University of Sheffield

Visualising Code_Saturne using `pyvista` and Python

Matthew Falcone

University of Sheffield, UK

8 May 2024

Contents

What post-processing data does Code_Saturne produce?

- Monitoring points:
  - `monitoring/*.csv` or `monitoring/*.dat` files.
  - Comes from probes specified in the input file or through user subroutine
  - Easy to process in Matlab or Python
- Surface or volume data
  - By default, boundary faces and interior cells are outputted with associated arrays
    - Boundary temperature, y^+, etc.
    - Velocity, pressure, etc.
  - By default saved in the Ensight Gold file format.
    - `postprocessing/*.case`
  - Others are available: MED, CGNS etc.
- In this presentation, we focus on processing the `.case` files.

What post-processing data does Code_Saturne produce?

- Monitoring points:
  - `monitoring/*.csv` or `monitoring/*.dat` files.
  - Comes from probes specified in the input file or through user subroutine
  - Easy to process in Matlab or Python
- Surface or volume data
  - By default, boundary faces and interior cells are outputted with associated arrays
    - Boundary temperature, y^+, etc.
    - Velocity, pressure, etc.
  - By default saved in the Ensight Gold file format.
    - `postprocessing/*.case`
  - Others are available: MED, CGNS etc.
- In this presentation, we focus on processing the `.case` files.

What post-processing data does Code_Saturne produce?

- Monitoring points:
  - `monitoring/*.csv` or `monitoring/*.dat` files.
  - Comes from probes specified in the input file or through user subroutine
  - Easy to process in Matlab or Python
- Surface or volume data
  - By default, boundary faces and interior cells are outputted with associated arrays
    - Boundary temperature, y^+, etc.
    - Velocity, pressure, etc.
  - By default saved in the Ensight Gold file format.
    - `postprocessing/*.case`
  - Others are available: MED, CGNS etc.
- In this presentation, we focus on processing the `.case` files.

Paraview: the conventional post-processing pipeline

- The `.case` files are usually visualised using paraview
- Widely-used visualisation package with a lot of functionality
  - GUI
  - Server-client mode: `pvserver`
    - Run on HPC with MPI, visualise on local machine
    - See ARCHER2 documentation
  - In-situ visualisation with Catalyst
  - Python scripting (`pvpython` or `pvbatch`)
    - 'Start trace'

Paraview: the conventional post-processing pipeline

- The `.case` files are usually visualised using paraview
- Widely-used visualisation package with a lot of functionality
  - GUI
  - Server-client mode: `pvserver`
    - Run on HPC with MPI, visualise on local machine
    - See ARCHER2 documentation
  - In-situ visualisation with Catalyst
  - Python scripting (`pvpython` or `pvbatch`)
    - 'Start trace'

Paraview: the conventional post-processing pipeline

- The `.case` files are usually visualised using paraview
- Widely-used visualisation package with a lot of functionality
  - GUI
  - Server-client mode: `pvserver`
    - Run on HPC with MPI, visualise on local machine
    - See ARCHER2 documentation
  - In-situ visualisation with Catalyst
  - Python scripting (`pvpython` or `pvbatch`)
    - 'Start trace'

Paraview: the conventional post-processing pipeline

- The `.case` files are usually visualised using paraview
- Widely-used visualisation package with a lot of functionality
  - GUI
  - Server-client mode: `pvserver`
    - Run on HPC with MPI, visualise on local machine
    - See ARCHER2 documentation
  - In-situ visualisation with Catalyst
  - Python scripting (`pvpython` or `pvbatch`)
    - 'Start trace'

VTK: Visualization Toolkit

- Paraview and Pyvista are based on the VTK.
- C++ library with interfaces for python and other languages.
- Python wrapper of VTK is nearly identical to the C++ functions and classes.
  - Very clunky
- Paraview's python scripting also inherits some of that clunkiness.
- My view is that where possible post-processing should be scripted
  - Easier to re-produce similar plots for different cases or papers.
  - Easier to make high-quality plots-they can be iteratively improved.
- The easier the scripting is, the better!

VTK: Visualization Toolkit

- Paraview and Pyvista are based on the VTK.
- C++ library with interfaces for python and other languages.
- Python wrapper of VTK is nearly identical to the C++ functions and classes.
  - Very clunky
- Paraview's python scripting also inherits some of that clunkiness.
- My view is that where possible post-processing should be scripted
  - Easier to re-produce similar plots for different cases or papers.
  - Easier to make high-quality plots-they can be iteratively improved.
- The easier the scripting is, the better!

VTK: Visualization Toolkit

- Paraview and Pyvista are based on the VTK.
- C++ library with interfaces for python and other languages.
- Python wrapper of VTK is nearly identical to the C++ functions and classes.
  - Very clunky
- Paraview's python scripting also inherits some of that clunkiness.
- My view is that where possible post-processing should be scripted
  - Easier to re-produce similar plots for different cases or papers.
  - Easier to make high-quality plots-they can be iteratively improved.
- The easier the scripting is, the better!

VTK: Visualization Toolkit

- Paraview and Pyvista are based on the VTK.
- C++ library with interfaces for python and other languages.
- Python wrapper of VTK is nearly identical to the C++ functions and classes.
  - Very clunky
- Paraview's python scripting also inherits some of that clunkiness.
- My view is that where possible post-processing should be scripted
  - Easier to re-produce similar plots for different cases or papers.
  - Easier to make high-quality plots-they can be iteratively improved.
- The easier the scripting is, the better!

VTK: Visualization Toolkit

- Paraview and Pyvista are based on the VTK.
- C++ library with interfaces for python and other languages.
- Python wrapper of VTK is nearly identical to the C++ functions and classes.
  - Very clunky
- Paraview's python scripting also inherits some of that clunkiness.
- My view is that where possible post-processing should be scripted
  - Easier to re-produce similar plots for different cases or papers.
  - Easier to make high-quality plots-they can be iteratively improved.
- The easier the scripting is, the better!

What is pyvista?

- Lightweight, 'pythonic' wrapper for VTK.
- Inherits from the VTK python classes but makes them more user friendly.
  - Fully compatible with VTK base classes.
  - Much lower learning curve.
- Much of the 'filters' you find in Paraview are in pyvista
  - Both are based on VTK.
  - For example:
    - 'Cell data to point data' in Paraview
    - `obj.cell_data_to_point_data()` or `obj.ctp()`
- Pyvista has easy access to the underlying data.

What is pyvista?

- Lightweight, 'pythonic' wrapper for VTK.
- Inherits from the VTK python classes but makes them more user friendly.
  - Fully compatible with VTK base classes.
  - Much lower learning curve.
- Much of the 'filters' you find in Paraview are in pyvista
  - Both are based on VTK.
  - For example:
    - 'Cell data to point data' in Paraview
    - `obj.cell_data_to_point_data()` or `obj.ctp()`
- Pyvista has easy access to the underlying data.

What is pyvista?

- Lightweight, 'pythonic' wrapper for VTK.
- Inherits from the VTK python classes but makes them more user friendly.
  - Fully compatible with VTK base classes.
  - Much lower learning curve.
- Much of the 'filters' you find in Paraview are in pyvista
  - Both are based on VTK.
  - For example:
    - 'Cell data to point data' in Paraview
    - `obj.cell_data_to_point_data()` or `obj.ctp()`
- Pyvista has easy access to the underlying data.

What is pyvista?

- Lightweight, 'pythonic' wrapper for VTK.
- Inherits from the VTK python classes but makes them more user friendly.
  - Fully compatible with VTK base classes.
  - Much lower learning curve.
- Much of the 'filters' you find in Paraview are in pyvista
  - Both are based on VTK.
  - For example:
    - 'Cell data to point data' in Paraview
    - `obj.cell_data_to_point_data()` or `obj.ctp()`
- Pyvista has easy access to the underlying data.

Examples!

- Look at a simple pipe flow case
  - First, Paraview...
- Example Jupyter notebooks (`*.ipynb` files) using JupyterLab.
  - You could use Jupyter with Visual Studio Code too
- Examples:
  - `reading_cs.ipynb`: Reading and plotting `.case` file
  - `common_filters.ipynb`: Q vortex identification criterion and λ_2, contour filter
  - `statistics.ipynb`: Computing TKE, $k = \overline{u_i' u_i'}/2$
- Optional ones:
  - `nice_figures.ipynb`: Combining with matplotlib to produce multiple subplots
  - `spatial_averaging.ipynb`: averaging pipe in θ.

Examples!

- Look at a simple pipe flow case
  - First, Paraview...
- Example Jupyter notebooks (`*.ipynb` files) using JupyterLab.
  - You could use Jupyter with Visual Studio Code too
- Examples:
  - `reading_cs.ipynb`: Reading and plotting `.case` file
  - `common_filters.ipynb`: Q vortex identification criterion and λ_2, contour filter
  - `statistics.ipynb`: Computing TKE, $k = \overline{u_i' u_i'}/2$
- Optional ones:
  - `nice_figures.ipynb`: Combining with matplotlib to produce multiple subplots
  - `spatial_averaging.ipynb`: averaging pipe in θ.

Examples!

- Look at a simple pipe flow case
  - First, Paraview...
- Example Jupyter notebooks (`*.ipynb` files) using JupyterLab.
  - You could use Jupyter with Visual Studio Code too
- Examples:
  - **`reading_cs.ipynb`:** Reading and plotting `.case` file
  - **`common_filters.ipynb`:** Q vortex identification criterion and λ_2, contour filter
  - **`statistics.ipynb`:** Computing TKE, $k = \overline{u_i' u_i'}/2$
- Optional ones:
  - `nice_figures.ipynb`: Combining with matplotlib to produce multiple subplots
  - `spatial_averaging.ipynb`: averaging pipe in θ.

Examples!

- Look at a simple pipe flow case
  - First, Paraview...
- Example Jupyter notebooks (`*.ipynb` files) using JupyterLab.
  - You could use Jupyter with Visual Studio Code too
- Examples:
  - **`reading_cs.ipynb`:** Reading and plotting `.case` file
  - **`common_filters.ipynb`:** Q vortex identification criterion and λ_2, contour filter
  - **`statistics.ipynb`:** Computing TKE, $k = \overline{u_i' u_i'}/2$
- Optional ones:
  - **`nice_figures.ipynb`:** Combining with matplotlib to produce multiple subplots
  - **`spatial_averaging.ipynb`:** averaging pipe in θ.

Using Paraview or Pyvista?

Paraview

- For having an initial look at data.
  - Nice to have a GUI
- Very large unstructured mesh
  - $N_{cell} \gg 100M$
  - MPI on HPC perhaps with server-client mode

Pyvista

- Up to reasonably large unstructured meshes $N_{cell} < 100M$.
- Produces publication quality plots easily

- Pyvista can be run multithreaded using OpenMP, but it isn't trivial
  - Build VTK from source!
  - I have a script!
  - You could use Apptainer/Singularity

Conclusions

- I have introduced pyvista for use with Code_Saturne
  - Can be used with other formats compatible with VTK
    - `*.xdmf` (CHAPSim or Incompact3D), `*.foam` (OpenFoam).
- Shown some brief examples of how to use it
- Given my opinion of where to use it compared with Paraview