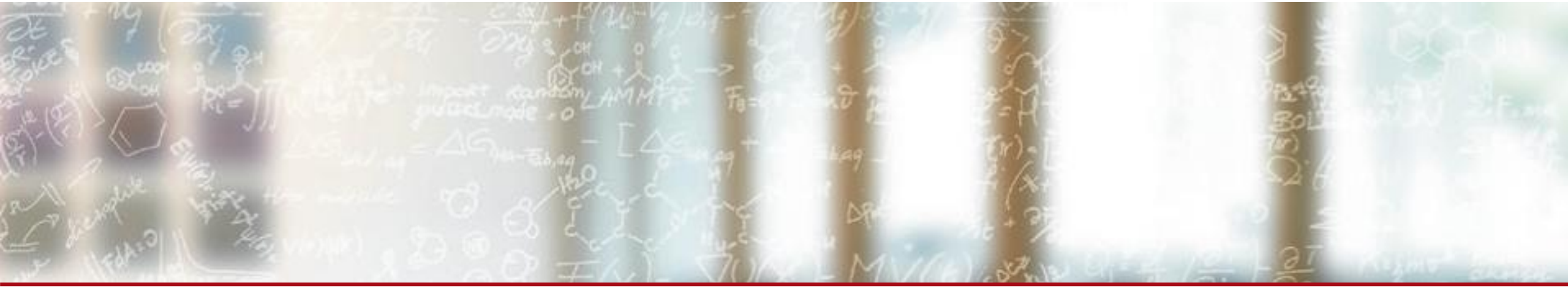
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Parallel data visualization

PDC Summer School 2023

Jean M. Favre, CSCS

August 24, 2023

What you will learn

- How to run an SMP-based ParaView session on the desktop
- How to run an MPI-based ParaView parallel session on your desktop and the PDC cluster
- Bottlenecks, or limitations
  - Parallel I/O
  - Parallel data extraction
  - Parallel graphics
  - Parallelism in a ParaView Python script
- Time parallelism

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Parallel data visualisation on the desktop

Multi-core nodes

- Threaded, shared-memory data processing on CPUs

A thread of execution is a sequence of instructions that can be executed concurrently with other such sequences in multithreading environments, while sharing the same address space.

- coarse-grained shared-memory computing. In ParaView/VTK, the SMPTools provide an abstraction over:
  - TBB (available on Dardel)
  - OpenMP
  - STD Thread (available on Dardel)

Demonstration on this laptop

- Intel® Core™ i7-12700H Processor

- export VTK_SMP_BACKEND_IN_USE=TBB or STDThread
- export VTK_SMP_MAX_THREADS=[0-20]

Shared memory parallel algorithms were kick-started in 2013

- I *grep*-ed for SMPTools and found it in 119 C++ class for ParaView 5.9
- I *grep*-ed for SMPTools and found it in 201 C++ class for ParaView 5.11

./IO/Geometry/vtkOpenFOAMReader.cxx
./IO/CGNS/vtkCGNSReader.cxx
./Imaging/Core/vtkImageDifference.cxx
./Imaging/Hybrid/vtkGaussianSplatter.cxx
./Rendering/Image/vtkDepthImageToPointCloud.cxx
./Parallel/DIY/vtkDIYGhostUtilities.cxx
./Filters/Geometry/vtkGeometryFilter.cxx
./Filters/GeometryPreview/vtkOctreeImageToPointSetFilter.cxx
./Filters/Core/vtkElevationFilter.cxx
./Filters/Core/vtkFlyingEdges3D.cxx
./Filters/Core/vtkResampleWithDataSet.cxx
./Filters/Core/vtkContour3DLinearGrid.cxx
./Filters/Core/vtkPolyDataNormals.cxx
etc…

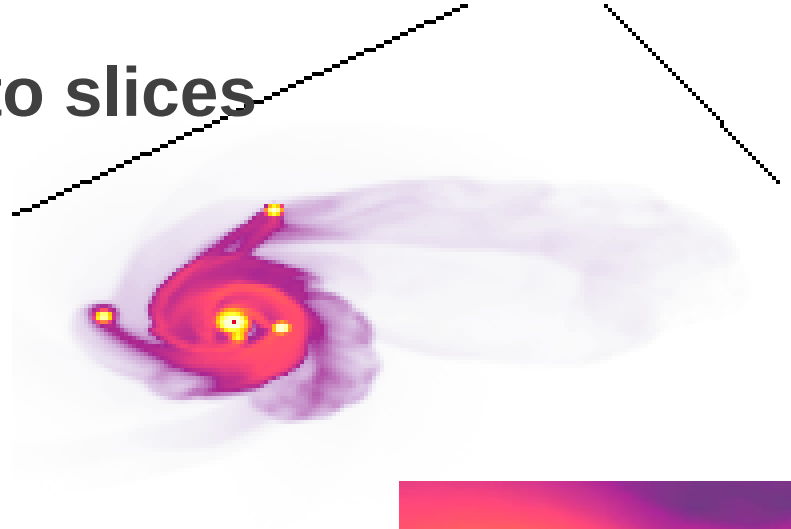
./Filters/Core/vtkCellCenters.cxx
./Filters/Core/vtkContour3DLinearGrid.cxx
./Filters/Core/vtkArrayCalculator.cxx
./Filters/Core/vtkStructuredDataPlaneCutter.cxx
./Filters/Core/vtkProbeFilter.cxx
./Filters/Extraction/vtkExtractGeometry.cxx
./Filters/ParallelDIY2/vtkRedistributeDataSetFilter.cxx
./Filters/Points/vtkHierarchicalBinningFilter.cxx
./Filters/Points/vtkPointOccupancyFilter.cxx
./Filters/Points/vtkMaskPointsFilter.cxx
./Filters/Points/vtkProjectPointsToPlane.cxx
./Filters/Points/vtkPCACurvatureEstimation.cxx
./Filters/Points/vtkPointSmoothingFilter.cxx
./Filters/Points/vtkPointInterpolator2D.cxx
./Filters/Points/vtkPointDensityFilter.cxx
./Filters/General/vtkPointConnectivityFilter.cxx
./Filters/General/vtkDiscreteFlyingEdges3D.cxx
./Filters/General/vtkWarpScalar.cxx
./Filters/General/vtkGradientFilter.cxx
etc…

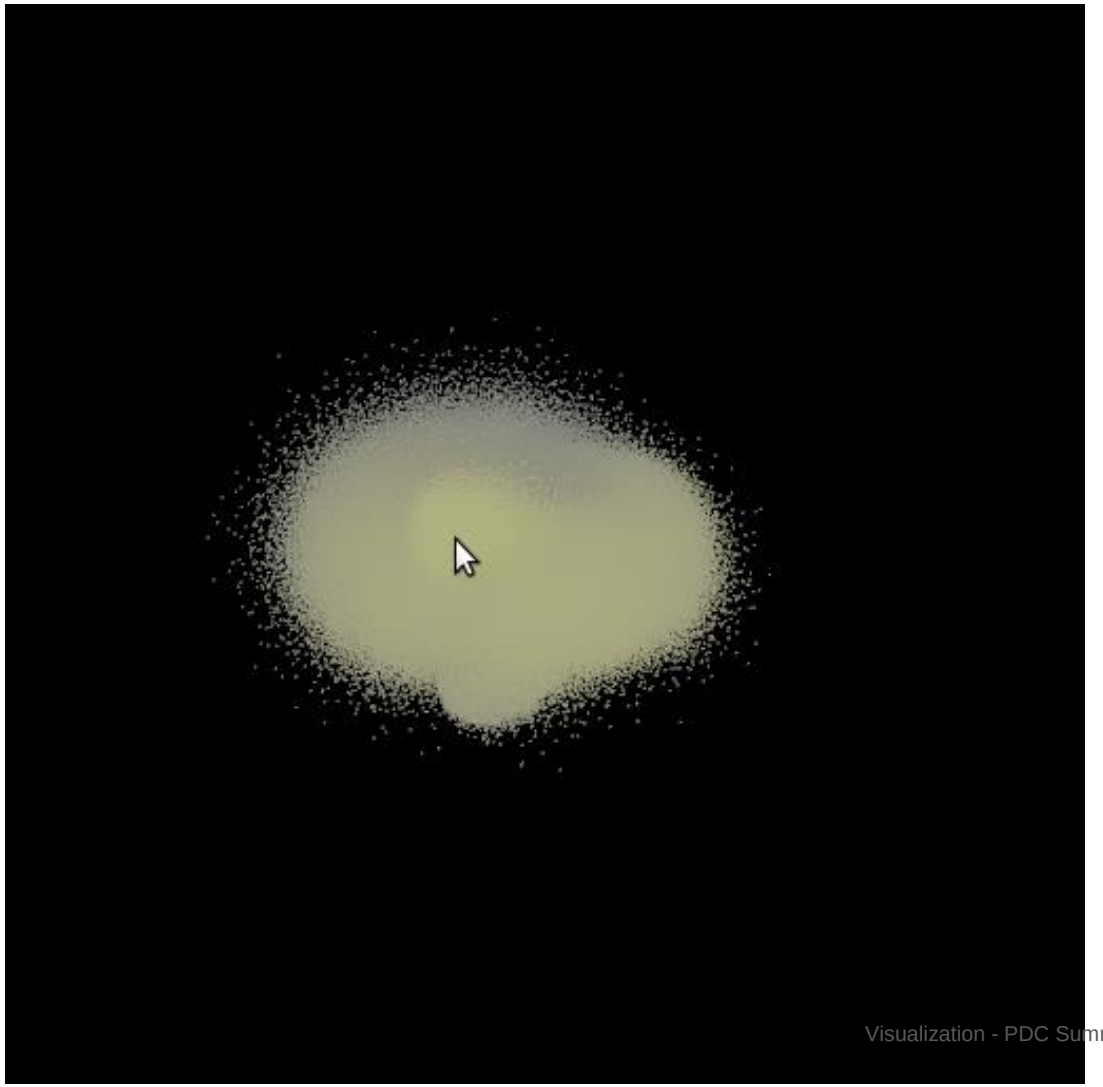
Multi-core nodes

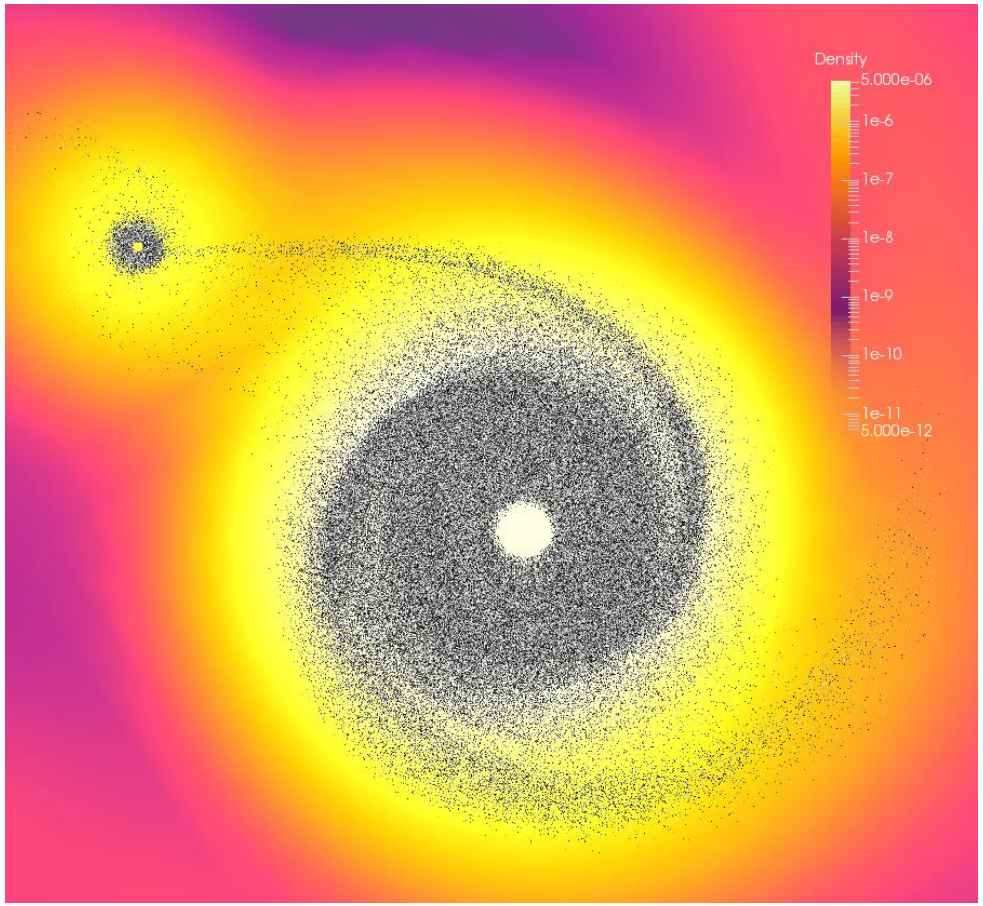
- In ParaView, many filters, will, by default, take advantage of some form of multi processing

- See also the Accelerated Algorithms

Example 1: SPH Particle clouds to slices

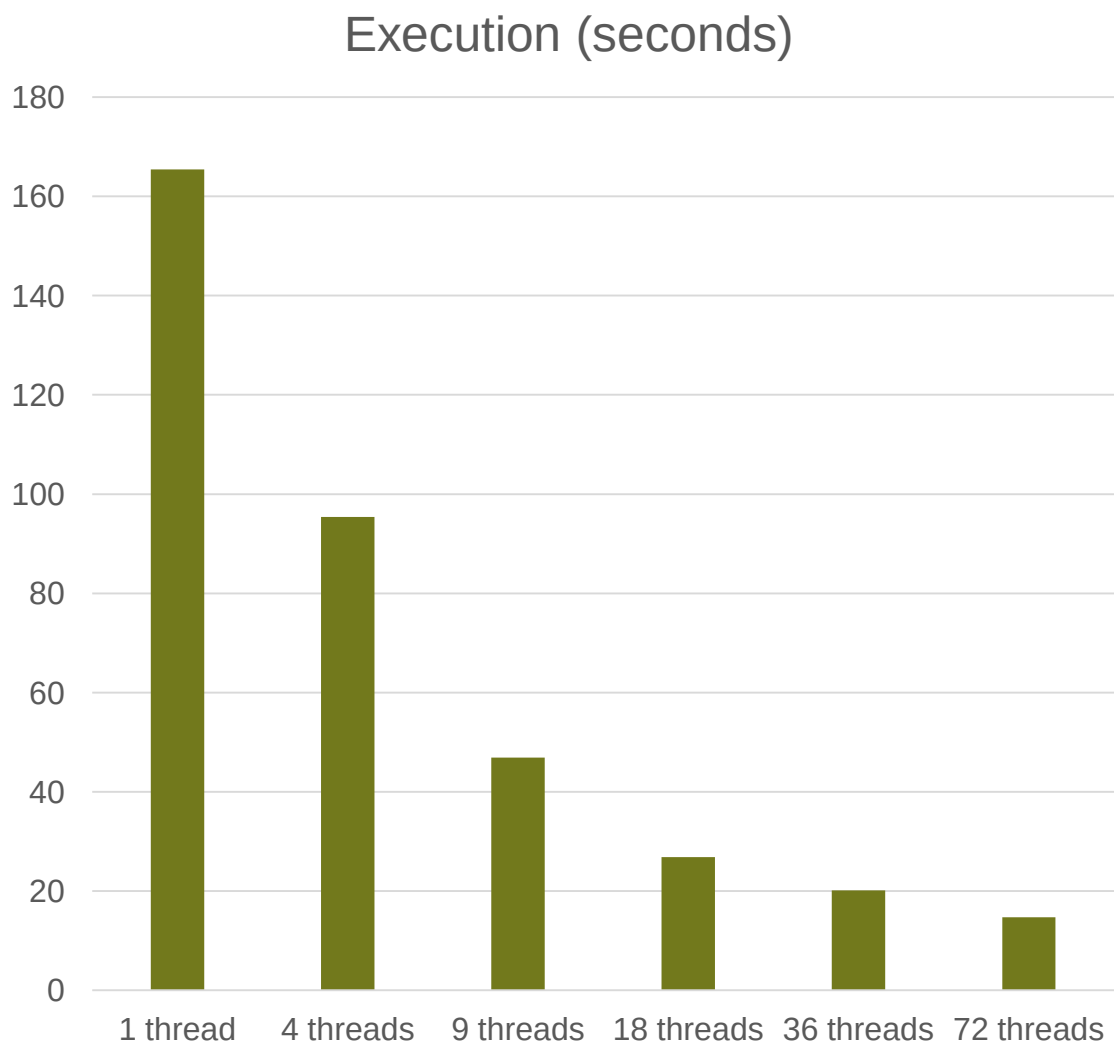




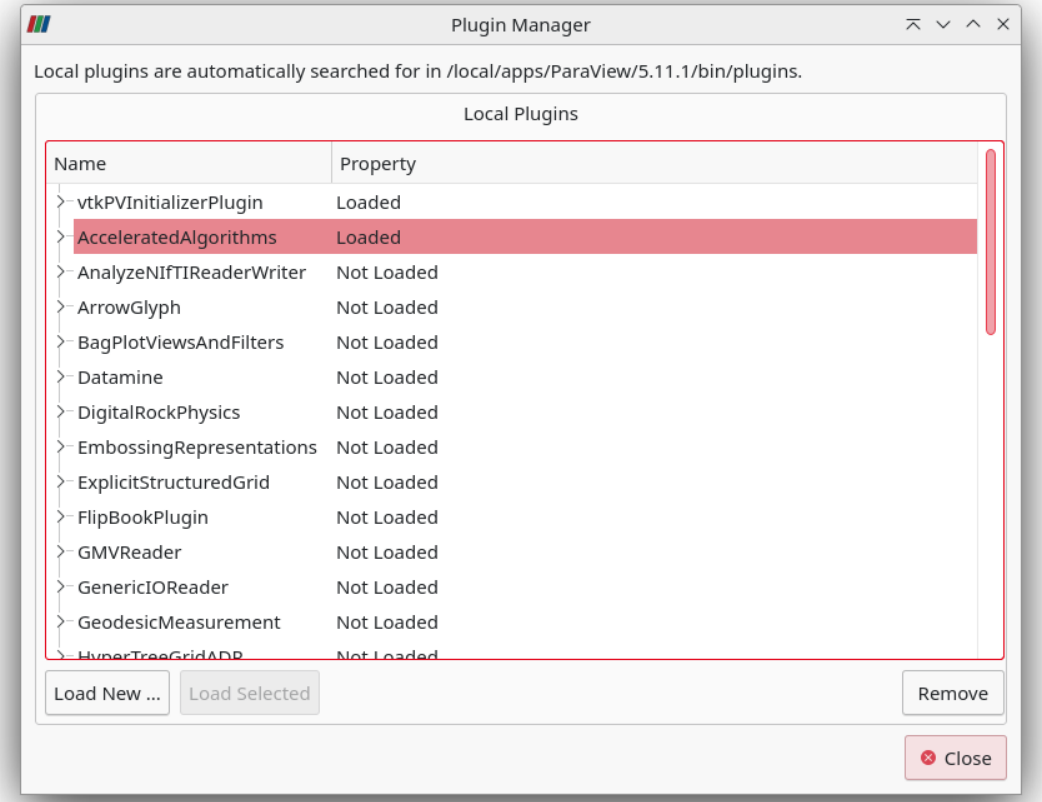


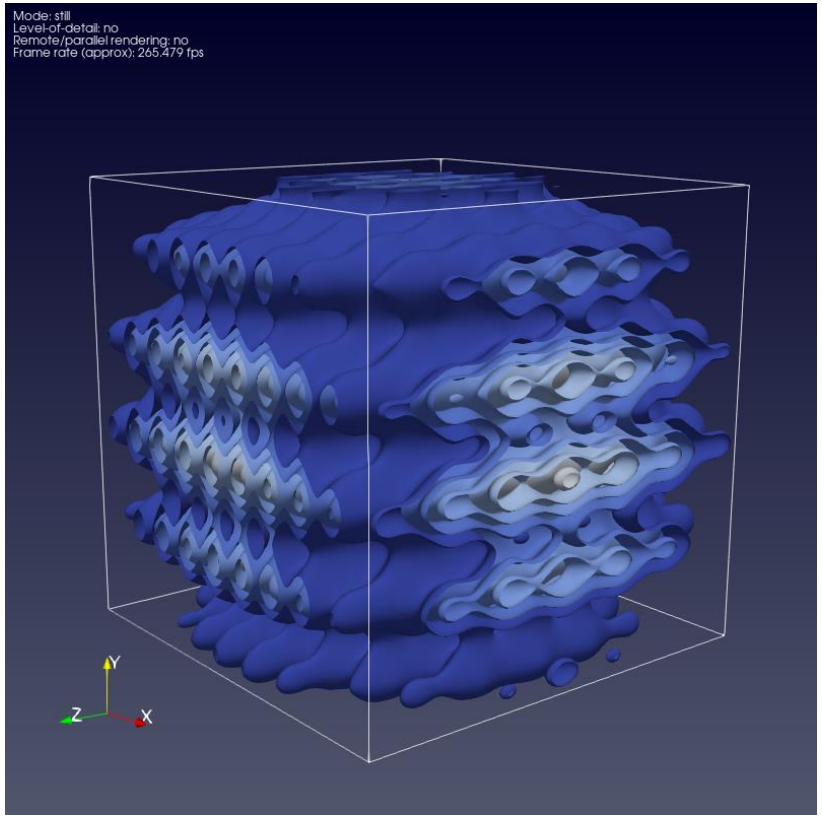
SPH data interpolation

- The vtkSPHInterpolator filter uses SPH (smooth particle hydrodynamics) kernels to interpolate a data source onto an input structure.
- A Point locator is a crucial part of the execution path, to accelerate queries about points and their neighbors.
- SMP-based acceleration provides a fast plane interpolation using parallelism-on-the-node with Intel TBB.
- Piz Daint Compute node: dual socket Intel® Xeon® E5-2695 v4 @ 2.10GHz (18 cores)



Exercise/demonstration: Contour or FlyingEdges





time pvbatch pvTwoContours.py -c flyingedge

real 0m4.898s
user 0m10.301s
sys 0m1.355s

time pvbatch pvTwoContours.py

real 0m14.251s
user 0m14.532s
sys 0m1.208s

VTK-m

- This is a toolkit of scientific visualization algorithms for emerging processor architectures (GPUs and coprocessors). VTK-m is designed for fine-grained concurrency and provides abstract data and execution models that can be utilized to construct a variety of scalable algorithms.
- Blog article

MPI-based parallel data visualisation

MPI-based parallelism

1. pvbatch
2. paraview + N pvserver

You may test simple things on your desktop, and later move to the Dardel supercomputer

The self-directed tutorial has collected together the best recipes to learn.

```
mpiexec –n 4 pvbatch pvScript.py
srun pvbatch pvScript.py
```

```
mpiexec –n 4 pvserver &

paraview –server=localhost pvscript.py
```

Remote and parallel visualization

- Reference manual

Did you know?

Remote and parallel processing are often used together, but they refer to different concepts, and it is possible to have one without the other.

In the case of ParaView, remote processing refers to the concept of having a client, typically `paraview` or `pvpython`, connecting to a `pvserver`, which could be running on a different, remote machine. All the data processing and, potentially, the rendering can happen on the `pvserver`. The client drives the visualization process by building the visualization pipeline and viewing the generated results.

Parallel processing refers to a concept where instead of single core — which we call a `rank` — processing the entire dataset, we split the dataset among multiple ranks. Typically, an instance of `pvserver` runs in parallel on more than one rank. If a client is connected to a server that runs in parallel, we are using both remote and parallel processing.

In the case of `pvbatch`, we have an application that operates in parallel but without a client connection. This is a case of parallel processing without remote processing.

Remote and parallel visualization … for big data

- How big is “big”?
- Structured grids are lighter
- Unstructured grids store their topology explicity
- Beware that often, filters in ParaView will change the topology in some way, and these filters will write out the data as an unstructured grid
- Must read:
  - Avoid Data Explosion
  - Culling Data

Data Parallelism/partitioning

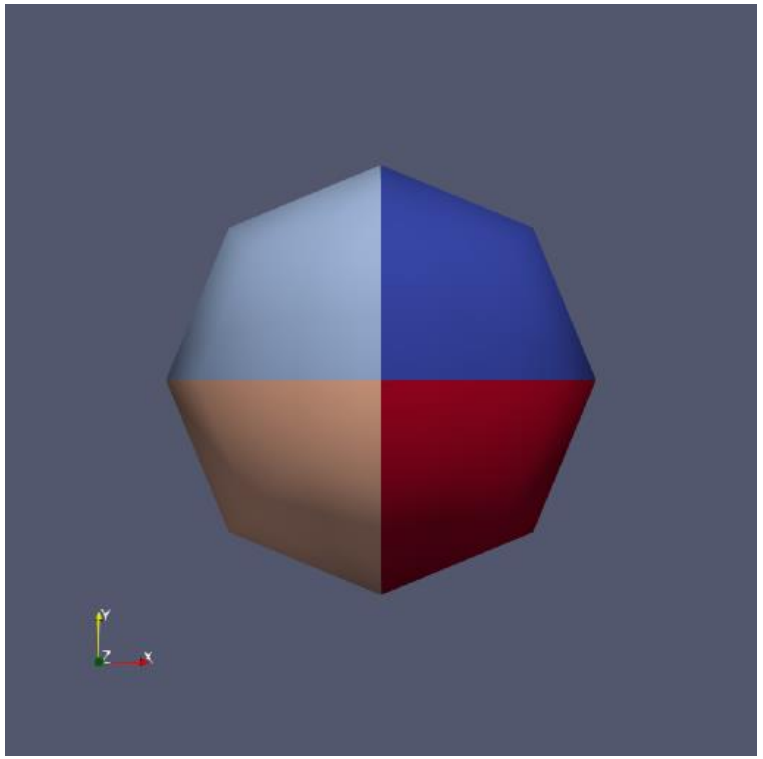
- So you want to do “real” parallelism, and split your data among multiple nodes.

  See also your lecture notes in the “introduction to MPI, pages 26-28”

- Now, to the details about splitting data...

“Hello Sphere” in parallel

```
from paraview.simple import *
nbprocs = servermanager.ActiveConnection.GetNumberOfDataPartitions()
Sphere() # a synthetic data generator
pid = ProcessIdScalars()
rep = Show()
# to scale for any number of processors.
rep.RescaleTransferFunctionToDataRange(False, True)
# to save an image
SaveScreenshot(format("sphere.procs=%02d.png" % nbprocs))
```

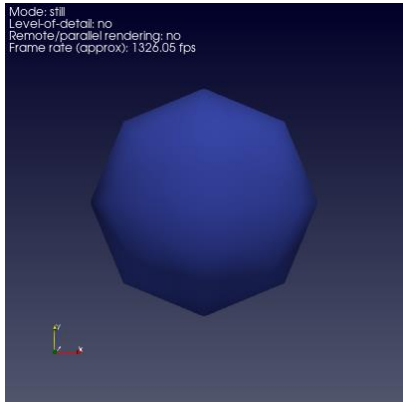


Can be executed *locally* with: **“mpiexec –n 4 ~~paraview~~ pvSphere.py”**

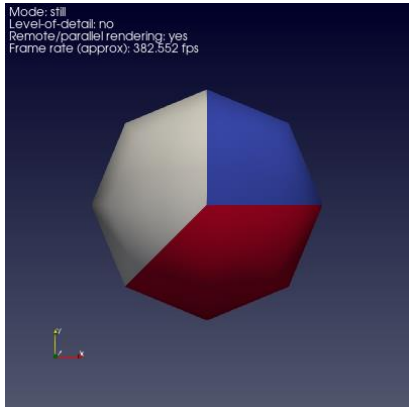
“mpiexec –n 4 pvbatch pvSphere.py”

Exercise/demonstration:

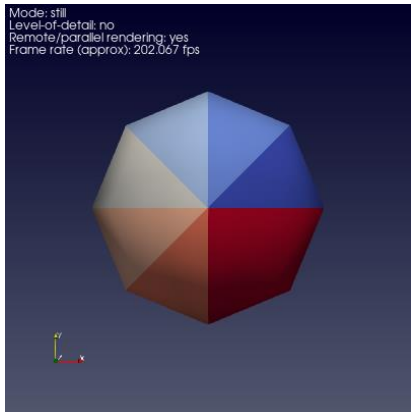
- pvbatch pvSphere.py



- mpiexec -n 3 pvbatch pvSphere.py



- mpiexec -n 7 pvbatch pvSphere.py



The Sphere Source example is parallel-aware

- What about your data-reader?

- Please. Plan ahead of time. Test. Ask for help

parallel-aware data readers. Survey Time!

Not all data readers are capable of reading data in parallel, and splitting (load balancing) their data amongst multiple servers

How many of you think that ParaView can read, in parallel:

- An OpenFoam dataset?
- A regular cartesian grid for Volume Rendering? (*.mhd)
- A regular cartesian grid using VTK XML vtkImageData format? (*.vti)
- A grid stored using VTK XML vtkUnstructuredGrid format? (*.vtu)

Details about the MetaIO data reader

- https://www.paraview.org/Wiki/ITK/MetaIO

ObjectType = Image

NDims = 3

BinaryData = True

BinaryDataByteOrderMSB = False

CompressedData = False

ElementSpacing = 0.5625 0.5625 1

DimSize = 2048 1024 512

ElementType = MET_FLOAT

ElementDataFile = volume.raw

Live Demonstration

A quick fix

```
from paraview.simple import *
f = MetaFileSeriesReader(FileNames=["volume.mhd"])
f.UpdatePipeline()
SaveData("volume.vti")
```

Live Demonstration

parallel-aware data readers

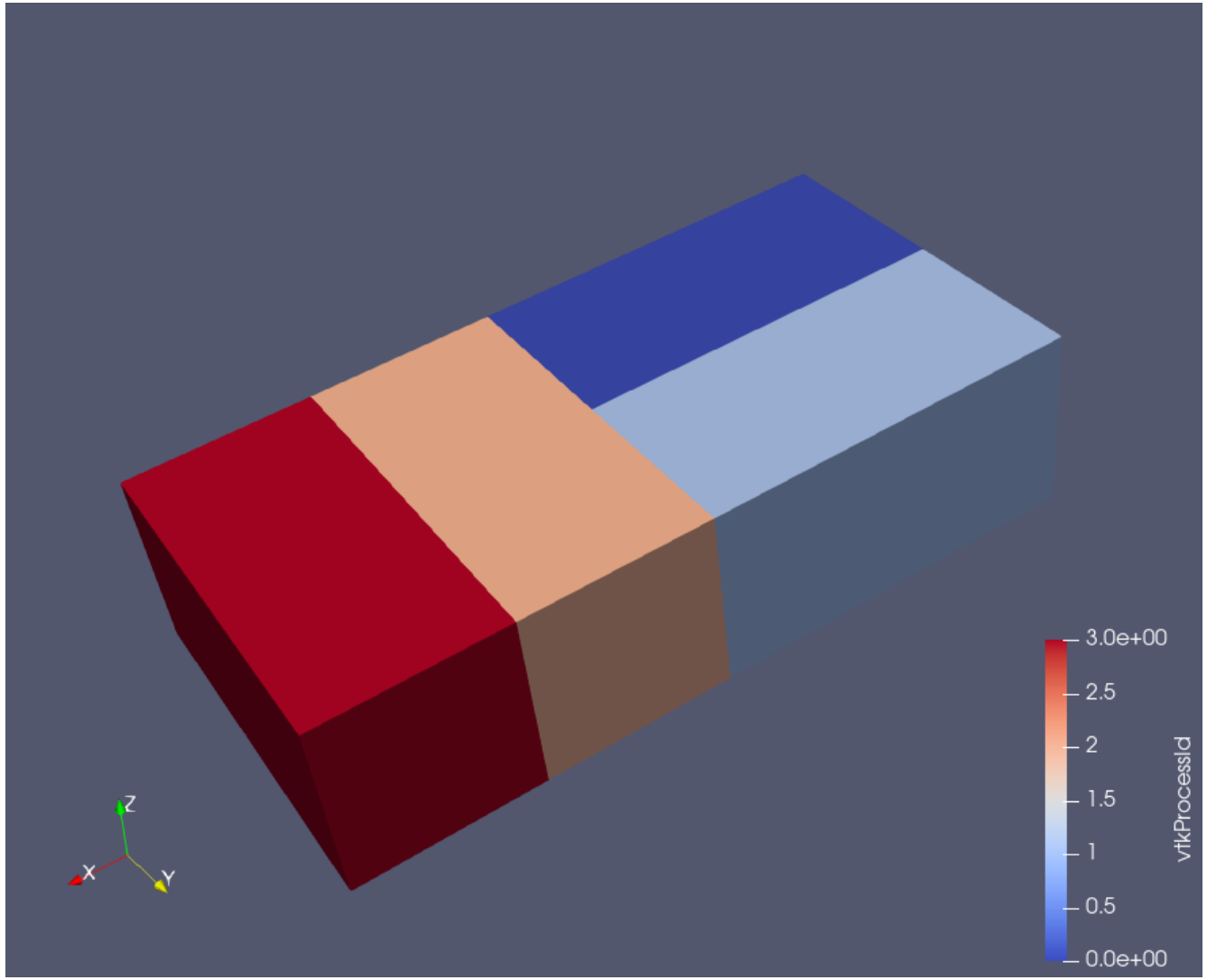
- An OpenFoam dataset in parallel? NO
- A regular cartesian grid for Vol Rendering? (*.mhd) NO
- A regular cartesian grid using VTK XML vtkImageData format? (*.vti) YES
- A grid stored using VTK XML vtkUnstructuredGrid format? (*.vtu) NO

Data partitioning. Use a data format that supports it

By the way, how did ParaView do that?

What a strange way to split the data!

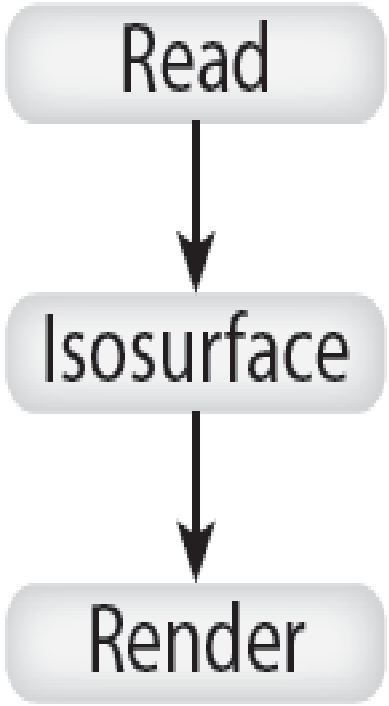
We now need notions about the “Visualization Pipeline”



Visualization Pipeline: Introduction

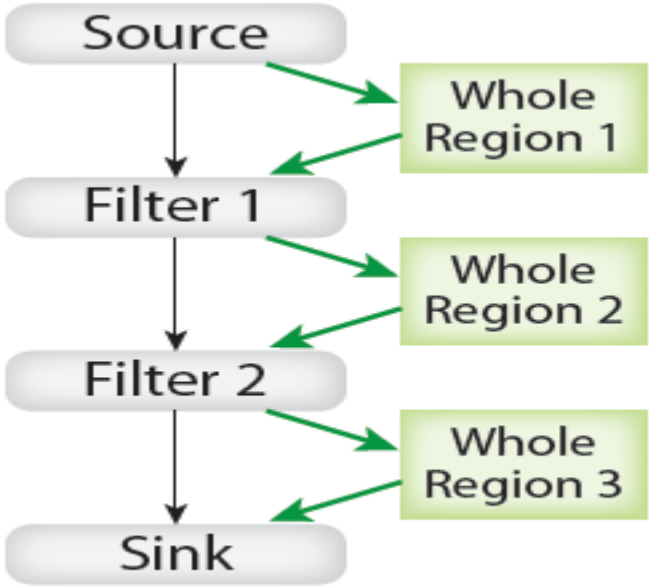
From a survey article by Ken Moreland, IEEE Transactions on Visualizations and Computer Graphics, vol 19. no 3, March 2013

«A visualization pipeline embodies a *dataflow network* in which computation is described as a collection of executable *modules* that are connected in a directed graph representing how data moves between modules. There are threee types of modules: *sources*, *filters* and *sinks*.»

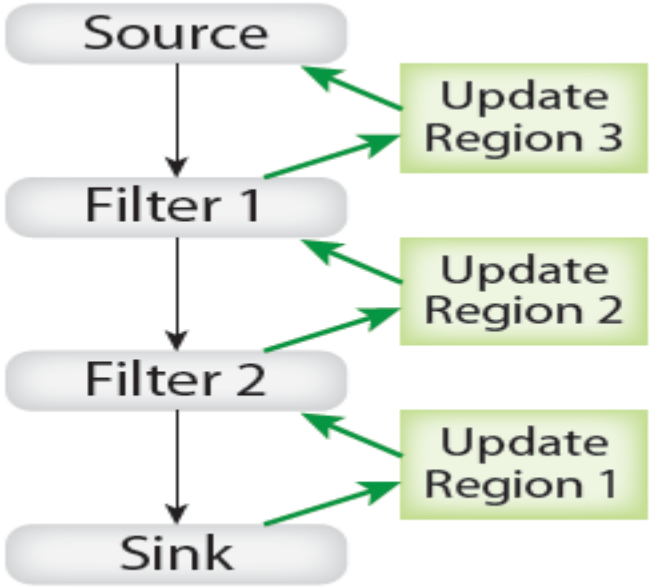


Visualization Pipeline: Metadata

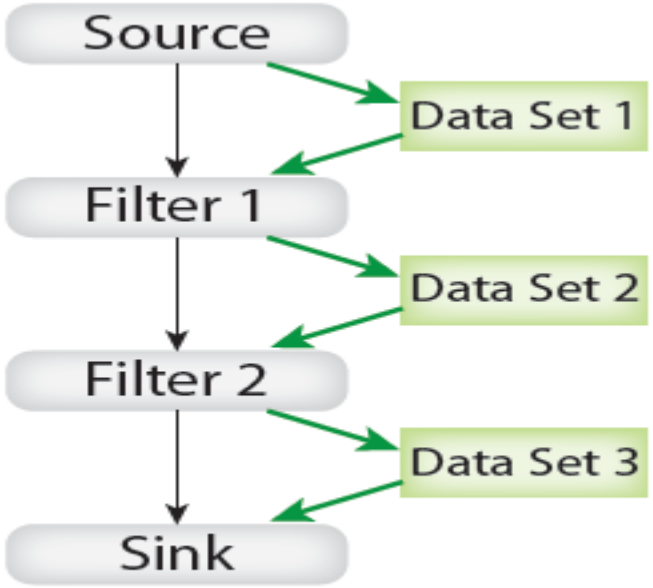
- 1st pass: Sources describe the region they can generate.
- 2nd pass: The application decides which region the sink should process.
- 3rd pass: The actual data flow through the pipeline



(a) Update Information



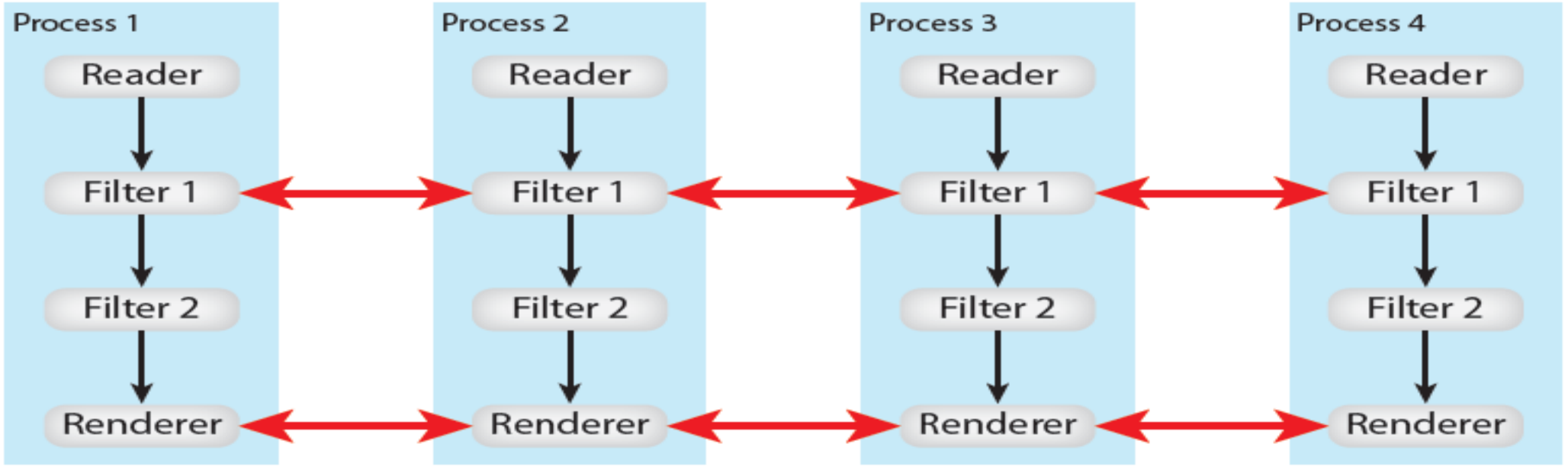
(b) Update Region



(c) Update Data

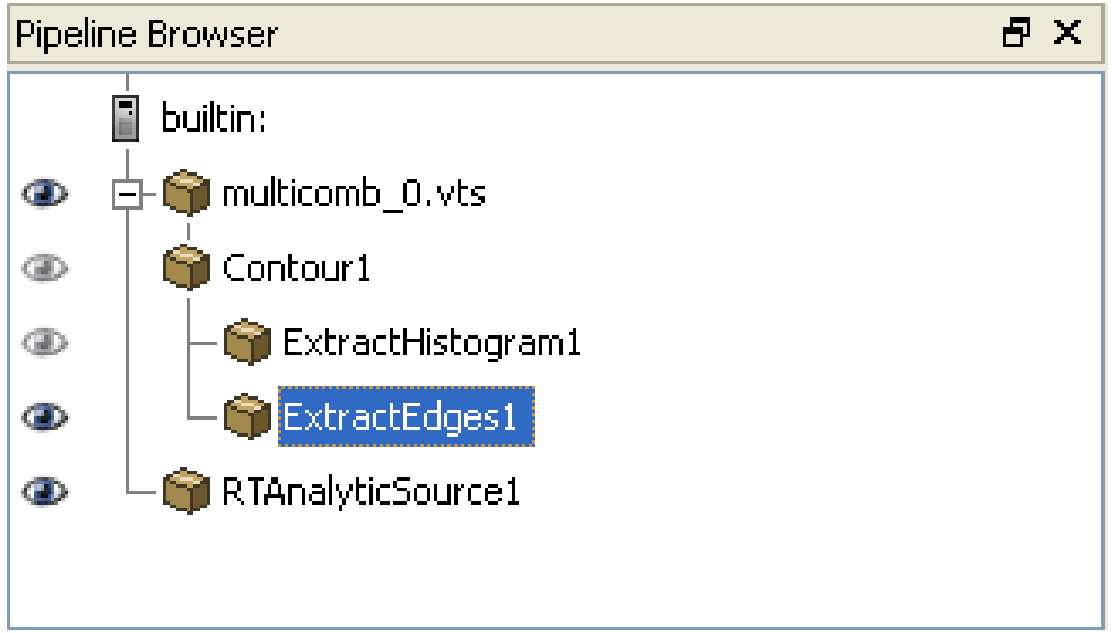
Visualization Pipeline: Data Parallelism

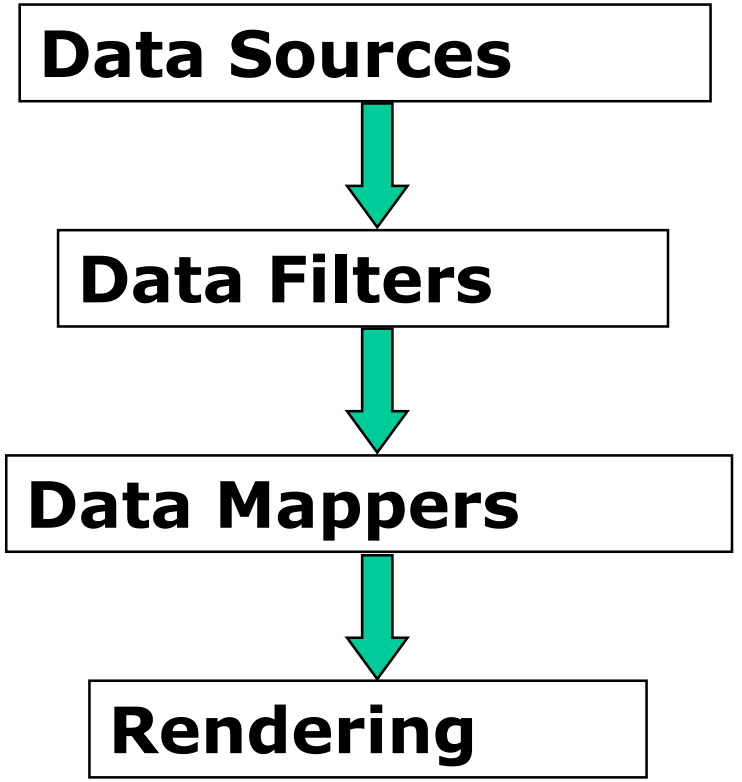
- Data parallelism partitions the input data into a set number of pieces, and replicates the pipeline for each piece.
- Some filters will have to exchange information (e.g. GhostCellGenerator)



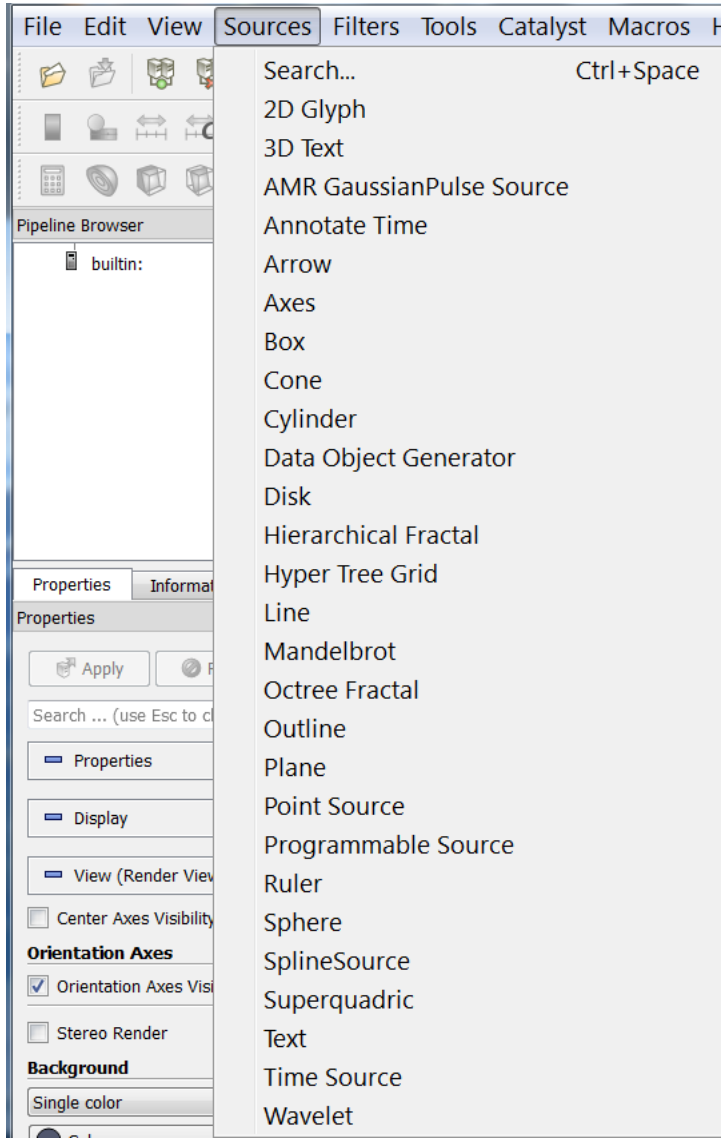
The VTK visualization pipeline

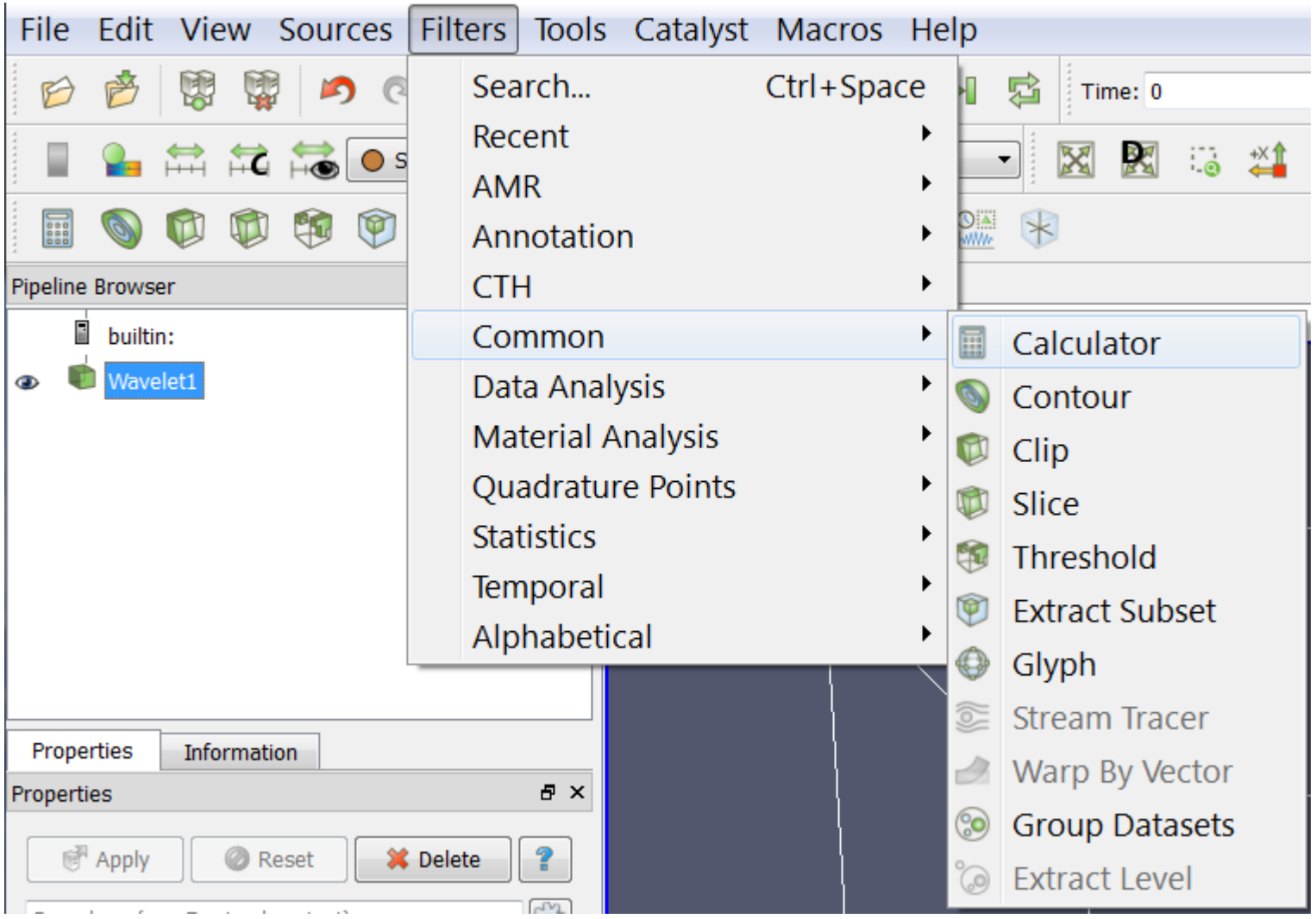
VTK's main execution paradigm is the *data-flow*, i.e. the concept of a downstream flow of data





Examples of Filters/Sources

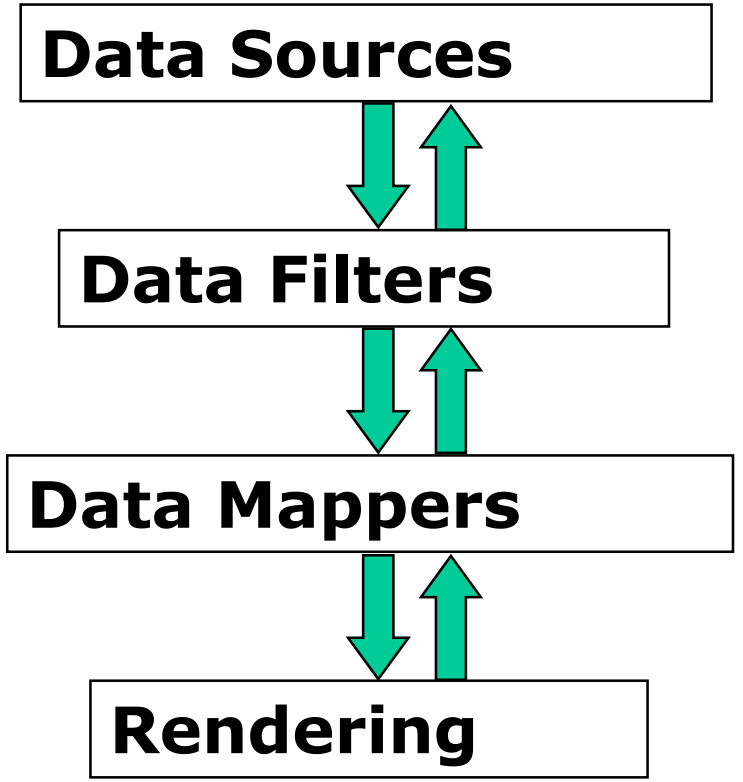




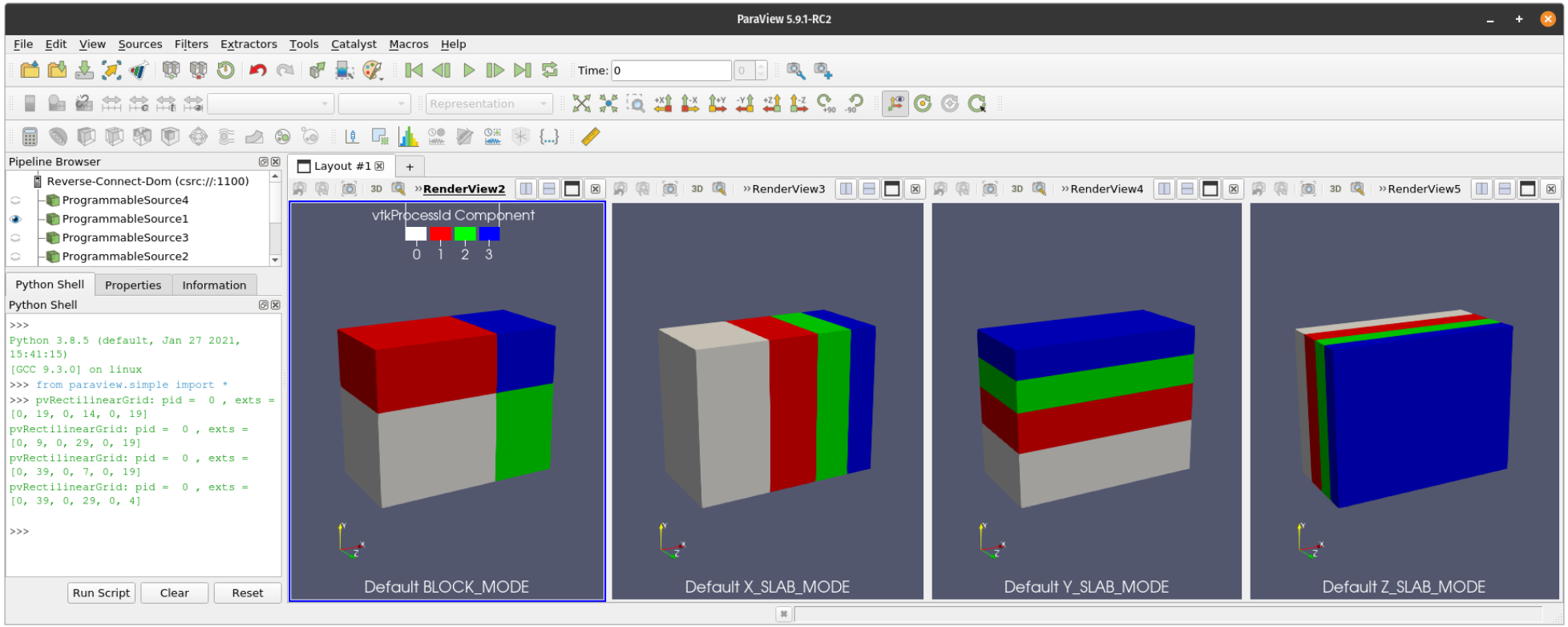
The VTK visualization pipeline

- VTK extends the *data-flow* paradigm
- VTK acts as an *event-flow* environment, where data flow downstream and events (or information) flow upstream
- ParaView's Rendering **triggers** the execution:

```
view = GetRenderView()
view.ViewTime = 5
Render()
```



Data partitioning



There are 4 ways to split structured data (by block, or along any of the I,J, K direction)

But ParaView only gives you one, by default. See *pvDistributedGrid.py*

There are also other ways to split data…

Think about grid-less particles, AMR or multi-block data

Does it matter? Should you feel concerned? ☺

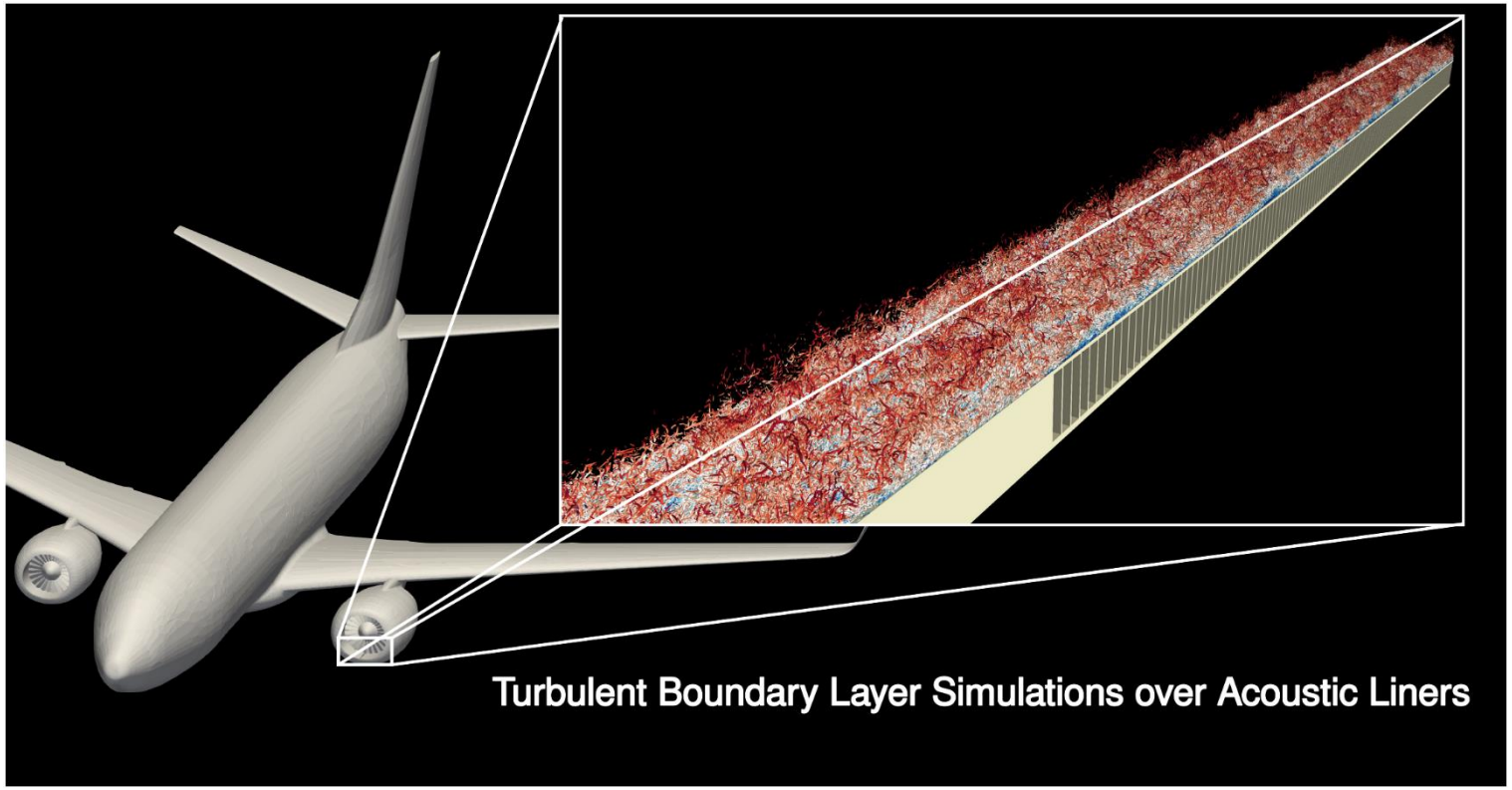
See our fluid flow visualization at the 2022 APS Division of Fluid Dynamics' Gallery of Fluid Motion "Turbulent Boundary Layer over Acoustic Liners"

The grid resolution used:

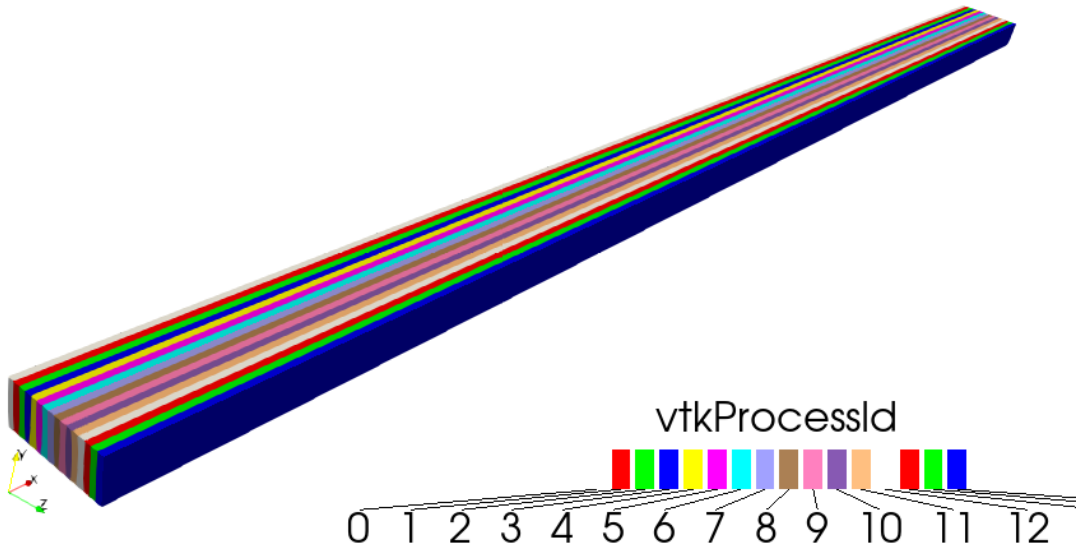
(Nx, Ny, Nz) = [21504, 448, 1120]

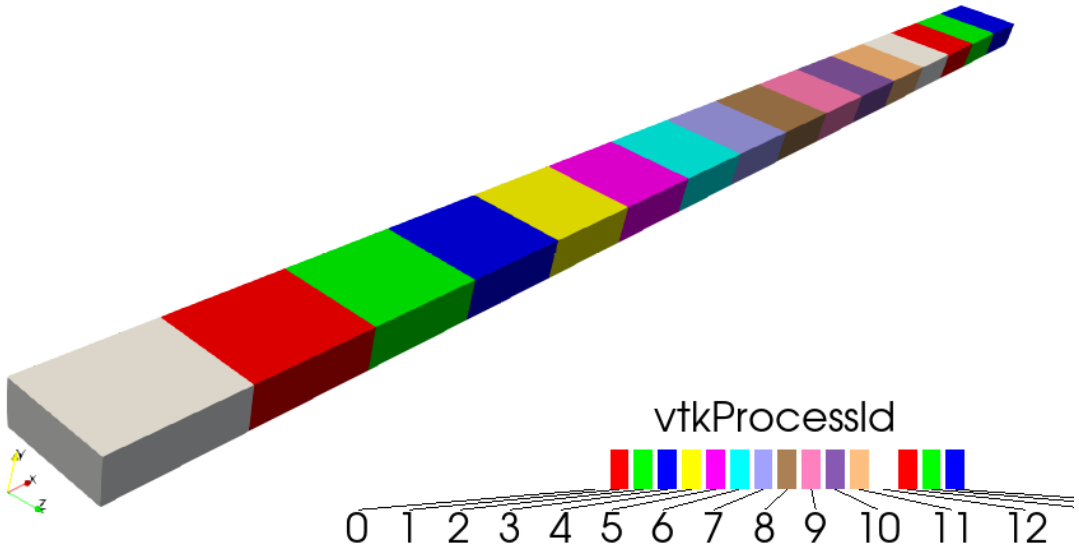
ParaView's default partitioning would split data along the X axis to get a "*globally balanced*" partitioning =>

Disastrous performance for I/O because the natural, "contiguous" ordering of bytes on disk follows the X,Y, Z ordering.



Does it matter? Should you feel concerned? ☺





The Good, ……and……..……The Ugly default!

Exercise/demonstration:

Locally (on your desktop):

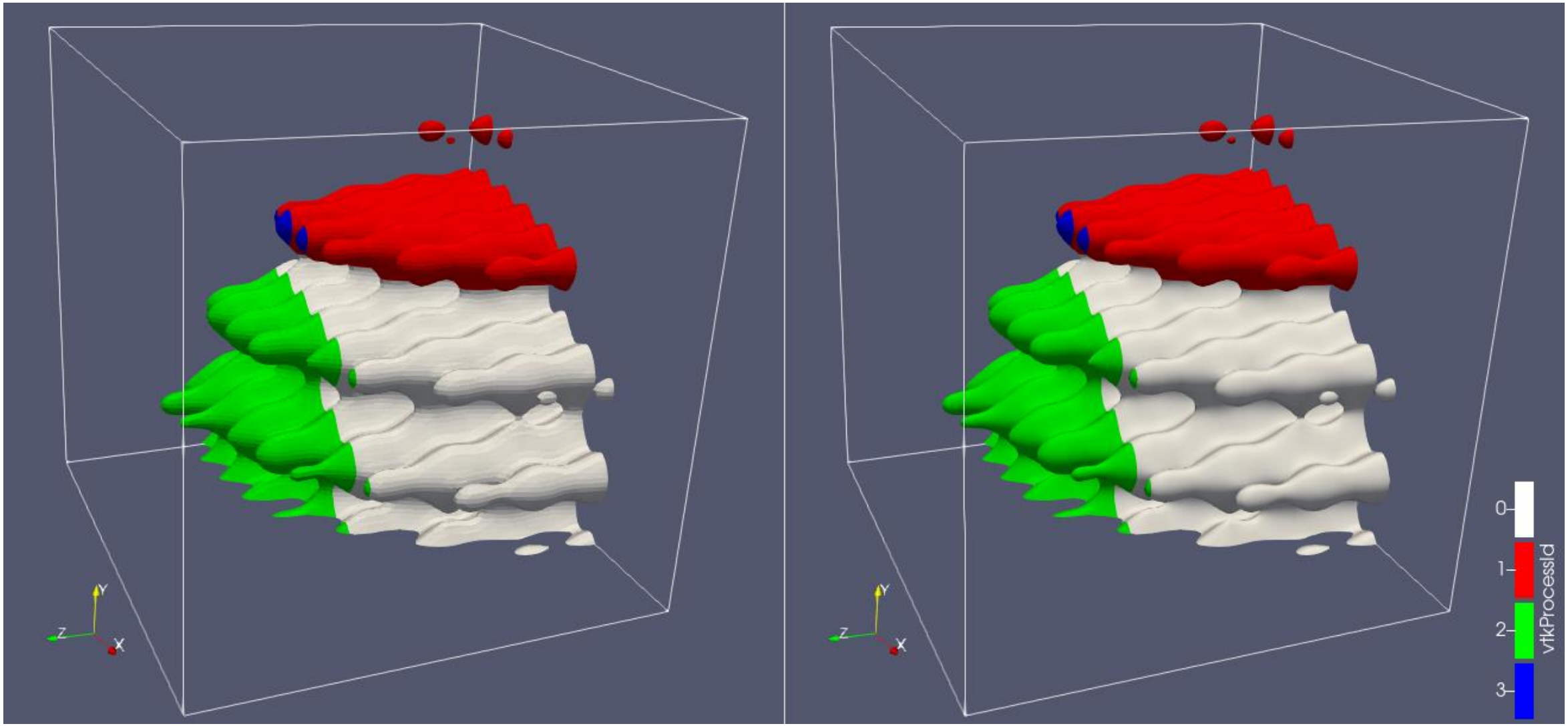
```
mpiexec -n 4 pvserver &
paraview --server=localhost  pvDistributedGrid.py
```

Edit the programmable source RequestInformation script to change the resolution of the grid (line 77) and re-run as many times as necessary to understand what is happening, and how it might apply to your own simulation data.

MPI-parallelism

- Transparent to the user
- Python scripts or interactive visualization should work without changes
- Beware of your I/O patterns and refer to the discussion about Data Extents and ghost-cells

Ghost-cells?



When the computation of ghost-cells is triggered by a filter

See *pvMakeGhostCells*.py*

- To do a better surface shading, we must use *Surface Normals* (right-side image)
- To compute *Normals*, we average normal vector of all vertices of a polygonal primitive
- What about when we are on the edge…?
- Typical of finite difference computational stencils. We ask neighbors for ghost cells.

- Implication => we most likely will re-read the data from disk with updated extents (the split indices)

Exercise/demonstration:

Locally (on your desktop):

```
mpiexec -n 4 pvserver &
paraview --server=localhost  pvMakeGhostCells0.py
```

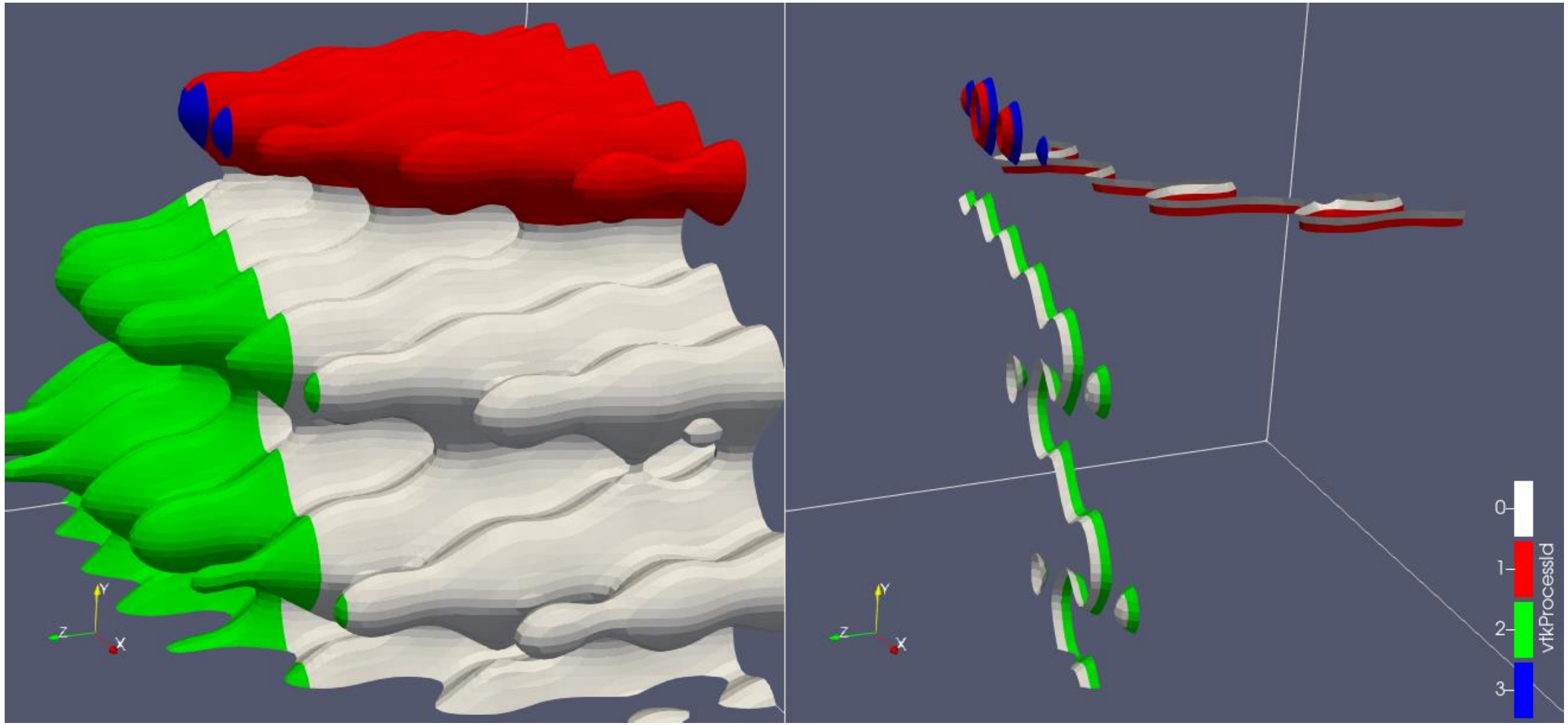
On Dardel:

```
module load  PDC/22.06 paraview/5.11.1
salloc -n 4 -t 10 -p main -A edu23.summer
srun –n 4 pvserver
```

<= execute on Dardel

on desktop => `paraview --server=Dardel --connect-id ?????  pvMakeGhostCells0.py`

Ghost-cells?



Exercise/demonstration:

Locally (on your desktop):

```
mpiexec -n 4 pvbatch pvMakeGhostCellsLog.py | grep "Execute Wavelet1 id"

mpiexec -n 4 pvbatch pvMakeGhostCellsLog.py --ghostcells|grep "Execute Wavelet1 id"
```

```
Execute Wavelet1 id: 300,  0.002962 seconds
Execute Wavelet1 id: 300,  0.004436 seconds
Execute Wavelet1 id: 300,  0.003002 seconds
Execute Wavelet1 id: 300,  0.003086 seconds
```

Does it matter?

- So, I have told you that using an MPI-based ParaView session is transparent to the user (no special tricks needed, no dependence on the rank and size of the MPI job).

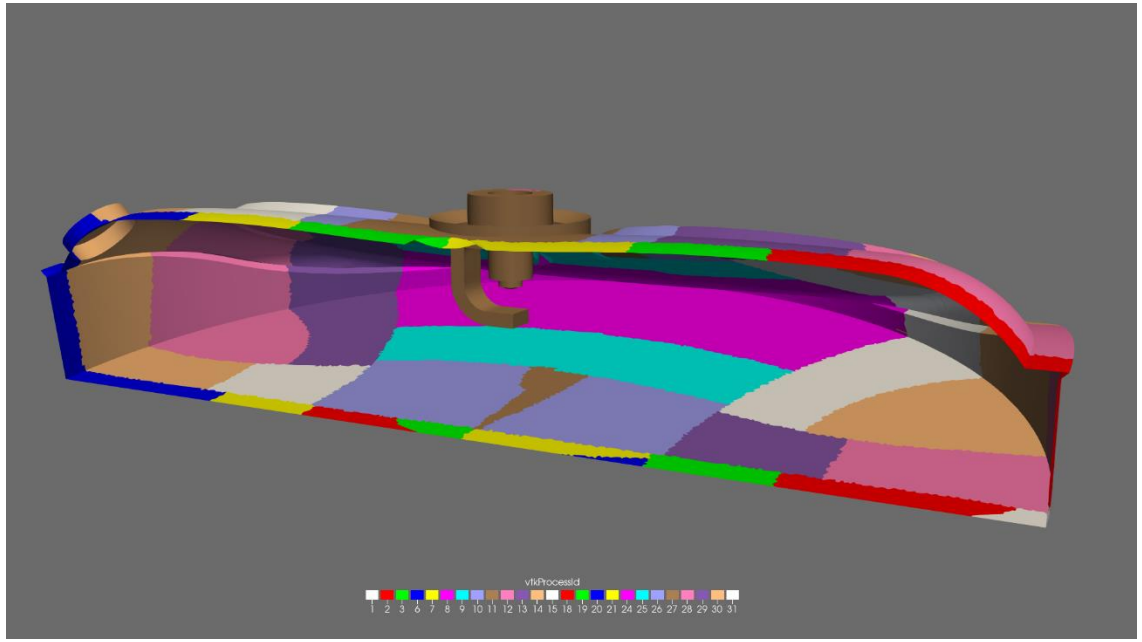
- So what? The Python script gets executed twice and we got the right image.

- What if reading the data involved 100 Gb and 1024 MPI tasks? ☺

Ghost Cells Generator

If your unstructured grid data is already partitioned satisfactorily but does not have ghost cells, it is possible to generate them using the Ghost Cells Generator filter.

For example, the ParaView NEK5000 file format reader, needs it to generate a surface geometry of a model, without the internal boundaries between spectral elements.

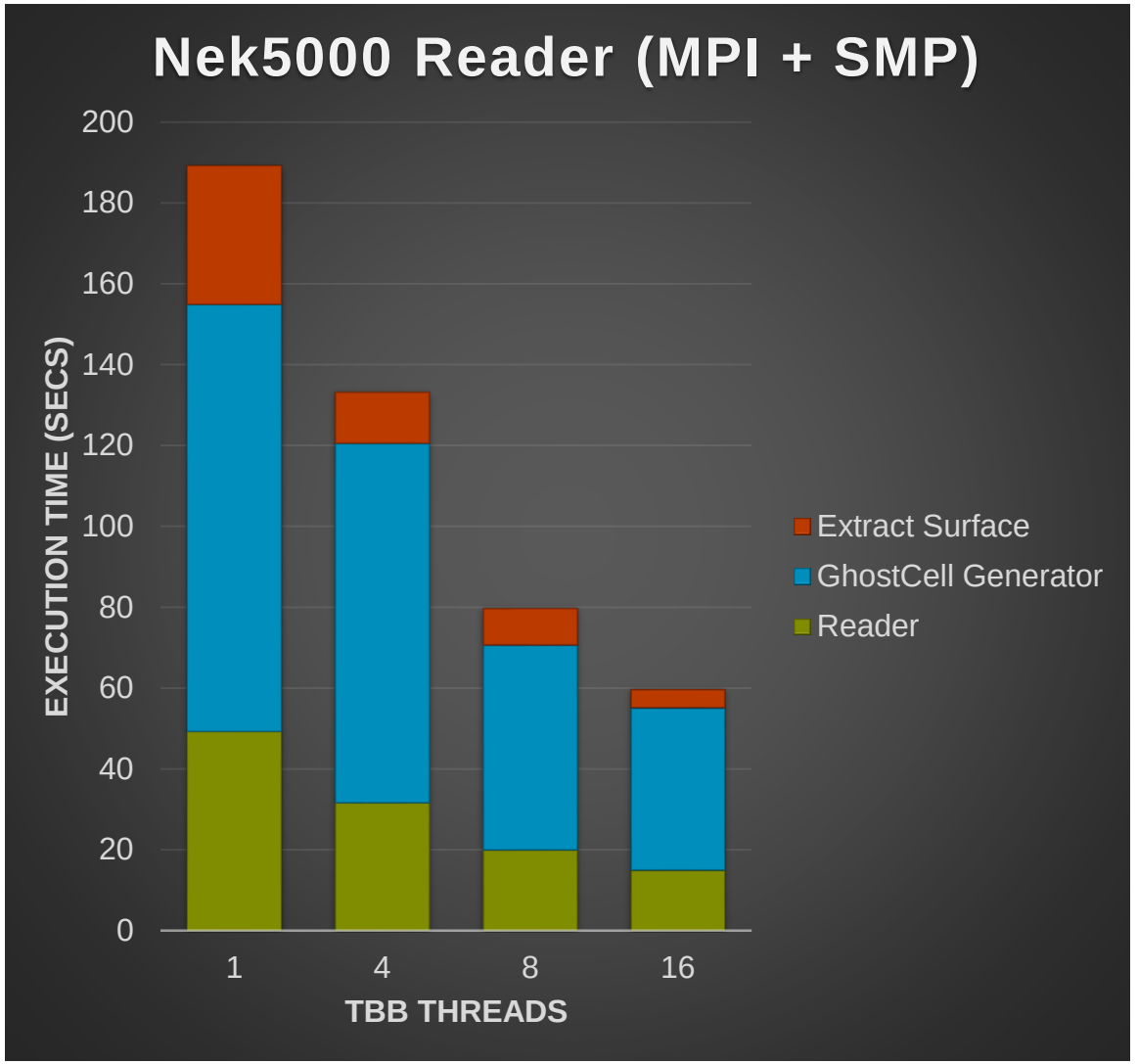


Mixed mode parallelism (SMP on the node and MPI across nodes)

Reading and using Nek5000 data

- The reading part is (of course) very dependent on disk I/O (MPI-based)
- A small part of reading the data is taken by the use of a vtkStaticCleanUnstructuredGrid to merge duplicate points (SMP-based)
- Extracting the solid geometry is done in two phases:
  - A Ghost-cell generator (SMP-based)
  - a surface extractor (SMP-based)

- Tests were run with a dataset made of 4,3 millions spectral elements of order 10, resulting in the generation of 3.2 billions hexahedra. Parallel reading is done 4 nodes, 32 MPI tasks and a maximum of 16 threads per task on an AMD EPYC 7742 64-Core single socket



Examples of hybrid parallelism:

- The DigitalRockPhysics plugin introduced in ParaView v 5.5. Release Notes
- The NEK5000 parallel reader introduced in ParaView v 5.11. Release notes

MPI-based parallel visualization on Dardel

Instructions from the web site suggest to allocate an interactive job with 128 cores

- salloc -n 128 -t 60 -p main -A edu23.summer

- This would allocate an MPI job with 128 tasks. A big overhead for our exercises…

I suggest using 8 MPI tasks only

- salloc -n 8 -t 10 -p main -A edu23.summer

Time parallelism

Embarassingly parallel visualization in the Time domain

- Very often, we have simulations producing many timesteps, where the visualization of each timestep can take a long time, but is independent of the other timesteps.

- You can split (or distribute) the processing load over multiple compute nodes

- man sbatch (and look for the --array option)
  - #SBATCH --array=0-299:25
  - pass the index $SLURM_ARRAY_TASK_ID to the ParaView Python script
  - srun pvbatch -- pvScript.py --frames ${SLURM_ARRAY_TASK_ID}
  - squeue –u$USER

pvScript.py

```
import argparse


parser = argparse.ArgumentParser()
parser.add_argument('--frames', required=True, type=int)
args, unknown = parser.parse_known_args()

start = args.frames



SaveAnimation("sequence.png", FrameWindow = [start, start+24])
```

Exercise/demonstration:

Write a SLURM job to write an animation of the Surface display of the Transient Double Gyre

split the computation of 300 timesteps by chunks of 25

Hints:

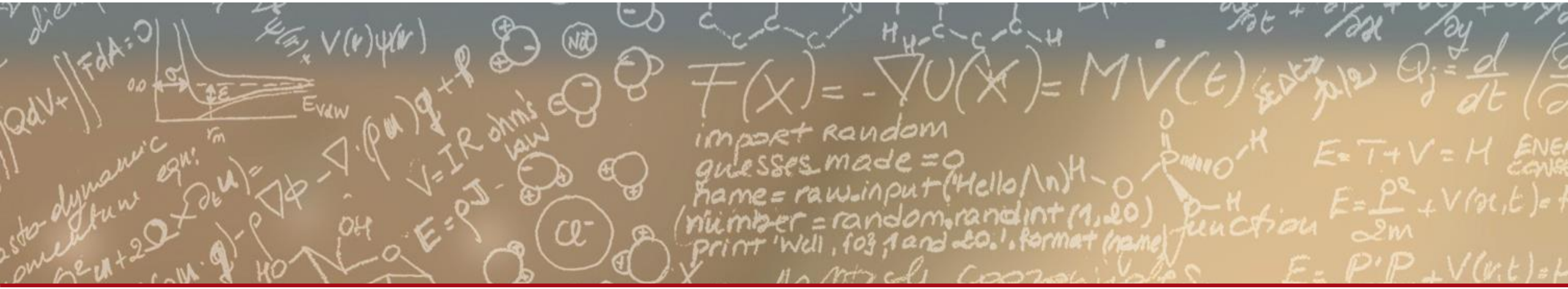
- renderView1.ViewSize = [1280, 720]
- SaveAnimation("LIC.png", ImageResolution=[1280, 720], FrameWindow=[start, start+24])

Add a time label

help(SaveAnimation)

Notice that filenames will be automatically numbered

ffmpeg -i LIC.%04d.png -c:v libx265 -tag:v hvc1 movie-file.mp4



Thank you for your attention.