NEITcool Thermal Model DOCUMENTATION

Multiparameter POD-Based Reduced Order Model Version 1 (V1) – Server Modeling

Thermal Model Overview (V1)

This code is designed to develop a data-driven Proper Orthogonal Decomposition (POD) based reduced order model (ROM) for thermal analysis at the server level using the Python programming language. This model provides fast and accurate thermo-fluid performance predictions, with speeds up to 100 times faster than traditional high-fidelity computational fluid dynamics (CFD) tools.

V1 uses the Open Compute Project’s Olympus 1U server [1] as the base case model. The model enables the prediction of the temperature profile within the server by the adjustment of several operational parameters, including CPU load fraction, server inlet temperature, and server inlet velocity. It also assumes a steady-state condition with consistent material/fluid properties, where air is the working fluid. The user will be provided with a maximum and minimum limiting range for each parameter, which they can adjust to view a scenario of interest within the range quickly and accurately. Solution visualization and additional post-processing, such as the development of contour plots and flow field examination, will be done through the open-source tool ParaView.

V1 Objective

This software aims to enable users to quickly and accurately predict the thermal-fluid performance of their server models without dependence on computational fluid dynamics (CFD) simulations after the training ROM.

Table of Contents

1. Using the Reduced Order Thermal Model
NEITcool Module Product Description ……………..……………………………..……………3
Key Features………………….…………………………………………………………………4
Open Compute Project Olympus Server Model …………………………………………………..4
2. Pre-Processing
CFD Simulations With Multiparameter Variations ……………………...……………………..6
Performing POD Reduced Order Model for Server Level ……………………………………..8
3. Post-Processing
Computations ……………………….…………………………………………………………..8
Solution Visualization and Post-Processing in ParaView…………………………………………8
4. Theory and Code
Proper Orthogonal Decomposition (POD) ………………………………..….………………..15

References………………………………………………………………………………….……..…18

1. Using the Reduced Order Thermal Model

NEITcool Module Product Description

The NEITcool module is an open-source thermal modeling tool written in Python. Version 1 of this software allows users to quickly and accurately predict the thermo-fluid performance of their server models without dependence on computational fluid dynamics (CFD) software after the training phase of the reduced-order model.

The multi-objective optimization software (MOSTCOOL) supports data exchange between NEITcool and other modules within the suite, such as Energy+ and reliability modeling, as illustrated in **Figure 1**. The current release presently involves only data exchange between the NEITcool and Energy+ modules, as shown in **Figure 2**. Future releases will include integration with the reliability software at the server level.

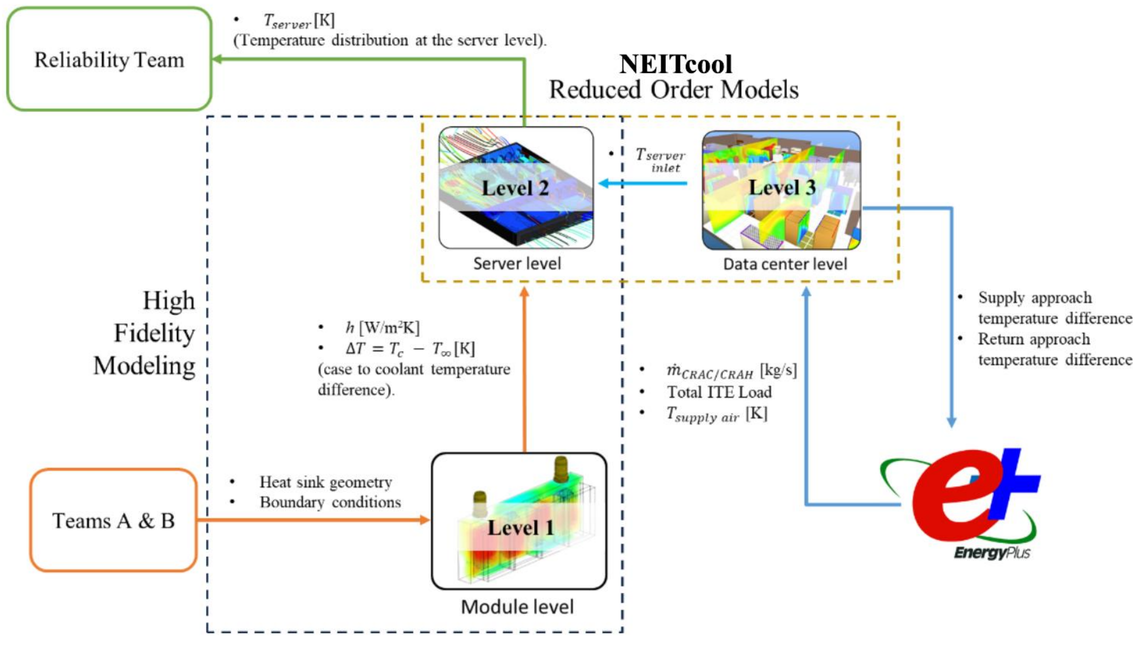


Figure 1. MOSTCOOL software integration scheme. Interaction between the thermal modeling team and other groups involves the exchange of inputs, wherein each team contributes information essential to the thermal modeling process, and outputs, signifying the results tailored to meet the specific needs of each respective team.

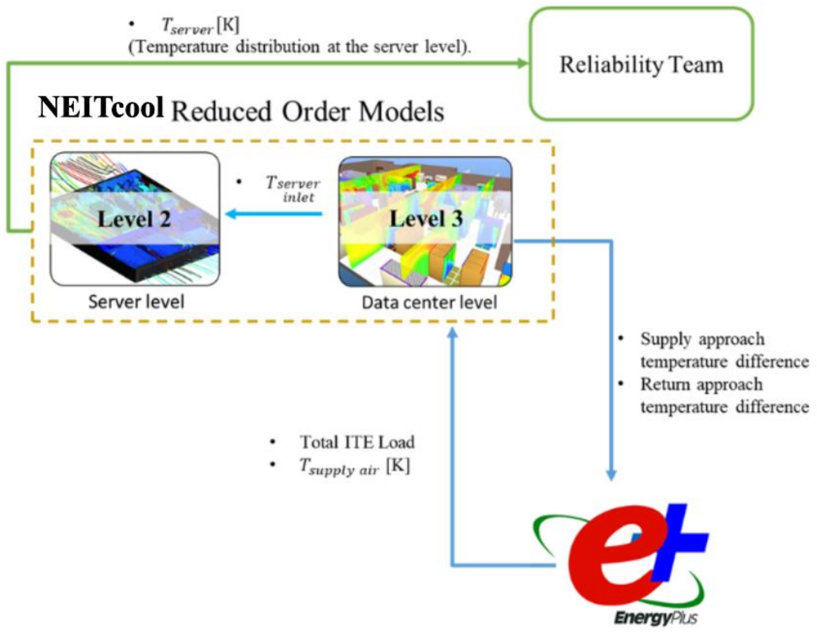

T_{server} [K]
(Temperature distribution at the server level).
Reliability Team
NEITcool Reduced Order Models
Level 2
Server level
$T_{server\ inlet}$
Level 3
Data center level
Supply approach temperature difference
Return approach temperature difference
Total ITE Load
$T_{supply\ air}$ [K]
EnergyPlus

Figure 2. This figure shows the current release integration only between the NEITcool module and Energy+. The schematic details the inputs and outputs for each module.

It includes a user-friendly graphical user interface that eliminates the need for any Python knowledge.

Key Features

- Graphical user interface
- Solution visualization and post-processing using ParaView V 5.12.0
- Quantity calculations such as surface averaging and minimum and maximum values
- Theory of mathematical concepts used in the reduced order model and some example Python code for those interested in low-level details

Open Compute Project Olympus Server Model

To help guide the user in using the thermal model, an example server problem and the steps needed to reach the final solution are dispersed throughout the pre-processing and post-processing steps of the documentation. The Olympus 1U server shown in **Figure 3 A** from Open Compute Project [1], [2] is the base server model for V1. The model was simplified to what is shown in **Figure 3 B** for the CFD simulations run to feed data to the POD model. The major heating component and the PBC board were kept, and the vapor chamber used to cool the CPUs in the original server were replaced with air cooled heat sinks. In these simulations, server inlet velocity, CPU load fraction, and server inlet temperature were varied to determine the temperature distribution across the server. The minimum and maximum ranges are provided in **Table 1**, and the server power levels are specified in **Table 2**.

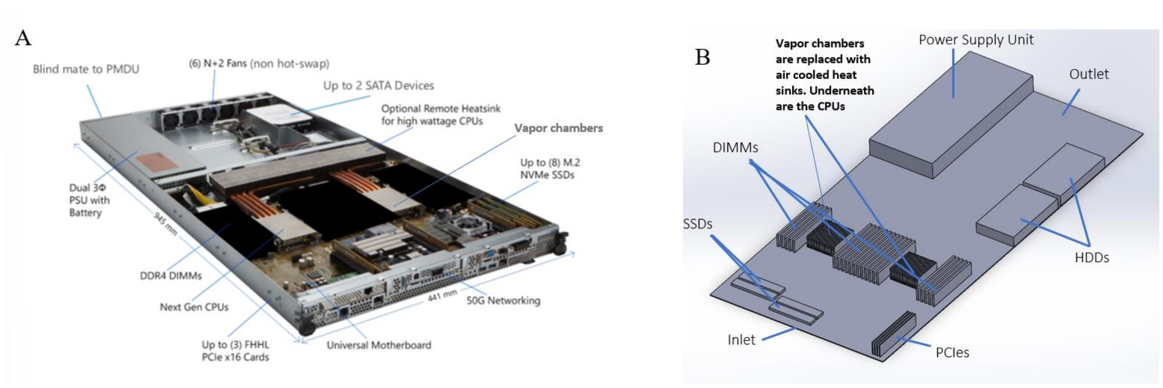


Figure 3. Open compute Project's Olympus 1U server. (A) Complete geometry for the server. The CAD model is provided in [1]. (2) Simplified CAD geometry of the Olympus 1U server. This was used to develop the simulation data needed to perform the POD.

Table 1. Simulation varied parameters and operating range limits

Parameter	**Range**
Server Inlet Velocity	5 m/s - 15 m/s
CPU Load Fraction	50% - 100%
Server Inlet Temperature	Limitless

Table 2. Server components' quantity, face area, volume, and power per component. Note that CPU load was the only component whose power varied in the simulations.

Component	**Quantity**	**Face area (cm^2)**	**Volume (cm^3)**	**Power (W) (each)**
1) RAM	22	40.50	20.25	10
2) SSD	4	39.00	7.80	20
3) PCIe	4	22.05	13.23	10
4) CPU	2	25.00	12.50	300-150 (varying load)
5) HDD	2	152.00	380.00	20
6) Power Supply	1	584.00	2236.00	20
Total	**-**	**-**	**-**	**1000**

2.Pre-Processing

A pre-processing step is conducted to develop the POD model. This step includes performing simulations to feed into the POD model for training and developing the POD model itself.

CFD Simulations with Multiparameter Variations

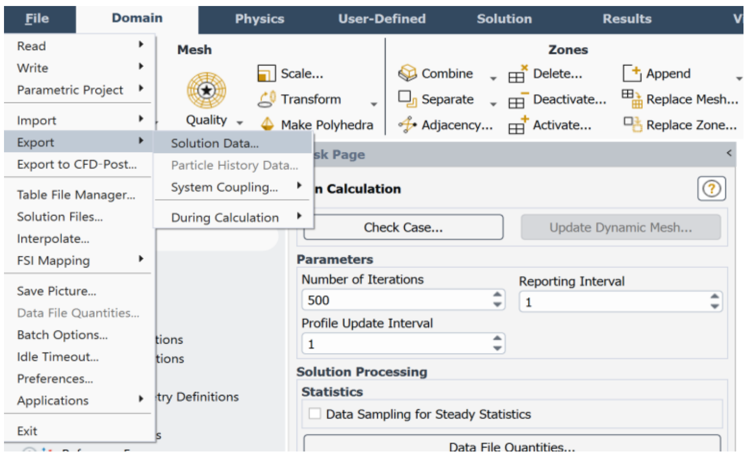
Initial CFD simulations at the server were done using the reduced-order model. This data-driven mathematical technique can predict the temperature distribution within the domain by interpolating between a set of simulated scenarios obtained from CFD. The temperature profile across the server was solved in each simulation case and saved as a CGNS file.

The CGNS file format is chosen as the exported file type as it contains all information needed for data treatment when performing SVD in Python, such as the spatial coordinates (*X*, *Y*, and *Z*) at a specific nodal location and the value of the field variable of interest (such as nodal temperature or nodal pressure). Any simulation software that can export solution data in CGNS format (e.g., ANSYS Fluent, OpenFOAM, COMSOL Multiphysics) can be used for the reduced-order model.

Additional information on the CGNS file format is provided in [3]. Below is an example of how to convert your solution into CGNS format in ANSYS Fluent.

Exporting solution to CGNS format in ANSYS Fluent

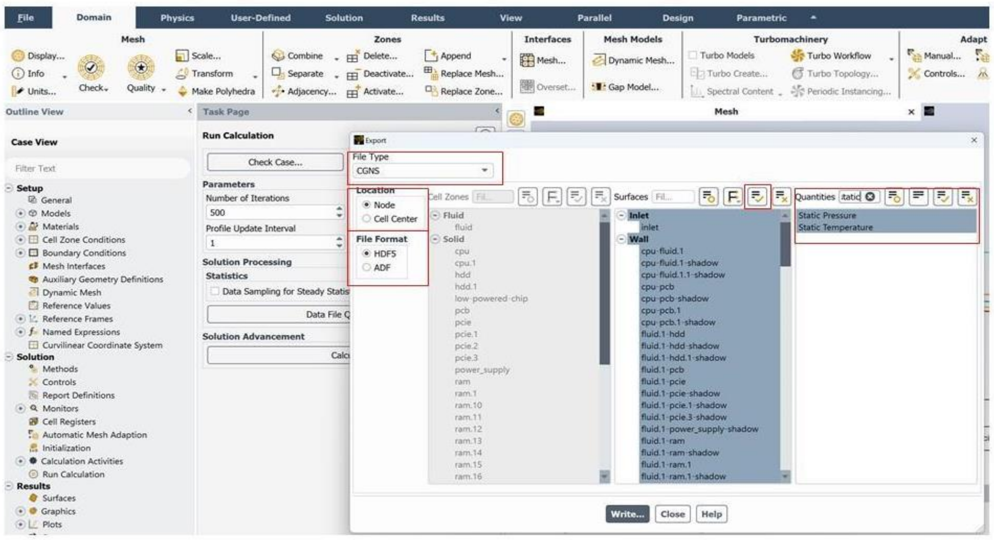
In the Fluent window, click on the "**File**" icon on the leftmost top ribbon, then select **"Export" ▯ "Solution Data",** as shown in Step 1 below.



Parameters
Number of Iterations: 500
Reporting Interval: 1
Profile Update Interval: 1

Step 1

A new window will open. As shown in **Step 2**, under "**File Type"**, select "**CGNS**" and ensure that the **"Node"** button is selected under "**Location"** and **"HDF5**" is selected for **File Format**. Ensure that you choose all surfaces in the "**Surfaces"** tab. This can be done by selecting the check mark next to "**Surfaces"** in the ribbon of the pop-up window. You should also choose the values of interest that you solved for. For example, in the Olympus 1U server model, we are interested in nodal temperature values, so we select the Static Temperature property. If you have an additional quantity you are interested in looking at, also select it.


Export

File Type: CGNS

Location: [✓ Node] Cell Center

File Format: [✓ HDF5] ADF

Surfaces (selected): inlet, cpu-fluid.1, cpu-fluid.1-shadow, cpu-fluid.1.1-shadow, cpu-pcb, cpu-pcb-shadow, cpu-pcb.1, cpu-pcb.1-shadow, fluid.1-hdd, fluid.1-hdd-shadow, fluid.1-hdd.1-shadow, fluid.1-pcb, fluid.1-pcie, fluid.1-pcie-shadow, fluid.1-pcie.1-shadow, fluid.1-pcie.3-shadow, fluid.1-power_supply-shadow, fluid.1-ram, fluid.1-ram-shadow, fluid.1-ram.1, fluid.1-ram.1-shadow

Quantities (selected): Static Pressure, Static Temperature

Step 2

Once you complete these steps, click **"Write"** and save your CGNS file in a known location.

It's important to note that the CFD simulation stage of developing the ROM is the most time-consuming part, as the POD model requires a substantial amount of data for training. In developing the POD model for the Olympus 1U server, about 20 simulations were performed to ensure reasonable accuracy.

However, once the POD model is completed, it is expected to provide fast and accurate predictions, significantly reducing the computational cost compared to running full CFD simulations.

Performing POD Reduced Order Model for Server Level

After the simulation phase is completed, the solution files are uploaded into the software for the POD. On the user side, nothing additional needs to be done with this step, as the POD ROM is done in the background in Python.

The basic premise of the POD model is that it reduces the dimensionality of a complex problem, identifies the most important patterns of variability, and then constructs a reduced model that captures these patterns. This reduced-order model can provide significant computational savings while maintaining accuracy in predicting system behavior.

If the user would like to learn more about the theory and process behind the POD method, some references are provided [4] - [8]. The basic theory and example code is also provided in this documentation's Theory and Code section.

3. Post-Processing

In the post-processing step, the user can actively use the GUI to predict a specific operational scenario by varying the three parameters in Table 1 and viewing their solution in ParaView.

Computations

Quantities such as the CPU's average or maximum surface temperature are calculated in the background after performing the POD. This step supports data exchange efforts between the NEITcool, Energy+, and reliability modules. For example, the average and maximum surface chip temperature can be computed and fed to the reliability module for failure modes assessment.

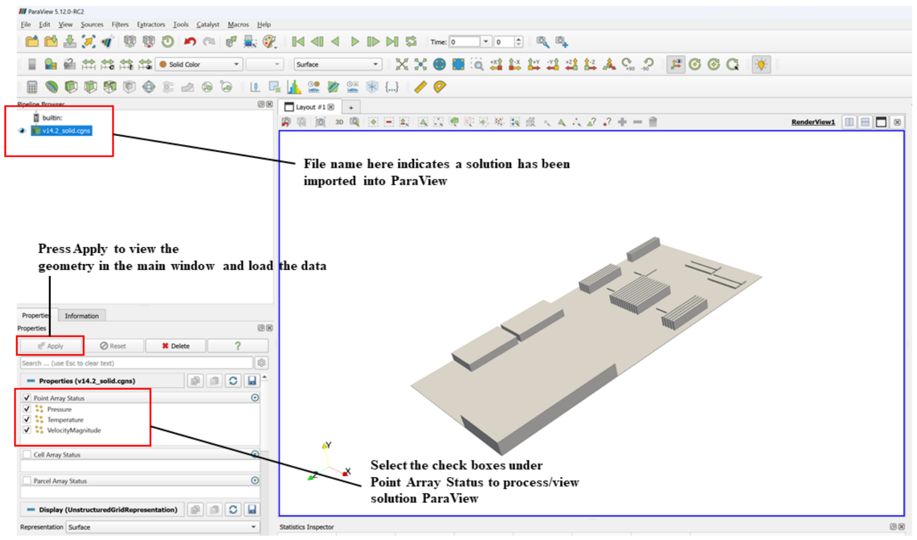
On the user’s part, the average CPU chip temperature is available for viewing in V1.

Solution Visualization and Post-Processing in ParaView

ParaView is an open-source data analysis and visualization tool that can process the solution from the POD model. The user will not need to do anything to open ParaView or load the predicted result into ParaView, as this is integrated with the GUI. However, they will need to understand basic controls in the tool. For additional detailed information, the user may refer to [9], but a tutorial on the necessary commands needed to view the solution and create contours is provided below for the Olympus 1U server.

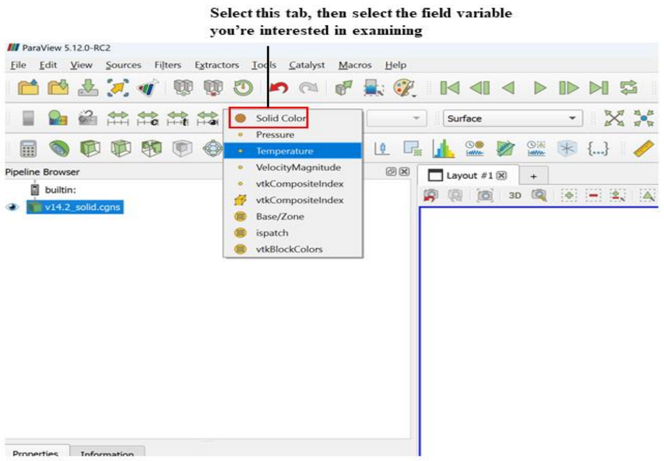
Visualizing Solution in ParaView

Once you give the predicted scenario to the GUI, you will automatically be directed to the ParaView page. Your main window will initially be blank. To view your geometry, click the "**Apply**" button as shown in **Step 1**. You can check all the boxes under the "**Point Array Status"** tab. This tab contains all the field variables stored inside the CGNS file.


File name here indicates a solution has been imported into ParaView
Press Apply to view the geometry in the main window and load the data
Select the check boxes under Point Array Status to process/view solution ParaView
Point Array Status
Pressure
Temperature
VelocityMagnitude

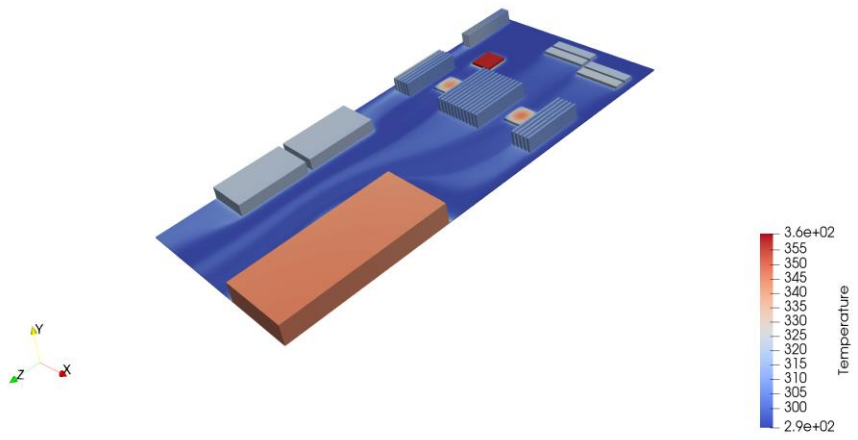
Step 1

Click on the tab with "**Solid Color"** in the middle ribbon and select the field variable you want to examine. In this example, the Temperature is selected.



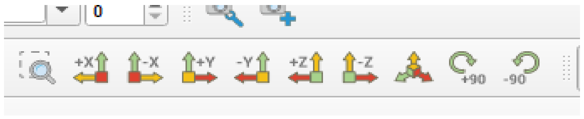
Step 2

The field variable profile will now be displayed on the geometry in the main window as shown in **Step 3.**



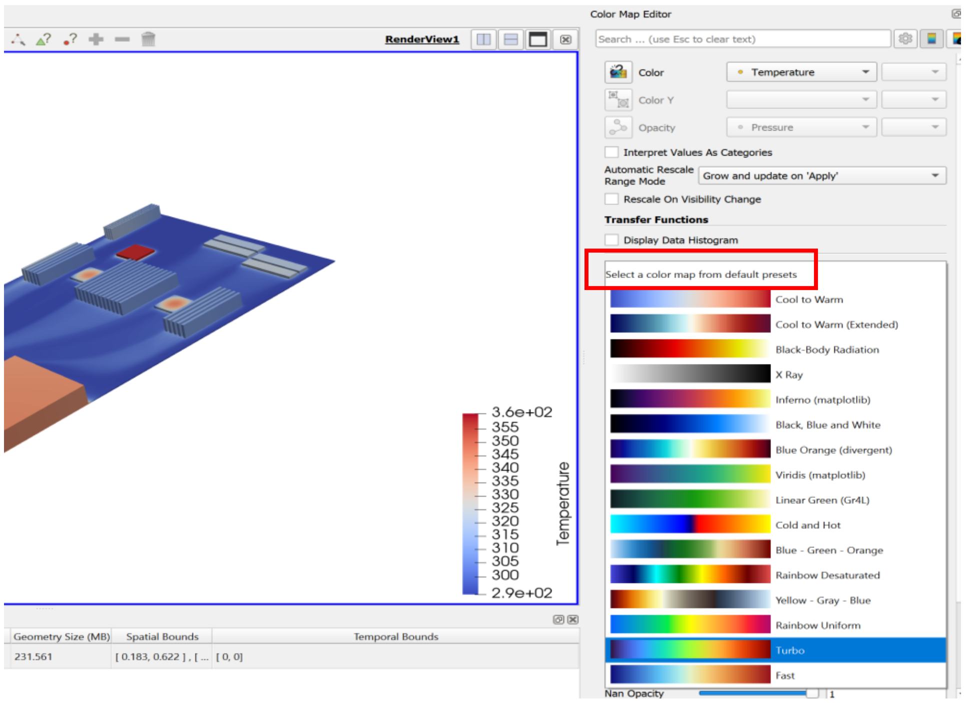
Step 3

To interact with the geometry, you may use the following commands in the second ribbon, as shown below in **Step 4**. You can left-click on your mouse on the main window to rotate the geometry freely or use the scroll wheel on your mouse to zoom in and out on the geometry.



Step 4

If you would like to change the color map type, in the Color Map Editor on the right of your screen, select the tab that says “**Select a color map from the default presets”** and choose any of the following:



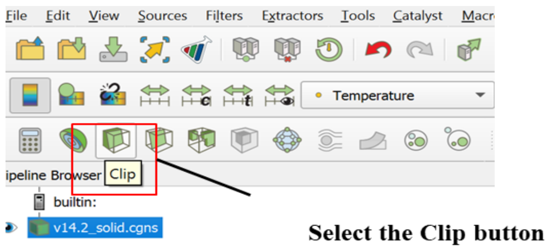
Color: Temperature
Color Y
Opacity: Pressure
Interpret Values As Categories
Automatic Rescale Range Mode: Grow and update on 'Apply'
Rescale On Visibility Change
Transfer Functions
Display Data Histogram
Select a color map from default presets
Cool to Warm
Cool to Warm (Extended)
Black-Body Radiation
X Ray
Inferno (matplotlib)
Black, Blue and White
Blue Orange (divergent)
Viridis (matplotlib)
Linear Green (Gr4L)
Cold and Hot
Blue - Green - Orange
Rainbow Desaturated
Yellow - Gray - Blue
Rainbow Uniform
Turbo
Fast
Nan Opacity 1

Geometry Size (MB)	Spatial Bounds	Temporal Bounds
231.561	[0.183, 0.622], [...	[0, 0]

Step 5

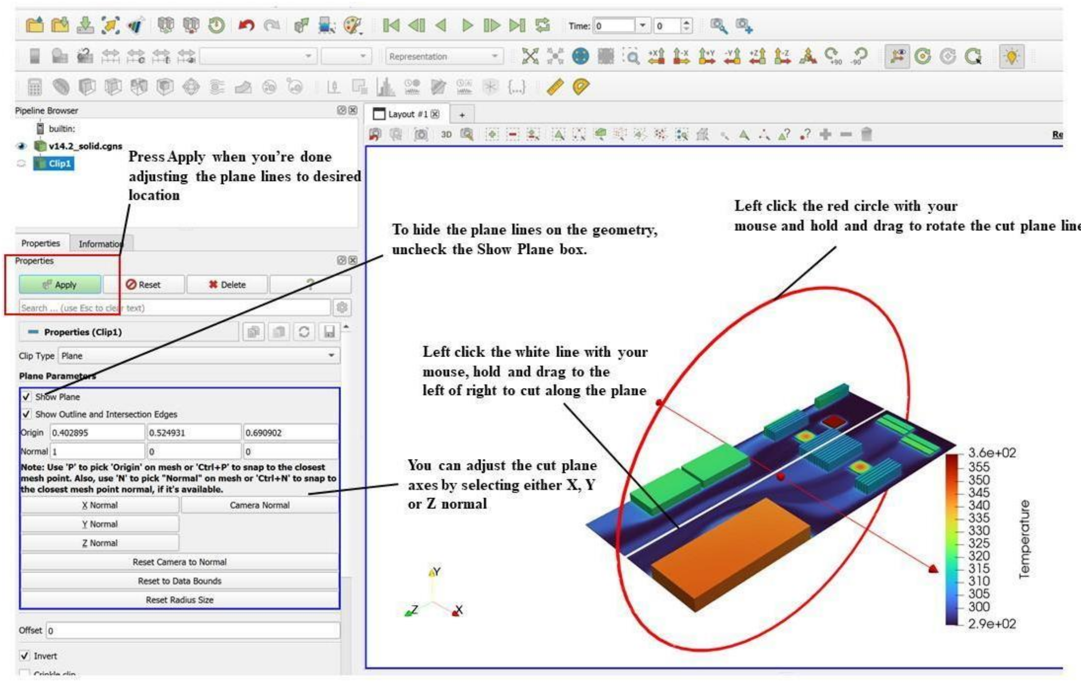
A standard colormap you can choose is the "Turbo" map, which is highlighted in blue in the schematic shown in **Step 5.**

To create a 2D contour plot for a specific part of the geometry, you may use the "**Clip**" button in the top ribbon, as shown in **Step 6**. This button is a filter that slices through your geometry and displays only a portion of the geometry that lies within the specified clipping places. This is particularly useful if you want to examine the internal structures of the model or if you want to focus on specific regions of interest.



Step 6

Once you select "**Clip",** you will see a white cut plane line and a red circle around your geometry. The white cut plane line through the geometry. You can left-click and hold on the white line and drag it along the plane to where you would like to cut the geometry. If you want to change the orientation of the cut plane, you can select either the “***X***”, “***Y***”, or “***Z***” **normal** buttons, as shown in **Step 7.** You can also rotate the cut plane by left-clicking and holding onto the red circle with your mouse. Once you're done adjusting, you can unselect the “**Show Plane”** checkbox to hide the plane, then press “**Apply”.**


Press Apply when you're done adjusting the plane lines to desired location

To hide the plane lines on the geometry, uncheck the Show Plane box.

Left click the red circle with your mouse and hold and drag to rotate the cut plane line

Left click the white line with your mouse, hold and drag to the left of right to cut along the plane

You can adjust the cut plane axes by selecting either X, Y or Z normal

Clip Type: Plane

Plane Parameters

✓ Show Plane

✓ Show Outline and Intersection Edges

Origin: 0.402895 | 0.524931 | 0.690902

Normal: 1 | 0 | 0

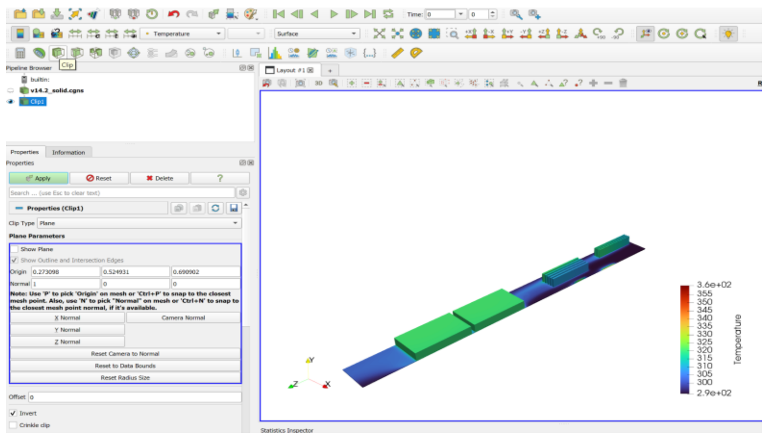
Note: Use 'P' to pick 'Origin' on mesh or 'Ctrl+P' to snap to the closest mesh point. Also, use 'N' to pick "Normal" on mesh or 'Ctrl+N' to snap to the closest mesh point normal, if it's available.

Offset: 0

✓ Invert

Step 7

The sliced plane is now viewable, as shown in Step 8.


Clip Type: Plane

Plane Parameters

Show Plane

✓ Show Outline and Intersection Edges

Origin: 0.273098 | 0.524931 | 0.690902

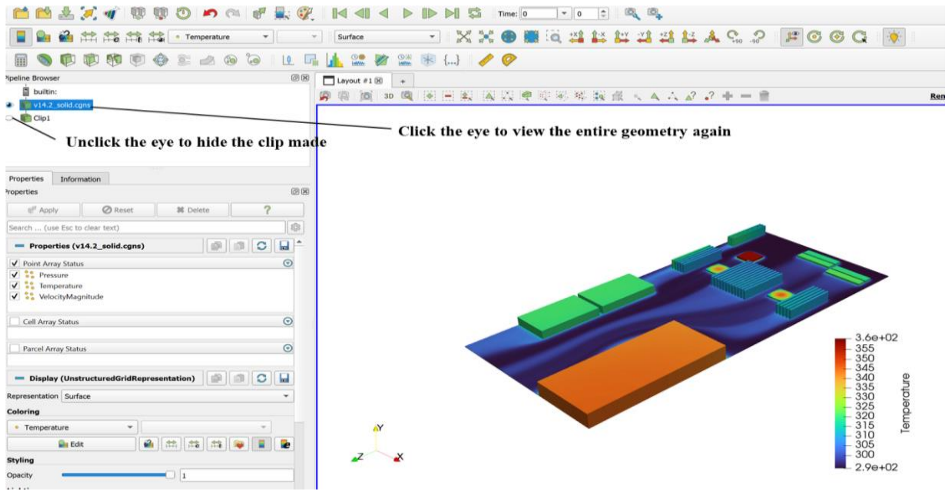
Normal: 1 | 0 | 0

Offset: 0

✓ Invert

Step 8

To review your complete geometry, unclick the eye of the Clip made, and reclick the eye for the .cgns file.



5. Theory and Code

A small theory section and an example Python code is provided below to allow the user to explore the approach used in developing a POD model.

Proper Orthogonal Decomposition

The Proper Orthogonal Decomposition (POD) method, also known as Principal Component Analysis (PCA) in statistics, is a widely used technique in fields such as fluid dynamics (both steady-state and transient problems), signal processing, and data analysis. The fundamental goal of POD is to decompose a multidimensional dataset into a set of orthogonal modes, which can be used to reconstruct the original data with reduced dimensionality while retaining most of its variance. The POD method essentially identifies the key patterns of variability in intricate data and builds a simplified model that retains these patterns. This streamlined model offers substantial computational efficiency while ensuring accuracy in predicting system behavior.

Singular Value Decomposition (SVD), a key component of POD, decomposes the simulation matrix and extracts the dominant modes of variability. These modes are then used to construct the reduced-order model in POD. SVD excels in extracting important features, enhancing data interpretability, and offering a linear algebraic framework for various mathematical analyses. The SVD decomposes a matrix into three other matrices, revealing its essential properties. For matrix A, the decomposition is $A = U\Sigma V^T$, where U and V are orthogonal matrices, and Σ is a diagonal matrix of singular values as shown in **Figure 3.**

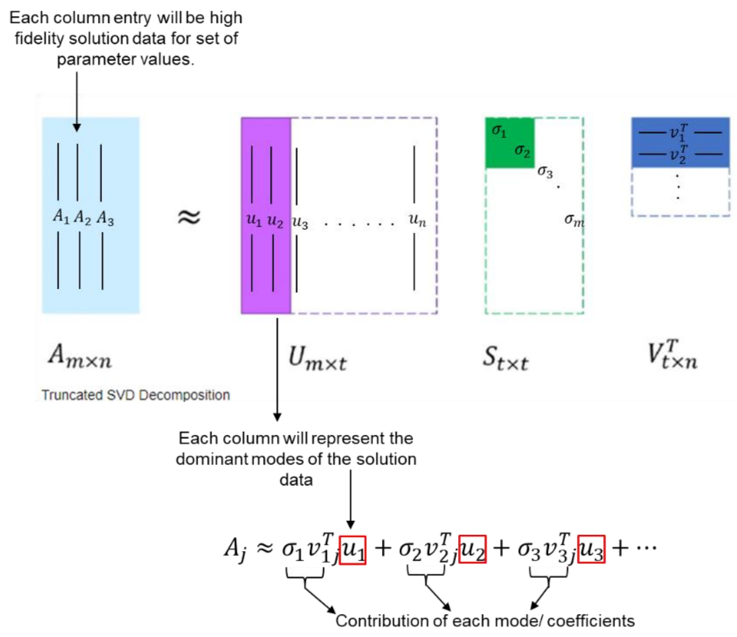

Each column entry will be high fidelity solution data for set of parameter values.
$A_1 A_2 A_3$
$\approx$
$u_1 u_2 u_3 \ldots u_n$
σ_1 σ_2 σ_3 σ_m
v_1^T v_2^T
$A_{m\times n}$
$U_{m\times t}$
$S_{t\times t}$
$V^T_{t\times n}$
Truncated SVD Decomposition
Each column will represent the dominant modes of the solution data
$A_j \approx \sigma_1 v_{1j}^T u_1 + \sigma_2 v_{2j}^T u_2 + \sigma_3 v_{3j}^T u_3 + \cdots$
Contribution of each mode/ coefficients

Figure 3. *Illustration of POD method by performing Singular Value Decomposition*

Performing SVD on a matrix A involves decomposing it into three matrices: U, Σ, and V, such that $A = U\Sigma V^T$, where U and V are orthogonal matrices, and Σ is a diagonal matrix containing the singular values of A. The steps to perform SVD are as follows:

1. **Compute the transpose of A (A^T):** If A is an $m\times n$ matrix, then A^T is an $n\times m$ matrix.
2. **Compute the product of A and A^T (AA^T):** Compute the matrix multiplication AA^T.
3. **Compute the product of A^T and A (A^TA):** Compute the matrix multiplication A^TA.
4. **Find the eigenvectors and eigenvalues of AA^T and A^TA:** Compute the eigenvectors and eigenvalues of AA^T. The eigenvectors form the columns of matrix V, and the square roots of the eigenvalues (after sorting in descending order) form the singular values along the diagonal of Σ. Compute the eigenvectors and eigenvalues of A^TA. The eigenvectors form the columns of matrix U.
5. **Construct U, Σ, and V:**

***U*:** The columns of U are the eigenvectors of A^TA, normalized to unit length.
Σ: Construct a diagonal matrix Σ by placing the square roots of the eigenvalues of AA^T (or A^TA) in descending order along the diagonal.
V: The columns of V are the eigenvectors of AA^T, normalized to unit length.

6. **Compute the pseudoinverse of *Σ (Σ+)*:** Replace the non-zero elements of Σ with their reciprocals and transpose the resulting matrix.
7. **Compute the SVD of *A*:** Compute the SVD of A as $A = U\Sigma V^T$.
8. **Optional: Reconstruct *A* (*$\hat{A}$*):** To reconstruct A, compute $\hat{A} = U\Sigma V^T$.

In the context of POD, A_j can be expressed as a sum of the products of POD modes (columns of U) and their coefficients (elements of V) as illustrated in **Figure 3**. This demonstrates how each POD mode contributes to the column A_j. Understanding this expansion provides insights into the underlying structure of the data and how different conditions influence it.

Truncated SVD is a variation of SVD in which only a few of the largest singular values (and corresponding vectors) are used. This approach is especially useful in data reduction and approximation. Truncated SVD reduces the dimensionality of the data, thereby simplifying the problem while retaining most of the significant information in the matrix.

Below is some base code for a simple single-parameter POD example to illustrate the basic structure of the POD model.

Simple POD Example

Below is an example code that explains how the POD code works according to the steps given in the theory section above. To interact directly with it, copy and paste the code and run it on a Python IDE.

Performing SVD

```python
import numpy as np

# Define the matrix A
A = np.array([[1, 2],
              [3, 4],
              [5, 6],
              [7, 8]])
print(A)

# ---------Calculating ATA --> VT---------
ATA = np.dot(A.T, A)
print("ATA\n", ATA)

# Calculate and print the eigenvectors and eigenvalues of ATA
eigenvalues_ATA, eigenvectors_ATA = np.linalg.eig(ATA)

print("\nEigenvalues of ATA:")
print(eigenvalues_ATA)

print("\nEigenvectors of ATA:")
print(eigenvectors_ATA)
```

```
# ---------Calculating AAT --> U---------

AAT = np.dot(A, A.T)
print("\n AAT\n", AAT)

# Calculate and print the eigenvalues of ATA
eigenvalues_AAT = np.linalg.eigvals(AAT)

# Calculate and print the eigenvectors and eigenvalues of AAT
eigenvalues_AAT, eigenvectors_AAT = np.linalg.eig(AAT)

print("\nEigenvalues of AAT:")
print(eigenvalues_AAT)

print("\nEigenvectors of AAT:")
print(eigenvectors_AAT)


# ---------------Perform SVD----------------
U, S, V = np.linalg.svd(A)

# Print U, S, and V matrices
print("\nU matrix:")
print(U)

print("\nS matrix (diagonal matrix of singular values):")
print(np.diag(S))

print("\nV matrix (transpose since NumPy returns V^T):")
print(V)

# Choose the rank for reconstruction
rank = 2

# Reconstruct the original matrix from the SVD components
A_reconstructed = np.dot(U[:, :rank], np.dot(np.diag(S[:rank]), V[:rank, :]))

print("\nReconstructed Matrix:")
print(A_reconstructed)
```

Reconstructing the *A* matrix

```
import numpy as np

# Define the matrix A
A = np.array([[1, 2], [3, 4], [5, 6], [7,8]])

# Perform SVD
U, S, V = np.linalg.svd(A)

# Calculate total energy
total_energy = np.sum(S**2)
print("Total energy =", total_energy)

# Choose the desired energy threshold (e.g., 90%)
energy_threshold = 0.9

# Calculate the rank needed to capture the specified energy threshold
cumulative_energy = np.cumsum(S**2) / total_energy
print("Cumulative energy", cumulative_energy )
rank = np.argmax(cumulative_energy >= energy_threshold) + 1

# Print the selected rank
print(f"\nSelected Rank to capture {energy_threshold*100}% energy: {rank}")
```

```
# Reconstruct the original matrix from the SVD components using the selected rank
A_reconstructed = np.dot(U[:, :rank], np.dot(np.diag(S[:rank]), V[:rank, :]))
print(V[:rank, :])

print("\nReconstructed Matrix:")
print(A_reconstructed)
```

References

[1] "Server/ProjectOlympus - OpenCompute." Accessed: Apr. 16, 2024. [Online]. Available: https://www.opencompute.org/wiki/Server/ProjectOlympus

[2] "Project Olympus Chassis Mechanical".

[3] "A User's Guide to CGNS - Introduction." Accessed: Apr. 16, 2024. [Online]. Available: https://cgns.github.io/CGNS_docs_current/user/intro.html

[4] A. Chatterjee, "An introduction to the proper orthogonal decomposition," *Curr. Sci.*, vol. 78, no. 7, pp. 808–817, 2000.

[5] N. Demo, M. Tezzele, and G. Rozza, "A non-intrusive approach for the reconstruction of POD modal coefficients through active subspaces," *Comptes Rendus Mécanique*, vol. 347, no. 11, pp. 873–881, Nov. 2019, doi: 10.1016/j.crme.2019.11.012.

[6] J. Weiss, "A Tutorial on the Proper Orthogonal Decomposition," in *AIAA Aviation 2019 Forum*, Dallas, Texas: American Institute of Aeronautics and Astronautics, Jun. 2019. doi: 10.2514/6.2019-3333.

[7] E. Samadiani and Y. Joshi, "Multi-parameter model reduction in multi-scale convective systems," *Int. J. Heat Mass Transf.*, vol. 53, no. 9, pp. 2193–2205, Apr. 2010, doi: 10.1016/j.ijheatmasstransfer.2009.12.013.

[8] R. Ghosh and Y. Joshi, "Rapid Temperature Predictions in Data Centers using Multi-Parameter Proper Orthogonal Decomposition." Accessed: Apr. 17, 2024. [Online]. doi: 0.1080/10407782.2013.869090

[9] "1. Introduction to ParaView — ParaView Documentation 5.12.0 documentation." Accessed: Apr. 16, 2024. [Online]. Available: https://docs.paraview.org/en/latest/UsersGuide/introduction.html