**GitHub：**https://github.com/DayBeha/HPC\_assignment

### Exercise 1 - Cythonize the STREAM Benchmark

**Task 1.1:** Develop a Cython version of the STREAM benchmark. Make sure to define the ctypes for obtaining full performance.

**Cython Code as follows:**

#cython: language\_level=3  
  
import numpy as np  
cimport numpy as np  
  
def copy(float[:] a, float[:] b):  
 # cdef float[:] c = np.empty(len(a), dtype=np.float32)  
 for i in range(len(a)):  
 b[i] = a[i]  
 # return c  
  
def scale(float scalar, float[:] a, float[:] b):  
 # cdef float[:] b = np.empty(len(a), dtype=np.float32)  
 for i in range(len(a)):  
 b[i] = scalar \* a[i]  
 # return b  
  
def add(float[:] a, float[:] b, float[:] c):  
 # cdef float[:] c = np.empty(len(a), dtype=np.float32)  
 for i in range(len(a)):  
 c[i] = a[i] + b[i]  
 # return c  
  
def triad(float[:] a, float scalar, float[:] b, float[:] c):  
 # cdef float[:] c = np.empty(len(a), dtype=np.float32)  
 for i in range(len(a)):  
 c[i] = a[i] + scalar \* b[i]  
 # return c

**Task 1.2:** Plot the bandwidth results varying the arrays' size. Answer the question: how does the bandwidth measured with Cython code compare to bandwidth obtained in Assignment II.

**bandwidth before Cython:**

STREAM\_ARRAY\_SIZE = 5

Copy time use:8.00 us data amount:280 bandwith:35.000 MB/s

add time use:87.00 us data amount:280 bandwith:3.218 MB/s

scale time use:152.00 us data amount:420 bandwith:2.763 MB/s

triad time use:31.00 us data amount:420 bandwith:13.548 MB/s

STREAM\_ARRAY\_SIZE = 10

Copy time use:2.00 us data amount:560 bandwith:280.000 MB/s

add time use:10.00 us data amount:560 bandwith:56.000 MB/s

scale time use:21.00 us data amount:840 bandwith:40.000 MB/s

triad time use:18.00 us data amount:840 bandwith:46.667 MB/s

STREAM\_ARRAY\_SIZE = 50

Copy time use:1.00 us data amount:2800 bandwith:2800.000 MB/s

add time use:8.00 us data amount:2800 bandwith:350.000 MB/s

scale time use:16.00 us data amount:4200 bandwith:262.500 MB/s

triad time use:18.00 us data amount:4200 bandwith:233.333 MB/s

STREAM\_ARRAY\_SIZE = 100

Copy time use:2.00 us data amount:5600 bandwith:2800.000 MB/s

add time use:8.00 us data amount:5600 bandwith:700.000 MB/s

scale time use:18.00 us data amount:8400 bandwith:466.667 MB/s

triad time use:21.00 us data amount:8400 bandwith:400.000 MB/s

STREAM\_ARRAY\_SIZE = 500

Copy time use:4.00 us data amount:28000 bandwith:7000.000 MB/s

add time use:10.00 us data amount:28000 bandwith:2800.000 MB/s

scale time use:22.00 us data amount:42000 bandwith:1909.091 MB/s

triad time use:25.00 us data amount:42000 bandwith:1680.000 MB/s

STREAM\_ARRAY\_SIZE = 1000

Copy time use:4.00 us data amount:56000 bandwith:14000.000 MB/s

add time use:12.00 us data amount:56000 bandwith:4666.667 MB/s

scale time use:21.00 us data amount:84000 bandwith:4000.000 MB/s

triad time use:25.00 us data amount:84000 bandwith:3360.000 MB/s

STREAM\_ARRAY\_SIZE = 5000

Copy time use:4.00 us data amount:280000 bandwith:70000.000 MB/s

add time use:17.00 us data amount:280000 bandwith:16470.588 MB/s

scale time use:26.00 us data amount:420000 bandwith:16153.846 MB/s

triad time use:91.00 us data amount:420000 bandwith:4615.385 MB/s

STREAM\_ARRAY\_SIZE = 10000

Copy time use:4.00 us data amount:560000 bandwith:140000.000 MB/s

add time use:36.00 us data amount:560000 bandwith:15555.556 MB/s

scale time use:52.00 us data amount:840000 bandwith:16153.846 MB/s

triad time use:60.00 us data amount:840000 bandwith:14000.000 MB/s

STREAM\_ARRAY\_SIZE = 50000

Copy time use:18.00 us data amount:2800000 bandwith:155555.556 MB/s

add time use:347.00 us data amount:2800000 bandwith:8069.164 MB/s

scale time use:242.00 us data amount:4200000 bandwith:17355.372 MB/s

triad time use:302.00 us data amount:4200000 bandwith:13907.285 MB/s

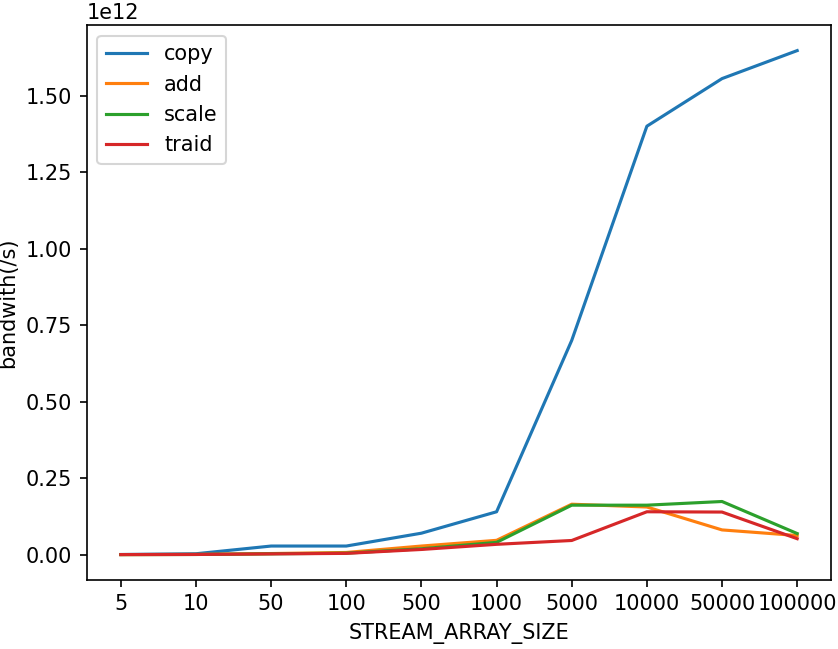
STREAM\_ARRAY\_SIZE = 100000

Copy time use:34.00 us data amount:5600000 bandwith:164705.882 MB/s

add time use:898.00 us data amount:5600000 bandwith:6236.080 MB/s

scale time use:1220.00 us data amount:8400000 bandwith:6885.246 MB/s

triad time use:1616.00 us data amount:8400000 bandwith:5198.020 MB/s



**bandwidth after Cython:**

STREAM\_ARRAY\_SIZE = 5

Copy time use:83.00 us data amount:280 bandwith:3.373 MB/s

add time use:29.00 us data amount:280 bandwith:9.655 MB/s

scale time use:10.00 us data amount:420 bandwith:42.000 MB/s

triad time use:11.00 us data amount:420 bandwith:38.182 MB/s

STREAM\_ARRAY\_SIZE = 10

Copy time use:13.00 us data amount:560 bandwith:43.077 MB/s

add time use:11.00 us data amount:560 bandwith:50.909 MB/s

scale time use:8.00 us data amount:840 bandwith:105.000 MB/s

triad time use:9.00 us data amount:840 bandwith:93.333 MB/s

STREAM\_ARRAY\_SIZE = 50

Copy time use:9.00 us data amount:2800 bandwith:311.111 MB/s

add time use:9.00 us data amount:2800 bandwith:311.111 MB/s

scale time use:7.00 us data amount:4200 bandwith:600.000 MB/s

triad time use:9.00 us data amount:4200 bandwith:466.667 MB/s

STREAM\_ARRAY\_SIZE = 100

Copy time use:9.00 us data amount:5600 bandwith:622.222 MB/s

add time use:10.00 us data amount:5600 bandwith:560.000 MB/s

scale time use:7.00 us data amount:8400 bandwith:1200.000 MB/s

triad time use:10.00 us data amount:8400 bandwith:840.000 MB/s

STREAM\_ARRAY\_SIZE = 500

Copy time use:12.00 us data amount:28000 bandwith:2333.333 MB/s

add time use:15.00 us data amount:28000 bandwith:1866.667 MB/s

scale time use:10.00 us data amount:42000 bandwith:4200.000 MB/s

triad time use:15.00 us data amount:42000 bandwith:2800.000 MB/s

STREAM\_ARRAY\_SIZE = 1000

Copy time use:17.00 us data amount:56000 bandwith:3294.118 MB/s

add time use:22.00 us data amount:56000 bandwith:2545.455 MB/s

scale time use:14.00 us data amount:84000 bandwith:6000.000 MB/s

triad time use:22.00 us data amount:84000 bandwith:3818.182 MB/s

STREAM\_ARRAY\_SIZE = 5000

Copy time use:45.00 us data amount:280000 bandwith:6222.222 MB/s

add time use:74.00 us data amount:280000 bandwith:3783.784 MB/s

scale time use:43.00 us data amount:420000 bandwith:9767.442 MB/s

triad time use:74.00 us data amount:420000 bandwith:5675.676 MB/s

STREAM\_ARRAY\_SIZE = 10000

Copy time use:82.00 us data amount:560000 bandwith:6829.268 MB/s

add time use:137.00 us data amount:560000 bandwith:4087.591 MB/s

scale time use:79.00 us data amount:840000 bandwith:10632.911 MB/s

triad time use:141.00 us data amount:840000 bandwith:5957.447 MB/s

STREAM\_ARRAY\_SIZE = 50000

Copy time use:385.00 us data amount:2800000 bandwith:7272.727 MB/s

add time use:672.00 us data amount:2800000 bandwith:4166.667 MB/s

scale time use:372.00 us data amount:4200000 bandwith:11290.323 MB/s

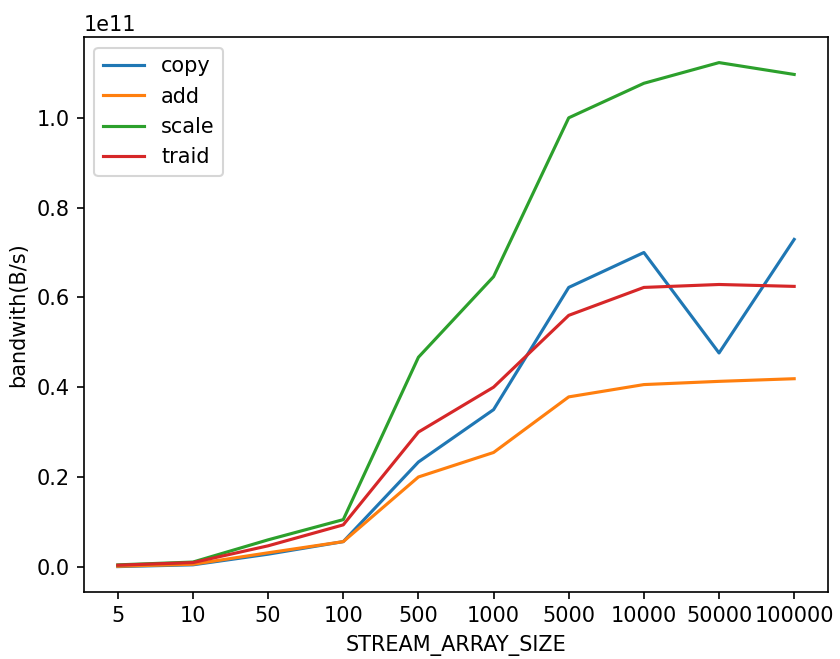
triad time use:681.00 us data amount:4200000 bandwith:6167.401 MB/s

STREAM\_ARRAY\_SIZE = 100000

Copy time use:749.00 us data amount:5600000 bandwith:7476.636 MB/s

add time use:1340.00 us data amount:5600000 bandwith:4179.104 MB/s

scale time use:741.00 us data amount:8400000 bandwith:11336.032 MB/s

triad time use:1355.00 us data amount:8400000 bandwith:6199.262 MB/s 

**Conclusion:**

1、As the plots shows, ‘copy’ get the highest bandwidth and high better than others before Cythonizing and ‘scale’ get the highest bandwidth after Cythonizing. It may because it’s copied as matrix before Cythonizing.

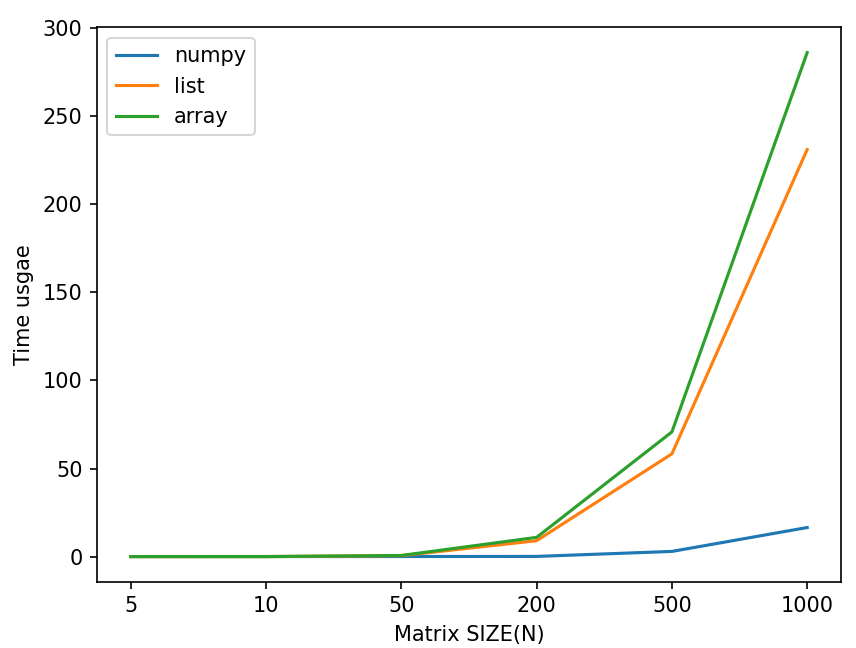
2、The performance after Cythonizing is no better than before which not as expectation. I both tried pass variable to functions in \*.pyx and define a new array. No better performance appear.

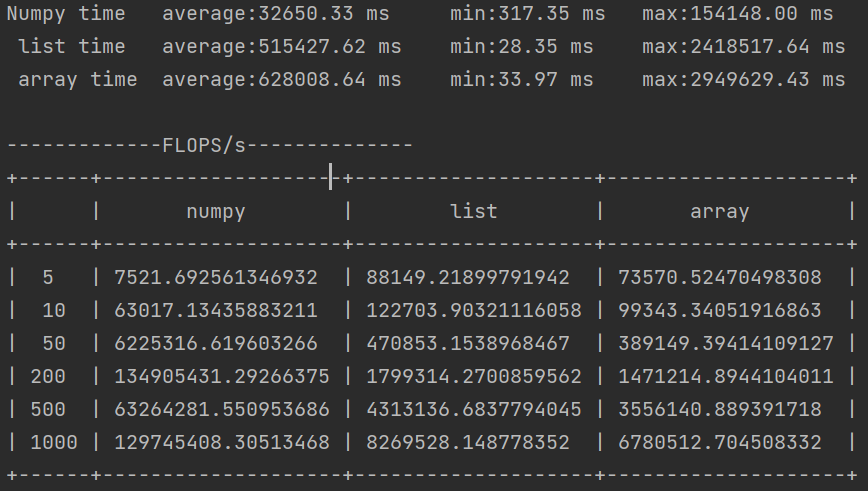
3、The bandwidth decrease when the STREAM\_ARRAY\_SIZE is too large(100000) before Cythonizing but not decrease after Cythonizing where bandwidth after Cythonizing is better than before. It may because ‘np.array’ need more memory than ‘cdef’.

### Exercise 2 - Gauss-Seidel for Poisson Solver

**Task 2.1:** Develop the Gauss-Seidel solver with Python List, array, or NumPy. Plot the performance varying the grid size.

Set grid size as [5,10,50,200,500,1000]

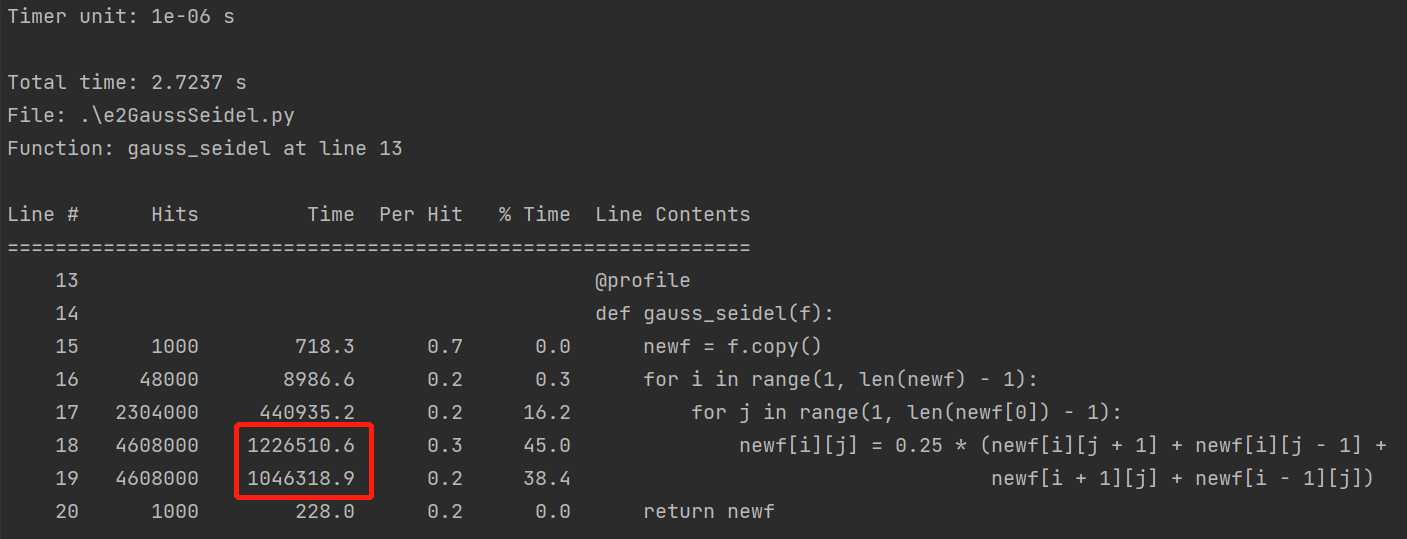




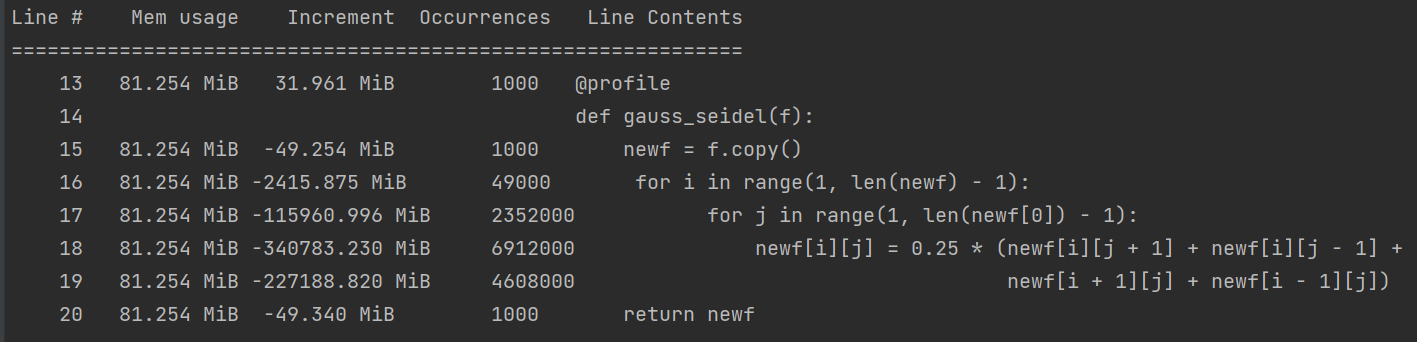
The performance is Numpy>list>array.

**Task 2.2:** Profile the code to identify the part of the code to optimize. You can use the tool of your choice.

**line\_profiler**

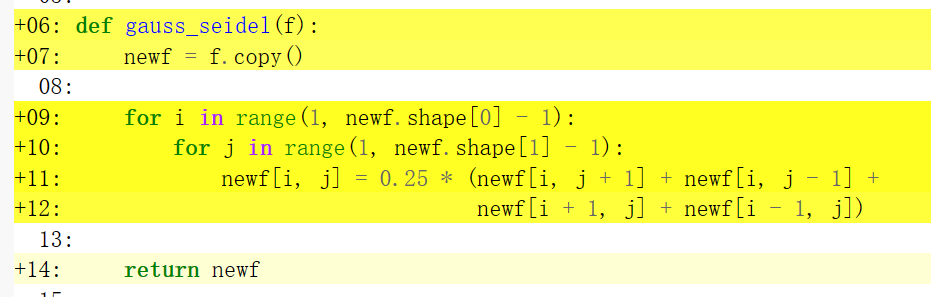


**memory\_profiler**



It can be seen that the code in loop takes the most time.

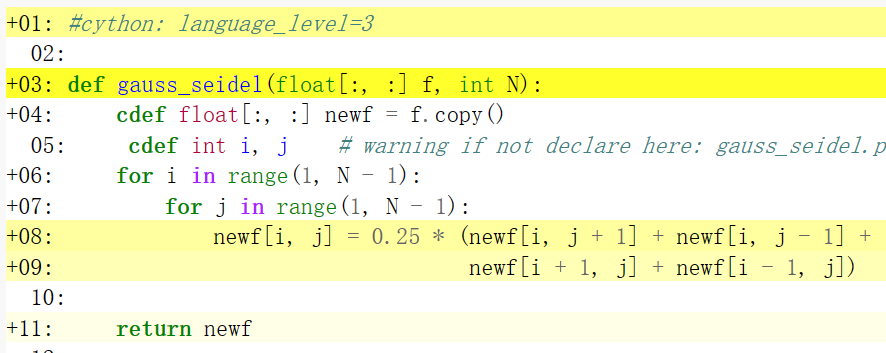
**Task 2.3:** Use the Cython Annotation tool to identify the parts to use Cython



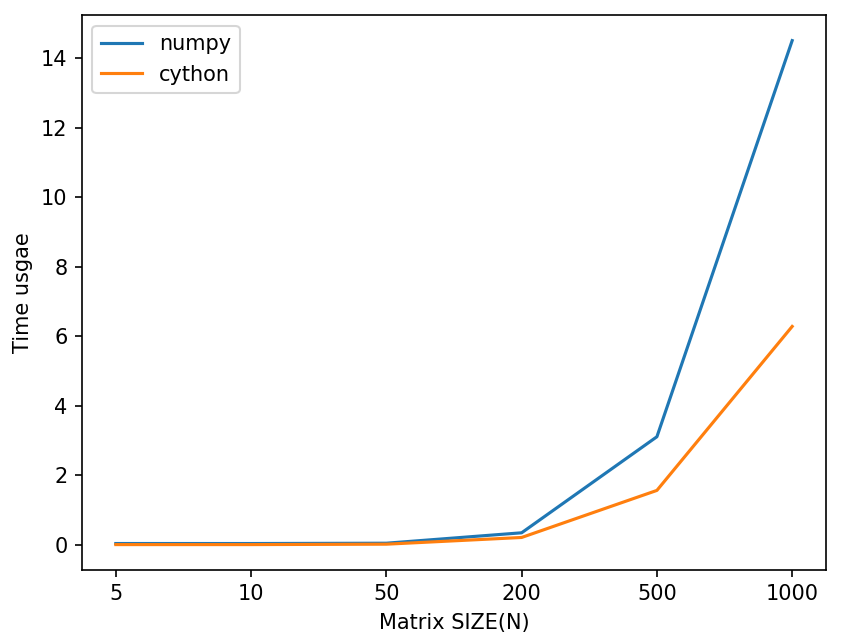
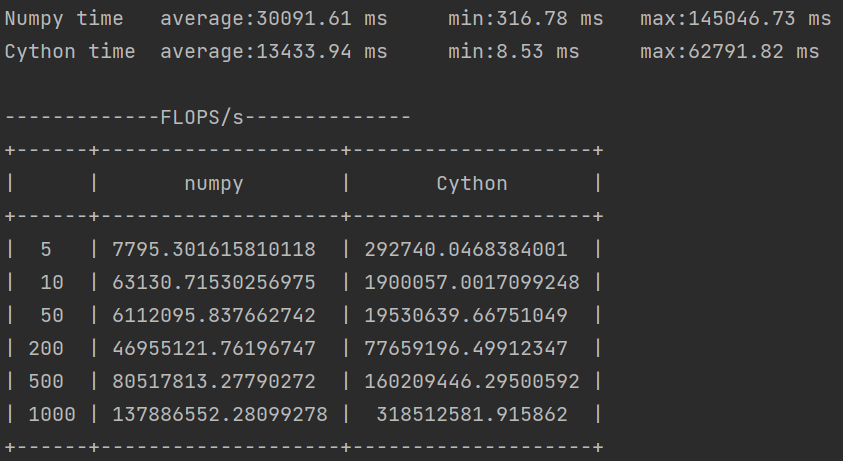
**It’s too yellow here, so all the code need to be update.**

**Task 2.4:** Use Cython to optimize the part you identified as the most computationally expensive. Compare the performance with the results obtained in Task 2.1.

**After optimize:**



Compare performance:

It looks better after cython is applied, taking about half time compared with numpy.

**Task 2.5:** Use PyTorch to port your code to Nvidia GPUs. For this you will need to express the two nested loops operations as numpy roll operations in 2D as we did for the diffusion code.

**Task 2.6:** Use CuPy to port your code to Nvidia GPUs. See [C.3 Tutorial - Introduction to CuPyC.3 Tutorial - Introduction to CuPy](https://canvas.kth.se/courses/37558/pages/c-dot-3-tutorial-introduction-to-cupy?wrap=1)

**Task 2.7:**Measure the performance (execution time) with GPU (PyTorch and cupy) and make a plot of the execution time varying the size the of the grid. Compare and comment of the performance difference with and without GPU.

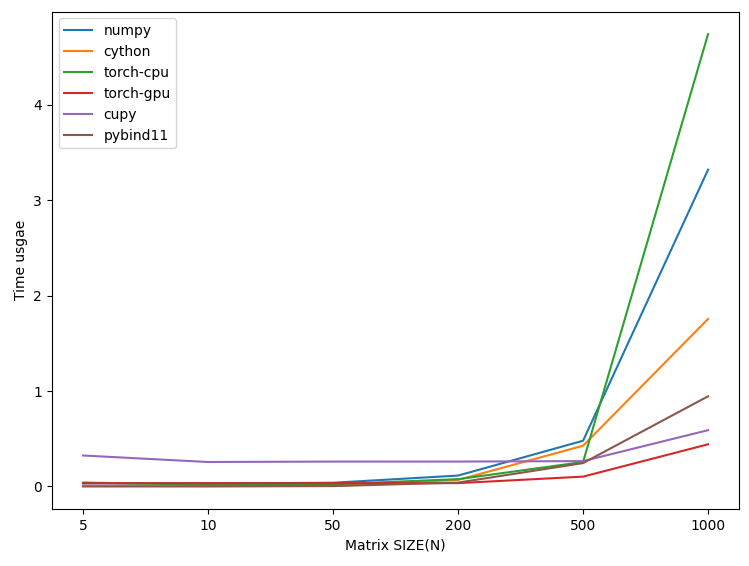
**Task 2.8:**Save the newgrid matrix as an hdf5 file using h5py  (see the tutorial [C.4 Tutorial - The HDF5 Library/ModuleC.4 Tutorial - The HDF5 Library/Module](https://canvas.kth.se/courses/37558/pages/c-dot-4-tutorial-the-hdf5-library-slash-module?wrap=1))

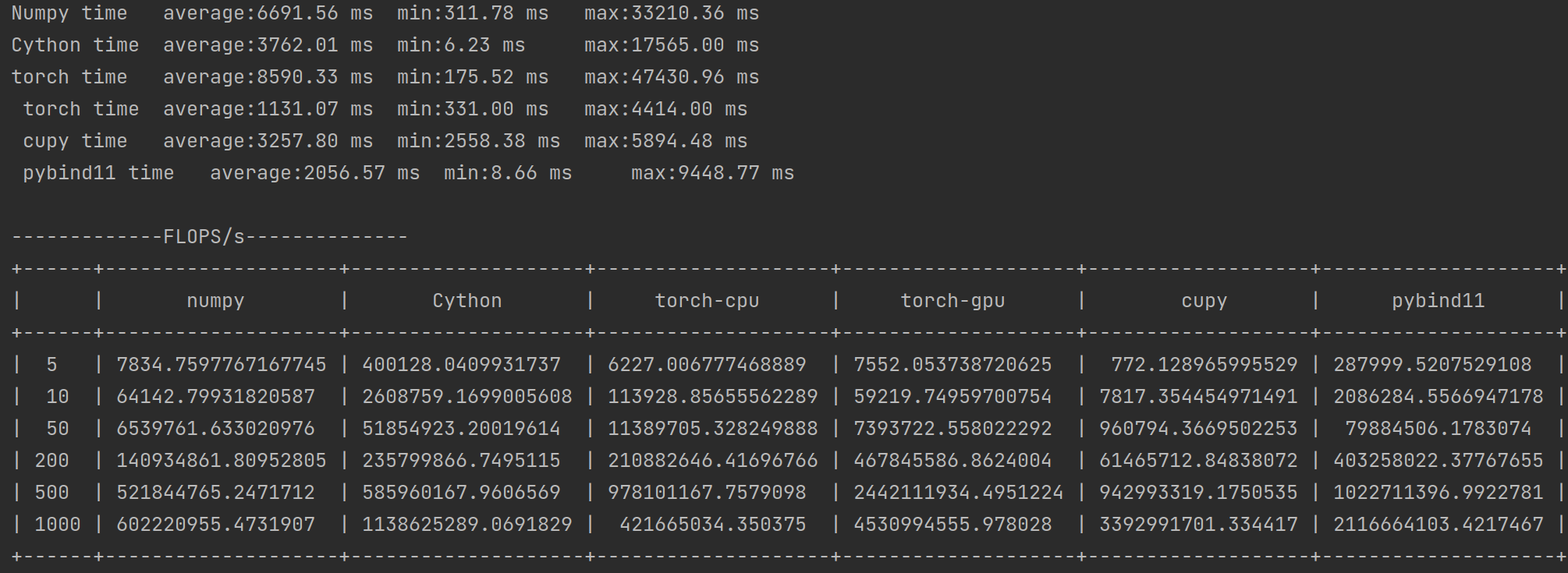
Its saved as file Assignment 3/newfs.hdf5 in github.

**Bonus Task B.1:** Develop the function you want to optimize in C/C++ or Fortran, make it a library, and couple C and Python. You can use the approach of your choice (see the lecture on foreign function interfaces). This is part of the bonus exercise.

**2.5-2.7 & B.1:** (codes in github Assignment 3/e2GaussSeidel\_gpu.py)

Set grid size as [5,10,50,200,500,1000]





As the plots shows:

1. Performance with GPU is highly better when Matrix Size is big enough and time usage is stable as Matrix Size increasing. Because of th advantage of Synchronous Computing of GPU.
2. With GPU performance of torch is better than cupy.
3. Without GPU performance is: pybind11>cython>numpy. In particular, torch in cpu is better than cython when Matrix Size is small, and no better than numpy as Matrix Size big enough.

**B1:** Created two libs by Cython and Pybind11, Assignment3/gauss\_seidel\_cython.pyx, Assignment3/gauss\_seidel\_c.cpp, respectively.

Bonus Exercise - Get familiar with VTK and Paraview

In this exercise, we ask you to experiment with one of the most used tools to visualize datasets coming from HPC simulations, Paraview. One of the most common and most used formats for visualization is called VTK.

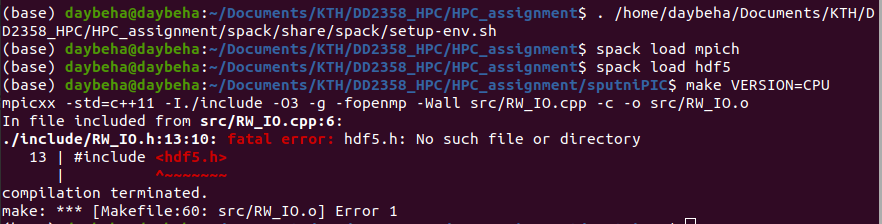
As part of the exercise, you will need to repeat all the different steps presented in the [C.6 Tutorial - VTK &amp; Visualization with ParaviewC.6 Tutorial - VTK &amp; Visualization with Paraview](https://canvas.kth.se/courses/37558/pages/c-dot-6-tutorial-vtk-and-visualization-with-paraview?wrap=1)

To install sputniPIC follow the instructions in the tutorial for paraview:

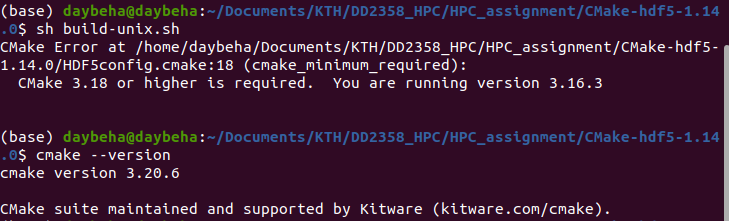
1. Install the **mpich and hdf5 libraries using spack**
2. Install the SputniPIC code
3. When you run sputniPIC, the code will create a number of VTK files in the data folder (be sure to create a directory called data otherwise, you will get an error and you can't run sputniPIC)
4. Download and install paraview on your system
5. Use paraview for the visualization

**Task 3.1:** Make a volume plot and a slice of the rhoe at the last step recorded in the vtk file as shown in the tutorial. Add the snapshots you took to your report.

Get error when make **sputniPIC**:



And when I’m trying to configure hdf5 from the source, I got weird ouput:

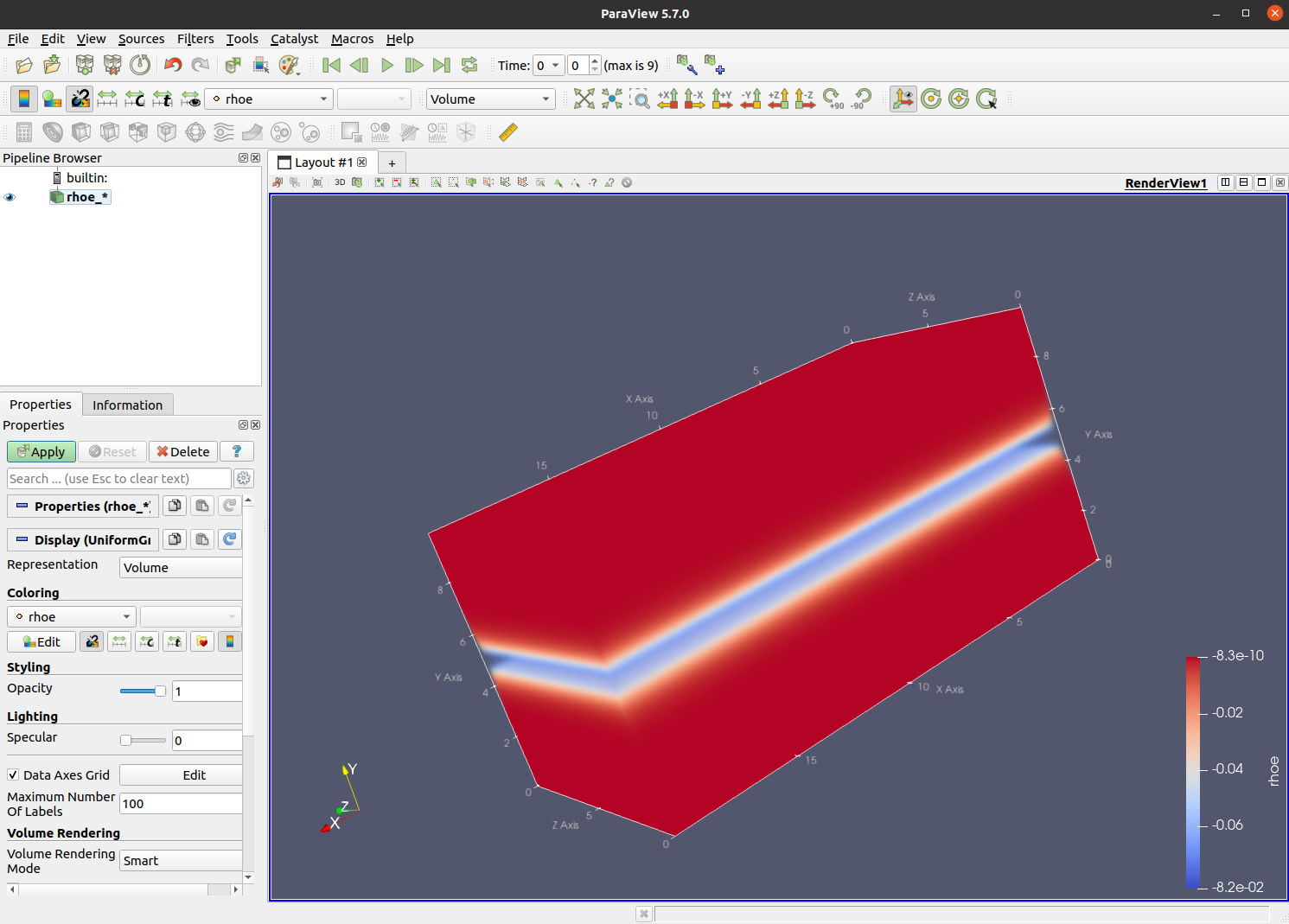


……

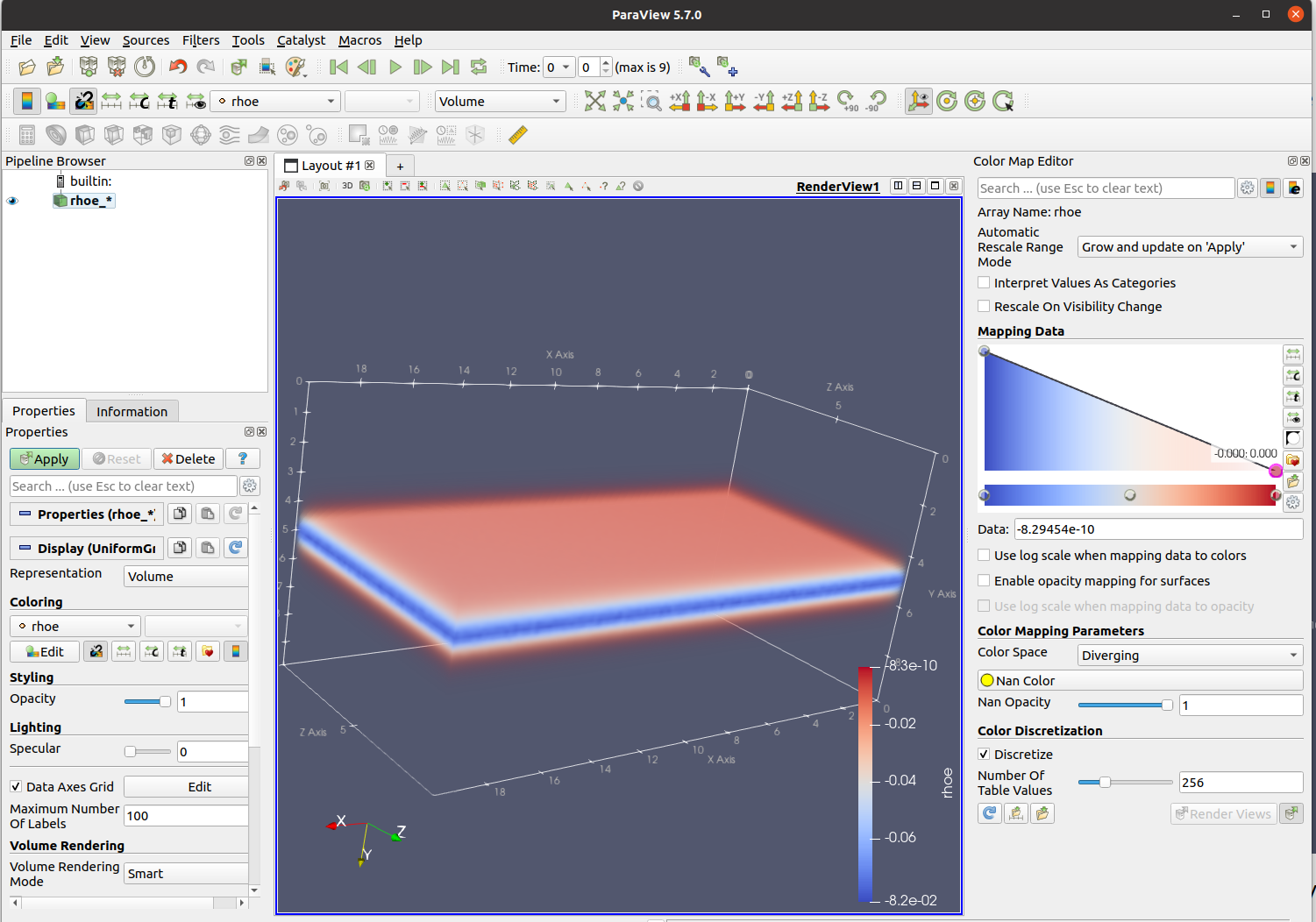
Lost of problems to load [hdf5](https://www.hdfgroup.org/downloads/hdf5/source-code/) and [mpich](https://www.mpich.org/downloads/) with spack using Ubuntu20.04.

Solved them by install hdf5 and mpich frome source and build with cmake.

**Successfully open rhoe\_\*.vtk** with **ParaView** using **Volume rendering:**

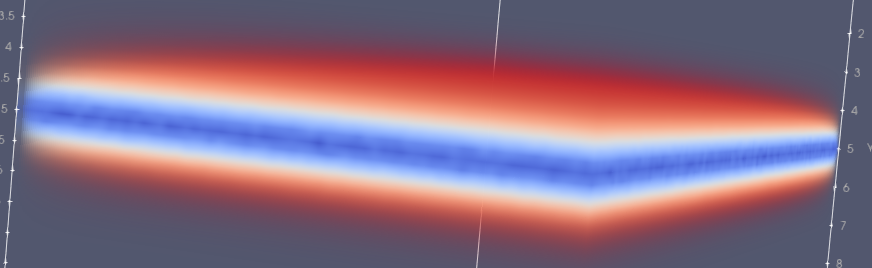


**reverse the coloring of Mapping Data:**

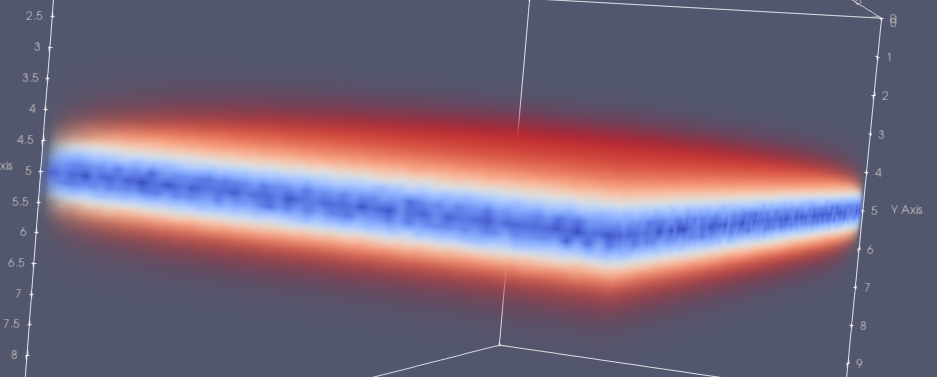


**play all the simulation output time step：**

time=0

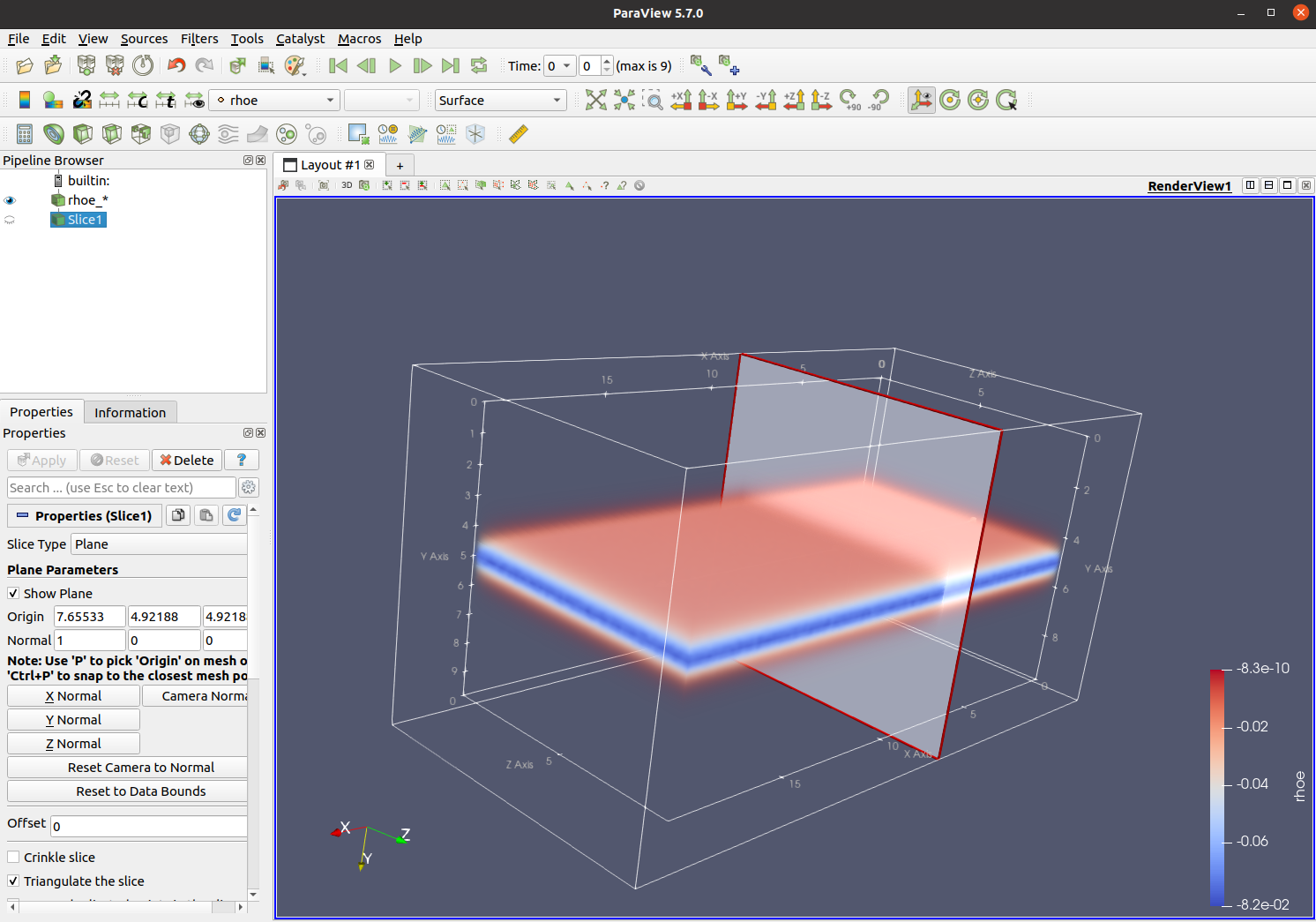


time=9 (max is 9)

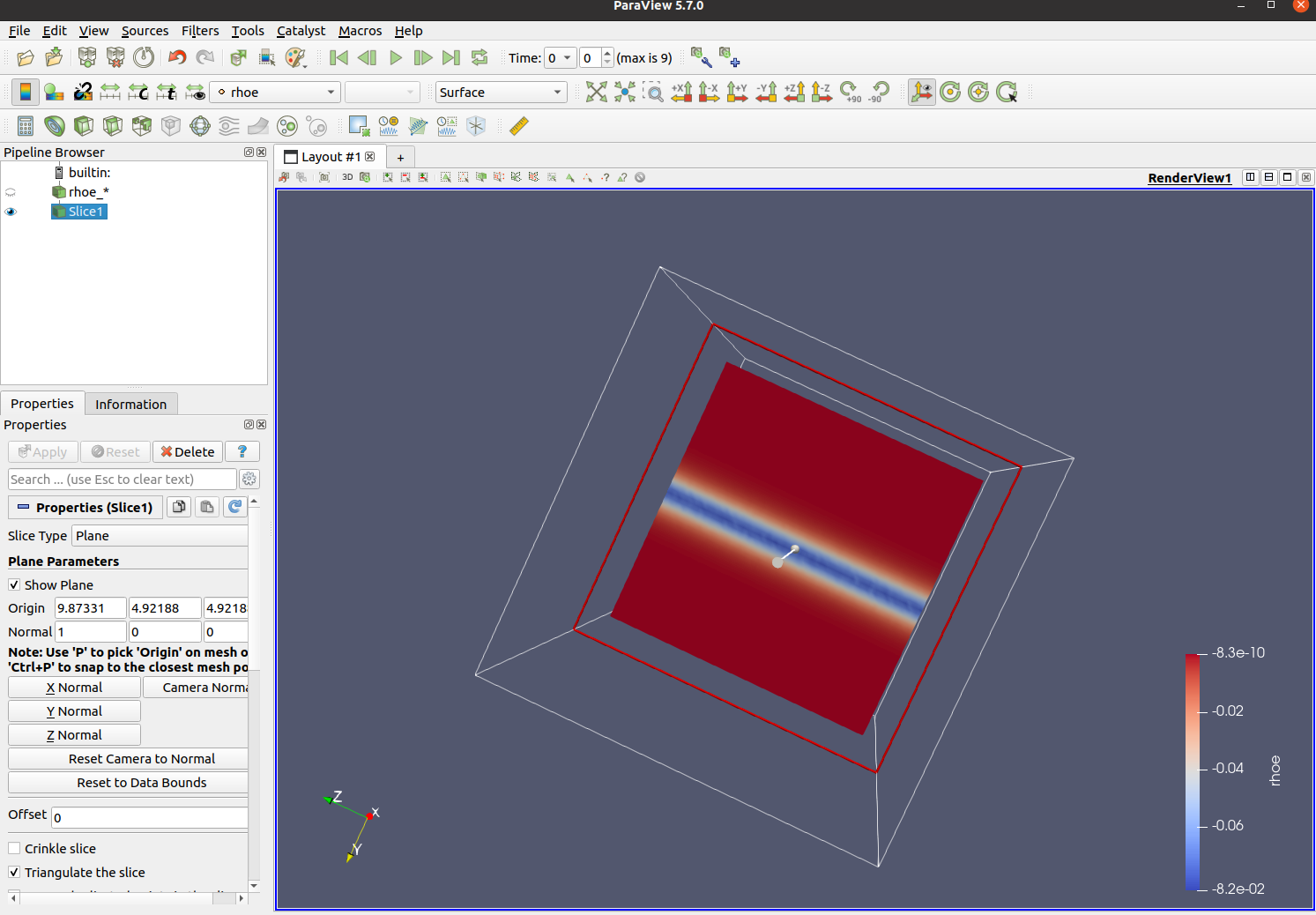


It’s different in the middle of volume. Seems clearer.

**To slice data:**



**data sliced:**



**show with rainbow desaturated:**

