Homework 2

Due Monday, April 15th at 11:59 PM

1 Problem 1: Channel Flow with a central narrowing

Simulating channel flow in 2D with a narrowing is the target.

The type of flow in absence of the narrowing is the well-known Hagen-Poiseuille flow (and by the way, Poiseuille was a physicist and physiologist who studied blood flows in a systematic way). Let's write the corresponding grid for a straight channel aligned along, say, the x axis.

When constructing the grid and using it in the LBM context you should apply what you have learned about grid management. Let's recall that in absence of open boundaries (inlet or outlet), each fluid node should always be surrounded by other fluid or wall nodes to preserve mass and momentum locally (no leakage rule).

The solution should employ two different options:

1. A periodic channel (aligned with the x direction): start from a quiescent fluid and apply a body acceleration uniformly through the fluid until you match the Reynolds number at steady state.

2. A non-periodic channel with inlet and outlet faces and by applying a uniform velocity at inlet and outlet with plug flow profiles.

Use no-slip boundary conditions at the walls and a Reynolds number of 10^{-2} computed from fluid velocity imposed at $x = 0$ (the inlet region for non-period channel) and characterstic length being H.

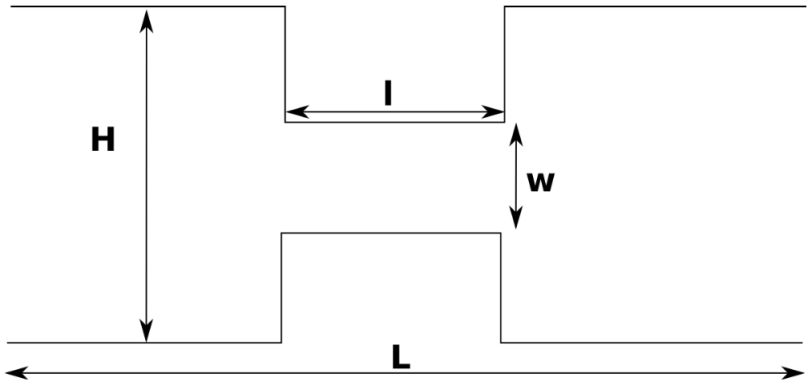

Figure 1: The geometry for the channel with central narrowing.

Solution

We first need to ensure a Reynolds number of 10^{-2}.
The Reynolds number is computed by

$$Re = \frac{uH}{\nu}$$

The characteristic length $H = 60$ here. u is set to be the velocity at $x = 0$. ν is the kinematic viscosity, computed from

$$\nu = c_s^2(\frac{1}{\Omega} - \frac{1}{2})$$

where $c_s^2 = \frac{1}{3}$ is sound speed squared. Ω is the relaxation frequency in BGK operator. In our code, we set $\Omega = 1$, indicating a full-relaxation scheme (except for the last question that requires changing Ω). Thus $\nu = \frac{1}{2}c_s^2 = 1/6$.
The desired u can be computed from

$$\begin{aligned} u &= \frac{Re \cdot \nu}{H} \\ &= \frac{0.01 \cdot 1/6}{60} \\ &= 2.8 \times 10^{-5} \end{aligned}$$

For periodic boundary (case 1), we tweak forcing so that the steady $u(x = 0)$ is around 2.8×10^{-5}. We find the forcing to be 1×10^{-8}. For non-periodic boundary, we directly apply this velocity to the boundary grid points. Both cases achieve steady state in about 8000 time steps.

For a fixed set of values for the geometry as in Fig. 1, (suggested values $L = 200$, $H = 60$, $l = 50$, $w = 30$), calculate the following quantities:

1. The pressure field
2. The velocity field
3. The volume flow rate
4. The average and maximum velocities in the channel

Solution

The pressure is computed from

$$p = c_s^2 \rho = \frac{1}{3}\rho$$

Fig 2 and Fig 3 show the macroscopic fields with the periodic (case 1) and non-periodic (case 2) boundary conditions.

Periodic Boundary Conditions

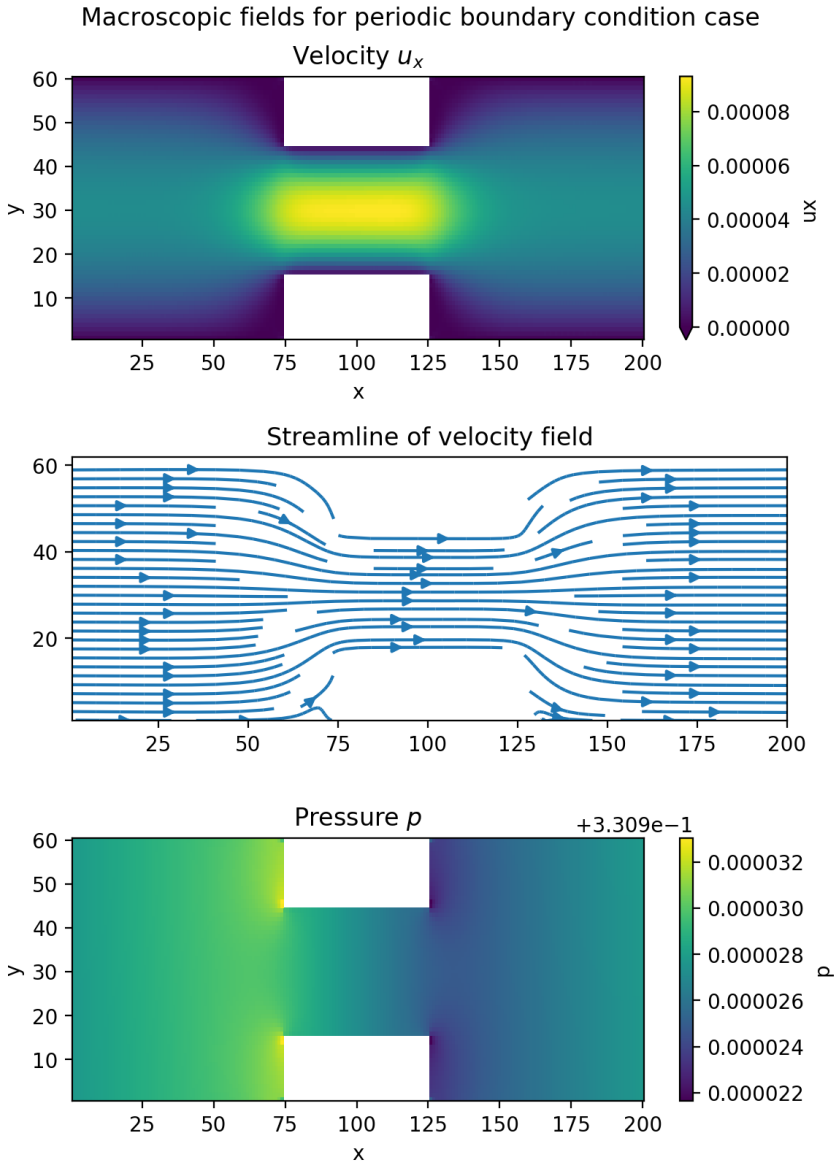

Figure 2: Macroscopic variables for periodic boundary case

Solution

Imposed Boundary Conditions

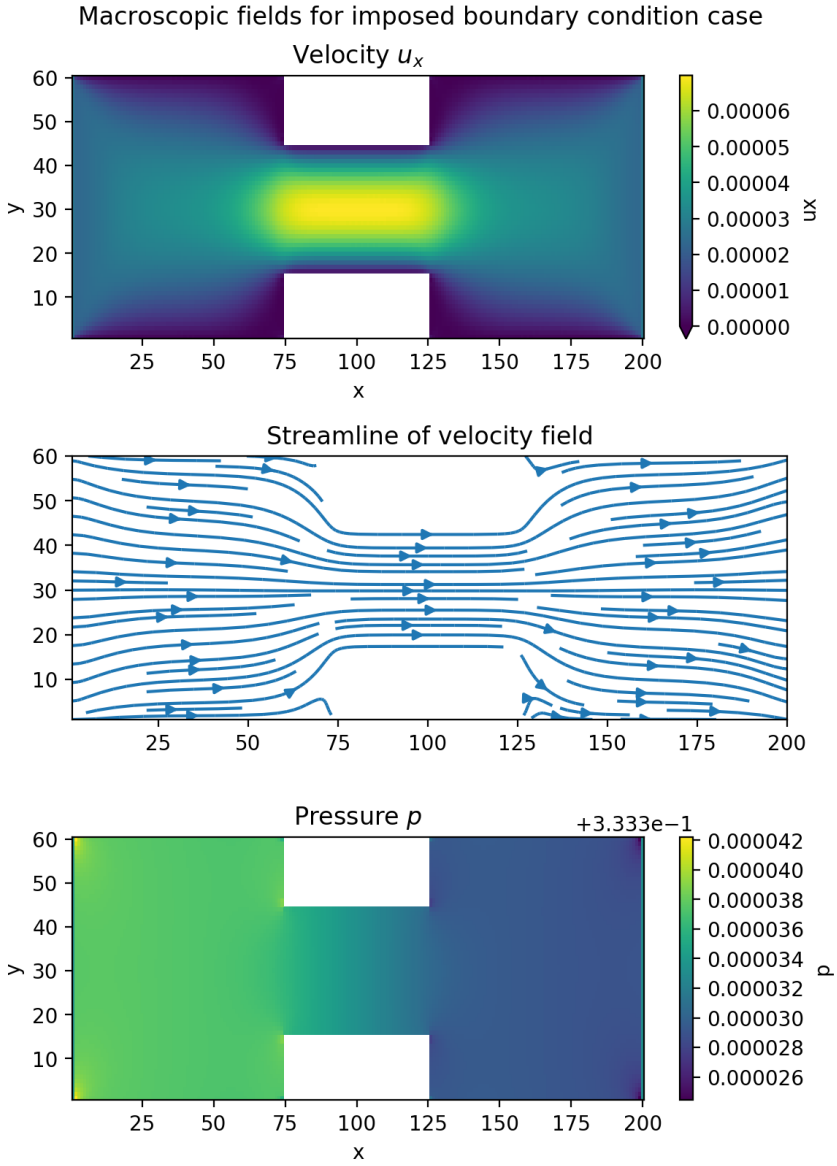

Figure 3: Macroscopic variables for non-periodic boundary case

Solution

Volume Flow Rate

The volume flow rates can be computed from the integral over x, on a specific cross-section $x = x_i$. Fig 4 and Fig 5 show the volume flow rate at all possible x, for two different boundary conditions. The rate is roughly constant at different x. For boundary case 1, $rate \approx 0.0018$; for boundary case 1, $rate \approx 0.0014$.

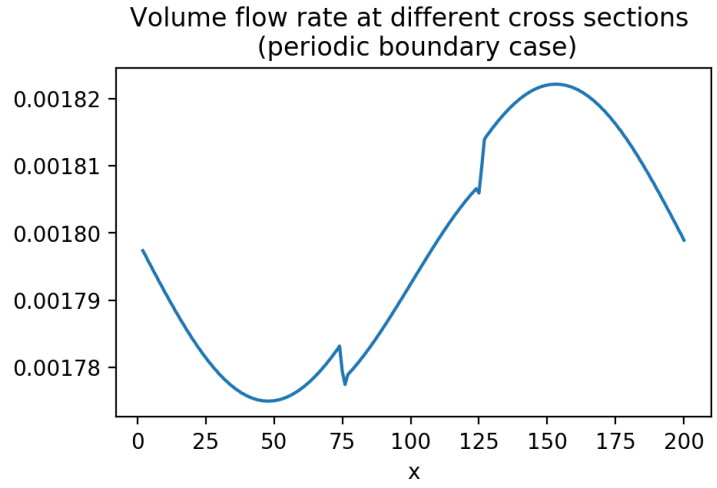

Figure 4: Volume flow rates for periodic boundary case

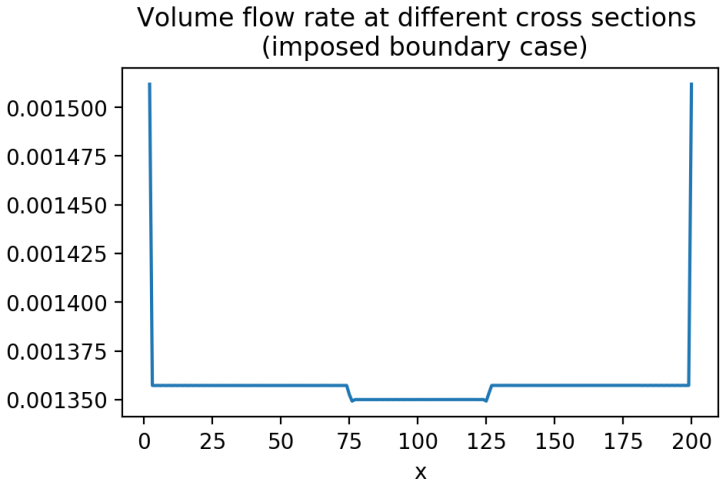

Figure 5: Volume flow rates for non-periodic boundary case

Solution

Average Velocity

The average velocity is 3.45×10^{-5} for periodic boundary and 2.61×10^{-5} for non-periodic. The maximum velocity is 9.30×10^{-5} for periodic boundary and 6.98×10^{-5} for non-periodic. u_y is orders of magnitude smaller than u_x and has negligible contribution to the total velocity magnitude.

We can further plot the average and maximum velocity at each cross-section $x = x_i$, as shown in Fig 8 and Fig 9. This velocity is significantly higher inside the narrowing region, consistent with the continuity equation of fluid. This result also matches our intuition of the system.

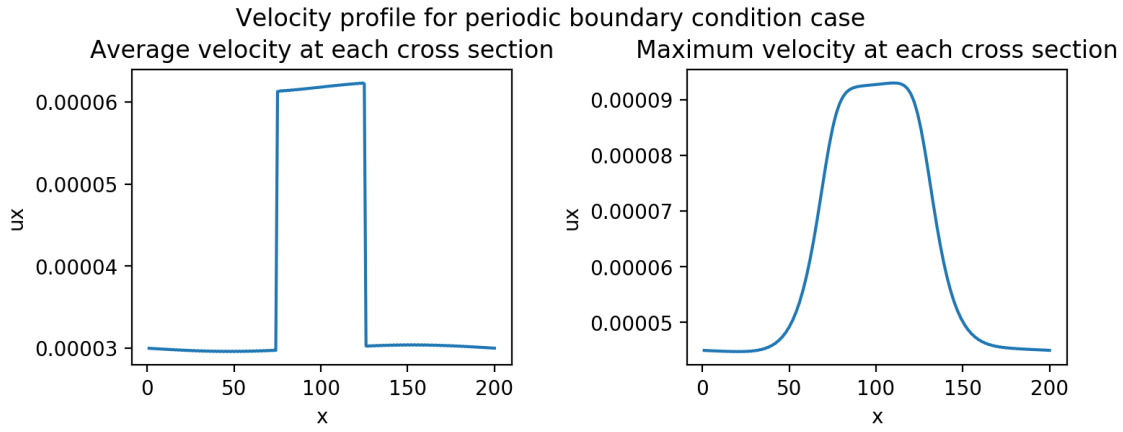

Figure 6: Average and maximum velocity at each cross-section, for periodic boundary case

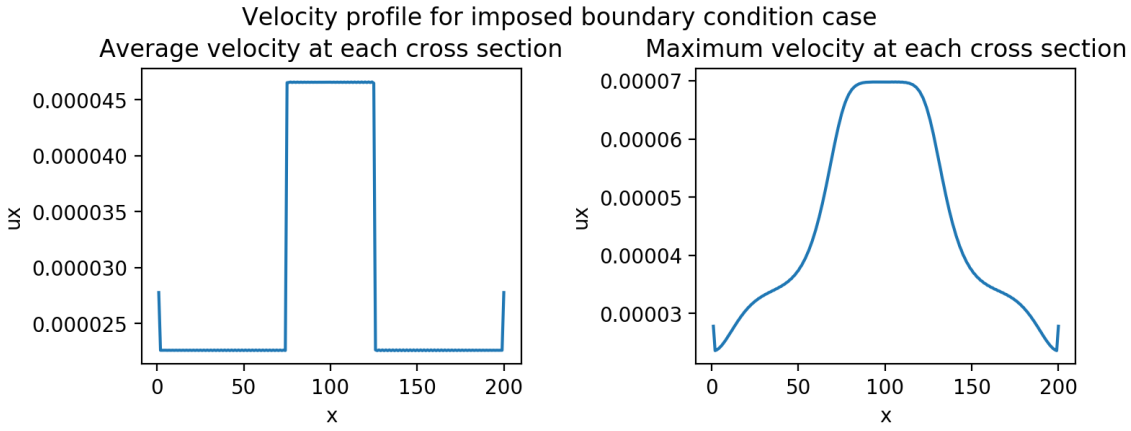

Figure 7: Average and maximum velocity at each cross-section, for non-periodic boundary case

Solution

We can further check the velocity profile at each cross-section. For 2D Poiseuille flow without walls, the analytical solution is known to be a parabola. With walls, the shape should still be parabola-like, as confirmed by Fig 8 and Fig 9. Inside the narrowing region ($x = 100$), the parabola is the sharpest. The only non-parabola-like profile is the non-periodic boundary case at $x = 0$, because we impose uniform velocity at the boundary.

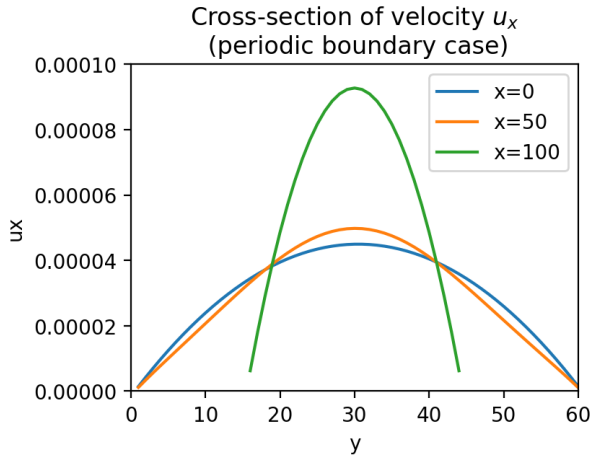

Figure 8: Velocity profile at three cross-sections, for periodic boundary case.

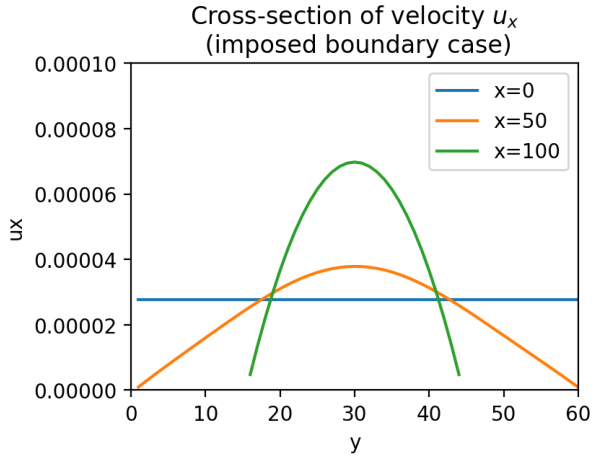

Figure 9: Velocity profile at three cross-sections, for non-periodic boundary case.

Repeat the simulation by varying the width of the narrowing w.
Plot: Show how the flow rate changes by varying w in the range $0 < w < 50$

Solution

Running the simulation with varying sizes of $w = 10, 20, 30, 40, 50$, we obtain the following velocity profiles. The steady state velocity u is shown in Fig 10 and Fig 11. The periodic cases has parabola-like velocity profile at the boundary, while the non-periodic we imposed to have uniform profile over y near boundary.

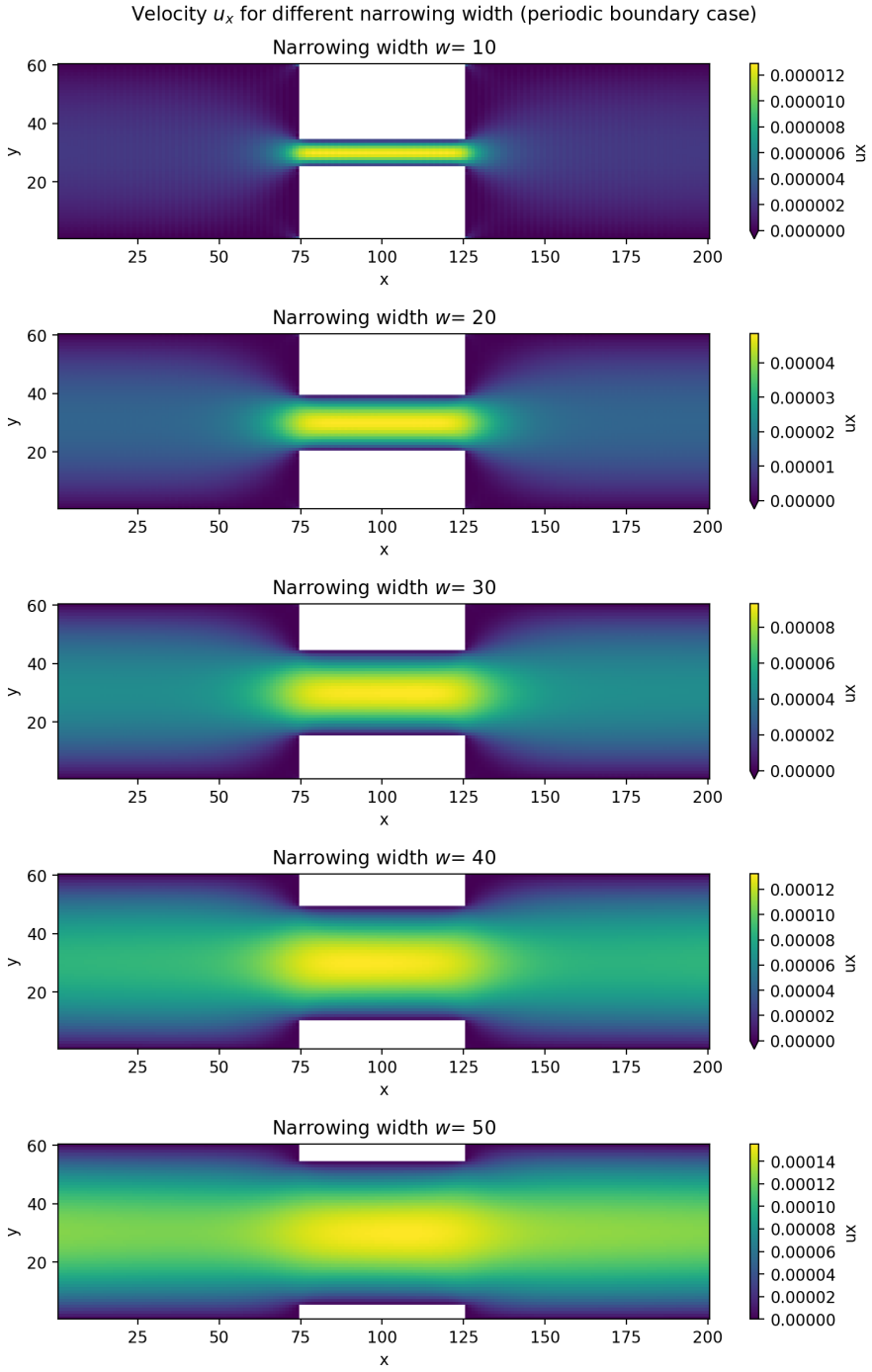

Figure 10: Velocity field with different narrowing width, for periodic boundary case.

Solution

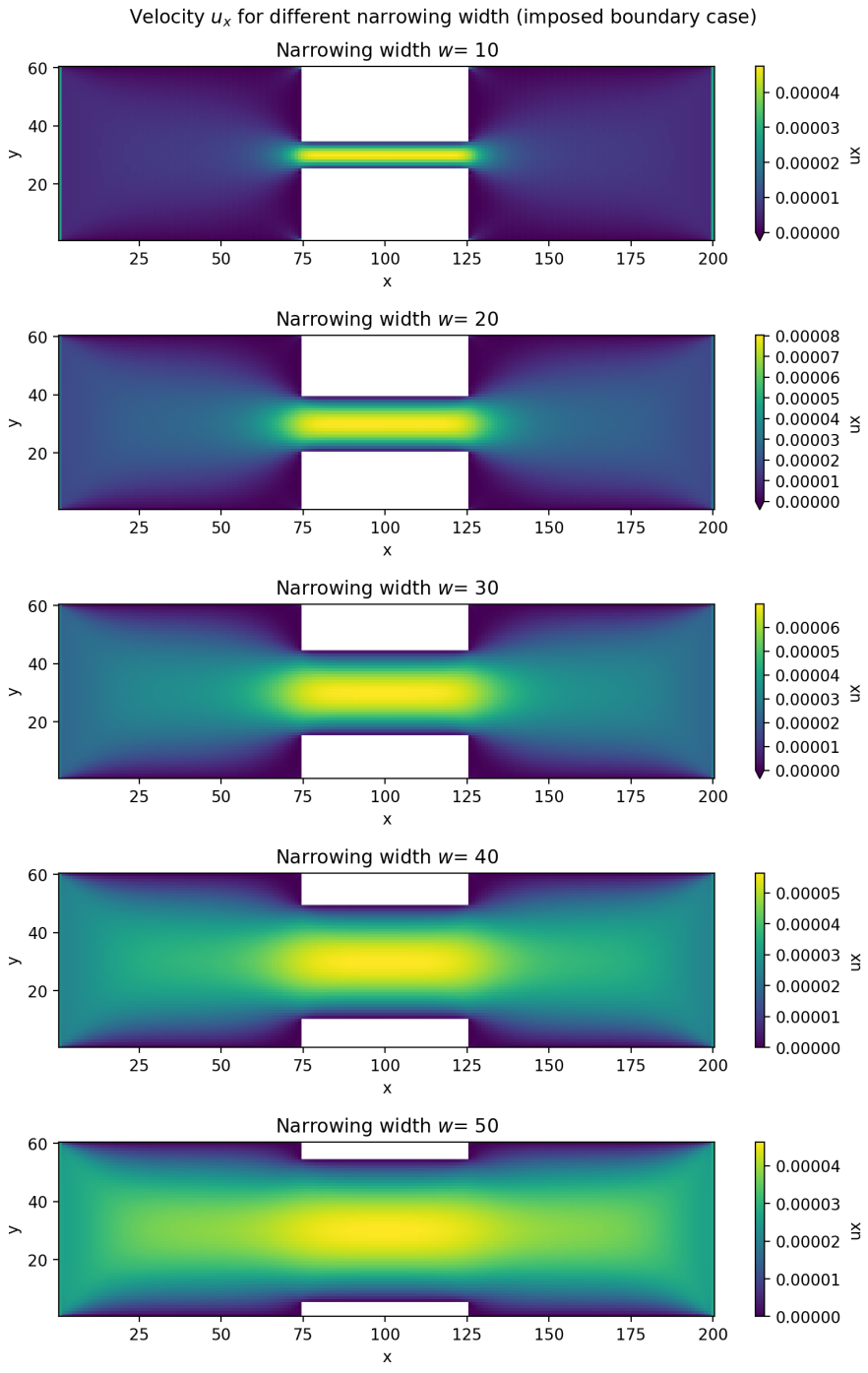

Figure 11: Velocity field with different narrowing width, for non-periodic boundary case.

Solution

We then compute the volume flow rate by integrating over y and then averaging over x. The rates for different w are shown in Fig 12 and Fig 13. In both cases, the rate increases with the channel width, as expected.

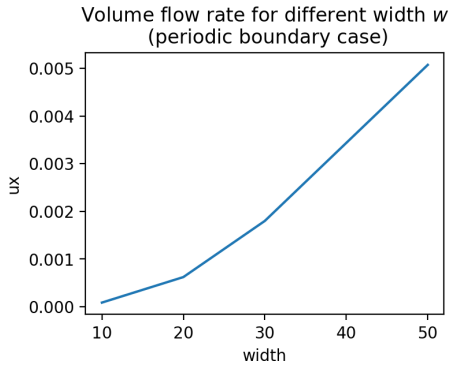

Figure 12: Volume flow rate (averaged) with different narrowing width, for periodic boundary case.

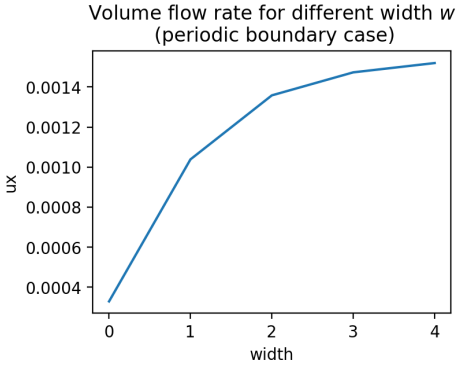

Figure 13: Volume flow rate (averaged) with different narrowing width, for non-periodic boundary case.

Compare the pressure at the narrowing with the simple Bernoulli estimate for inviscid flows, stating that

$$\frac{1}{2}\rho u_o^2 + p_o = \frac{1}{2}\rho u_i^2 + p_i$$

where u_o and p_o are cross-sectional averages of velocity and pressure at $x = 0$ for a periodic system or at the inlet otherwise, and u_i and p_i are the same quantities at the center of the narrowing.

Plot: Plot the obtained pressure vs the Bernoulli estimate by varying the LBM kinematic viscosity in the range $0.05 : 0.1667$.

Solution

We find that the term $\frac{1}{2}\rho u^2$ is orders of magnitude smaller than the pressure p. This is because the velocity u is very small (a result of the very small Reynolds number). The pressure p is almost constant (only 0.002% perturbation) over the entire domain. This is demonstrated by Fig 14 and Fig 15, where we can see that $\frac{1}{2}\rho u^2 + p \approx p$. The pressure p is almost 0.166 everywhere, and the velocity perturbation is negligible. In this case, the Bernoulli estimate almost just uses p_0 to approximate p_i. The relative error is less than 0.002%.

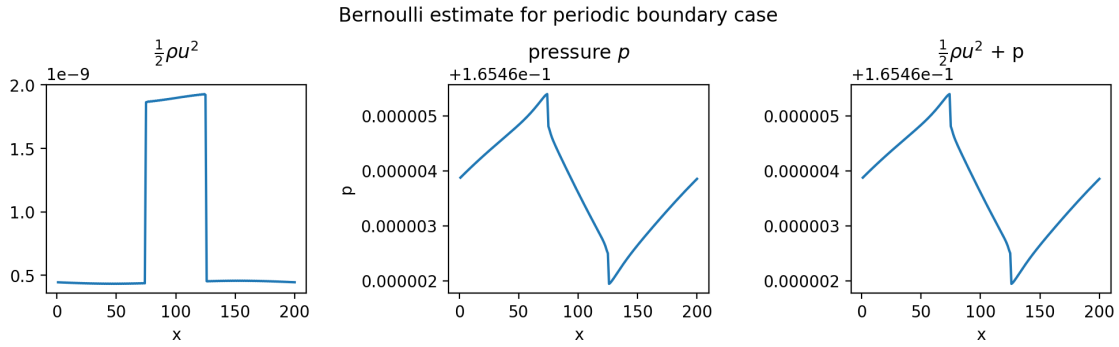

Figure 14: Major quantities used in Bernoulli estimate, for periodic boundary case.

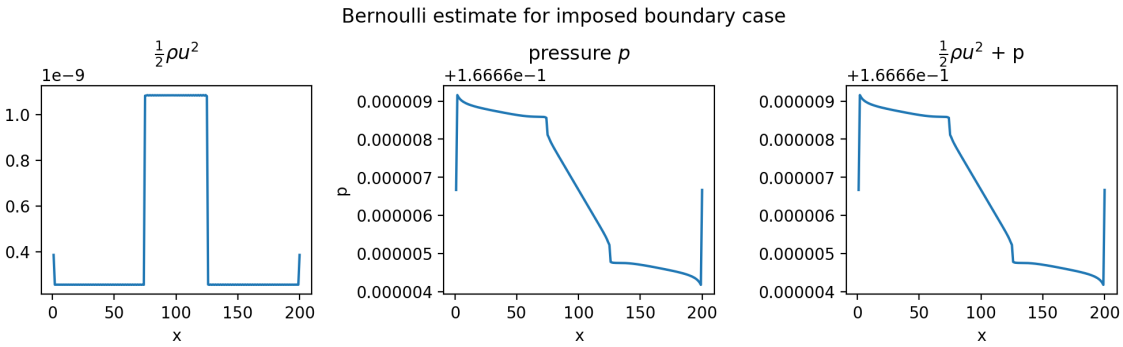

Figure 15: Major quantities used in Bernoulli estimate, for non-periodic boundary case.

Solution

Finally we vary the LBM kinematic viscosity by tweaking the relaxation parameters Ω. For all the previous cases we set $\Omega = 1.0$, thus $\nu = 1/6 = 0.1667$, which is the upper bound of ν. For the lower bound $\nu = 0.05$, the corresponding $\Omega = 1.54$. The velocity fields with different viscosity are shown in Fig 16. Smaller viscosity leads to higher velocity magnitude, as expected.

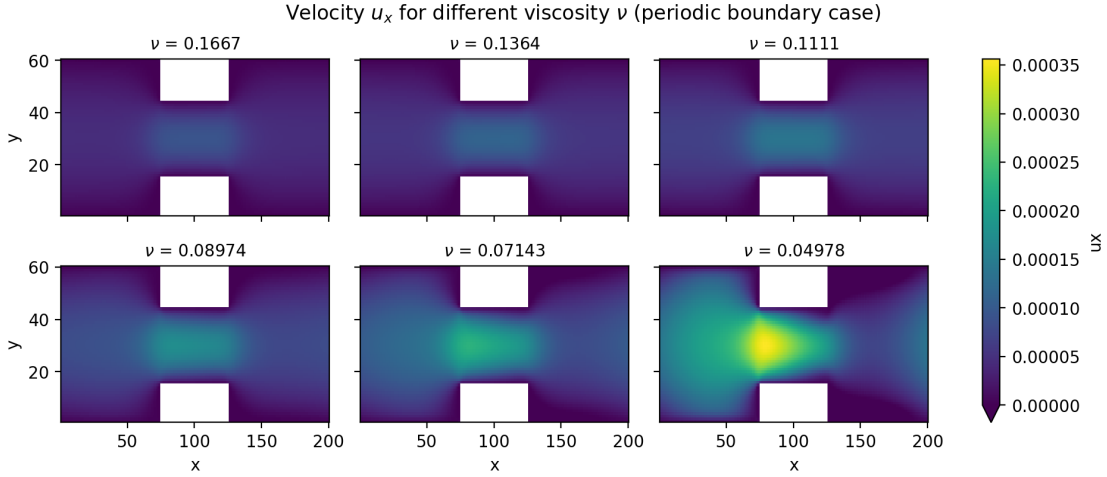

Figure 16: Velocity field for different viscosity ν. Here uses Periodic boundary. The non-periodic one has similar behavior.

For different viscosity ν, the Bernoulli estimate has similar behavior, as shown in Fig 16. The kinetic energy term $\frac{1}{2}\rho u^2$ becomes smaller with higher viscosity, as expected. The pressure term p, however, stays almost constant, and domintates the total team $\frac{1}{2}\rho u^2 + p$.

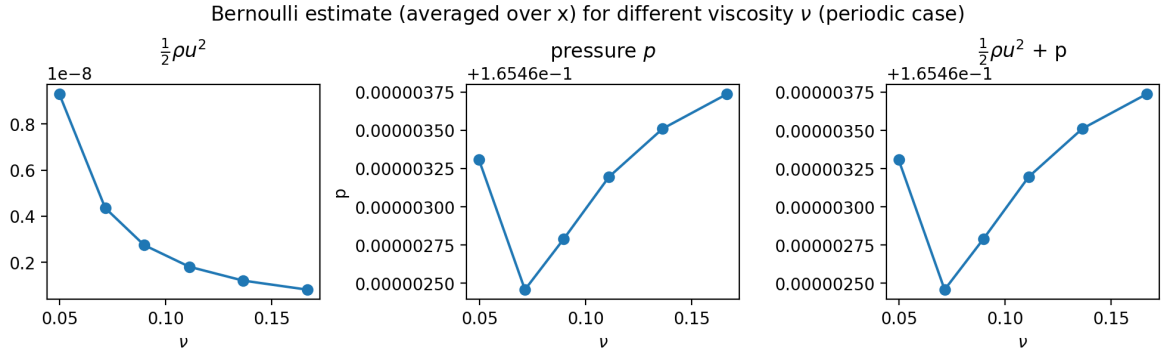

Figure 17: Major quantities used in Bernoulli estimate (averaged over x), for different viscosity ν. Here uses Periodic boundary. The non-periodic one has similar behavior.

Solution

All figures can generated by the Python notebook JiaweiZhuang/use_LBM_solver.ipynb.

Solution

In conclusion, we simulated a periodic channel and a non-periodic channel using the LBM. The results were computed rather quickly highlighting the efficiency of the LBM at producing reliable results when simulating flows.
To test the validity of our approach and design for implementing LBM we originally coded our design in Python. The results appeared accurate and held to the boundary conditions we constrained the problem to. Additionally, the organization that we used for the code follows that of Sauro Succi. We then translated our code and methods to C in order to achieve more efficient computational speeds enabling us to run our LBM to higher iteration numbers quicker for both cases. We interpreted the output of the C method in Python for better visualization using various libraries.
The periodic channel can be found on the file "LBM_2D_channel_v1.c" and the continuous channel can be found in the file "LBM_2D_channel_v2.c". In the files one can change various parameters associated with the environment that the flow is in such as H, L, w, l, the forces, etc.

2 Problem 2: Basic CUDA and GPUs

This problem was submitted by Dan Willen `daniel.p.willen@gmail.com`

Consider the equation

$$\frac{\partial u}{\partial t} = D\nabla^2 u, \tag{1}$$

on the domain $0 \leq x, y \leq \pi$, with $u = u(x, y, t)$, D the diffusive constant, and ∇^2 the Laplace operator. The boundary conditions are of the homogeneous Dirichlet type:

$$u(x = 0, y) = u(x = \pi, y) = u(x, y = 0) = u(x, y = \pi) = 0, \tag{2}$$

and the initial condition is

$$u(x, y, t = 0) = \sin(x)\sin(y). \tag{3}$$

The solution to this equation is

$$u(x, y, t) = \sin(x)\sin(y)\exp(-2Dt) \tag{4}$$

Using a second-order central difference in space and first order forward difference in time, the discretization of (1) is

$$u_{i,j}^{(n+1)} = u_{i,j}^{(n)} + \frac{D\Delta t}{\Delta x^2}\left[u_{i+1,j}^{(n)} + u_{i-1,j}^{(n)} + u_{i,j+1}^{(n)} + u_{i,j-1}^{(n)} - 4u_{i,j}^{(n)}\right], \tag{5}$$

where $u_{i,j}^{(n)}$ is the value of u at the n^{th} time step at grid point (i, j), $\Delta t = \Delta x^2/4D$ is the time step size, and Δx is the grid spacing in the x and y directions.

1. Using Cuda, solve the discretized equations up to a time $t = \pi^2/D$ using the Jacobi method. A skeleton code is provided to assist you, with comments in the locations you should make changes.

Step I – Declare, allocate, and initialize memory for the field variable `u` on the CPU. You should allocate enough memory for the grid, which has size $nx \times ny$ and initialize `u` to the initial condition. Make sure you free the memory at the end of the program.

Step II – Declare and allocate the GPU memory for `_u`, `_u_new` and `_error`. Copy the CPU memory to the GPU; the other two arrays have been initialized to zero for you. Make sure you free the memory at the end of the program.

Step III – Set up the kernel execution configuration for the GPU kernel based on the input domain size and the maximum threads per dimension, which is set at the top of the file as a `#define`. You will need to determine the number of threads per block and the number of blocks in each direction, as well as set up the `dim3` variables.

Step IV – Write a GPU kernel that advances to the next timestep using the Jacobi method. This should be done in parallel, not in serial.

Step V – Write a GPU kernel that calculates the error between the numerical and analytic solutions at each point. Be careful to compare solutions at the correct timestep – `_u_new` is at $t = (n+1)\Delta t$ and `_u` is at $t = n\Delta t$. Using this result, a parallel reduction using the Thrust parallel algorithms library has been provided to calculate the total error in order to find the average percent error at each grid point.

Step VI – At the end of the loop, copy the data back to the CPU.

Your program should take as an input the number of grid cells on one side of the domain. This has been set up for you such that the program can be run from the command line like:

```
./jacobi_solver.cu n
```

where $nx = ny =$ `n` is the size of one side of the square domain. The program should output the percent difference between your result and the analytic solution, averaged over all of the grid nodes:

$$\epsilon = \frac{1}{n_x n_y} \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \frac{u_{i,j}^{(n)} - U_{i,j}^{(n)}}{U_{i,j}^{(n)}}, \tag{6}$$

where U is the analytic solution.

If you have time, discuss the following:

- How does the error change as you increase the resolution? Does this behavior make sense?
- How does the runtime scale with the resolution? You can get a rough estimate by using the bash command `time` when executing your program, like:

  ```
  ./jacobi_solver.cu n
  ```

  and using the result for `real`.

- How does the runtime change as you change the kernel execution configuration? e.g., play around the `MAX_THREADS_DIM` parameter as well as different schemes for setting up the thread blocks.
- Use shared memory in the Jacobi kernel

Solution

We developed the code based on the skeleton provided in `parallel.cu`. In addition to the basic requirements stated in the 5 steps in the problem statement, we used the constant memory and the shared memory to accelerate the computation. In particular, the constant memory is applied to store the constant variables such as `nx` and `ny`, and the shared memory is applied to cache the field variable `_u` as each of its element would be accessed for multiple times by different threads in one Jacobi iteration. Our completed code has been uploaded here.

The code is compiled and tested on Odyssey, the HPC cluster maintained by Harvard FAS Research Computing. We observed how the performance scale with the resolution as well as the block dimension. The results are displayed in Figure 18 and Figure 19.

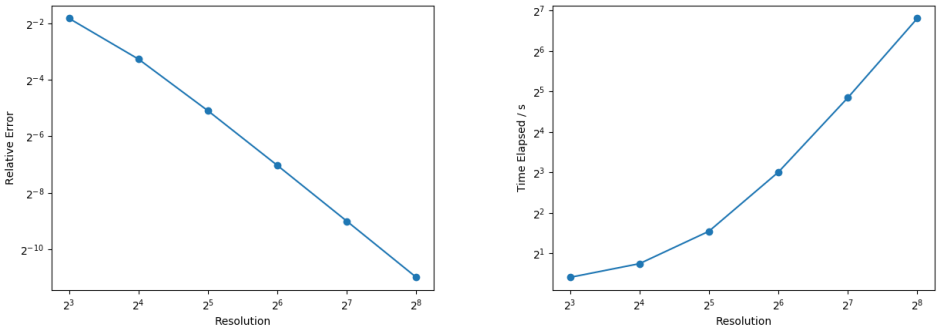

Figure 18: Performance scale with the resolution. `MAX_THREAD_NUM`=18.

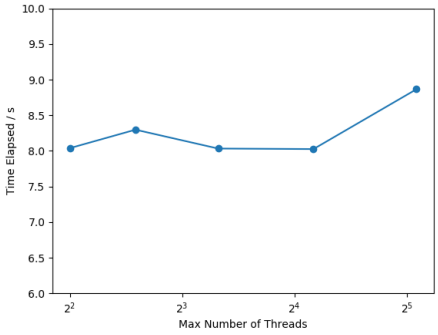

Figure 19: Performance scale with the block dimension. The grid is 64×64.

As we enhance the resolution (denoted by the grid size $N \times N$), we see that the relative error shrinks approximately with $O(\frac{1}{N^2})$, and the elapsed time increases approximately with $O(N^2)$, which makes sense for our Jacobi iteration setup. On the other hand, the performance does not seem to vary a lot with block dimension. Due to hardware restriction, we cannot try `MAX_THREADS_DIM` $\geq 2^6$. Based on the trend in Figure 19, the runtime might increase with larger block dimensions.