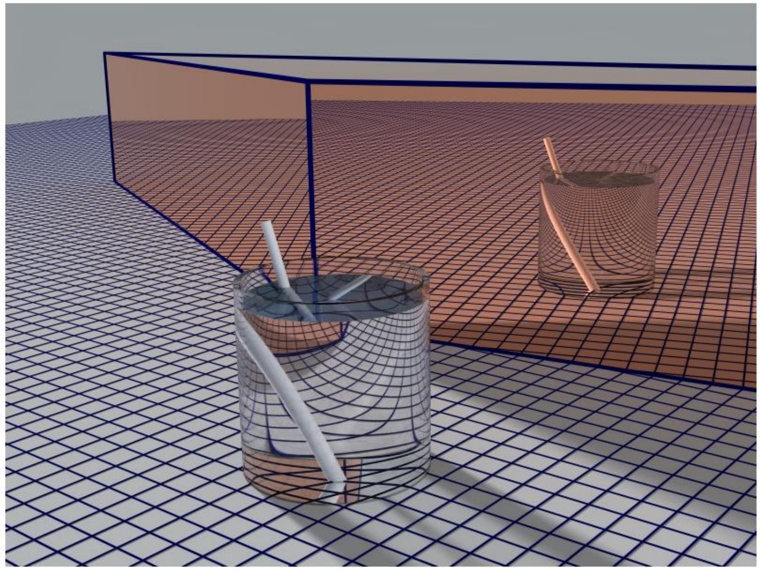
```
renderView1.AdditionalLights = [light1, light2]
renderView1.OSPRayMaterialLibrary = materialLibrary1
```

In [6]:

```
renderView1.ViewSize = [800, 600]
```

In [7]:

```
# import the PVDisplay widget, then instantiate one for renV
from ipyparaview.widgets import PVDisplay
disp = PVDisplay(renderView1)

# show the widget once
display(disp)
```



In [8]:

```
from paraview.modules.vtkPVClientServerCoreRendering import vtkPVOpenGLInformation

info = vtkPVOpenGLInformation()
info.CopyFromObject(None)
print("Vendor:   %s" % info.GetVendor())
print("Version:  %s" % info.GetVersion())
print("Renderer: %s" % info.GetRenderer())
```

```
Vendor:   NVIDIA Corporation
Version:  4.6.0 NVIDIA 418.39
Renderer: Tesla P100-PCIE-16GB/PCIe/SSE2
```

- Ray-tracing is executed on the GPU

  renderView1.BackEnd = 'OptiX pathtracer‘

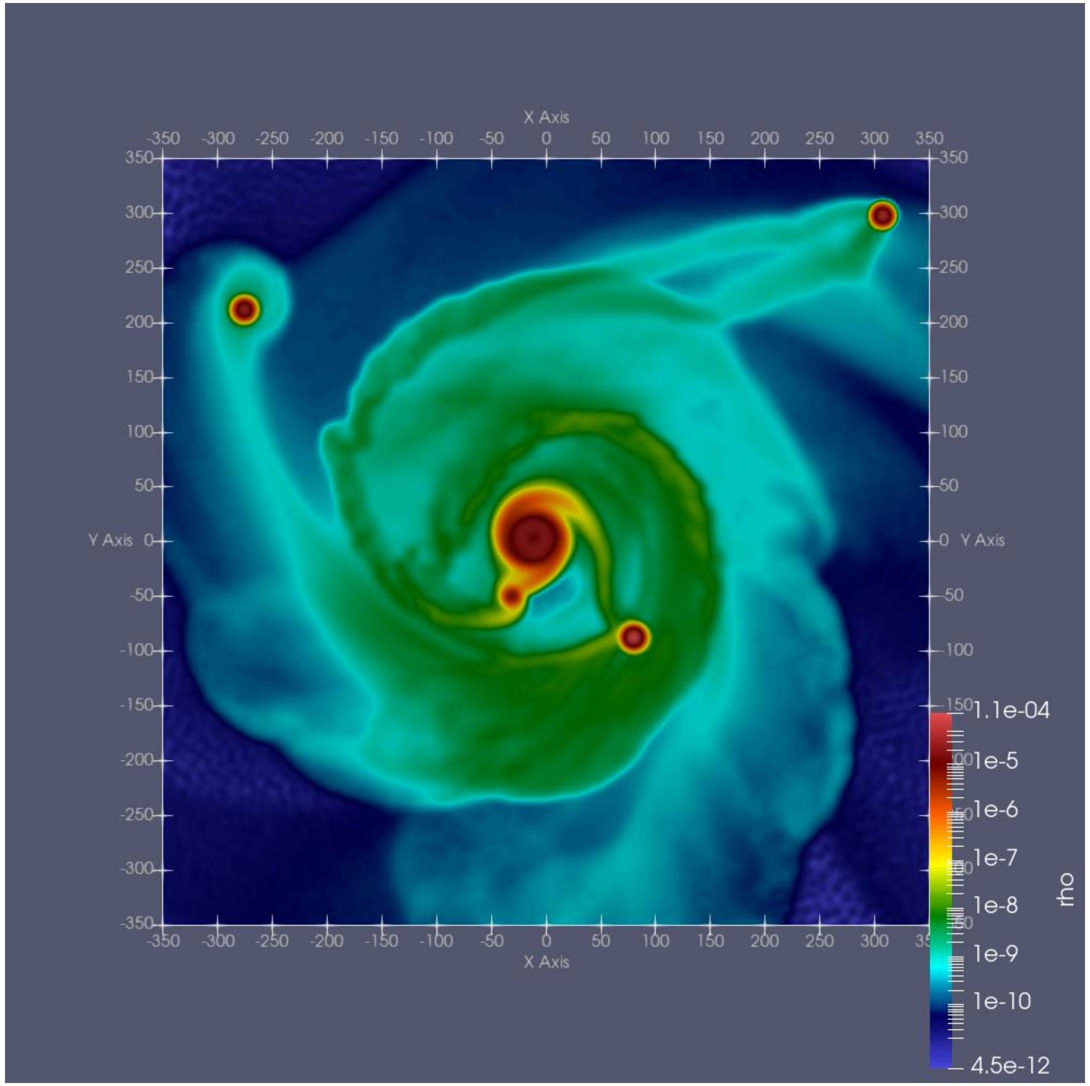
- Or runs on all available CPU threads

  renderView1.BackEnd = 'OSPRay raycaster‘

  renderView1.BackEnd = 'OSPRay pathtracer‘

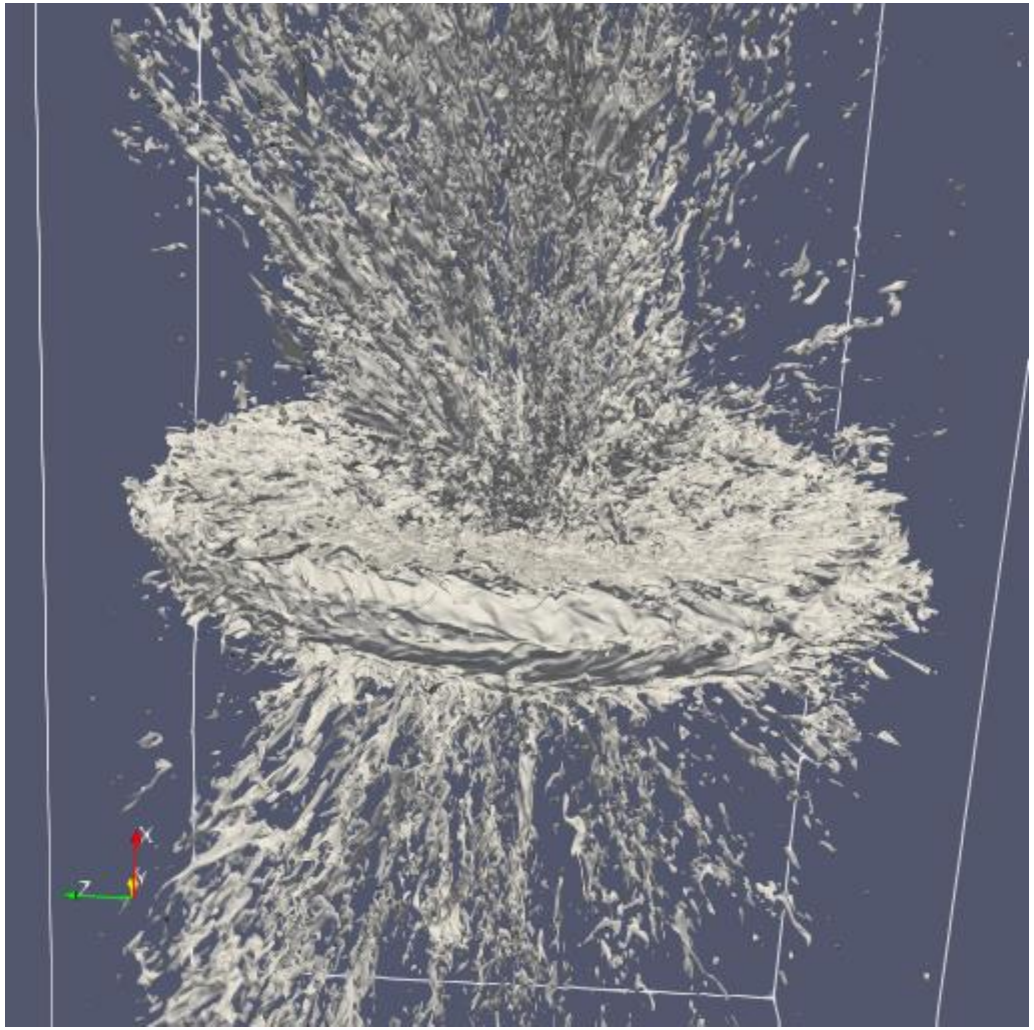
SMP-accelerated filters: the vtk SPH Interpolator

- ParaView-SMP-Parallel.ipynb



“Accelerated Algorithms plugin

- LargeData-Contours.ipynb
- How to load an additional plugin?



Questions?

- Use the chat for Q/A

Your wish list?

What do you wish to have to improve your experience with ParaView (or 3D visualization) at CSCS?

Send me direct email jfavre@cscs.ch to discuss it further.

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich