The VM2D Open Source Code for Incompressible Flow Simulation by Using Meshless Lagrangian Vortex Methods on CPU and GPU

Kseniia S. Kuzmina
Bauman Moscow State Technical University,
Ivannikov Institute for System Programming of the RAS
Moscow, Russia
kuz-ksen-serg@yandex.ru

Ilia K. Marchevsky
Bauman Moscow State Technical University,
Ivannikov Institute for System Programming of the RAS
Moscow, Russia
iliamarchevsky@mail.ru

Evgeniya P. Ryatina
Bauman Moscow State Technical University,
Ivannikov Institute for System Programming of the RAS
Moscow, Russia
evgeniya.ryatina@yandex.ru

***Abstract*—The main features and data structure of the VM2D code is described. VM2D is the original code that implements the meshless Lagrangian vortex methods for two-dimensional viscous incompressible flows simulation. The code is open source and supports parallel technologies OpenMP, MPI and Nvidia CUDA.**

The VM2D code can be useful for flow simulation around airfoils as well as system of airfoils and unsteady hydrodynamic loads computation. It is possible to simulate flow around immovable airfoils, airfoils moving according to the given law and solve fluid-structure interaction problems in weakly coupled and strongly coupled statement. Known algorithms of vortex methods and original improvements developed by authors are implemented.

The current version of source code of the VM2D is available on GitHub under GNU GPL license (https://github.com/vortexmethods/VM2D)

***Index Terms*—vortex method, incompressible flow, hydrodynamic loads, fluid-structure interaction, hydroelasticity, open source code, boundary integral equation**

I. Introduction

Vortex methods of flow simulation around airfoils are based on considering the vorticity as a primary computed variable and on the fundamental principle, which was discovered by prof. N. E. Zhukovsky in 1906: an immovable airfoil influences the inviscid incompressible flow just as attached vortex sheet placed on its surface line [1]. So it is possible to replace airfoil with vortex sheet and to determine somehow the intensity of this vortex sheet. For some airfoils of simple shapes (elliptical airfoils and Zhukovsky wing airfoils) it can be found by using conformal mappings technique, such solutions can be used as benchmarks for verification of the numerical algorithms.

When solving the Navier — Stokes equations the classical Zhukovsky approach remains correct in principle both for movable and immovable airfoils. However, it is necessary to simulate vorticity flux [2] from the body surface to the flow. It means that the vortex sheet which simulates the airfoil surface line influence should be considered as free vortex sheet instead of the attached one. According to Lighthill's approach this phenomena can be modelled as a result of vorticity generation on the surface line K due to vorticity flux action within the small time period. Then the vorticity, which is concentrated in this vortex sheet with intensity $\gamma(\boldsymbol{r})$, $\boldsymbol{r} \in K$, becomes part of the vortex wake and moves in the flow according to the governing equations. In order to simulate movable airfoil in viscous flow both attached and free vortex sheets and attached source sheet could be considered.

There are many modifications of vortex methods. The key difference between them is the approach to the viscous forces modeling. From this point of view, stochastic and deterministic algorithms can be distinguished. The stochastic approach, for example, the random walk method (called also Random Vortex Method) [3] was one of the first methods of taking viscosity effects into account. According to it, the diffusion of vorticity is simulated by a stochastic process. There are two deterministic approaches: the methods of circulations redistribution and the diffusive velocity methods. Among the circulations redistribution method, the Particle Strength Exchange method [4] and Vorticity Redistribution Method [5] can be pointed out. PSE is quite popular, and there are various two- and three-dimensional implementations [6]–[10].

In the present paper, only two dimensional flows are considered, and deterministic approach is used, called the Viscous Vortex Domains method (VVD) [11], [12], which is pure Lagrangian and belongs to a class of diffusive velocity methods. According to this approach, the vortex particles retain their circulations and move according to the velocity field, which is a superposition of the convective and diffusive velocities.

The research is funded by Russian Science Foundation (proj. 17-79-20445).

This paper describes the authors implementation of the VVD method — the VM2D code, which is open source and available on the GitHub platform. Formerly, the structure and options of the VM2D code has been described in [13]. However, the development of the VM2D continues, so the aim of this paper is to describe the current status (September, 2019) and capabilities of the VM2D code.

The VM2D code is cross-platform and it has a modular structure. Parallel algorithms in VM2D are implemented using OpenMP, MPI and Nvidia CUDA technologies, that allows performing computations on multiprocessor systems with classical (CPU) architecture and using GPU accelerators. In VM2D it is possible to simulate unsteady incompressible flows around airfoil or system of airfoils, calculate hydrodynamic loads acting the airfoils, solve fluid-structure interaction (FSI) problems when the airfoils move under the hydrodynamic loads. It is also possible to simulate internal flows in VM2D.

II. Flow variables reconstruction

In meshless Lagrangian vortex methods which are used for incompressible flow simulation, vorticity is primary computed variable, while velocity and pressure fields can be reconstructed by using known vorticity distribution in the flow.

Vorticity distribution in the flow is simulated by set of vortex elements — elementary vorticity fields, which correspond to circular vortices with some given shape of vorticity distribution, for example, Rankine's vortex or Lamb's vortex (Fig. 1).

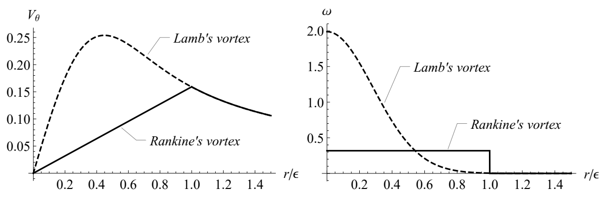

Fig. 1. Circumferential velocity induced by Rankine's and Lamb's vortices with unit circulation and vorticity distributions; ϵ is Rankine's vortex radius, Lamb's vortex parameters are such that more, than 0,998 of total vorticity is inside the ϵ-circle; difference of velocities at $r/\epsilon > 1$ is less than 0,2 %

By default, Rankine's vortex model is used, in order to use smooth velocity profile of the Lamb's vortex, the following directive should be specified: `#define LAMBVORTEX`

A. Velocity reconstruction

The simplest way to compute flow velocity at arbitrary point $\boldsymbol{r}$ in the flow domain S is the Biot-Savart law usage:

$$\boldsymbol{V}(\boldsymbol{r}) = \int\limits_S \frac{\Omega(\boldsymbol{\xi})}{2\pi} \frac{\boldsymbol{k} \times (\boldsymbol{r} - \boldsymbol{\xi})}{|\boldsymbol{r} - \boldsymbol{\xi}|^2} dS_\xi + \boldsymbol{V}_\infty,$$

where $\Omega(\boldsymbol{\xi})$ is vorticity field, $\boldsymbol{k}$ is unit vector, orthogonal to the flow plane; $\boldsymbol{V}_\infty$ is incident flow velocity; for simplicity we assume the airfoils to be immovable.

The computational cost of this method, however is extremely high, it is proportional to N^2, where N is number of vortex elements, because it is necessary to take into account mutual influence of all pairs of vortices. In practice it strongly restricts the capabilities of vortex methods, because time of computations becomes unacceptably high at $N \sim 10\,000 \ldots 100\,000$ even with using parallel algorithms.

In order to solve this problem fast approximate methods are being used; the most suitable are the following:

- method based on multipole expansion [14];
- fast method based on solution of auxiliary Poisson equation for stream function on a coarse mesh using Fast Fourier Transform (FFT) [15].

The computational complexity of both methods is proportional to $N \log N$, their accuracies are comparable at optimal parameters choice. Note, that in the current version of the code fast methods are not implemented, so it is strongly recommended to use GPUs in computations.

B. Pressure reconstruction and loads computation

In order to reconstruct the pressure distribution in the flow domain, an analogue of the Cauchy — Lagrange integral can be used [16]. However, in practice, as a rule it is necessary to determine hydrodynamic loads (forces and torque) acting the airfoil in the flow. It is possible to use for this purpose the integral formulae derived by prof. G.Ya. Dynnikova and adapted to several types of problems being solved by means of vortex methods [12], [17], [18]:

- flow around an immovable airfoil;
- flow around a rigid airfoil in translational motion;
- flow around a rigid airfoil in rotational motion;
- flow around a rigid airfoil in arbitrary motion.

These integral formulae have been obtained by analytical integration of the pressure distribution over the airfoil surface line. There is also approximate formula for viscous stresses computation.

III. Vorticity generation on airfoil surface line

Vorticity in viscous incompressible flow is being generated only at an airfoil surface line. The intensity of vortex sheet which is being formed during small time period is described by boundary integral equation. There are two possible ways to write down such equation.

- Singular integral equation of the first kind

$$-\frac{1}{2\pi} \oint\limits_K \underbrace{\frac{(\boldsymbol{r} - \boldsymbol{\xi}) \cdot \boldsymbol{\tau}(\boldsymbol{\xi})}{|\boldsymbol{r} - \boldsymbol{\xi}|^2}}_{Q_n(\boldsymbol{r}, \boldsymbol{\xi})} \gamma(\boldsymbol{\xi}) dl_\xi = f_n(\boldsymbol{r}), \quad \boldsymbol{r} \in K,$$

  with Hilbert-type kernel Q_n, where $\boldsymbol{\tau}$ is unit tangent vector at the corresponding point of the airfoil surface line. In order to provide the computation of the principal value of the integral in Cauchy sense, specially derived quadrature formulae should be used. In most cases Discrete Vortices Method-type quadrature formulae are used [19], their accuracy is not very high and they require special surface line discretization, that is not easy to provide in case of airfoils of complex shape or when the airfoil is deformable.

- Fredholm-type integral equation of the second kind

$$\frac{1}{2\pi}\oint\limits_K \underbrace{\frac{(\boldsymbol{r}-\boldsymbol{\xi})\cdot\boldsymbol{n}(\boldsymbol{\xi})}{|\boldsymbol{r}-\boldsymbol{\xi}|^2}}_{Q_\tau(\boldsymbol{r},\boldsymbol{\xi})}\gamma(\boldsymbol{\xi})dl_\xi - \frac{\gamma(\boldsymbol{r})}{2} = f_\tau(\boldsymbol{r}), \quad \boldsymbol{r}\in K,$$

  where $\boldsymbol{n}$ is unit normal vector, which kernel Q_τ is bounded by one half of maximal curvature of the airfoil (it means that in case of smooth airfoil the kernel is uniformly bounded function).

  For this approach the hierarchy of numerical schemes is developed based on Galerkin approach with piecewise-constant, piecewise-linear and piecewise-quadratic basis and projection functions and differen approaches to airfoil surface line discretization [20]–[22].

Right hand sides of both equations are normal and tangent components of flow velocity at the corresponding point of the airfoil surface line, respectively

$$f_n(\boldsymbol{r}) = -\boldsymbol{V}_\omega(\boldsymbol{r})\cdot\boldsymbol{n}(\boldsymbol{r}), \quad f_\tau(\boldsymbol{r}) = -\boldsymbol{V}_\omega(\boldsymbol{r})\cdot\boldsymbol{\tau}(\boldsymbol{r}),$$

where

$$\boldsymbol{V}_\omega(\boldsymbol{r}) = \boldsymbol{V}_\infty + \sum_{i=1}^{N}\boldsymbol{Q}^{(i)}(\boldsymbol{r})$$

is the influence of incident flow and vortex elements which simulate vortex wake. It is important to take into account correctly the influence of vortices, which are placed in proximity to the airfoil, first of all, in boundary layer. Numerical experiments show that the second approach and the numerical schemes, based on it, are much more accurate in comparison with the first approach [20].

IV. Vorticity evolution in the flow

Vortex wake evolution simulation in inviscid incompressible flow in vortex method is rather easy: it is necessary to solve numerically the system of ordinary differential equation

$$\frac{d\boldsymbol{r}_i}{dt} = \boldsymbol{V}(\boldsymbol{r}_i), \;\; i = 1,\ldots,N,$$

where $\boldsymbol{r}_i$ is position of the i-th vortex element. It means that vorticity is just being transferred in the flow domain with flow velocity.

In the framework of VVD [11], [12] method motion of the vortex elements simulating vortex wake is described by the ODE system

$$\frac{d\boldsymbol{r}_i}{dt} = \boldsymbol{V}(\boldsymbol{r}_i) + \boldsymbol{W}(\boldsymbol{r}_i), \;\; i = 1,\ldots,N,$$

while their circulations remain the same.

For the so-called diffusive velocity computation

$$\boldsymbol{W}(\boldsymbol{r}) = -\nu\nabla\ln|\Omega(\boldsymbol{r})| = -\nu\frac{\nabla\Omega(\boldsymbol{r})}{\Omega(\boldsymbol{r})},$$

which is proportional to kinematic viscosity ν and depends on vorticity distribution in the flow domain in proximity to point $\boldsymbol{r}$ and on the shape of the flow region boundary (if there is such in neighborhood to $\boldsymbol{r}$), efficient approach is known, which is also considered as an important part of VVD method.

V. The open source VM2D code

Vortex method is powerful tool for numerical simulation in number of engineering application: from aerodynamical coefficients estimation for aircrafts and aircraft trails simulation to hydroelastic oscillations simulation of structural elements which interact with the flow, and industrial aerodynamics problems solving. The range of the problems for which vortex methods are applicable, is limited by the flow regimes with low subsonic Mach numbers, when the influence of flow compressibility can be neglected.

For such problems, especially hydroelastic ones, vortex methods can be very efficient, at least in comparison with "traditional" mesh methods, where it is necessary to deform or reconstruct mesh at every time step. Moreover, for vortex methods computational cost of hydroelastic problem solution remains nearly the same as for flow simulation around immovable airfoil, and, in fact it is much easier to implement numerical scheme with strong coupling (again, in comparison with mesh methods).

But at the same time vortex methods are not implemented in widespread software. Scientific groups, which develop new modifications of vortex methods, of course, have their own codes, but such codes are 'private', they aren't available for other scientists and engineers. As far as authors know, there are no open-source codes that implement vortex methods (both 2D and 3D). The authors have developed such a code `VM2D` for 2D incompressible flow simulation around movable and immovable airfoils, including solving coupled fluid-structure interaction (FSI) problems. Source code is open and it is available on GitHub: https://github.com/vortexmethods/VM2D.

A. The structure of the VM2D code

The source code is written in C++ language and has modular structure. It is cross-platform software and can be compiled under Windows, Linux and MacOS by using MSVC, GCC, Intel C++ Compiler, Clang compilers (as well as other ones supporting the C++11 standard). The Eigen external library is used in VM2D for the numerical solution of linear equations systems. The OpenMP, MPI and Nvidia CUDA [23] technologies are used for computation acceleration on multi-core and multiprocessor cluster systems, including hybrid architectures with graphic accelerators [23]. There is also doxygen-documentation for the VM2D, to date only in Russian.

B. Problems description in VM2D

It is possible to use `VM2D` for solution of one particular problem as well as for solution of the set of similar (or not similar) problems. For every problem to be solved separate directory should be created with the same name as problem's name. List of problems should be saved in `problems` text file, which normally has the following structure:

```
problems = {
wing00(np = 1, angle =  0, tau = 0.015),
wing05(np = 2, angle =  5, tau = 0.015),
wing10(np = 4, angle = 10, tau = 0.010)
};
```

In the simplest case it is enough to specify just empty brackets; in this case the parameter values are set by default. Parameter `np` set number of processors, which will be used for problem solving in parallel mode by using MPI technology (by default equal to 1). Note, that for every multicore processor OpenMP technology is used for parallelization of the algorithm in shared-memory mode. Other parameters specified in brackets will be passed to the corresponding passport files where they can be reached through the names consist of `$` and parameter name, i. e., for the described case, in the passport files, parameters `$angle` and `$tau` are available: the first one takes the values 0.0, 5.0 and 10.0 for corresponding problems and the second one — 0.015, 0.015 and 0.010.

When solving several similar tasks that differ only in some input parameters (for example, incident angle or time step, as shown in the example above), you can use the following feature, which allows to eliminate the need for manual preparation of passport files of same type. File `problems` should contain a list of tasks, and for each task the name of "universal" passport file that contains common data for all tasks should be set as the value of the parameter `copyPspFile`. In this case, the parameters to be changed in the universal passport file must be specified through formal variables with `$` and their specific individual values must be listed as parameters in parentheses after the task name in the file `problem`. In this case, directories with the necessary passports for all tasks will be generated automatically, that simplifies the initial data preparation procedure. The example of `problems` file for described case is shown below.

```
problems = {
wing00(np = 1, angle =  0, tau = 0.015,
       copyPspFile="wingBase/uniPassport"),
wing05(np = 2, angle =  5, tau = 0.015,
       copyPspFile="wingBase/uniPassport"),
wing10(np = 4, angle = 10, tau = 0.010,
       copyPspFile="wingBase/uniPassport")
};
```

The following syntaxis is used in the input files:

- `//` — single-line comment;
- `/* ...*/` — multi-line comment;
- end of line — the same as the space;
- `;` — separating character between parameters;
- `,` — separating character between parameters in a list (in brackets) or separating character between components of arrays or vectors;
- spaces and tabs are ignored;
- the names of parameters are case insensitive.

Here is an example of the `passport` file.

```
//Physical Properties
rho = 1.0;
vInf = {1.0, 0.0};
vRef = 1.0;
nu = 0.001;

//Time Discretization Properties
timeStart = 0;
timeStop = 10.0;
timeAccel = 1.0;
dt = $tau;
nameLength = 5;

saveTXT = 0;
saveVTK = 10;
saveVP = 100;

//Wake Discretization Properties
eps = 0.015;
epscol = 0.010;
distFar = 10.0;
delta = 1e-5;
vortexPerPanel = 1;
maxGamma = 0.002;

//Airfoils and wake
airfoil = {
 wing(
  basePoint = {0.0, 0.0},
  scale = 1.0,
  angle = $angle,
  mechanicalSystem = mech1
 )
};
fileWake = {};
fileSource = {};
```

The sense of parameters in the passport is described in Table I.

TABLE I
MAIN PARAMETERS IN VM2D PASSPORT FILE

Parameter name	Brief description
Physical properties	
`rho`	density of the flow
`vInf`	incident flow velocity
`vRef`	reference velocity value
`nu`	kinematic viscosity of the flow
Time discretization properties	
`timeStart`	physical time at which the simulation starts (normally 0.0)
`timeStop`	physical time at which the simulation stops
`timeAccel`	time period for incident flow uniform acceleration from zero velocity to `vInf` value
`dt`	time step
`nameLength`	number of digits in name of the files with simulation results (by default 5)
`saveTXT`	period (in steps) for vortex wake storage to text file
`saveVTK`	period (in steps) for vortex wake storage to VTK file
`saveVP`	period (in steps) for velocity and pressure computation at specified points and storage of the result
Wake discretization properties	
`eps`	typical radius of the vortex element
`epscol`	maximal distance at which two vortices could be merged into one summary vortex (only if some additional conditions are satisfied)
`distFar`	distance at which vortex wake is removed form the simulation
`delta`	small distance at which vortex elements are placed over the airfoil surface line after being generated
`vortexPerPanel`	minimal number of vortices generated at each panel
`maxGamma`	maximal value of the vortex element circulation

Files with airfoils geometry are stored in `settings/airfoils` directory; they are text files with very simple format, which is absolutely clear from examples. Note that the airfoil surface line is specified by the panels endings; the direction of traversal is counterclockwise. For each airfoil the following parameters can be specified after the file name inside the brackets:

- `basePoint` — point where the airfoil reference point should be placed (by default is (0.0, 0.0));
- `scale` — scale factor for the airfoil (by default equal to 1.0);
- `angle` — angle of incidence (by default equal to 0.0);
- `inverse` — switcher, that should be set to 'true' for internal flow simulation ('false ' by default);
- `mechanicalSystem` — numerical scheme for coupling strategy implementation in coupled FSI problems; the value should correspond to one of the values from the list of mechanical systems specified in `mechanics` file.

In passport file all the parameters being used in simulation should be defined. However, for some of them there are predefined default values. Default values with the lowest priority are specified just inside the code, defaults with higher priority can be specified by user in `defaults` file. The decryption (in term of integer values) for verbal expressions of some options (such as keyword `mechanicsRigidImmovable` means that airfoil is rigid and immovable. In the file `switchers`, each keyword is associated with integer values used in source code.

If the flow around system of airfoils should be simulated, it is possible to specify more than one airfoil; in this case the corresponding section of the passport file has the following structure:

```
airfoil = {
 square_160points(
  basePoint = {0.0, 0.0},
  angle = 45.0,
  scale = 1.0
 ),
 circle_200points(
  basePoint = {1.2, -0.2},
  angle = 0.0,
  scale = 0.5
 ) };
```

In this example the interference phenomenon for two airfoils is being simulated: small (2 times scaled) circular airfoil placed behind the square airfoil (installed at angle of incidence $45°$) in its vortex wake. All other parameters will be set to default values.

If user wants to deal with previously simulated vorticity distribution, the positions and circulations of the vortices can be uploaded by specifying the corresponding file name in section `fileWake` of the passport. Files with vortex wakes description should be stored in `settings/wakes` directory.

The internal flows can be simulated by uploading the sets of sources and sinks. Their positions and intensities should be specified in the corresponding file.

The coordinates of the points where it is necessary to calculate the velocity field and pressure in the flow region should have been previously stored to the file `pointsVP`.

C. Documentation

Programmer's guide to `VM2D` is being generated automatically by using `doxygen` tool. It includes full information about all the classes implemented in `VM2D`: description of all the class members and class methods. Relationships between the classes are also shown in graphical mode, as well as execution diagrams of the functions. Html-version of documentation is available at https://vortexmethods.github.io/VM2D/ and it is being updated automatically via `Travis-CI` service after every modification of source code and its push on GitHub.

D. Results of simulation

The results of simulation are being saved in files in the directory with the same name as the problem being solved. Files, which contain the description of vortex wake at particular time steps (every `saveTXT` or `saveVTK` steps) are saved in subdirectory `snapshots` it text or/and VTK format.

Files containing the values of the velocity and pressure, computed at the specified points, are stored in a subdirectory `velocityPressure`. For the convenience of processing this data in different cases, there are two formats for saving data. If it is necessary to save the values of pressures and velocities at all points at different files at each time step, the points should be listed in section `points` of the file `pointsVP`. If it is necessary to save evolution of velocity and pressure in different files for each point, the points must be listed in the `history` section. One can use both options together.

Hydrodynamic loads acting the airfoils are saved for all time steps in `forces-airfoil-n` files; positions and velocities of the airfoils are saved in `position-airfoil-n` files; time statistics is being stored in `timestat` file; program log is being shown on screen (it can be redirected to file by using standard command prompt/shell operator <).

E. Main classes in VM2D

The `main` function (the entry point of C++ program) is determined in `VM2D.cpp` file. Its structure is very simple; `queue` class instance is created, list of problems to be solved is loaded and 'numerical conveyer' is being started. It runs until all the problem from the list, described in `problems` file are solved. All available processors, which work by using MPI, are being split into subgroups, according to number of processors required for simulation in particular problems. If number of processors is less then total number of processors required for simultaneous solution of all the problems from the list, class `queue` works as real queue with 'fifo' discipline. Note, that all the processors subgroups work in asynchronous mode; global synchronization is performed every some ΔT seconds (the so-called 'time quantum'), when the queue state is being updated, finished problems are replaced with new ones, *etc*.

The basic and auxiliary classes defined in `VM2D` are listed in Table II:

TABLE II
CLASSES DEFINED IN VM2D

Class name	**Brief description**
	Basic classes
`Queue`	stores the list of problems to be solved, organizes its solution in parallel mode according to number of required processors and total number of available processors
`Task`	stores the state (in the `queue`) of particular problem and its description (passport)
`Passport`	stores full definition of the problem (its passport)
`World2D`	describes all the properties and current state of the particular problem from the queue; the instance of this classs is the main object in numerical simulation of the flow around airfoils
`Airfoil`	*abstract class* which describes the geometry of the airfoil
`Sheet`	determines attached and free vortex sheets and attached source sheets which are placed on airfoils surface lines
`VirtualWake`	vortex elements which simulate vortex sheet when it is transferred to the vortex wake
`Wake`	describes vortex wake
`Boundary`	*abstract class* which determines the numerical scheme being used for integral equation solution with respect to unknown free vortex sheet intensity
`Velocity`	*abstract class* which determines numerical method of velocities computation in flow domain
`Mechanics`	*abstract class* which determines model of coupled hydroelastic problem and coupling
`MeasureVP`	velocity and pressure computation
	Auxiliary classes
`numvector`	*template class* which determines the array of specified length consist of variables of specified type; for 'numerical' vectors all basic arithmetic operations are defined, including vector product (`operator^`) for 3-dimensional vectors consist of `float` and `double` variables
`Point2D`	inherits `numvector<double, 3>` and has the necessary MPI-descriptor
`Vortex2D`	stores properties of vortex element (its position and circulation) and includes the corresponding MPI-descriptor
`WakeDataBase`	data base of vortex elements; base class for classes Wake and VirtualWake
`Gpu`	provides the ability to perform calculations on the GPU using Nvidia CUDA technology
`Parallel`	stores the properties of the MPI-communicator created for the particular problem
`Preprocessor`	stores the tools for input files preprocessing: comments exclusion, linebreaks exchange with spaces, multiple spaces and tabs removal; the result is used as input data for `StreamParser`
`StreamParser`	stores the tools for input files (after being preprocessed) parsing; it is used for all text files reading, including `problems`, `defaults`, `passport` files loading, *etc*
`LogStream`	a tool for displaying messages about the programme monitoring process and the state of each problem to be solved
`Times`	class with structures for time statistics assembling ant tools for its saving to `timestat` file.

In files `defs.h` and `defs.cpp` the necessary "universal" constants and functions are defined.

F. Abstract classes implementations

There are four main abstract classes in `VM2D`:

- `Airfoil`,
- `Boundary`,
- `Velocity`,
- `Mechanics`,

which implementations correspond to different modifications of vortex methods; some of them are briefly described in the beginning of this paper.

The inheriting classes have names, which consist of the name of parent class and some additional words that specifies the particular method implemented there.

Class `Airfoil` determines the way of airfoil surface line discretization. For the day, there are only one implementation `AirfoilRect`, which corresponds to airfoil surface line discretization by curvilinear panels. In the future, the class `AirfoilCurv` will be implemented for curvilinear airfoil surface line discretization.

The class `Boundary` determine the way of no-slip boundary condition satisfaction at the airfoil surface line. At the moment, only the class `BoundaryConstLayerAver` is implemented which corresponds to the "tangent" scheme with Galerkin approach with piecewise-constant basis and projection functions [20]. It is assumed that other implementations will be created that correspond to numerical schemes with piecewise-linear and piecewise-quadratic basis and projection functions and curvilinear airfoil surface line discretization.

The class `Velocity` determines the way of computations of velocities of vortex elements. At the moment, the class `VelocityBiotSavart` is implemented, which involves direct calculation of the velocities of vortex elements according to Biot — Savart law. This algorithm have complexity $O(N^2)$. In real problems, the number of vortex elements can reach hundreds of thousands or even millions, which leads to extremely large time costs for calculating velocities. The usage of parallel technologies in this case does not allow to radically solve this problem, so there is a need to use approximate fast methods, such as fast multipole method [14] and method based on Fast Fourier Transform [15]. The authors have made prototypes of parallel algorithms for these methods, which are currently at the testing stage. In the future, they will be included to VM2D as inherited classes from the `Velocity` class.

In the class `Mechanics`, the model of coupled hydroelastic problem is described, which determines the possible movement of airfoils and coupling strategy for mechanical and hydrodynamic subsystems. A list of the implementations for this class are given in Table III.

G. VM2D compilation and run

In order to compile `VM2D` code it firstly should be downloaded or cloned from `GitHub` repository:
`https://github.com/vortexmethods/VM2D`

Then, in the directory where the repository have been cloned, subdirectory `build` should be created and from there

TABLE III
IMPLEMENTATIONS FOR THE ABSTRACT CLASS AIRFOIL

Class name	Brief description
RigidImmovable	flow simulation around rigid immovable airfoil
RigidGivenLaw	flow simulation around rigid airfoil at its arbitrary prescribed motion according to the given law
RigidOscillPart	flow simulation around rigid airfoil which moves in straight-line under the linear viscoelastic constraint; weakly coupled strategy for coupling mechanical and hydrodynamic subsystems is used
RigidOscillMon	flow simulation around rigid airfoil which moves in straight-line under the linear viscoelastic constraint; monolithic approach is used
RigidRotateMon	flow simulation around rigid rotating airfoil; monolithic approach is used

the command "`cmake ..`" should be executed (or with necessary `CMake` parameters). The MPI implementation and `Eigen` library should be preliminarily installed and correctly configured in you system, of course, in addition to some C++ compiler (C++11 standard compatible).

If CUDA Toolkit is installed on the system, the program will be automatically configured in the mode of graphic accelerator usage with Nvidia CUDA technology. For correct work, it is necessary to specify the video card architecture in the file `CMakeLists.txt`.

After such preparations, the code should be compiled. In order to run the simulation, it is necessary simply to execute the program from the directory where `problems` file with list of the problems description is placed; if it is necessary — in parallel mode by using `mpirun`/`mpiexec`. In addition, using MPI technology, one can use multiple graphic accelerators.

VI. FUTURE DEVELOPMENTS

As a further development of VM2D, the following options are expected:

- implementation of algorithms for fast computation of vortex element velocities: a fast multipole method and a method based on an approximate solution of the Poisson equation using a Fast Fourier Transform;
- implementation of the numerical schemes for boundary integral equation solution based on curvilinear airfoil surface line discretization and Galerkin method with piecewise-linear and piecewise-quadratic basis functions;
- refactoring of the block `Mechanics`, with the aim of ensuring greater algorithm flexibility, that will allow to expand the list of statements of the problems to be solved.

VII. CONCLUSIONS

The main features and data structure of the VM2D code is described. VM2D is the original code that implements the two-dimensional vortex methods for two-dimensional viscous incompressible flows simulation. The code is open source and supports parallel technologies MPI, OpenMP and Nvidia CUDA.

REFERENCES

[1] G. Tokaty, A history and philosophy of fluid mechanics. Dover: Courier Corporation, 1994.

[2] M. J. Lighthill, "Introduction. Boundary layer theory" in Laminar boundary layers. J. Rosenhead, Ed. New-York: Oxford University Press, 1963, pp. 54–61.

[3] A. J. Chorin, "Numerical study of slightly viscous flow," J. Fluid Mech, vol. 57, No. 4, pp. 785–796, March 1973.

[4] P-A. Raviart, Méthodes particulaires. Lecutre Notes, Ecole d'été d'analyse numérique, Centre d'étude du Bréau-sans-nappe, France, 1987.

[5] S. Shankar, and L. VanDommelen, "A new diffusion procedure for vortex methods," J. Comput. Phys, vol. 127, No. 1, pp. 88–109, August 1996.

[6] G.-H. Cottet, B. Michaux, S. Ossia, and G. VanderLinden, "A comaprison of spectral and vortex methods in three-dimensional incompressible flows," J. Comput. Phys, vol. 175, pp. 702–712, January 2002.

[7] H. N. Najm, "A Hybrid Vortex Method with Deterministic Diffusion" in Vortex Flows and Related Numerical Methods. Eds: J. T. Beale, G.-H. Cottet, S. Huberson, NATO ASI Series (Series C: Mathematical and Physical Sciences), Springer, Dordrecht, vol. 395, pp. 207–222, 1993.

[8] P. Ploumhans, G. S. Winckelmans, and J. K. Salmon, "Vortex particles and tree codes: I. Flows with arbitrary crossing between solid boundaries and particle redistribution lattice; II. Vortex ring encountering a plane at an angle," ESAIM Proc., vol. 7, pp. 335–348, 1999.

[9] J. D. Eldredge, A. Leonard, and T. Colonius, "A general deterministic treatment of derivatives in particle methods," J. Comput. Phys., vol. 180, pp. 686–709, 2002.

[10] O. S. Kotsur, and G. A. Shcheglov, "Implementation of the particle strength exchange method for fragmentons to account for viscosity in vortex element method," Herald of the Bauman Moscow State Technical University, Series Natural Sciences, vol. 3, pp. 48–67, 2018.

[11] G. Ya. Dynnikova, "The Lagrangian approach to solving the time-dependent navier-stokes equations" Doklady Physics,vol. 49, No. 11, pp. 648–652, 2004.

[12] P. R. Andronov, D. A. Grigorenko, S. V. Guvernyuk, and G. Ya. Dynnikova, "Numerical simulation of plate autorotation in a viscous fluid flow," Fluid Dynamics, vol. 42, No. 5, pp. 719–731,2007.

[13] K. S. Kuzmina, I. K. Marchevsky, and E. P. Ryatina, "Open source code for 2D incompressible flow simulation by using meshless Lagrangian vortex methods," Proceedings of Ivannikov ISPRAS Open Conference, pp. 97–103, 2017.

[14] L. Greengard, and V. Rokhlin, "A fast algorithm for particle simulations," J. Comput. Phys., vol. 73, No. 2, pp. 325–348, 1987.

[15] G. Morgenthal, and J. H. Walther, "An immersed interface method for the Vortex-In-Cell algorithm," Comp. Struct., vol. 85, pp. 712–726, 2007.

[16] G. Ya. Dynnikova, "An analog of the Bernoulli and Cauchy – Lagrange integrals for a time-dependent vortex flow of an ideal incompressible fluid," Fluid Dynamics, vol. 35, No. 1, 2000, pp. 24–32.

[17] S. V. Guvernyuk, and G. Ya. Dynnikova, "Modeling the flow past an oscillating airfoil by the method of viscous vortex domains," Fluid Dynamics, vol. 42, No. 1, pp. 1–11, 2007.

[18] G. Y. Dynnikova, and P. R. Andronov, "Expressions of force and moment exerted on a body in a viscous flow via the flux of vorticity generated on its surface," Europ. J. of Mech., B/Fluids, vol. 72, pp. 293–300, 2018.

[19] I. K. Lifanov, Singular integral equations and discrete vortices. Utrecht: VSP, 1996.

[20] K. S. Kuzmina, I. K. Marchevskii, and V. S. Moreva, "Vortex Sheet Intensity Computation in Incompressible Flow Simulation Around an Airfoil by Using Vortex Methods," Mathematical Models and Computer Simulations, vol. 10, No. 3, pp. 276–287, 2018.

[21] K. S. Kuzmina, I. K. Marchevskii, V. S. Moreva, and E. P. Ryatina, "Numerical scheme of the second order of accuracy for vortex methods for incompressible flow simulation around airfoils" Russian Aeronautics, vol. 60, No. 3, pp. 398–405, 2017.

[22] K. S. Kuzmina, and I. K. Marchevskii, "On the calculation of the vortex sheet and point vortices influence at approximate solution of the boundary integral equation in two-dimensional vortex methods of computational hydrodynamics," Fluid Dyn., vol. 54, No. 7, pp. 92–102.

[23] K. S. Kuzmina, I. K. Marchevsky, and E. P. Ryatina, "On CPU and GPU parallelization of VM2D code for 2D flows simulation using vortex method," 6th European Conf. on Computational Mechanics (ECCM 6) 7th European Conf. on Computational Fluid Dynamics (ECFD 7), Glasgow, UK, pp. 2390–2401, June 2018.