

Turbulent Flow Simulation on HPC-Systems - Worksheet 1: Getting Started

| **Name:** *Batuhan Erden* | **Matriculation Number:** *03738750* |

Part 1: VTK Stencil

In the **apply** function, given a cell, we are expected to get the pressure (*at the center of the cell*) and velocity values (*average across the cell*) from **Flow Field**. After getting those values, I stored them in the object by making use of the **vector** data structure in order to access the values of the particular cells during the **write** operation. I also needed to know the **First Corner**, which is the start of the *White Region*. For that reason, I simply kept the first iterated *cell indexes*. You can find the implementation of the **apply** function [here](#).

As a second step, I have implemented the **write** functionality, which writes **VTK** files with a time information (in *microseconds*) appended to the filename. For that I have created separate function for *writing the nodes/points*, *writing the pressures* and *writing the velocities* for readability purposes. For keeping this report as short as possible, I decided not to include the code blocks. However, feel free to check it [here](#)!

Keep in mind that we have more nodes than we have cell values because **VTK** differentiates between vertex (or node) and cell values, and we use quadrilateral or octahedral cells on a uniform grid. For example, if we had $20 * 10 = 200$ cells in a **2D** space, we would have $(20 + 1) * (10 + 1) = 231$ nodes.

The functions **getPosX** and **getDx** are utilized to write the points.

Part 2: Handling the Ghost Cells and always working in the White Region:

As you might remember from the lecture, we have boundaries around our workable region (*for ease of coding and parallelization*). As can be seen in the picture [here](#), in addition to the *default boundary* where the workable region is covered by a single cell space, we also have some additional ghost cells that we need to take care. For that reason, I have used the **lowOffset** and **highOffset** parameters of the **Iterator** class, which is *bounded iteration* where by default, the *1-layer of default boundary* is already handled. That's why instead of setting **lowOffset** to **2** and **highOffset** to **1**, I set them **1** and **0** respectively. After setting those values correctly, the **Iterator** now iterates only in the *White Region*.

Part 3: Plotting the VTK when the "time" is right!

- **dt_convergence** = adaptive time stepping, always changing.
- **dt_vtk** = time step size given in the parameters.
- **dt**, the actual time step.

We always wish things that cannot always happen, like having **dt_convergence** and **dt_vtk** same all the time! Since we are required to plot **every 0.01 seconds (or whatever is set in the parameter)**, I had to handle the cases where **dt_convergence** and **dt_vtk** are not the same:

1) **dt_convergence = dt_vtk**

Everything works as expected!

2) $dt_{convergence} > dt_{vtk}$

dt_{vtk} acts as an upper bound for dt . We simply set $dt = dt_{vtk}$ here.

3) $dt_{convergence} < dt_{vtk}$

The most problematic case. In this case, we are doing as many time steps as we can until we reach the next point in time to plot.

Part 4: The results on ParaView

I have used ParaView to check my results. I have both printed 2D and 3D and analyzed both pressure and velocity. To keep the report short, I have not included any images here. Please check [here for pressure](#) and [here for velocity](#).

Part 5: Flow Physics and Profiling (Questions Answered)

1) Study of the scenarios cavity, channel and channel with backward-facing step in more detail:

- ***The influence of the Reynolds number on the velocity and pressure field. In particular,***
 - ***How many vortices can you observe in the cavity scenario, depending on the choice of the Reynolds number? Where are these vortices located?***

As we increase the *Reynolds number*, the vorticity distribution seems more homogeneous. That is, the vortices are distributed more evenly throughout the surface. In contrast, lower *Reynolds number* resulted in more inhomogeneous distribution where the vortices lie mostly around the borders.

- ***How does the velocity/pressure field of the channel flow depend on the Reynolds number?***

As can be seen in the graph [here](#) where **Velocity (Orange)** is plotted over **Pressure (Red)** with the *Reynolds number* set to **500 (Left)** and **100 (Right)**, it is not hard to deduce that setting the *Reynolds number* high resulted in a higher **velocity/pressure field** as **velocity** values got bigger while **pressure** values got smaller. In a nutshell, **velocity/pressure field** and the *Reynolds number* has a positive correlation.

- ***How is the velocity/pressure field behind the backward-facing step affected by the choice of the Reynolds number?***

As can be seen in the graph [here](#), the results look similar with that of channel flow. Again, we can say that **velocity/pressure field** and the *Reynolds number* has a positive correlation.

- ***The influence of the length and height of the backward-facing step on the flow field.***

As can be seen in the screenshot [here](#), I have tested different length and height settings of the backward-facing step. It definitely has an influence on the flow field. When we look at the flow field at the top (the default one), we can see that the pressure is getting closer to **0.8**. However, when we look at the ones at the bottom, in both cases, the pressure gets to a point higher than default one.

On the bottom left, we have smaller length and height values. As can be seen easily, the pressure is getting closer to even **1.0** whereas on the bottom right, where we have larger length and height values, the pressure was lower (close to **0.9**).

- ***The influence of the used mesh on the overall accuracy and time step size. You may restrict your considerations to the channel flow scenario for this sub-task.***

The mesh structure used has a significant effect on the overall accuracy and time step size. One can see [here](#) how the **stretched mesh (left)** converged faster (*More accurate!*) than **uniform mesh (right)**. According to the time step, **stretched mesh (left)** has a **dt** of 0.000228466 whereas **uniform mesh (right)** has a **dt** of 0.0496032, which is higher.

2) Investigation of the sequential performance using gprof:

As can be seen in the output of the **gprof** command (*Performance analysis for Cavity 2D*) [here](#) (for more **gprof** results, see [here](#)), the case is not as expected. Actually, the **write** routine takes absolutely no time compared to the other routines. I ran the performance analysis for each of the example cases, and the results are not that different and the routines that takes most of the time are the same.

For the Cavity2D case, the **apply** routine took 37,50% of the time followed by **getVelocity** that took 25,00% of the time. When I checked my implementation of the **apply** routine and the implementation of **getVelocity**, I saw no possible improvements since I believe they are implemented in the best way possible. However, as you can see under the *calls* section, there are too many calls to those routines. **apply** is called **231231** time and **getVelocity**, which gets called under **apply**, is called **3051048** times. So, we immediately can see there is room for improvement. We can for example make use of the **MPI** to parallelize the simulation so that multiple processes work on those *bottleneck* routines.

I thank you for the opportunity of gaining hands-on experience with such algorithms, and I truly appreciate your effort.

Batuhan Erden