A Python Implementation in Graphic Processing Unit of a Lattice Boltzmann Model for Unstable Three-dimensional Flows in Immersed Permeable Media

Gustavo Boroni[1], Nicolás Silin[2], Alejandro Clausse[1,3,4]

[1]CONICET and National University of Central Buenos Aires, 7000 Tandil, Argentina.

[2]CONICET and Instituto Balseiro, 8400 Bariloche, Argentina.

[3]Comisión Nacional de Energía Atómica, Libertador 8250, 1429 Buenos Aires, Argentina.

[4]Corresponding author, Email clausse@exa.unicen.edu.ar

ABSTRACT

The implementation of a lattice Boltzmann model for three-dimensional (3D) permeable media with localized drag forces is presented. The model was previously introduced for two-dimensional (2D) geometries and follows the basics of the immersed boundary method. Permeable flows are much less stable than their counterparts in porous media and generally produce large coherent flow structures, like vortex lines, rolls, and wakes. Also, in permeable media the small-scale geometry often needs to be represented to a high degree of detail in order to capture certain transport phenomena, like micro-convection or pollination. Hence, both, calculation speed and memory requirements are under strain. The present model was implemented in a Graphic Processing Unit (GPU) showing excellent performance in the calculation of stable and unstable flows in a rectangular channel partially obstructed by an array of parallel wires. In particular, the model is able to deal with small and medium spatial scales without losing the heterogeneous nature of permeable flows in the homogenization process. The algorithm to manage memory issues is described in detail, and the results of the test case for stable and unstable conditions show the capability of the method to simulate this type of flows.

I. INTRODUCTION

The Lattice Boltzmann method (LBM) has become a popular solver of the Navier-Stokes equations, particularly in the incompressible range at low and moderate Reynolds numbers. It has been successfully applied to solve oscillatory flows, heterogeneous geometries and multiphase flows, among other flows. A recent update on the advances in LBM can be found in Succi (2018). From the numerical perspective, LBM is a fully explicit method to solve transport equations using more variables than the strictly necessary to characterize the macroscopic flow. A set of population functions define the local state at an appropriate mesoscale, which represent the fluid dynamics via discrete kinetic equations, emulating molecular advection and interactions. Selected averages of the population functions are then construed as the macroscopic field variables, such as velocity, pressure and shear stress. In particular, it has been proved (Succi, 2018) that LBM recovers the Navier-Stokes

equations to second order accuracy, by means of the BGK approximation (Bhatnagar et al.,1954) and Chapman-Enskog technique (Chapman and Cowling, 1970).

Fluid flow in heterogeneous highly permeable media has a wide range of applications, from heat exchange structures to pollination, forestation and water resources management. Permeable flows are much less stable than its counterparts in porous media. However, when the interaction between the fluid and sparse solid structures is characterized by a low Reynolds number, the permeable medium usually damps considerably the short wavelength fluctuations, and brings about instead large scale resonant coherent flow structures, like vortex lines, rolls and wakes (Chang and Constantinescu, 2012; Ledda et al, 2018). Much effort has been devoted to understand the characteristics of wakes behind permeable obstructions, like disks (Cummins et al., 2017) and arrays of cylinders (Tang et al., 2019). In these cases recirculation bubbles detaching from the obstructions and eventually disappearing have been reported, which is a phenomenon that is not observed in solid obstructions. The importance of the small scale phenomena in complex obstruction has been stressed by previous researchers. Bem Meftah and Mossa (2013) developed a theoretical model of the turbulent flow within a square array of objects. Tang et al. (2020) calculated numerically the flow in presence of a square cylinder array, investigating the interplay of instabilities at large and small scales.

Unlike porous media, the application of LBM in permeable media has much less development. One of the reasons of this is that in most of the ranges and geometries of interest, the flow is unstable and develops sustained oscillations, which requires longer calculations given the usually short time steps required by LBM to simulate real fluids. The other barrier is the need to represent the small scale geometry to a certain degree of detail, which is relevant for certain transport phenomena, like micro-convection or pollination. Hence, both, calculation speed and memory requirements are under strain.

Nevertheless, much of the methods developed to simulate porous media with LBM can be arguably extended to permeable media. Actually, the field-particle dual nature of LBM is of much advantage, since the particle-like dynamics fits elegantly to emulate preferential channeling typical of interstitial regions, whereas the field averaging homogenizes the small-scale information in a consistent fashion. These assets make LBM an excellent numerical framework for the treatment of flows in permeable media, which has progressively been extended to systems of increasing complexity, accompanying the growth of computing power. As a recent reference review states (Succi, 2018): “Indeed, the modeling of multiscale-heterogeneous media, whereby the local transport coefficients, such as the permeability, change from place to place, and from scale to scale, still poses an outstanding computational challenge.”

The popular way to represent a porous medium in LBM is to couple the Boltzmann transport equation to some effective medium, whereby the spatial cells are provided with Darcy-like constitutive drag laws (Babu and Narasimhan, 2010). This generally requires a

resistance tensor which turns into an effective anisotropic volume drag force (Kang et al, 2002). Alternatively, the method of gray nodes treats the porous cells by means of partial boundary conditions, bouncing back only a fraction of the populations (Zhu and Ma, 2013). In a recent version of this model a fraction of the population is bounced back and another fraction is allocated to the null-velocity population, which proved to be more robust than the volume drag force scheme (Zhu and Ma, 2018). The mass-conserved volumetric LBM (Yu et al., 2014; An et al., 2017) is another proposed scheme for porous media, where the cells are categorized by means of an effective pressure whose evolution is dependent on the solid volume fraction of each cell. There are also numerous variants implementing similar effective laws (Guo and Shu, 2013).

This work presents a LBM model that simulates the permeable medium with localized drag forces, which are designed following the immersed boundary method (Peskin, 2002). The method was introduced in previous works for two dimensional geometries, where it was shown that it is capable of reproducing oscillatory flows with great accuracy and acceptable computational costs (Boroni et al., 2015; Clausse et al., 2019). However, since most real case scenarios of permeable flows involve three dimensional phenomena (e.g., vortex lines bending), it remained to extend the model to a full three-dimensional (3D) scheme. The model was implemented in a Graphic Processing Unit (GPU) showing excellent performance in the calculation of stable and unstable flows in a rectangular channel partially obstructed by a permeable medium. The permeable medium represents an array of parallel wires, perpendicular to the flow direction, which can produce both stable and oscillatory flows.

II. LATTICE BOLTZMANN MODEL OF LOCALIZED PERMEABLE MEDIA

The Lattice Boltzmann model of immersed permeable media (LBIPM) was introduced in a previous paper for two dimensional domains (Boroni et al, 2015; Clausse et al, 2019). In this section we present the direct extension of LBIPM to 3D geometries. LBIPM is designed to simulate flows in permeable media by means of localized drag forces, which are used to model immersed sparse structures.

It is well known that LBM is a numerical method to solve transport equations that is ideal for parallelization, as it is fully explicit. The way the method manages to handle complex flows is by using more variables than the strictly necessary to characterize the macroscopic flow. This is achieved by means of a discrete kinetic representation supported on a regular grid. The basic form of LBM is the following:

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i \Delta x, t + \Delta t) = f_i(\boldsymbol{x}, t) - \frac{1}{\tau_i}[f_i(\boldsymbol{x}, t) - f_i^e(\boldsymbol{x}, t)] + S_i(\boldsymbol{x}, t) \quad (1)$$

where $f_i(\boldsymbol{x},t)$ and $S_i(\boldsymbol{x},t)$ stands for a distribution density and source at the grid coordinate $\boldsymbol{x}$ and time *t*, which is undergoing a spatial displacement $\boldsymbol{e}_i\Delta x$ in a time step Δt. The vectors $\boldsymbol{e}_i$ form a finite set of directions generally defined by a neighborhood in the underlying lattice. In the present study the D3Q19 model (Fig. 1) will be used, which has 19 displacements $\boldsymbol{e}_i$ labeled from 0 to 18.

The magnitude τ_i is a relaxation parameter which is used to control the viscosity and stabilize the scheme. Here we used the simplest version, where a single value τ is used for all directions. Eq. (1) approaches the Navier-Stokes equations provided that the equilibrium function $f_i^e(\boldsymbol{x},t)$ satisfies a set of constitutive conditions related to the moments of $f_i(\boldsymbol{x},t)$ respect to $\boldsymbol{e}_i$. A popular scheme complying with these conditions is:

$$f_i^e = \rho\omega_i\left[1+\frac{3}{e^2}(\boldsymbol{e}_i\cdot\boldsymbol{u})-\frac{3}{2e^2}(\boldsymbol{u}\cdot\boldsymbol{u})+\frac{9}{2e^4}(\boldsymbol{e}_i\cdot\boldsymbol{u})^2\right] \tag{2}$$

where $e=\Delta x/\Delta t$ is the so-called grid speed unit and:

$$\rho=\sum_i f_i \tag{3}$$

$$\boldsymbol{u}=\frac{e}{\rho}\sum_i f_i\boldsymbol{e}_i \tag{4}$$

are interpreted as the fluid density and the flow velocity. The coefficients ω_i are 1/3 for the resting particles, 1/18 for the Cartesian directions and 1/36 for the diagonal directions. In such case, the relaxation parameter τ is related to the kinematic viscosity of the fluid by:

$$\nu=(2\tau-1)\frac{\Delta x^2}{6\Delta t} \tag{5}$$

The permeable medium is represented by a drag force imposed in a neighborhood of each point in the domain that is part of an obstacle. The drag forces are incorporated following the immersed-boundary technique (Peskin, 2002; Boroni et al, 2015), via narrow, but smooth, spatial distributions $\delta_k(\boldsymbol{x})$ around each drag point $\boldsymbol{x}_k$, which does not need to be a node of the lattice. Each $\boldsymbol{x}_k$ introduces a Darcy volumetric force $\boldsymbol{F}_k$ upon its surround given by:

$$\boldsymbol{F}_k=-\frac{\rho\nu}{\kappa_l}\boldsymbol{u}_k \tag{6}$$

where $\boldsymbol{u}_k$ is the average velocity in the neighborhood of $\boldsymbol{x}_k$, and is defined as:

$$\boldsymbol{u}_k=\sum_{\boldsymbol{x}}\delta_k(\boldsymbol{x})\boldsymbol{u}(\boldsymbol{x}) \tag{7}$$

with:

$$\delta_k(\boldsymbol{x})=\mathrm{C}(r)\phi(x-x_k)\phi(y-y_k)\phi(z-z_k) \tag{8}$$

$$\phi(\xi) = \begin{cases} 1 + \cos\left(\frac{\pi\xi}{r}\right) & if \ \ |\xi| < r \\ 0 \quad otherwise \end{cases} \tag{9}$$

The functional form given by Eq. 8 was proposed and analyzed by Peskin (2002). $C(r)$ is a normalization factor ensuring that δ_k satisfies (Peskin, 2002; Uhlmann, 2005):

$$\sum_{x} \delta_k(\boldsymbol{x}) = 1 \tag{10}$$

The summations in Eqs. 7 and 9 are performed over all the cells of the grid, although the distribution δ_k restricts the effect only to the cells within the zone of influence of the reference point $\boldsymbol{x}_k$. Accordingly, Eq. 7, 8 and 10 leads to:

$$C(r) = \left[\sum_{n\geq -r}^{n\leq r} \left(1 + \cos\frac{\pi n}{r}\right)\right]^{-3} \quad ; \quad n \in \mathbb{Z} \tag{11}$$

Finally, the contribution of all the solid reference points $\boldsymbol{x}_k$ to the grid cell located at $\boldsymbol{x}$ is given by the source term:

$$S_i(\boldsymbol{x}) = 3\omega_i \Delta t \left[\frac{\boldsymbol{e}_i - \boldsymbol{u}(\boldsymbol{x})}{e^2} + 3\frac{\boldsymbol{e}_i \cdot \boldsymbol{u}(\boldsymbol{x})}{e^4}\boldsymbol{e}_i\right] \cdot \boldsymbol{F}(\boldsymbol{x}) \tag{12}$$

where

$$\boldsymbol{F}(\boldsymbol{x}) = \sum_{k} \delta_k(\boldsymbol{x}) \boldsymbol{F}_k \tag{13}$$

Eq. 12 is obtained by the second order expansion in Hermitte polynomials of the population functions (Shan et al., 2006).

The control parameters κ_l and r in Eqs. 6 to 11 are used to represent the obstacle specifics, like shape, size and roughness. κ_l is a local permeability that should not be confused with the macroscopic permeability of the medium, κ_m; and r is an influence length accounting for small-scale boundary layer effects (Clausse et al, 2019). An assessment of the relation between κ_l and κ_m is given in the Appendix.

Regarding the boundary conditions, here we follow the scheme proposed by Hecht and Harting (2010), where the unknown variables from outside the domain are defined by the local distribution functions and appropriate correctors.

III. GPU IMPLEMENTATION OF LBIPM D3Q19

The LBM has several advantages over traditional methods from the computational point of view, the main one is that the method is highly parallelizable, which makes it ideal for high performance applications on GPU and for easily dealing with some practical issues like

complex boundaries and multi-components (Succi, 2018). In the last decade there were several works reporting different implementation approaches to LBM on GPU. Bailey et al. (2009) presented an interesting implementation increasing the occupancy of a GPU multiprocessor and introducing an efficient storage method that greatly reduces GPU RAM requirements. Rinaldi et al (2012) proposed a single-step LBM algorithm with a reversed collision–propagation scheme that maximizes the GPU memory bandwidth. Tran et al (2017) achieved a high performance parallelization by minimizing the memory access playing with the cache locality. Wen and Ma (2019) applied a LBM-GPU implementation for high-resolution motion of smoke in real time. A review of different memory access patterns for LBM-GPU implementation can be found in Herschlag et al. (2018).

TABLE 1. Characteristics of the GPU NVIDIA Titan XP.

PARAMETER	**VALUE**
Memory size	12 Gb
Memory type	GDDR5X
Nominal memory clock	1426 MHz
Effective memory clock	11408 MHz
Memory interface width	384 bits
Memory bandwidth	547.58 GB/s

In the present work, the LBIPM D3Q19 model was implemented using a GPU NVIDIA compatible with the CUDA technology (NVIDIA®, 2020). The main parameters of the GPU are shown in Table 1. In this section, we describe the implementation of the algorithm LBIPM D3Q19, optimized for memory allocation efficiency and high speed computation. The structure of the implementation is based on an improvement of a previous implementation for the D2Q9 model, which requires much less memory (Boroni et al, 2015). The same matrix representation used in 2D was maintained, although special care was taken in minimizing the data transfer between GPU and CPU. This is particularly important because the aim of the tool is to simulate unstable 3D flows, where information during transients is relevant. In fact, one of the purposes of the present work is to show the management of information extraction related to the variations of large quantities of data generated by the GPU. Under certain conditions the optimum performance of the GPU is in conflict with the production of accurate calculations, and so there are compromises to be done between both criteria.

Although GPU calculations are much faster, certain tasks are more efficiently performed by the CPU. In order to avail ourselves of the advantages of both architectures, it was convenient to implement in the CPU all sequentially dominated functions typical of input-output management, like input preprocessing, output post processing, and data rendering and visualization. In turn, the GPU should be dedicated to perform the core calculations of the LBM model.

A. BASIC ALGORITHM

The basic algorithm of LBIPM D3Q19 is the following:

1. Data preprocessing, including the configuration of the spatial grid and the location of the drag points, and setting of boundary conditions and external forces.
2. Memory allocation on the host CPU and GPU device for f_i, S_i and x_k, and copy of the contents of the host to the device.

3. Loop the following sequence until a stop criterion is reached

   3.1. Calculation of the collision term using the current values of f_i (Eq. 2).
   3.2. Calculation of the drag force (Eq.12) and the corresponding source (Eq.11).
   3.3. Streaming step (Eq. 1).
   3.4. Application of boundary conditions, and actualization of f_i.
   3.5. Generation of special outputs at predefined calculation steps of the loop.

4. Post processing and rendering of output data.

The stages 3.1 to 3.4 (in italic) are executed in the GPU, whereas all the other steps are executed by the CPU. Stage 3.5 extracts relevant data from the GPU during the calculation, and is defined by the user according to the objectives of the numerical study. Since communications between the GPU and the CPU are expensive in execution time, it is important to optimize the design of these outputs avoiding redundancies and conveying the maximum information possible. In the present case, we will show how transient features of the flow can be visualized and characterized by identifying strategic points around the permeable medium. Also, strategic data outputs of the instantaneous state of the variables are convenient, which may be used later to restart the calculation including changes of control parameters of interest.

B. IMPLEMENTATION IN CUDA PYTHON

CUDA is based on special kernel functions, which execute in the GPU (NVIDIA®, 2020). The key feature of the kernels is that the number of parallel execution threads can be easily controlled. Threads compound blocks, and blocks compound a grid. Thus, every kernel is executed in several threads per block, and several blocks per grid (Rinaldi et al., 2015). The algorithm was implemented on a GPU NVIDIA Titan Xp using CUDA Python and Numba (Python compiler from Anaconda for execution on CUDA). Numba allows programming CUDA compiling blocks of Python code in CUDA kernels.

Python has become a very popular programming language and is currently used in a wide range of applications. Python is particularly successful in scientific computation, where several external libraries are used, such as PyCUDA (Klöckner et al., 2012), Numba-Numpy (Van Der Walt et al., 2011) and SciPy (Virtanen et al., 2020). PyCUDA uses C++ code for the Nvidia's CUDA API and Python code for the general program flow in the CPU and access to the data of the GPU. Numba is an open source compiler that uses Python

syntax either on CPU or GPU, having the advantage that a single language is used throughout the whole implementation. The drawback is that Python is slower than C or C++, especially for heavy computations. However, recent studies report that, used appropriately, the performances of CUDA-Numba and C-CUDA are comparable (Oden, 2020).

The main algorithm of LBIPM-D3Q19 is given in Figs. 2 and 3. The block LOAD CONFIGURATIONS shows the implementation of stage 1. To import cases, the code loads files containing the input/simulation parameters and the geometry of LBIPM D3Q19. The blocks ALLOCATE MEMORY ON THE HOST and ALLOCATE MEMORY ON THE DEVICE AND COPY HOST TO DEVICE correspond with stage 2. The data transmission from host to device and device to host uses the Numba instructions to_device(…) and copy_to_host(…) respectively.

Using the abstract kernels of algorithm 3.1, the grid/block organization for each kernel call (spatial grid and permeable points) is defined in code block CONFIGURE BLOCKS, within the limits set by the GPU. For study cases where a single permeable point is associated to a single cell in the spatial grid, it is convenient to use the same block specification. In such case, the access to the permeable points should be also changed accordingly. The main difference between both specifications is the dimensions of blocks and threads. When the permeable points are associated to the spatial grid, blocks are defined multi-dimensionally, whereas a single dimension is used for allocating the permeable points with independent permeable blocks.

The iterative loop of the stage 3 is implemented in the block SIMULATION LOOP, which includes calls of CUDA kernels of stages 3.1 to 3.4. Fig. 4 shows the code performing stage 3.2, which considers a single thread for each permeable reference point x_k. The communication of data between the stages articulated by kernels is performed by means of parameters. In order to keep the orders and types of the parameters, they are declared in the kernels and functions of the GPU by means of jit decorators (Lam et al., 2015).

A worth-mentioning issue is the management of data output within the calculation cycle. There are two conditions. The first condition is used to extract information of interest at specific times of the calculation. The second is used to get an image of the whole set of variables necessary to restart the calculation in case needed. These communications should be minimized as much as possible, since they can increase dramatically the cost in computational time.

Fig. 5 shows how the LBM grid (a) and the drag points (b) are distributed in the GPU. The grid size is (XMAX,YMAX,ZMAX). To allocate memory on the CPU the following NUMBA commands are used:

```
f_host = np.zeros(shape=(19,XMAX,YMAX,ZMAX), dtype=np.float32)  //f_i
ftemp_host    =    np.zeros(shape=(19,XMAX,YMAX,ZMAX),    dtype=np.float32)
//advencted f_i
S_host = np.zeros(shape=(19,XMAX,YMAX,ZMAX), dtype=np.float32) //S_i
```

To allocate memory on the GPU and copy the CPU data to the GPU, the following MUMBA commands are used:

```
f_device=cuda.to_device(f_host)
ftemp_device=cuda.to_device(ftemp_host)
S_device=cuda.to_device(S_host)
```

The relation between the LBM cells and the id_threads in the GPU, which will describe how physical domain is distributed on the GPU, the following Python sentences: are used:

```
griddimLBM = ZMAX, YMAX, 1          //blocks per grid
blockdimLBM = XMAX, 1, 1            //threads per block
```

In our implementation, each thread corresponds to a plane of constant X. Hence, the size of the thread block is XMAX. To ensure that there is one thread per cell, the grid is created with (ZMAX * YMAX) blocks. The cell and its id_thread relate to each other as:

```
z = cuda.blockIdx.x
y = cuda.blockIdx.y;
x = cuda.threadIdx.x;
```

Fig. 6 shows an example of the CUDA Python code for the kernel of the streaming step. The invocation of this kernel is:

```
Streaming[griddimLBM,blockdimLBM](XMAX,YMAX,ZMAX,f_device,ftemp_device,S_
device)
```

IV. APPLICATION STUDY

To verify the performance of the proposed implementation we have chosen a straight rectangular channel partially blocked by a regular array of fine wires. Fig. 7 shows a diagram of the system configuration. The geometry was used in a previous work to validate the 2D implementation of the LBIPM against experimental measurements (Dalponte et al., 2012; Boroni et al., 2015). One characteristic of this flow configuration is that it presents a linear instability that gives place to regular harmonic oscillations in the zone where free flow and flow through the permeable region coexist. This feature allows us to study unstable flow conditions without entering into fully turbulent flows. In all cases the boundary conditions are periodic in the streamwise direction, and bounce-back (i.e., null velocity) at the bottom and top walls of the channel. For the 3D case, the lateral walls also have bounce-back conditions.

The driving pressure head is ensured by imposing a constant and uniform volume force in every cell of the domain. The force is calibrated in each case in order to set the same flow rate in 2D and 3D. The corresponding Reynolds numbers are defined as:

$$Re = \frac{UL}{\nu} \tag{13}$$

where U is the average velocity at the inlet, L is the channel height, and ν the kinematic viscosity. This procedure matches with the imposition of a constant pressure gradient in the direction of the volumetric force in steady state and when compressibility effects are negligible. It is equivalent to the action of a constant and uniform gravity field in the flow direction.

Two flow regimes were tested, laminar steady state and laminar unstable sustained oscillatory flow, corresponding to Reynolds numbers 100 and 200 respectively. In the oscillatory case, the Reynolds number is calculated with the inlet velocity averaged also in time. Table 2 details the Euler numbers for each case, defined as:

$$Eu = \frac{p_2 - p_1}{\frac{1}{2}\rho U^2} \tag{14}$$

where p_1 and p_2 are the inlet and exit pressure, respectively, and ρ is the average density.

TABLE 2. Euler numbers resulting for each study case.

Re	Geometry	Eu
100	2D	4.6
	3D	6
200	2D	3.5
	3D	4.0

TABLE 3. Assessment of the grid units. L is the channel length in metric units (1440 mm) and ν is the kinematic viscosity. ZMAX is the number of cells of the channel length. The current relaxation parameter is $\tau = 0.56$.

Fluid	$\Delta x = \frac{L}{\text{ZMAX}}$	$\Delta t = (2\tau - 1)\frac{\Delta x^2}{6\nu}$	$u_{LBM} = \frac{\Delta x}{\Delta t}$
Air (20º C)	1 mm	$1.33\ 10^{-3}$ s	0.75 m/s
Water (20º C)		$2\ 10^{-2}$ s	0.05 m/s

All results are presented in dimensionless form. Table 3 shows the equivalence in metric units for air and water.

Figures 8 to 11 show the contour maps of the streamwise component of the velocity calculated with the 2D model and the 3D model at the central vertical plane, for the stable case and the unstable case. In the latter, the map corresponds to the average velocity. The main features of the velocity fields are:

- The velocity magnitude is much lower inside the permeable region. This is consistent with previous experimental and theoretical results (Silin et. al, 2011), and is caused by high viscous stresses inside the permeable region.
- The free flow structure experiences a contraction and an expansion at the beginning and after the permeable region, respectively.
- Downstream the permeable region, there is a recirculation in the lower part of the channel and a region of lower velocity nearby the upper wall. Both effects are more pronounced in the 2D case. The difference stems from the influence of the lateral walls in the 3D case, which reduces the momentum flux generating pressure gradients that diminish the boundary layer by the upper wall.
- For Re = 200 the flow in the free region over the obstructed zone does not complete the development, whereas for low Re = 100 it develops in the first quarter of the obstruction length.
- The LBIPM is able to resolve the channeling effects around the small obstacles in the permeable medium. This feature produces a more realistic flow around the positions occupied by the sparse solid structures, a feature that is generally more relevant in permeable media than in porous media.

While velocity fields in the stable case are rather similar in both, 2D and 3D scenarios, important differences manifest in the dynamics of the unstable case (Re = 200). Figs. 12 and 13 show the maps of the standard deviation of the streamwise and vertical velocity component, comparing the 2D results with the central plane in the 3D case. These maps evince the regions where the velocity fluctuations are higher. The fluctuations of the streamwise velocity peak in the recirculation behind the obstruction in both cases, but the spatial patterns differ, concentrating closer to the bottom in the 2D case. On the other hand, the pattern of fluctuations of the vertical velocity are much more complex in the 3D case, which is caused by the interaction of the lateral fluctuations which are absent in 2D. Fig. 14 shows the map of the standard deviation of the lateral velocity component in the central plane of the 3D case. It can be seen that the spot of high lateral fluctuation coincide with corresponding spots on the maps of the other components, located about z = 110 and y = 20. This spot is produced by a secondary flow induced by the lateral walls. Actually, there is also a second spot about z = 1250 and y = 30, which is also observed in the fluctuations of the three velocity component.

The secondary flow induced by the lateral components can be visualized in the current lines shown in Fig. 15. In order to capture the change of the flow structure triggered by the instabilities at the higher Re number, 3D contour maps of each velocity component were produced, showing the central plane together with three cross sections perpendicular to the stream located in the obstructed region, just after the obstruction and at the downstream in the expansion wake. Figs. 16 and 17 show the 3D color maps of the streamwise velocity at Re 100 and 200 respectively. It can be seen that at high Re a region of higher velocity appears in the lower central part of the channel, bespeaking of a jet concentration produced by the secondary flow. This effect is also reflected in the corresponding 3D maps of the vertical and lateral velocities, Figs. 18 to 21. Particularly striking is the complexity of the cross-section of the lateral velocity at the wake, where the antisymmetric patterns reveal a complicated pattern of vortices.

Table 4. Performance metrics of the implementation of LBIPM.

Grid size	Number of spatial cells	Number of permeable points	Cell updates per sec	Permeable point update per sec
1×54×1440	77760	183	4.6 E7	1.1 E5
40×54×1440	3.1104 E6	7320	1.9 E9	4.5 E6
80×54×1440	6.2208 E6	14640	3.4 E9	8.0 E6
40×114×1440	6.5664 E6	7320	4.1 E9	4.6 E6
180×114×1440	2.9549 E7	60390	2.6 E9	5.2 E6
220×114×900	2.2572 E7	49610	3.0 E9	6.7 E6

It is worth mentioning that the patterns shown in Figs. 17, 19 and 21 (for Re = 200) were built using the average velocity field. The actual flow is unstable and oscillates around the values represented by the colors of these maps. Figs. 22 and 23 depict the temporal evolution of the velocity components along a vertical centerline located at z = 1000, just behind the obstructed pack where the intensity of the fluctuations peaks. It can be seen that the 2D oscillations are much more regular than those in 3D. Fig. 24 compares the Fourier spectra of the downstream velocity at three points along the vertical centerline at z = 1000, namely, at y = 10, 20 and 40, for both 2D and 3D simulation results at Re = 200. The power spectra present peaks in the range of frequencies from 10^{-3} to 10^{-2}. For the 2D results the main oscillation peak is located at a slightly lower frequency as compared to the case of the 3D results. In order to compare the local non-linear dynamics of the oscillations in 2D and 3D, Fig. 25 depicts the phase-space trajectories taking coordinates of instantaneous downstream and vertical velocity components, corresponding to the same three points whose spectra are shown in Fig. 24. It can be seen that the shape of the attractors is

different in the 2D and 3D solutions, and the oscillations are more symmetrical in the 2D case, consistent with the temporal maps.

The implementation of the LBIPM in GPU shows an excellent performance. The main metrics are presented in Table 4. The optimum performance for the GPU NVIDIA Titan XP use in the present study was registered for a grid of 220×114×900 cells, achieving a computation rate of 3000 MLUPS.

V. CONCLUSIONS

A GPU implementation of the 3D model LBIPM for flows through and around a permeable medium simulated by localized drag forces was presented. The parameters of the localized drag forces let the user adjust the resistance force and its spatial distribution to produce a realistic flow around the sparse solid structures. This enables retaining the heterogeneous nature of the flow while avoiding the high cost of modelling them in greater detail. Using drag points, which can be arranged both in regular or irregular arrays, permeable media with different shapes and permeabilities can be straightforwardly emulated.

The algorithm was applied to simulate a straight rectangular channel partially occupied by a regular array of cylindrical wires that form a permeable obstruction. The 3D simulations are compared against a previously developed and validated 2D LBM model. Two conditions were simulated, the first corresponds to a laminar flow (RE = 100) while the second presents an instability that brings about growing waves over and in the wake behind the permeable medium (Re = 200). The results show that flow fields obtained from the two models largely agree, but the 3D model captures secondary flow structures that propagate downstream from the end of the obstruction. Also there are moderate differences in the growth of the boundary layer and on the recirculation bubble downstream from the end of the obstruction between the 2D and 3D cases. For the case of unstable flow it is observed that the flow oscillations obtained in 3D are less regular than in 2D. While the power spectra obtained in each scenario barely show significant differences, the time evolution produced in the 3D case exhibits frequent losses of coherence. In turn, the 2D case shows more persistent regularity during longer time intervals. The intensity of the velocity fluctuations at the end of the permeable obstructions is similar for both scenarios, but the fluctuations in the 3D case diminish significantly earlier than in 2D. These differences between indicate that, even for this relatively low Reynolds number, the fluctuations produced by the flow instability have a three dimensional nature that is captured by the present model. In general terms, it was shown that by modeling a permeable medium with a set of drag points it is possible to deal with small and medium spatial scales without losing the heterogeneous nature of permeable flows in the homogenization process.

The numerical results presented in Section 4 call for further experimental verification. Unfortunately this requires capturing features that clearly unveil the 3D nature of the flow, which in this case proved to be extremely challenging. Nevertheless, the present numerical predictions are a useful reference source for future benchmarks and for guiding the design of adequate experimental setups. In particular, modal decomposition techniques applied to experimental measurements and the corresponding numerical simulations can be significant.

ACKNOWLEDGEMENTS

This work was partially supported by Project SIIP 2019 06/C595 of the National University of Cuyo, Argentina. The Titan Xp devise used for this research was donated by the NVIDIA Corporation.

APPENDIX: ESTIMATION OF THE EFFECTIVE PERMEABILITY OF THE MEDIUM

An algebraic relation between the local permeability parameter of the model, κ_l, and the permeability of the medium, κ_m, can be estimated by considering a representative volume element (RVE) consisting of the minimum set of grid cells whose tessellation generates the medium. Each drag point $\boldsymbol{x}_k$ contained in the RVE introduces a force $\boldsymbol{F}_k \Delta x^3$, where $\boldsymbol{F}_k$ is given by Eq. 6. The net force per unit volume in the RVE is then:

$$\boldsymbol{F}_{RVE} = \frac{1}{\Delta x^3 N_{RVE}} \sum_{\boldsymbol{x}_k \in RVE} \boldsymbol{F}_k \, \Delta x^3 = -\frac{\rho \nu}{N_{RVE} \kappa_l} \sum_{\boldsymbol{x}_k \in RVE} \boldsymbol{u}_k$$

where N_{RVE} is the number of grid cells of the RVE. Assuming that the velocity field inside the RVE is approximately uniform, the summation of velocities can be written as:

$$\sum_{\boldsymbol{x}_k \in RVE} \boldsymbol{u}_k \cong N_k \langle \boldsymbol{u} \rangle$$

where N_k is the number of drag points contained in the RVE and $\langle \boldsymbol{u} \rangle$ is the average velocity in the RVE.

The effective permeability of the RVE is defined by the relation between $\boldsymbol{F}_{RVE}$ and $\langle \boldsymbol{u} \rangle$, that is:

$$\boldsymbol{F}_{RVE} = -\frac{\rho \nu}{\kappa_m} \langle \boldsymbol{u} \rangle$$

Then the effective permeability of the media is approximately given by:

$$\kappa_m \cong \frac{N_{RVE}}{N_k}\kappa_l$$

It should be stressed that this formula is merely a rough assessment, since the velocity is never completely uniform inside the RVE. Actually, strictly speaking, κ_m depends on the direction of the flow.

In general, κ_m can be assessed by numerically calculating the average velocity $\langle \boldsymbol{u} \rangle$ in a periodic RVE, driven by an external constant and uniform volume force $\boldsymbol{F}$. For example, the global permeability κ_x in direction *x* is:

$$\kappa_x = \frac{\rho \nu \langle u_x \rangle}{F_x}$$

Moreover, if the drag points are arranged in asymmetric configurations respect to the streamwise direction, transversal velocity gradients may appear inside the RVE, which in turn will induce lift forces in addition to drag. And of course, there is the matter of temporal variation of the spatial flow distribution inside the RVE, which would render κ_m time dependent. A recent formal treatment of these homogenization problems can be found in Blanco et al. (2017).

DATA AVAILABILITY

The data that supports the findings of this study are available within the article.

REFERENCES

An, S., Yu, H, Yao, J., GPU-accelerated volumetric lattice Boltzmann method for porous media flow, Journal of Petroleum Science and Engineering 156, 546–552 (2017).

Babu, V., Narasimhan, A., Investigation of vortex shedding behind a porous square cylinder using lattice Boltzmann method. Physics of Fluids 22, 053605 (2010).

Bailey P., Myre J., Walsh S., Lilja D. J. and Saar M. O., Accelerating Lattice Boltzmann Fluid Flow Simulations Using Graphics Processors. International Conference on Parallel Processing (2009).

Ben Meftah, M., Mossa, M., Prediction of channel flow characteristics through square arrays of emergent cylinders. Physics of Fluids 25, 045102 (2013).

Blanco, P., Clausse, A., Feijóo, R., Homogenization of the Navier-Stokes equations by means of the multi-scale virtual power principle, Comp. Meth. Applied Mech. Eng. 315 760–779 (2017).

Bhatnagar, P.L., Gross, E.P., Krook, M., A model for collision processes in gases, I. Small amplitude processes in charged and neutral one-component systems. Phys. Rev. 94, 511 (1954).

Boroni, G., Silin, N., Dalponte, D., Dottori, J., Clausse, A., Lattice-Boltzmann Modeling of Unstable Flows amid Arrays of Wires. Computers and Fluids 120, 37-45 (2015).

Chang, K., Constantinescu, G., Numerical simulation of flow past a porous cylinder with 20% solid volume fraction, J. Comp. Fluids Eng. 17, 87-92 (2012).

Chapman, S. and Cowling, T.G., The Mathematical Theory of Non-uniform Gases: an Account of the Kinetic Theory of Viscosity, Thermal Conduction and Diffusion in Gases. Cambridge Univ. Press (1970).

Clausse, A., Silin, N., Boroni, G., A Multiscale Method for Producing Homogenized Drag Laws of a Permeable Medium by Conflating Experimental Data with Lattice-Boltzmann Simulations, Int. J. Num. Meth. Heat Fluid Flow 29, 4394-4407 (2019).

Cummins, C., Viola, I., Mastropaolo, E., Nakayama, The effect of permeability on the flow past permeable disks at low Reynolds numbers. Physics of Fluids 29, 097103 (2017).

Dalponte, D., Silin, N., Clausse, A., Gas flow in a channel semiobstructed by a porous media, J. Porous Media 15, 927-936 (2012).

Guo, Z. and Shu, C., Lattice Boltzmann Method and its Applications in Engineering, Springer Series in Advances in Computational Fluid Dynamics, Vol. 3 (2013).

Hecht M. and Harting J., Implementation of on-site velocity boundary conditions for D3Q19 lattice Boltzmann simulations, J. Stat. Mech. Theory Exp., 2010, P01018 (2010).

Herschlag G., Lee S., Vetter J. S. and Randles A., GPU Data Access on Complex Geometries for D3Q19 Lattice Boltzmann Method. IEEE International Parallel and Distributed Processing Symposium (2018).

Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., Fasih, A., PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. Parallel Computing 38, 157–174 (2012).

Kang, Q., Zhang, D., Chen, S., Unified lattice Boltzmann method for flow in multiscale porous media, Phys. Rev. E, 66, 056307 (2002).

Ledda, P., Siconolfi, L. Viola, F., Gallaire, F., Camarri, S., Suppression of von Kármán vortex streets past porous rectangular cylinders. Physical Rev. Fluids, 3, 103901 (2018).

NVIDIA CUDA Programming Guide, NVIDIA® Corporation (2020).

Lam, S., Pitrou, A., Seibert, S., Numba: a LLVM-based Python JIT compiler, Proc. Second Workshop LLVM Compiler Infrastructure HPC, Article 7, pp. 1–6 (2015).

Oden, L., Lessons learned from comparing C-CUDA and Python-Numba for GPU-Computing, Proceedings of the 28th Euromicro Int. Conf. on Parallel, Distributed Network-

Based Processing, Vasteras, Sweden, 216-223 (2020), doi: 10.1109/PDP50117.2020.00041.

Peskin, C., The immersed boundary method. Acta Numerica 11, 479-517 (2002).

Rinaldi, P., Dari, E., Vénere, M., Clausse, A., A Lattice-Boltzmann Solver for 3D Fluid Simulation on GPU, Simulation Modelling Practice and Theory 25, 163–171 (2012).

Shan, X., Yuan, X,, Chen, H., Kinetic theory representation of hydrodynamics: a way beyond the Navier–Stokes equation, J. Fluid Mech. 550, 413–441 (2006).

Silin, N., Converti, J., Dalponte, D., Clausse, A., Flow instabilities between two parallel planes semi-obstructed by an easily penetrable porous medium, J. Fluid Mech. 689, 417-433 (2011).

Succi, S., The Lattice Boltzmann Equation for Complex States of Flowing Matter, Oxford Univ. Press (2018).

Tang, T., Yu, P., Shan, X., Chen, H., The formation mechanism of recirculating wake for steady flow through and around arrays of cylinders. Physics of Fluids 31, 043607 (2019).

Tang, T., Yu, P., Shan, X., Li, J., Yu, S., On the transition behavior of laminar flow through and around a multi-cylinder array. Physics of Fluids 32, 013601 (2020).

Tran N., Lee M., and Hong S., Performance Optimization of 3D Lattice Boltzmann Flow Solver on a GPU. Hindawi Scientific Programming (2017).

Uhlmann, M., An Immersed Boundary Method with Direct Forcing for the Simulation of Particulate Flows, J. Comput. Phys. 209, 448-476 (2005).

Van Der Walt, S., Colbert, S., Varoquaux, G.,The NumPy array: a structure for efficient numerical computation. Comp. Sci. Eng. 13, 22 (2011).

Virtanen, P. et al., SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17, 261-272 (2020).

Wen J., Ma H., Real-time smoke simulation based on vorticity preserving lattice Boltzmann method. Vis. Comput, 35, 1279–1292 (2019).

Yu, H., Chen, X., Wang, Z., Deep, D., Lima, E., Zhao, Y., Teague, S.D., Mass conserved volumetric lattice Boltzmann method for complex flows with willfully moving boundaries, Phys. Rev. E 89, 063304 (2014).

Zhu, J. and Ma, J., An improved gray lattice Boltzmann model for simulating fluid flow in multi-scale porous media, Adv. in Water Res. 56, 61 (2013).

Zhu, J. and Ma, J., Extending a Gray Lattice Boltzmann Model for Simulating Fluid Flow in Multi-Scale Porous Media, Scientific Reports 8, 5826 (2018).

FIGURE CAPTIONS

Figure 1. Diagram of the discrete velocity set corresponding to the LBM scheme D3Q19. Index 19 corresponds to stagnation.

Figure 2. Main algorithm of LBIPM-D3Q19 (part 1)

Figure 3. Main algorithm of LBIPM-D3Q19 (part 2)

Figure 4. CUDA code implementing stage 3.2 (drag forces).

Figure 5. Diagram of the memory allocation in the GPU.

Figure 6. CUDA Python code for the kernel of the streaming step.

Figure 7. Schematic diagram of the geometry of the application study. The zoom shows the detail of the permeable structure formed by a regular array of thin wires. All lengths are expressed in grid units given by the lattice cell size.

Figure 8. Color contour maps of the streamwise component of the velocity calculated with the 3D model at the central vertical plane in the flow direction, for the stable case (Re = 100). All magnitudes are expressed in grid units. Note the heterogeneous nature of the velocity field inside the permeable region.

Figure 9. Color contour maps of the streamwise component of the velocity calculated with the 3D model at the central vertical plane, in the flow direction, for the unstable case (Re = 200). All magnitudes are expressed in grid units. Note the heterogeneous nature of the velocity field inside the permeable region.

Figure 10. Color contour maps of the streamwise component of the velocity calculated with the 2D model at the central vertical plane, in the flow direction, for the stable case (Re = 100). All magnitudes are expressed in grid units. Note the heterogeneous nature of the velocity field inside the permeable region.

Figure 11. Color contour maps of the streamwise component of the velocity calculated with the 2D model at the central vertical plane, in the flow direction, for the unstable case (Re = 200). All magnitudes are expressed in grid units. Note the heterogeneous nature of the velocity field inside the permeable region.

Figure 12. Color maps of the standard deviation of the streamwise velocity component, for the unstable condition (Re = 200), comparing the 2D results (top) with the central plane in the 3D case (bottom). All magnitudes are expressed in grid units.

Figure 13. Color maps of the standard deviation of the vertical velocity component, for the unstable condition (Re = 200), comparing the 2D results (top) with the central plane in the 3D case (bottom). All magnitudes are expressed in grid units.

Figure 14. Color maps of the standard deviation of the lateral velocity component, for the unstable condition (Re = 200) in 3D. All magnitudes are expressed in grid units. The spot of high lateral fluctuation coincide is located about z = 110 and y = 20.

Figure 15. Current lines for the unstable 3D case (Re = 200). The color maps show the magnitude of the average vertical velocity component at each plane (see. Fig. 15). The current lines are particle trajectories.

Figure 16. 3D color map of the streamwise velocity component at Re = 100 (stable). Compare with the unstable case in Fig. 13.

Figure 17. 3D color map of the average streamwise velocity component at Re = 200 (unstable). Compare with the stable case in Fig. 12.

Figure 18. 3D color map of the vertical velocity component at Re = 100 (stable). Compare with the unstable case in Fig. 15.

Figure 19. 3D color map of the average vertical velocity component at Re = 200 (unstable). Compare with the stable case in Fig. 14.

Figure 20. 3D color map of the lateral velocity component at Re = 100 (stable). Compare with the unstable case in Fig. 17.

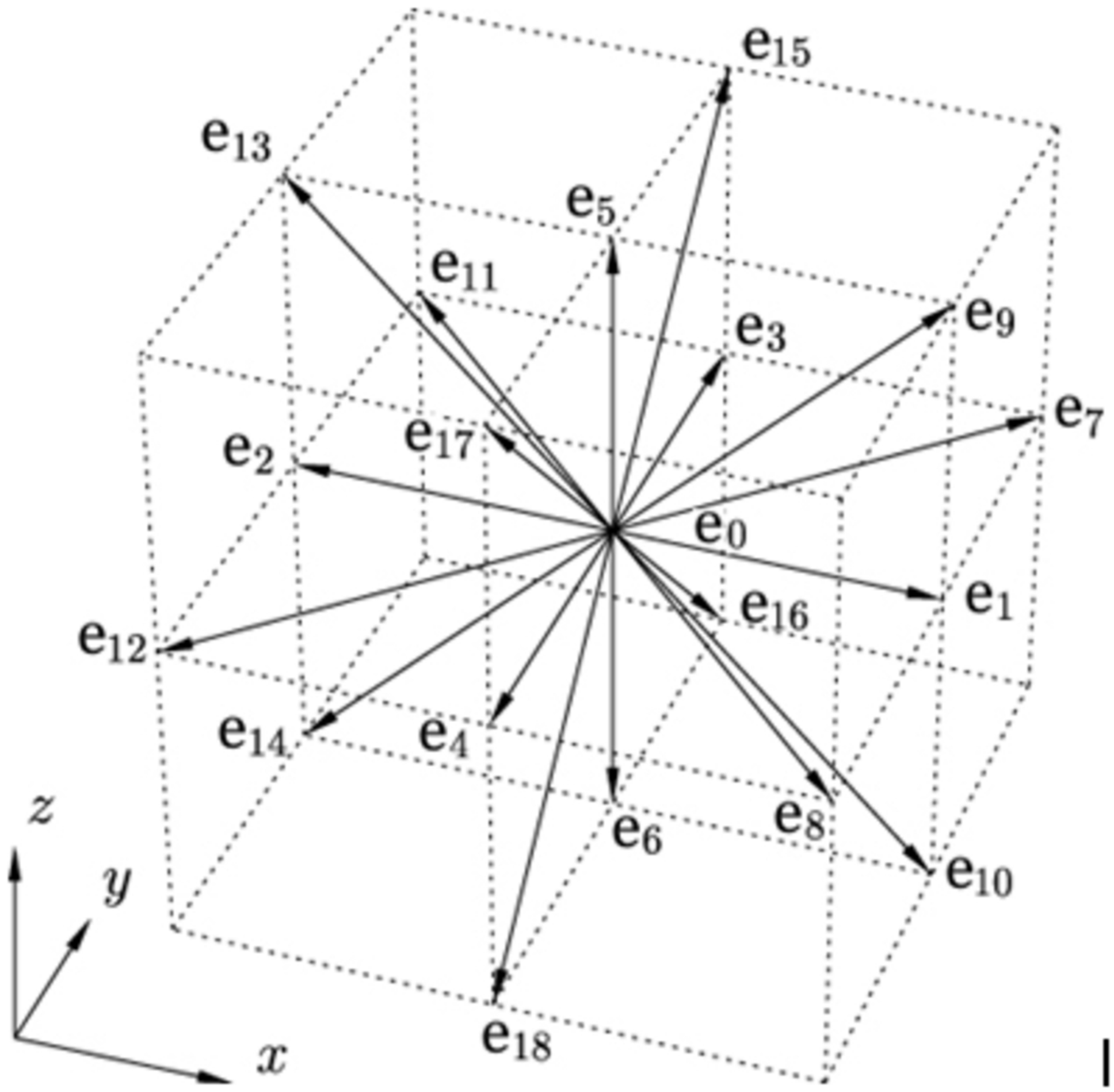
Figure 21. 3D color map of the average lateral velocity component at Re = 200 (unstable). Compare with the stable case in Fig. 16.

Figure 22. Map of the temporal evolution of each velocity component along a vertical centerline located at z = 1000, for the 3D case. All magnitudes are expressed in grid units. Although the oscillations are chaotic, transients of regular oscillations are observed (see between times 280 and 380).

Figure 23. Map of the temporal evolution of each velocity component along a vertical centerline located at z = 1000, for the 2D case. All magnitudes are expressed in grid units. Comparing with Fig. 18, the oscillations are much more regular in the 2D case.

Figure 24. Fourier spectra of the downstream velocity at three points along the vertical centerline at z = 1000, namely, at y = 10 (black), 20 (blue) and 40 (red), for both 2D and 3D simulation results at Re = 200.

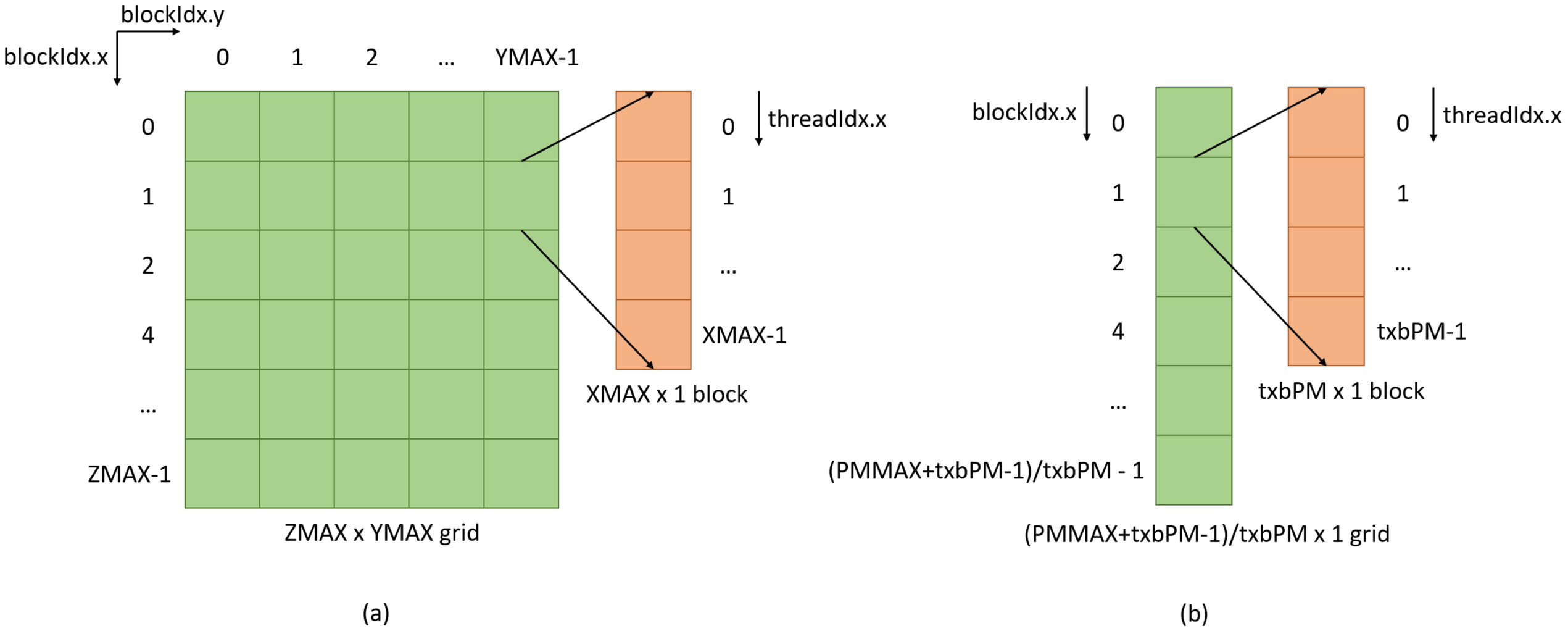
Figure 25. Phase-space trajectories of instantaneous downstream and vertical velocity components, corresponding to the same three points whose spectra are shown in Fig. 20.


e15
e13
e5
e11
e9
e3
e7
e17
e2
e0
e1
e16
e12
e14
e4
e8
e6
e10
e18
z
y
x

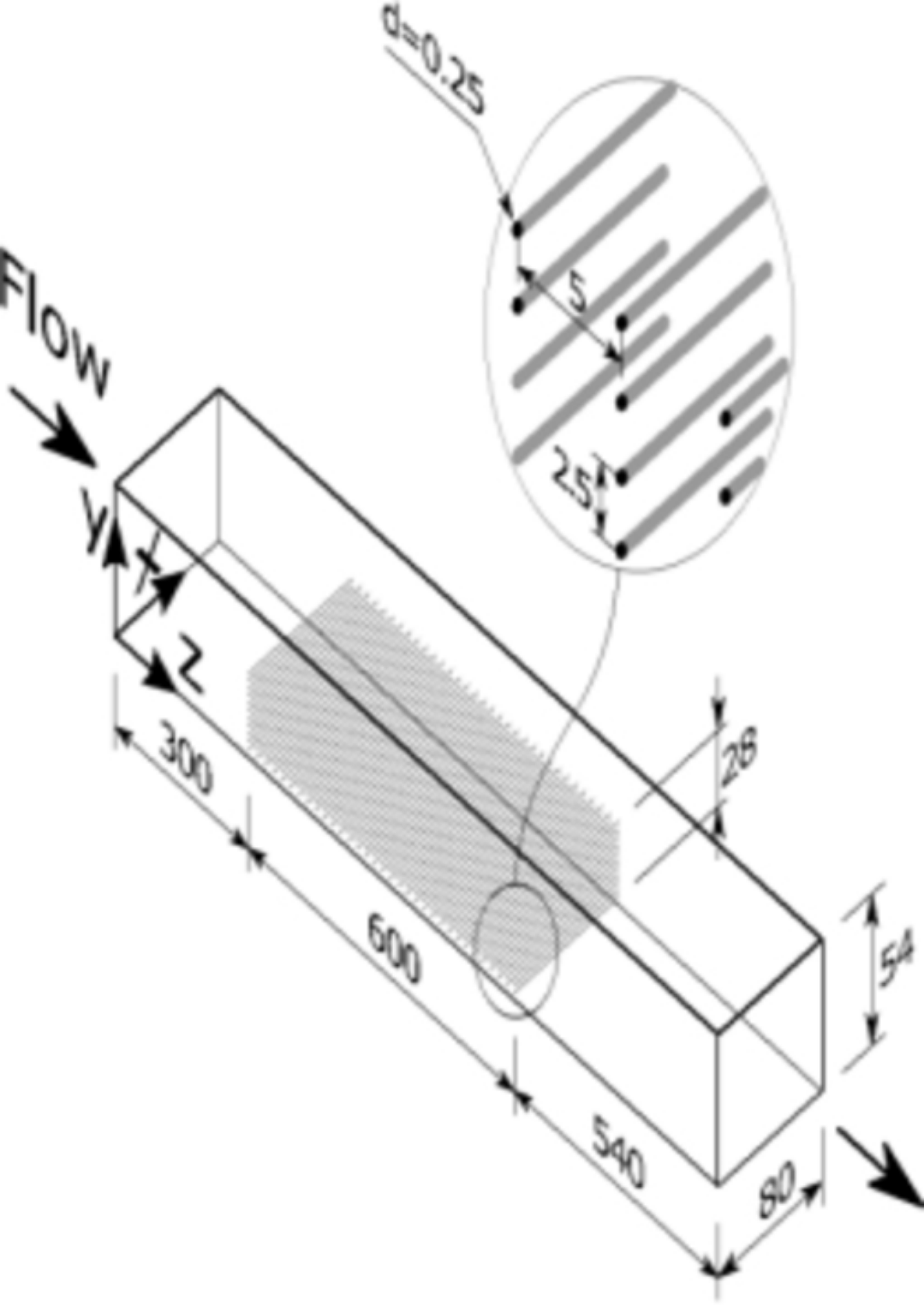
```
//Algorithm LBIPM-D3Q19 v.2020
def LBM_PM_Simulation():

"""LOAD CONFIGURATIONS"""
 //Grid size (XMAX, YMAX, ZMAX), PMMAX permeable reference points
 load_LBM_configuration("""XMAX,YMAX,ZMAX,MAXITERATION,boundary,inlet,outlet,rho,tau,.""")
 load_PM_configuration("""PMMAX,X,...""")
 load_history_results("""means,velocity respect to time,...""")

 """ALLOCATE MEMORY ON THE HOST"""
 f_host= np.zeros(shape=(19,XMAX,YMAX,ZMAX), dtype=np.float32) //f_i
 ftemp_host= np.zeros(shape=(19,XMAX,YMAX,ZMAX), dtype=np.float32) //f_i advected
 S_host= np.zeros(shape=(19,XMAX,YMAX,ZMAX), dtype=np.float32) //S_i
 external_forces_host= np.zeros(shape=(3,XMAX,YMAX,ZMAX), dtype=np.float32) //define other types of external forces
 X_host = np.zeros(shape=(3,PMMAX), dtype=np.float32)// Permeable points
 load_partial_results(f_host,S_host)
 ftemp_host[:,:,:,:]=f_host[:,:,:,:];
 X_host[0:PMMAX,:]=X[0:PMMAX,:]

 """ALLOCATE MEMORY ON THE DEVICE AND COPY HOST TO DEVICE"""
 f_device=cuda.to_device(f_host)
 ftemp_device=cuda.to_device(ftemp_host)
 S_device=cuda.to_device(S_host)
 external_forces_device=cuda.to_device(external_forces_host)
 X_device=cuda.to_device(X_host)

"""CONFIGURE BLOCKS"""
griddimLBM = ZMAX, YMAX, 1
blockdimLBM = XMAX, 1, 1
txbPM = 32
griddimPM = int((PMMAX+txbPM-1)/txbPM), 1, 1
blockdimPM = txbPM, 1, 1
```

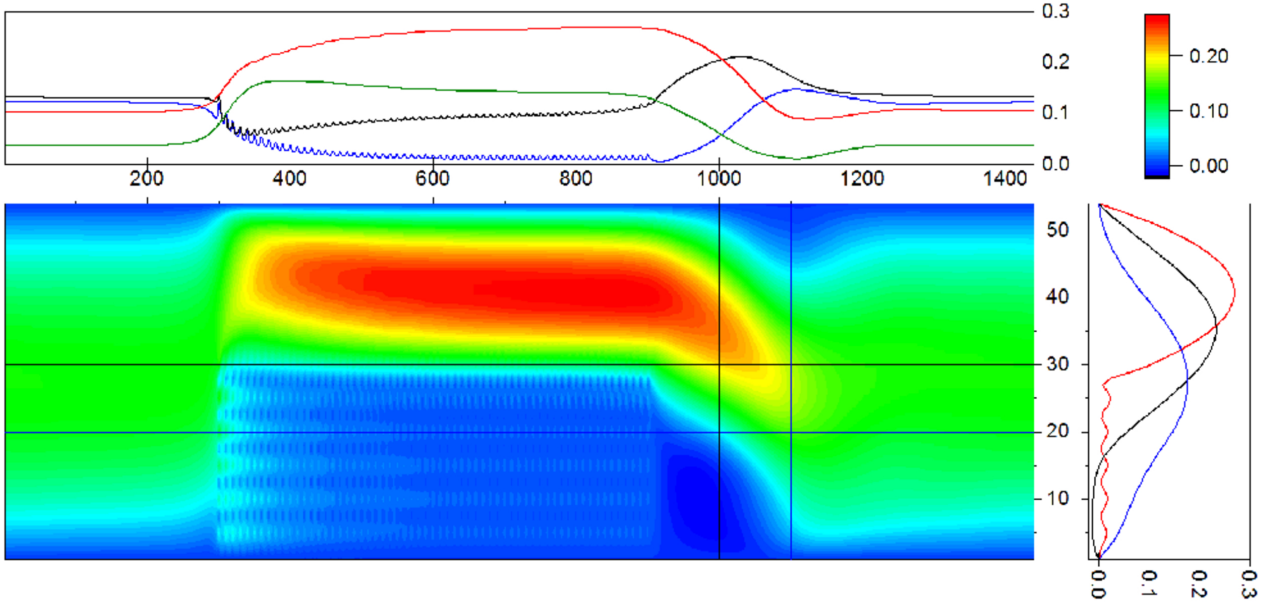
```
//MAIN ALGORTITHM cont.

"""SIMULATION LOOP"""
iteration=0
while (iteration < MAXITERATION):
collision[griddimLBM,blockdimLBM](XMAX,YMAX,Z
MAX,f_device,ftemp_device,
external_forces_device,"""tau,rho,..."""")
fluid_pm_interaction[griddimPM,blockdimPM](XM
AX,YMAX,ZMAX,PMMAX,ftemp_device,
S_device,X_device,"""radio,drag          force
parameters,..."""")
streaming[griddimLBM,blockdimLBM](XMAX,YMAX,Z
MAX,f_device,ftemp_device,S_device)
inlet_outlet_boundary_conditions[griddimLBM,b
lockdimLBM](XMAX,YMAX,ZMAX,ftemp_device,
external_forces_device,"""boundary,inlet,outl
et,rho,tau,..."""")

if condition_1 is true partial results are
displayed:
"""copy to host"""
ftemp_device.copy_to_host(ftemp_host)
"""plot(...,ftemp_host,...)"""

if condition_2 is true partial results are
saved:
"""copy to host"""
ftemp_device.copy_to_host(ftemp_host)
"""save(...,ftemp_host,...)"""
iteration+=1

"""COPY TO HOST"""
ftemp_device.copy_to_host(ftemp_host)
S_device.copy_to_host(S_host)

"""SAVE OUTPUT"""
save_intermediate_results(...,iteracion_acumu
lada,ftemp_host,S_host)
save_local_results(XMAX,YMAX,ZMAX,ftemp_host)
save_vtk_fluid_pm(...)
save_history_results(...)
```

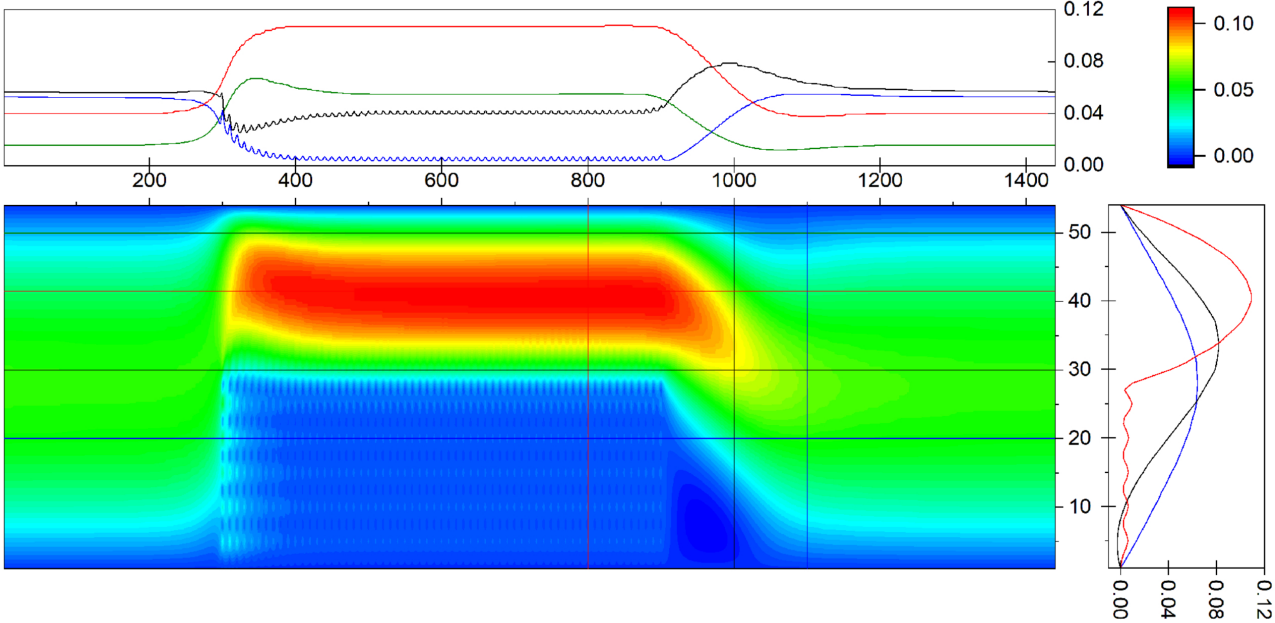
```
Kernel: compute fluid_pm(...){
k = point of reference index of PM
iniX=Max(1,int(Xx(k)-radius)) //cell index of
radius-neighbor
        iniY=Max(1,int(Xy(k)-radius))
        iniZ=Max(1,int(Xz(k)-radius))
        endX=Min(Lx,int(Xx(k)+radius))
        endY=Min(Ly,int(Xy(k)+radius))
        endZ=Min(Lz,int(Xz(k)+radius))
        x=iniX..endX{
        y=iniY..endY{
        z=iniZ..endZ{

if (|x-Xx(k)|<radius)&(|y-Xy(k)|<radius)&(|z-
Xz(k)|<radius){ //Eq. ()
δh=calculateδh([x,y,z],[Xx(k),Xy(k),Xz(k)])//
Eq. ()
ukx=ukx+calculateulbx(fi,x,y,z)*δh //Eqs. ()
uky=uky+calculateulby(fi,x,y,z)*δh //Eqs. ()
ukz=ukz+calculateulbz(fi,x,y,z)*δh //Eqs. ()
}}}}
Fkx=-visclb*ukx/K //Eq. ()
Fky=-visclb*uky/K //Eq. ()
Fkz=-visclb*ukz/K //Eq. ()
x=iniX..endX{
y=iniY..endY{
z=iniZ..endZ{

if (|x-Xx(k)|<radius)&(|y-Xy(k)|<radius)&(|z-
Xz(k)|<radius){//Eq. ()
δh=calculateδh([x,y,z],[Xx(k),Xy(k),Xz(k)])//
Eq. ()
fpx=Fkx*δh //Eq. ()
fpy=Fky*δh //Eq. ()
fpz=Fkz*δh //Eq. ()
i=0..Q{//Q possible velocity directions
auxSilb=calculateSilb(fi,[fpx,fpy,fpz],x,y,z,
i) //Eq. ()
atomicAdd(Silb(x,y,z,i),auxSilb)
}}}}}}
```

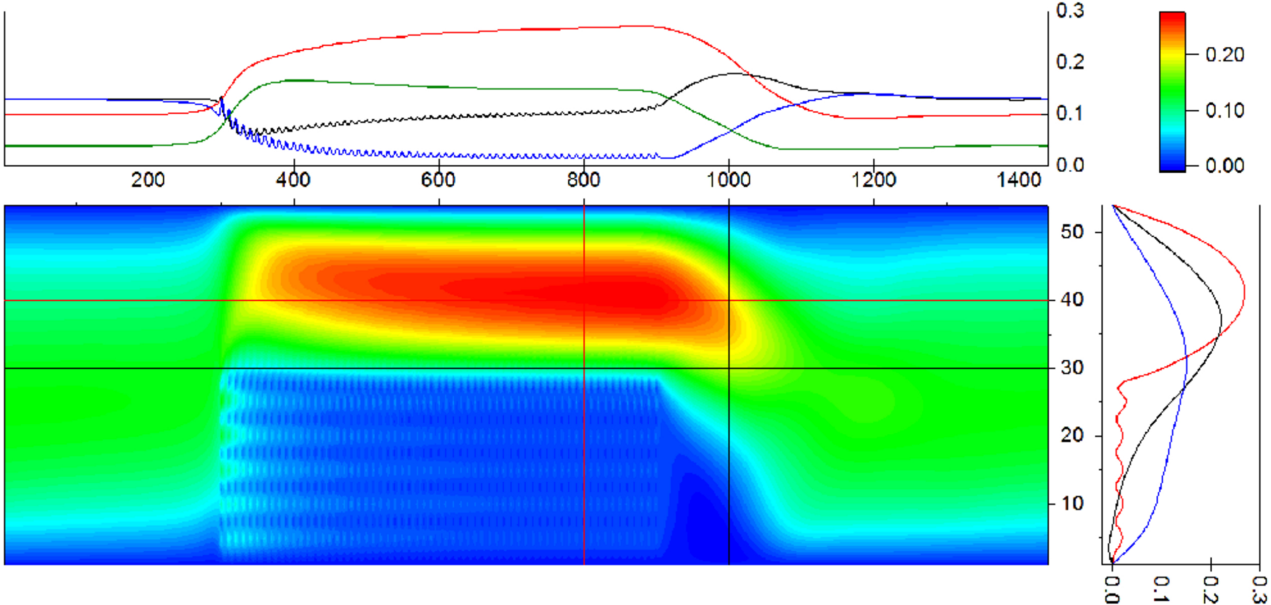

blockIdx.y
blockIdx.x
0
1
2
...
YMAX-1
0
1
2
4
...
ZMAX-1
ZMAX x YMAX grid
0
threadIdx.x
1
...
XMAX-1
XMAX x 1 block
(a)
blockIdx.x
0
1
2
4
...
(PMMAX+txbPM-1)/txbPM - 1
(PMMAX+txbPM-1)/txbPM x 1 grid
0
threadIdx.x
1
...
txbPM-1
txbPM x 1 block
(b)

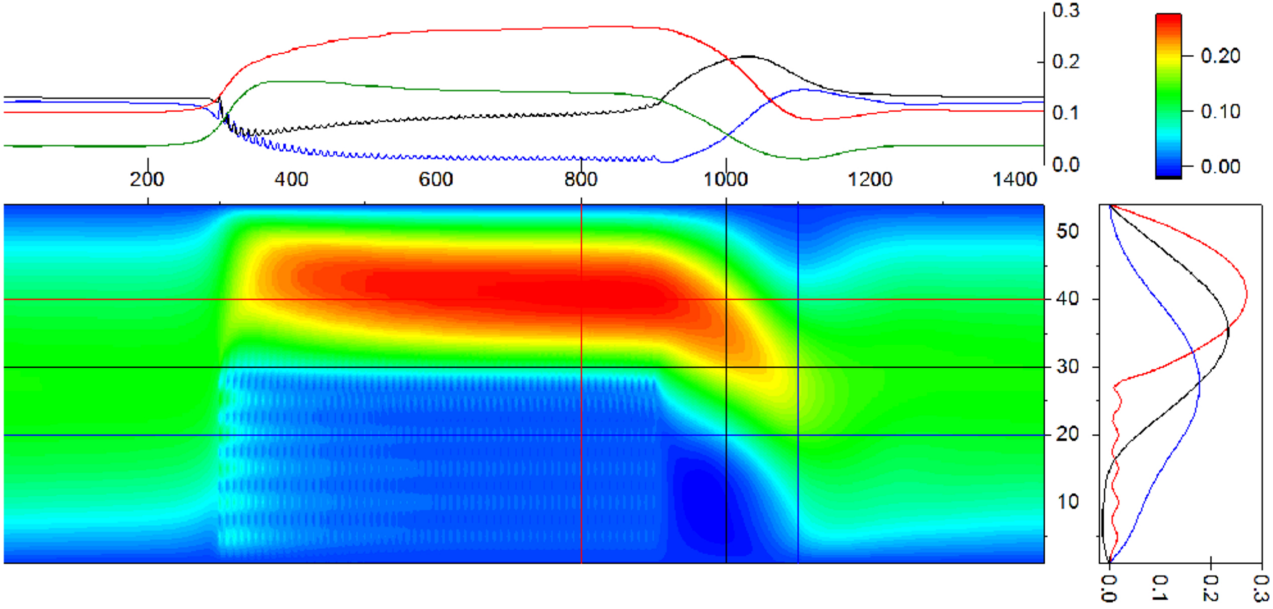
```
def streaming(XMAX,YMAX,ZMAX,f,ftemp,S):
z = cuda.blockIdx.x
y = cuda.blockIdx.y;
x = cuda.threadIdx.x;
if ((x<XMAX) and (y<YMAX) and (z<ZMAX)):
x_e = ((x+1) % XMAX)
y_n = ((y+1) % YMAX)
z_t = ((z+1) % ZMAX)
x_w = (x-1+XMAX) % XMAX
y_s = (y-1+YMAX) % YMAX
z_b = (z-1+ZMAX) % ZMAX
ftemp[0,x,y,z] = f[0,x,y,z] + S[0,x,y,z]
ftemp[1,x_e,y,z] = f[1,x,y,z] + S[1,x_e,y,z]
ftemp[2,x_w,y,z] = f[2,x,y,z] + S[2,x_w,y,z]
ftemp[3,x,y_n,z] = f[3,x,y,z] + S[3,x,y_n,z]
ftemp[4,x,y_s,z] = f[4,x,y,z] + S[4,x,y_s,z]
ftemp[5,x,y,z_t] = f[5,x,y,z] + S[5,x,y,z_t]
ftemp[6,x,y,z_b] = f[6,x,y,z] + S[6,x,y,z_b]
ftemp[7,x_e,y_n,z] = f[7,x,y,z] +
S[7,x_e,y_n,z]
ftemp[8,x_e,y_s,z] = f[8,x,y,z] +
S[8,x_e,y_s,z]
ftemp[9,x_e,y,z_t] = f[9,x,y,z] +
S[9,x_e,y,z_t]
ftemp[10,x_e,y,z_b] = f[10,x,y,z] +
S[10,x_e,y,z_b]
ftemp[11,x_w,y_n,z] = f[11,x,y,z] +
S[11,x_w,y_n,z]
ftemp[12,x_w,y_s,z] = f[12,x,y,z] +
S[12,x_w,y_s,z]
ftemp[13,x_w,y,z_t] = f[13,x,y,z] +
S[13,x_w,y,z_t]
ftemp[14,x_w,y,z_b] = f[14,x,y,z] +
S[14,x_w,y,z_b]
ftemp[15,x,y_n,z_t] = f[15,x,y,z] +
S[15,x,y_n,z_t]
ftemp[16,x,y_n,z_b] = f[16,x,y,z] +
S[16,x,y_n,z_b]
ftemp[17,x,y_s,z_t] = f[17,x,y,z] +
S[17,x,y_s,z_t]
ftemp[18,x,y_s,z_b] = f[18,x,y,z] +
S[18,x,y_s,z_b]
```

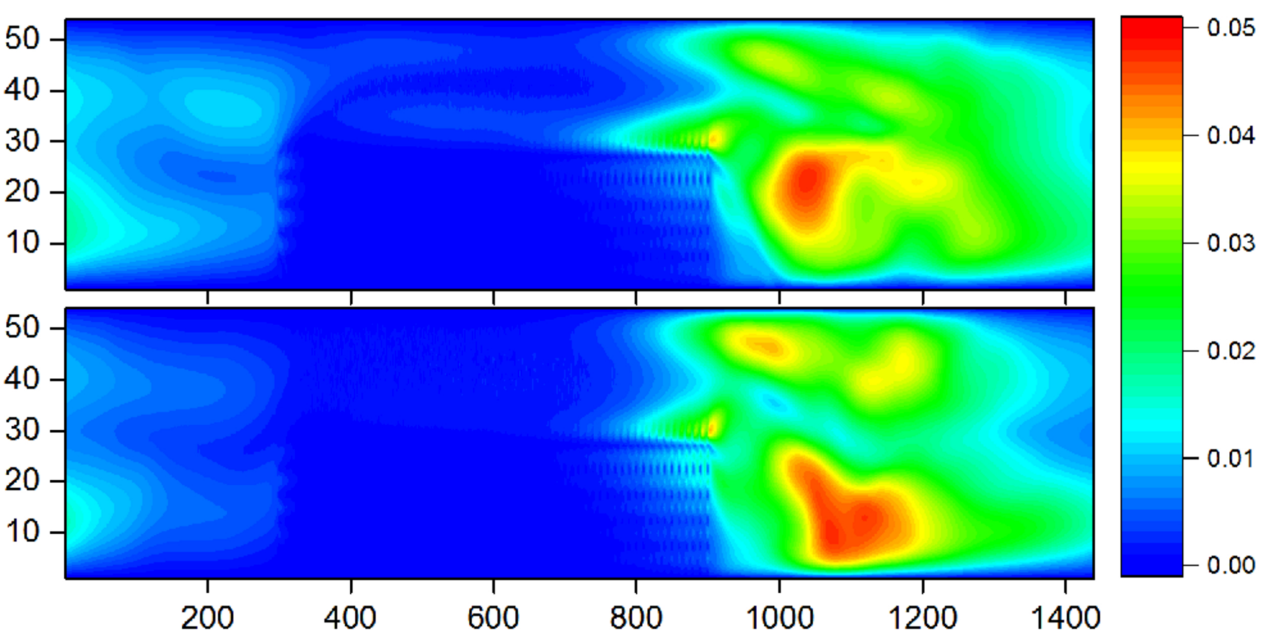

d=0.25
5
25
FLOW
Y
X
Z
300
600
540
80
26
54

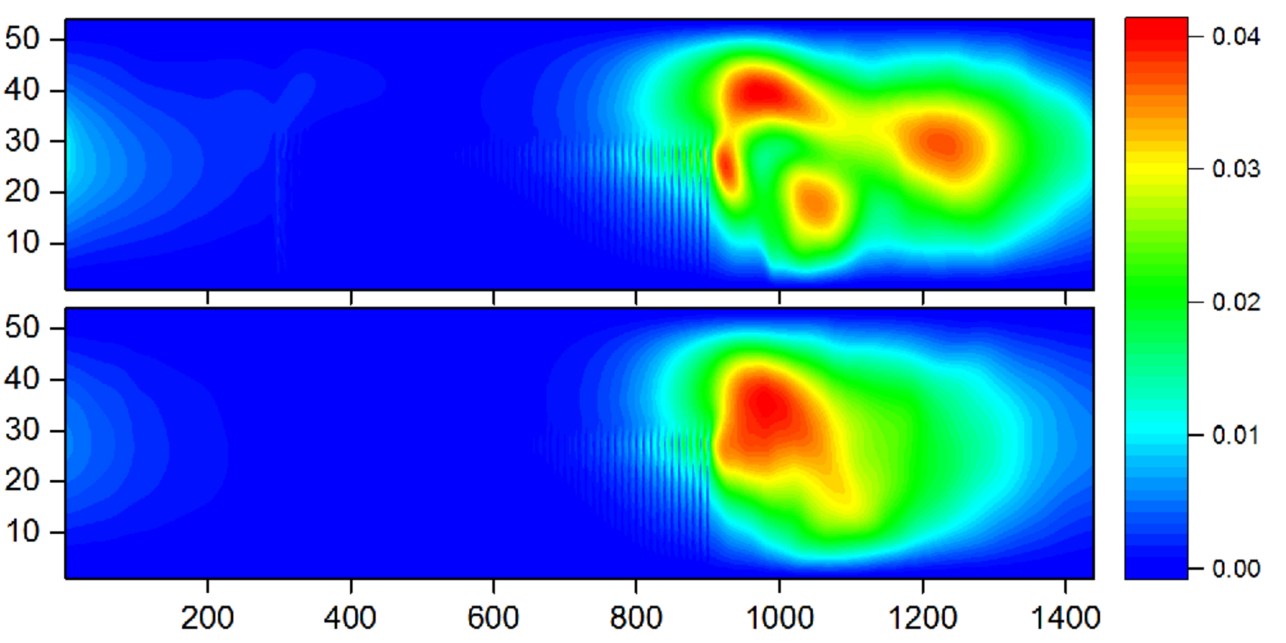

0.3
0.2
0.1
0.0
200
400
600
800
1000
1200
1400
0.20
0.10
0.00
50
40
30
20
10
0.0
0.1
0.2
0.3

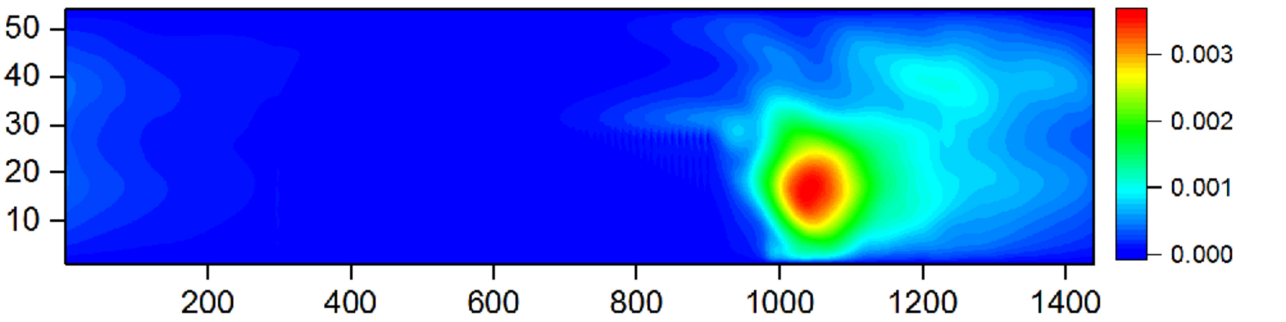

0.12
0.08
0.04
0.00
200
400
600
800
1000
1200
1400
0.10
0.05
0.00
50
40
30
20
10
0.00
0.04
0.08
0.12

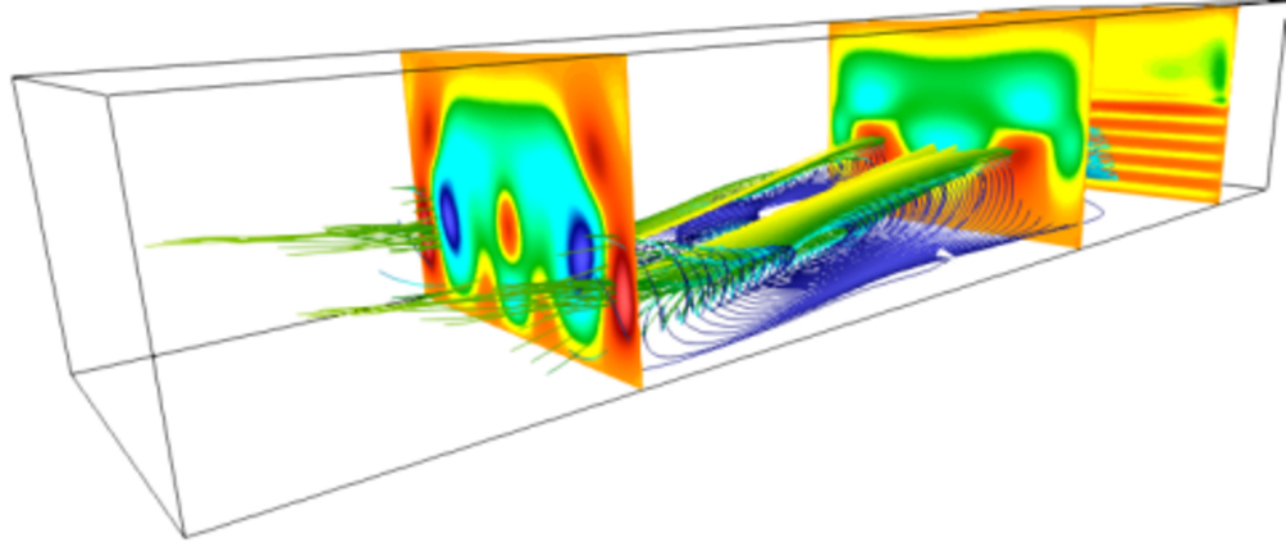

0.3
0.2
0.1
0.0
200
400
600
800
1000
1200
1400
0.20
0.10
0.00
50
40
30
20
10
0.0
0.1
0.2
0.3

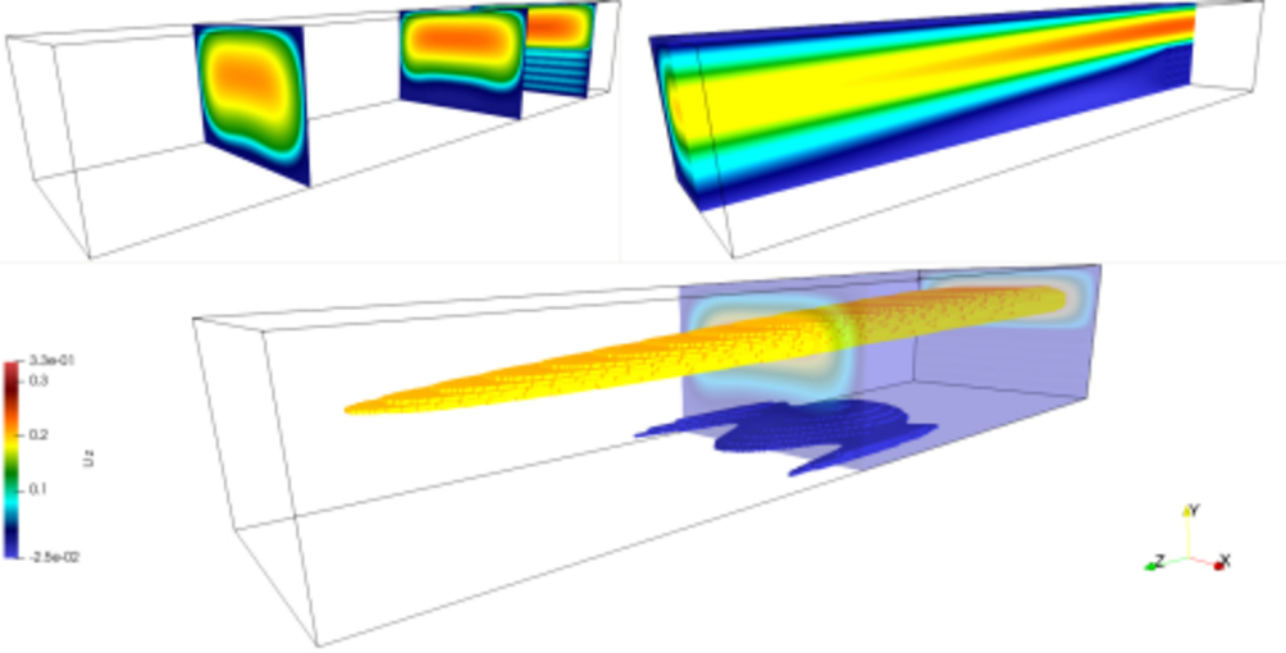

0.3
0.2
0.1
0.0
200
400
600
800
1000
1200
1400
0.20
0.10
0.00
50
40
30
20
10
0.0
0.1
0.2
0.3

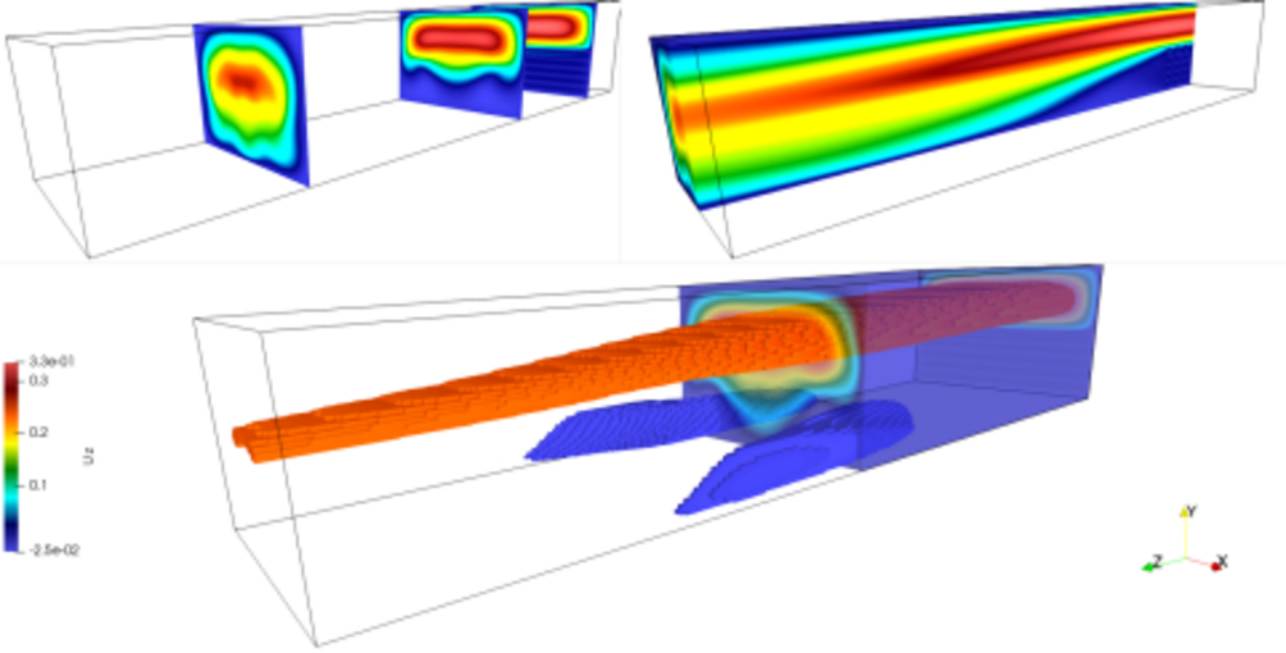

50
40
30
20
10
50
40
30
20
10
200
400
600
800
1000
1200
1400
0.05
0.04
0.03
0.02
0.01
0.00

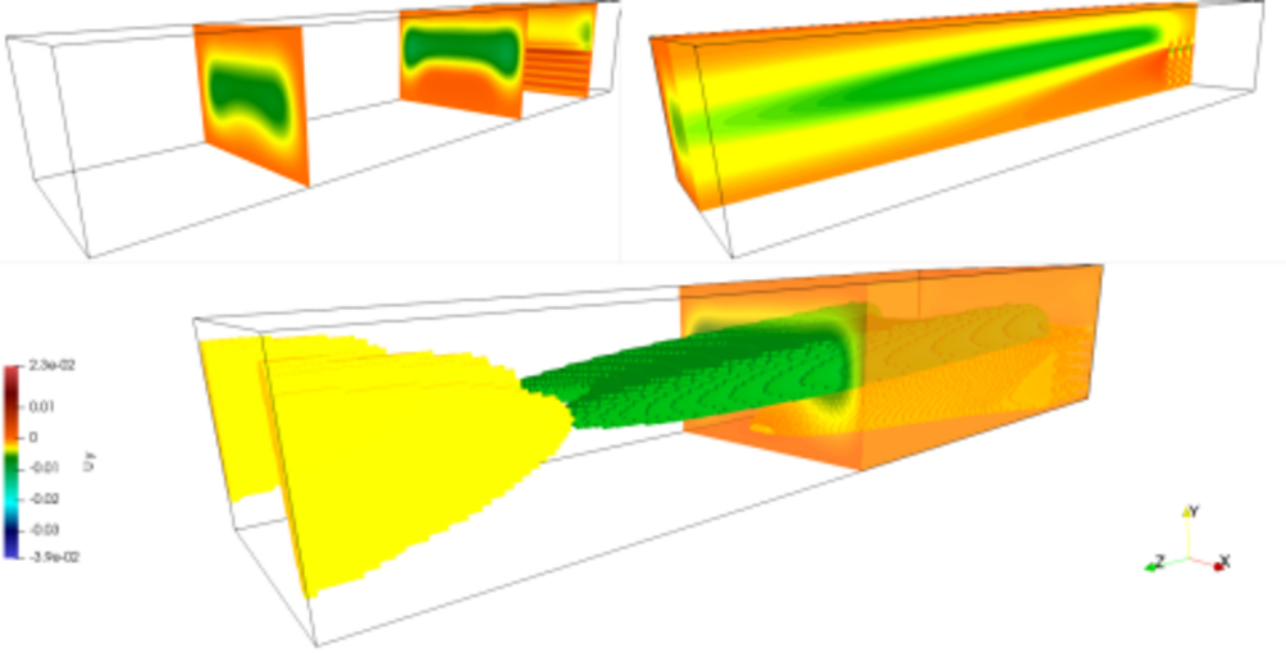

50
40
30
20
10
50
40
30
20
10
200
400
600
800
1000
1200
1400
0.04
0.03
0.02
0.01
0.00

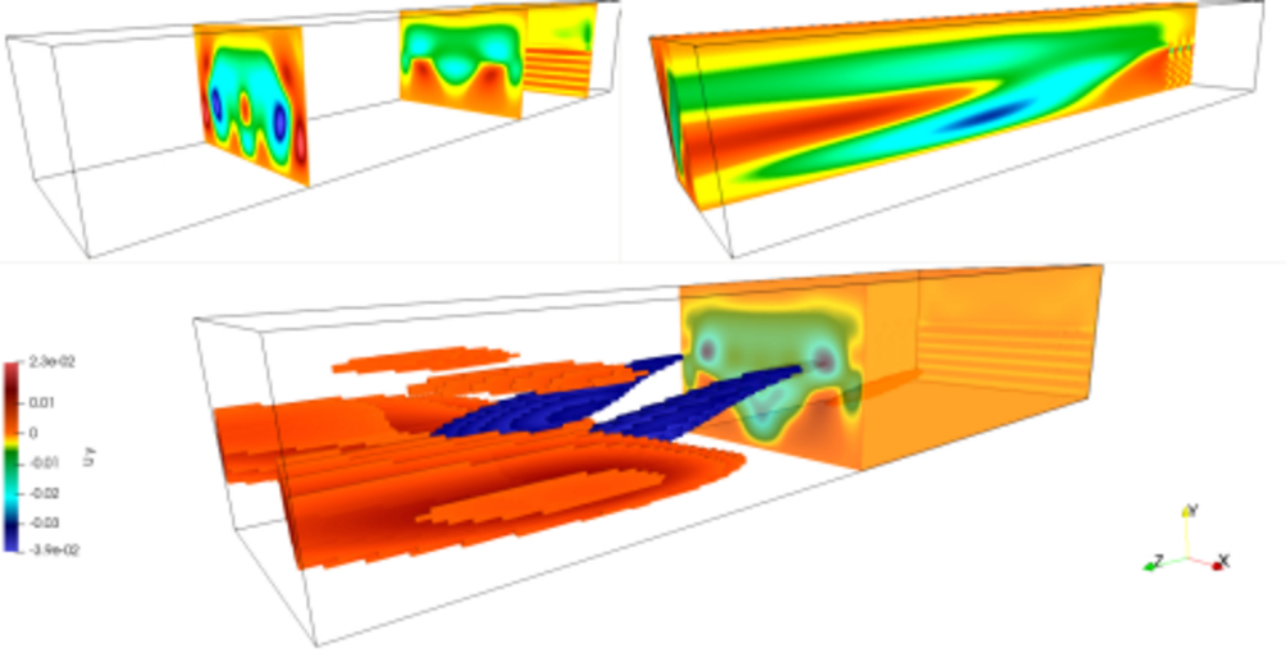

50
40
30
20
10
200
400
600
800
1000
1200
1400
0.003
0.002
0.001
0.000

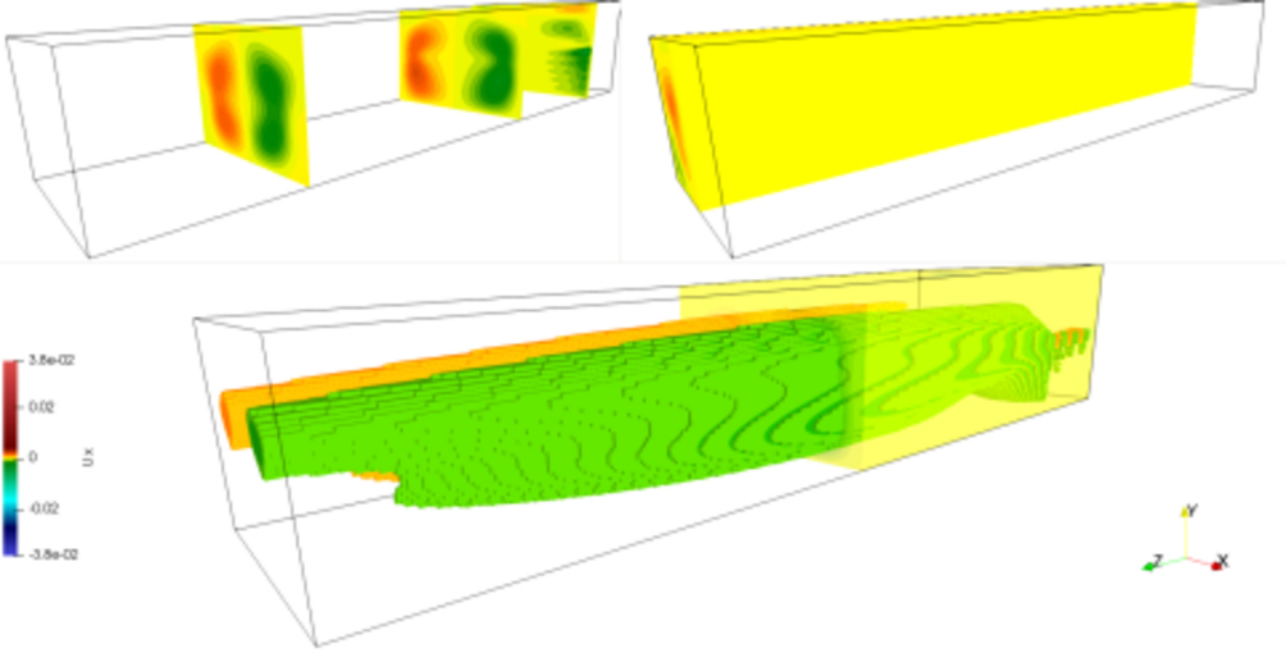


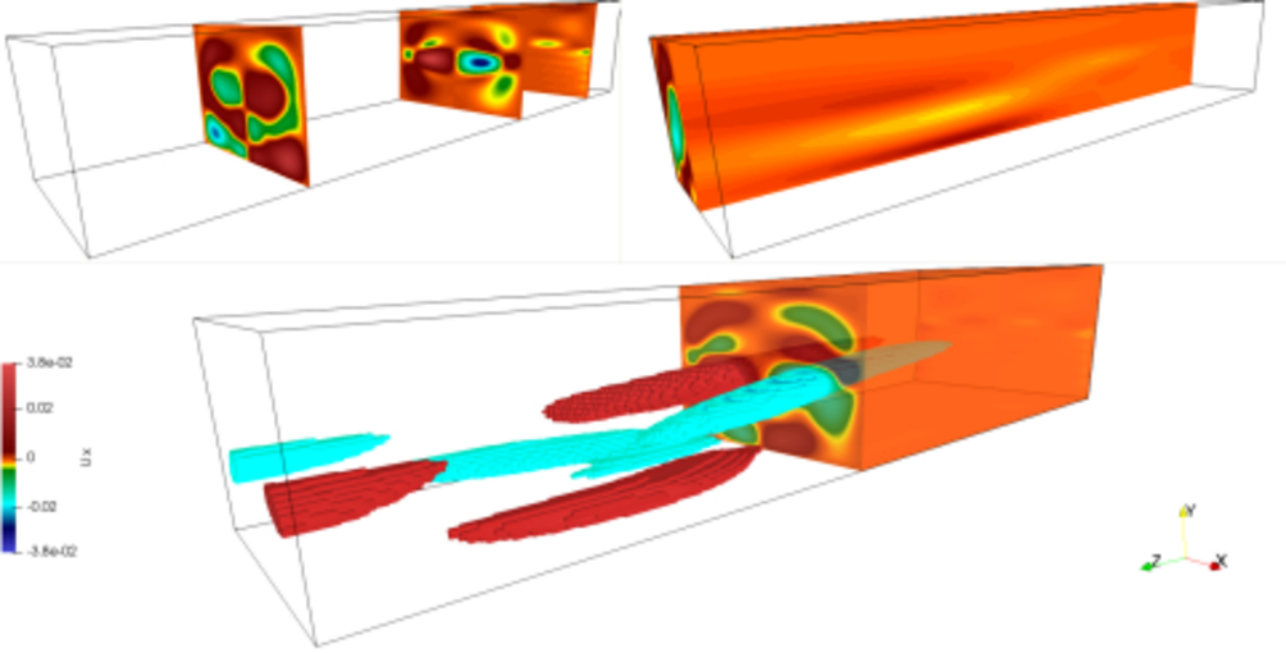

3.3e-01
0.3
0.2
0.1
-2.5e-02
Uz
Y
Z
X

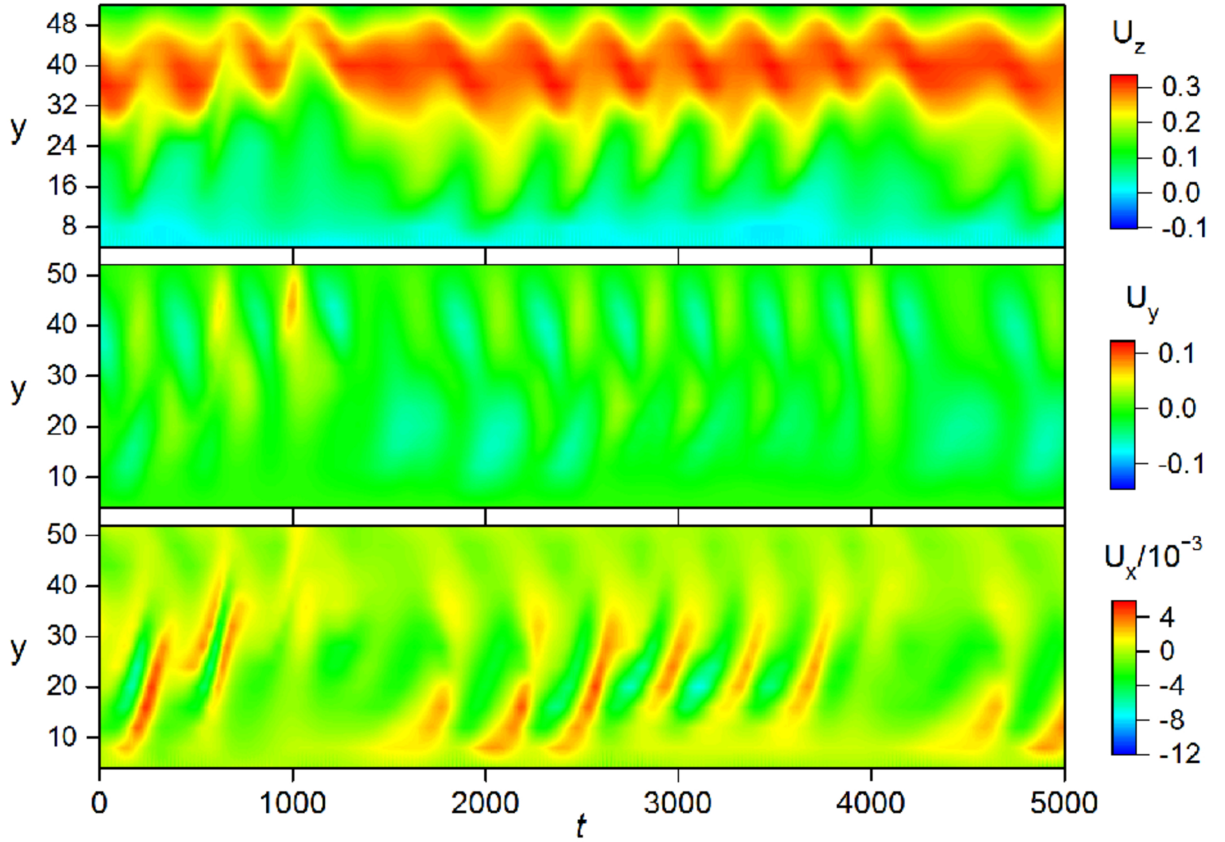

3.3e-01
0.3
0.2
0.1
-2.5e-02
S
Y
Z
X

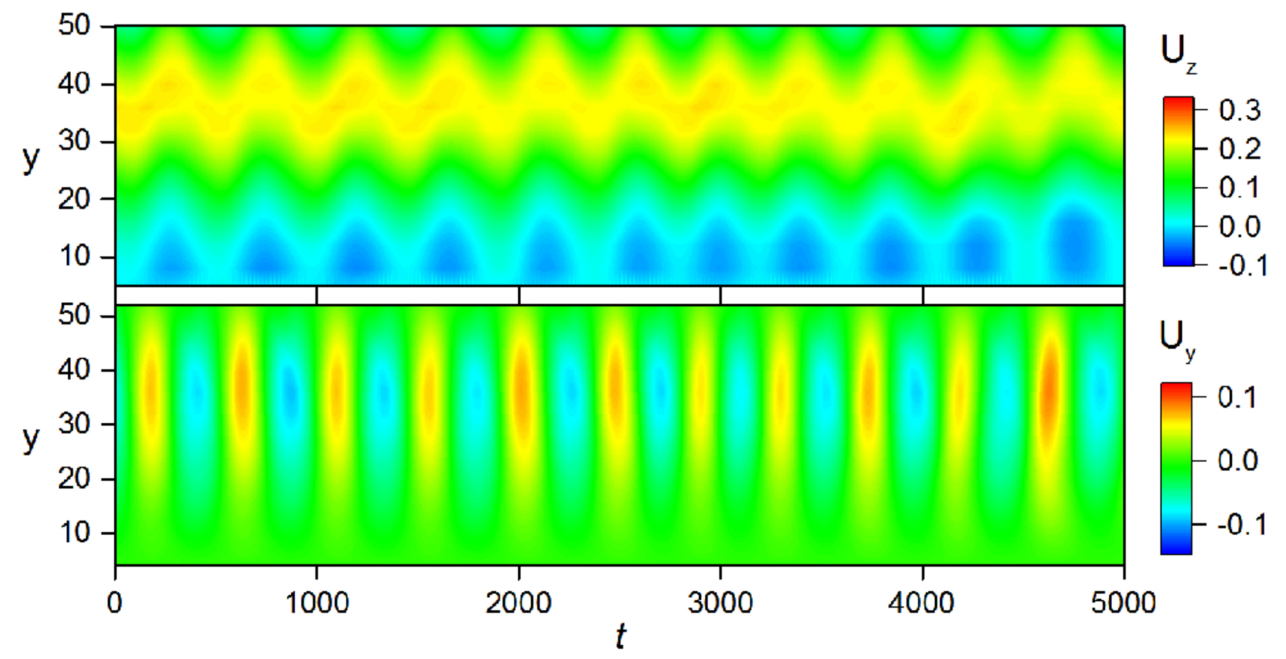

2.3e-02
0.01
0
-0.01
-0.02
-0.03
-3.9e-02
Y
Y
Z
X

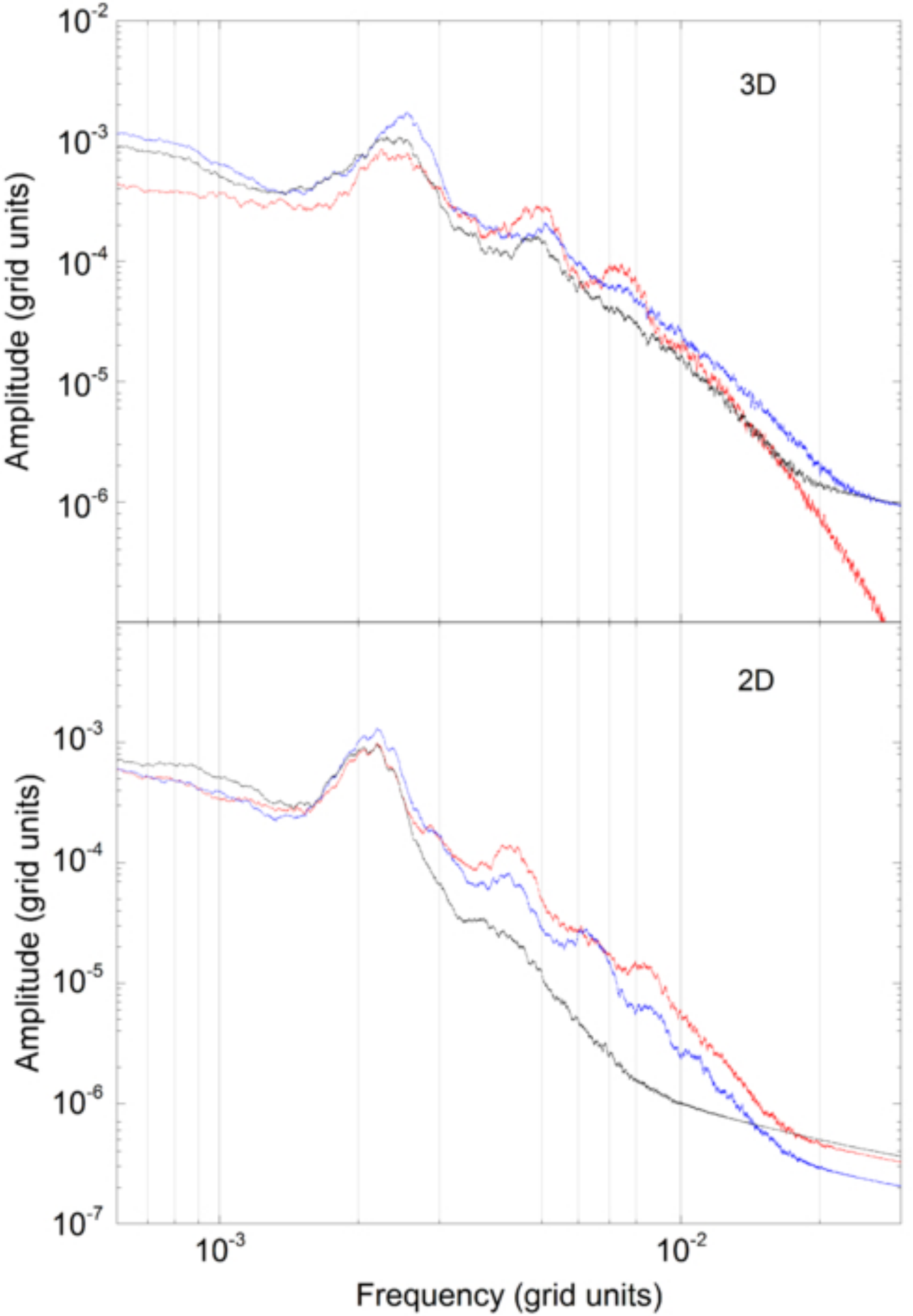

2.3e-02
0.01
0
-0.01
-0.02
-0.03
-3.9e-02
Y
Y
Z
X

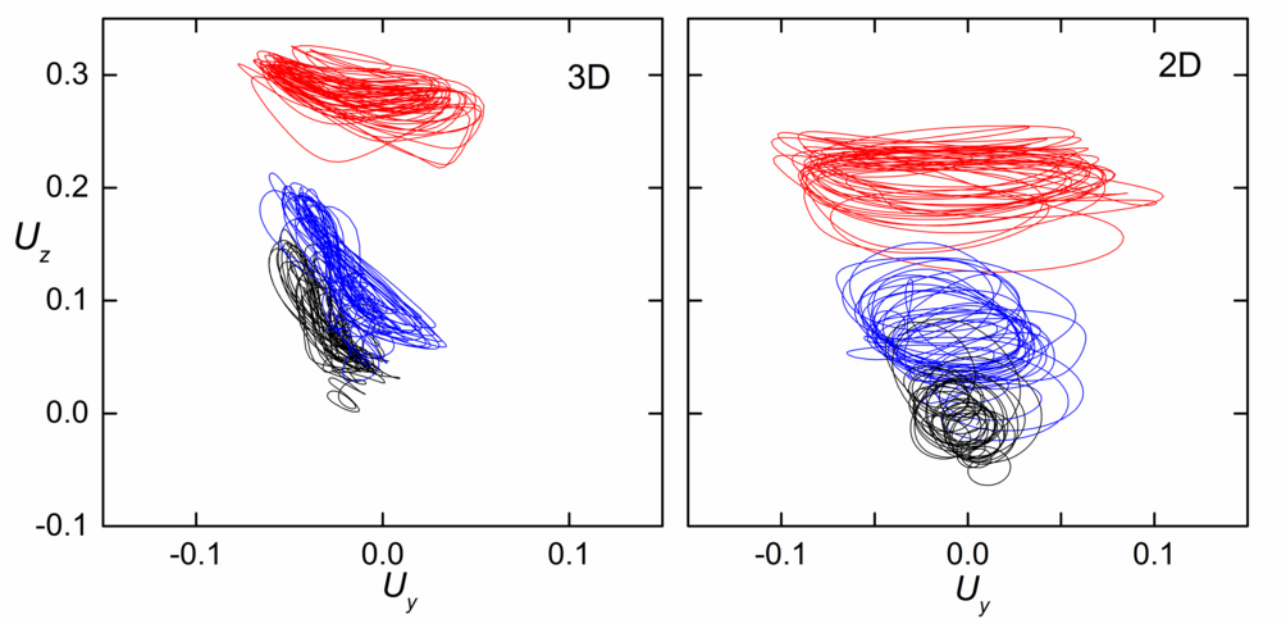

3.8e-02
0.02
0
-0.02
-3.8e-02
U:X
Y
Z
X


3.8e-02
0.02
0
-0.02
-3.8e-02
U:x
Y
Z
X


y
48
40
32
24
16
8
U_z
0.3
0.2
0.1
0.0
-0.1
y
50
40
30
20
10
U_y
0.1
0.0
-0.1
y
50
40
30
20
10
$U_x/10^{-3}$
4
0
-4
-8
-12
0
1000
2000
3000
4000
5000
t


y
50
40
30
20
10
U_z
0.3
0.2
0.1
0.0
-0.1
y
50
40
30
20
10
U_y
0.1
0.0
-0.1
0
1000
2000
3000
4000
5000
t


3D
2D
Amplitude (grid units)
Amplitude (grid units)
Frequency (grid units)
10^-2
10^-3
10^-4
10^-5
10^-6
10^-3
10^-4
10^-5
10^-6
10^-7
10^-3
10^-2


3D
2D
0.3
0.2
0.1
0.0
-0.1
U_z
-0.1
0.0
0.1
U_y
-0.1
0.0
0.1
U_y