The Nvidia CUDA technology capabilities in two-dimensional flows simulation by vortex methods

K.S. Kuzmina, I.K. Marchevsky, E.P. Ryatina

Bauman Moscow State Technical University
Applied mathematics dep.

Institute for System Programming
of Russian Academy of Science

ISPRAS Open
December 5-6, 2019

Introduction

Vortex method

Advantages:

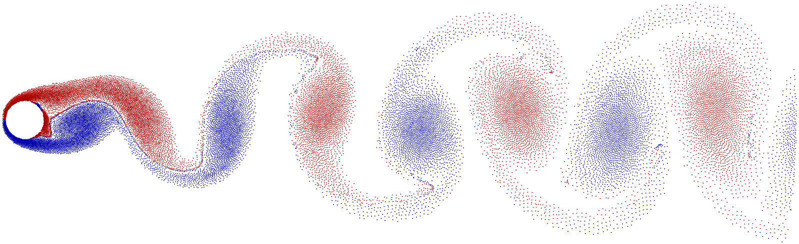
- meshless lagrangian methods;
- low memory requirements and computational time.

Disadvantages:

- range of applicability: incompressible flows and low Reynolds numbers;
- computational complexity of vortex influence calculation $O(N^2)$.

Available codes

1. **Vvflow:**
   author: Dynnikov Ya. A.
   - September 2018
   - closed source code
   - OpenMP
2. **VM2D:**
   authors: Marchevsky I.K., Kuzmina K.S., Ryatina E.P.
   - December 2017
   - open source code
   - OpenMP, MPI, CUDA



Vortex method. Main ideas

- The vorticity $\vec{\Omega}(\vec{r},t) = \nabla \times \vec{V}(\vec{r},t)$ is a primary computed variable.
- The Navier — Stokes equations in Helmholtz-type form (in 2D case)

$$\frac{\partial \vec{\Omega}}{\partial t} + \nabla \times \big(\vec{\Omega} \times (\vec{V} + \vec{W})\big) = 0$$

can be considered as a transfer equation for $\vec{\Omega} = \Omega \vec{k}$, which moves with the velocity $\vec{V} + \vec{W}$, where

$$\vec{W}(\vec{r},t) = -\nu \frac{\nabla \Omega}{\Omega} \text{ — диффузионная скорость.}$$

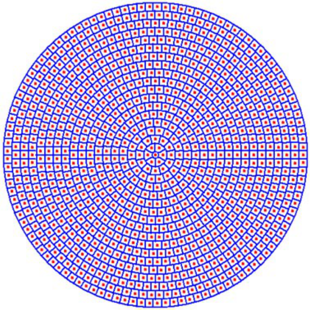
- Vorticity is generated only on the surface line of the airfoil: $\gamma(\vec{r})$, $\vec{r} \in K$ — unknown vortex sheet intensity, which can be found from no-slip condition satisfaction.
- Convective velocity reconstruction (Biot – Savart law):

$$\vec{V}(\vec{r},t) = \vec{V}_\infty + \frac{1}{2\pi} \int_{S(t)} \frac{\vec{\Omega}(\vec{\xi},t) \times (\vec{r} - \vec{\xi})}{|\vec{r} - \vec{\xi}|^2} dS + \frac{1}{2\pi} \oint_{K(t)} \frac{\vec{\gamma}(\vec{\xi},t) \times (\vec{r} - \vec{\xi})}{|\vec{r} - \vec{\xi}|^2} dl_K +$$

$$+ \frac{1}{2\pi} \oint_{K(t)} \frac{\vec{\gamma}_{\text{att}}(\vec{\xi},t) \times (\vec{r} - \vec{\xi})}{|\vec{r} - \vec{\xi}|^2} dl_K + \frac{1}{2\pi} \oint_{K(t)} \frac{q_{\text{att}}(\vec{\xi},t)(\vec{r} - \vec{\xi})}{|\vec{r} - \vec{\xi}|^2} dl_K .$$

Viscous vortex domains method (VVD)

Vorticity distribution simulation



Vortex element circulation in the i-th cell

$$\Gamma_i = \iint\limits_{S_i} \Omega(\vec{r}) dS_r, \qquad \vec{r}_i \in S_i.$$

Vorticity distribution:

$$\Omega(\vec{r}, t) = \sum_{i=1}^{N} \Gamma_i \delta(\vec{r} - \vec{r}_i),$$

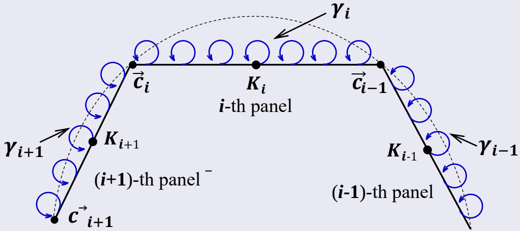
where N is number of vortex elements, δ is two-dimensional Dirac function.

Equations of vortex elements motion

$$\begin{cases} \dfrac{d\Gamma_i}{dt} = 0, \\ \dfrac{d\vec{r}_i}{dt} = \vec{V}(\vec{r}_i, t) + \vec{W}(\vec{r}_i, t), \end{cases} \qquad i = 1, \ldots N.$$

Airfoil approximation

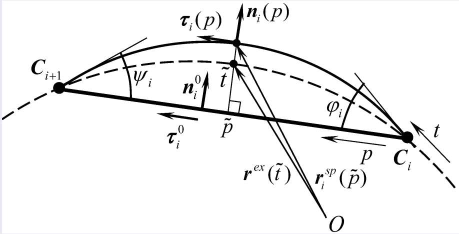
Rectilinear panels

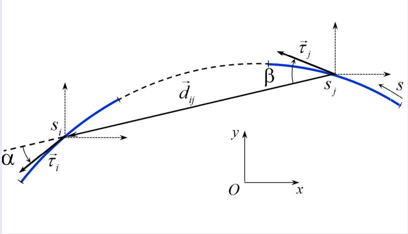


$\gamma(\vec{r}) = \gamma_i$ for $\vec{r} \in K_i$;

L_i — the length of the i-th panel;
$\vec{K}_i$ — the i-th panel center;
$\vec{c}_i$ — the i-th vortex element generation point.

Curvilinear panels





Vortex sheet intensity computation

No-slip condition

$$\vec{V}_-(\vec{r},t) = \vec{V}_K \quad \Leftrightarrow \quad \underbrace{\vec{V}_- \cdot \vec{n} = \vec{V}_K \cdot \vec{n}}_{N\text{-scheme}} \quad \Leftrightarrow \quad \underbrace{\vec{V}_- \cdot \vec{\tau} = \vec{V}_K \cdot \vec{\tau}}_{T\text{-scheme}},$$

$\vec{V}_-(\vec{r},\, t) = \vec{V}(\vec{r},\, t) - \dfrac{\gamma(\vec{r},\, t) - \gamma_{att}(\vec{r},\, t)}{2}\vec{\tau}(\vec{r},\, t) + \dfrac{q_{att}(\vec{r},t)}{2}\vec{n}(\vec{r},t)$ — velocity limit value.

Boundary integral equation

N-scheme: Singular equation of the 1-st kind

$$\frac{1}{2\pi}\oint\limits_K Q_n(\vec{r},\, \vec{\xi})\gamma(\vec{\xi})\, dl_\xi = f_n(\vec{r}), \qquad Q_n = \frac{1}{2\pi}\frac{(\vec{r}-\vec{\xi})\cdot\vec{\tau}(\vec{r})}{|\vec{r}-\vec{\xi}|^2}.$$

T-scheme: Fredholm-type equation of the 2-nd kind

$$\frac{1}{2\pi}\oint\limits_K Q_\tau(\vec{r},\, \vec{\xi})\gamma(\vec{\xi})\, dl_\xi - \frac{1}{2}\gamma(\vec{r}) = f_\tau(\vec{r}), \quad Q_\tau = \frac{1}{2\pi}\frac{(\vec{r}-\vec{\xi})\cdot\vec{n}(\vec{r})}{|\vec{r}-\vec{\xi}|^2}.$$

Solution representation (Galerkin approach): $\gamma(\vec{r}) = \sum\limits_{s=0}^{m}\sum\limits_{j=1}^{n_p}\gamma_j^s\varphi_j^s(\vec{r}),$

n_p — number of panels; m — order of basis functions (constant, linear, quadratic);
γ_j^s — unknown coefficients.

VM2D code

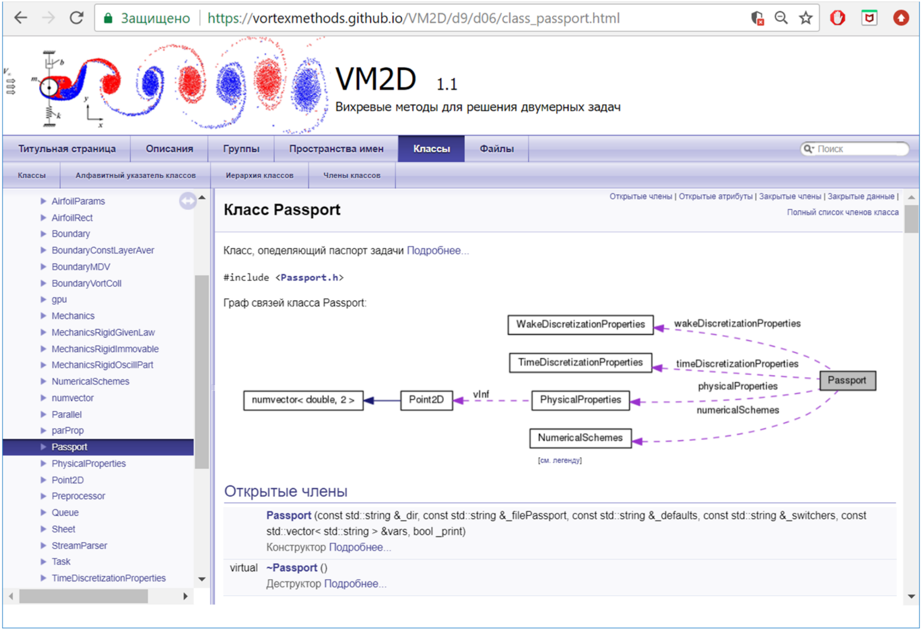
Main features

- C++ language (GCC, Intel C++ Compiler, MSVC)
- Cross-platform software (Linux, Windows, MacOS)
- Parallel technologies: OpenMP, MPI, NVidia CUDA
- External library: Eigen (eigen.tuxfamily.org)
- Open sourse code: https://github.com/vortexmethods/VM2D/
- Documentation: http://vortexmethods.github.io/VM2D/
- Module structure

```
##  ## ##   ##  ####  #####
##  ## ### ### ##  ## ##  ##
##  ## ## # ##    ## ##  ##
 ####  ##   ##   ##   ##  ##
  ##   ##   ## ###### #####
```

Versions

- VM2Dv.1.0 (December 1, 2017, ISPRASOpen 2017)
- VM2Dv.1.1 (April 2, 2018, PAVT 2018)
  - \+ Moving airfoils; MPI + OpenMP
- VM2Dv.1.2 (June 14, 2018, ECCM-ECFD 2018)
  - \+ FSI problems; Rotated airfoils
- VM2Dv.1.3 (September 26, 2018, VoenMeh (Saint Petersburg) 2018)
  - \+ Velocity and pressure fields calculation in fixed points
- VM2Dv.1.4 (April 2, 2019, PAVT 2019)
  - \+ Nvidia CUDA


Защищено
https://vortexmethods.github.io/VM2D/d9/d06/class_passport.html
VM2D 1.1
Вихревые методы для решения двумерных задач
Титульная страница
Описания
Группы
Пространства имен
Классы
Файлы
Поиск
Классы
Алфавитный указатель классов
Иерархия классов
Члены классов
AirfoilParams
AirfoilRect
Boundary
BoundaryConstLayerAver
BoundaryMDV
BoundaryVortColl
gpu
Mechanics
MechanicsRigidGivenLaw
MechanicsRigidImmovable
MechanicsRigidOscillPart
NumericalSchemes
numvector
Parallel
parProp
Passport
PhysicalProperties
Point2D
Preprocessor
Queue
Sheet
StreamParser
Task
TimeDiscretizationProperties
Открытые члены | Открытые атрибуты | Закрытые члены | Закрытые данные |
Полный список членов класса
Класс Passport
Класс, опеделяющий паспорт задачи Подробнее...
#include <Passport.h>
Граф связей класса Passport:
WakeDiscretizationProperties
wakeDiscretizationProperties
TimeDiscretizationProperties
timeDiscretizationProperties
Passport
physicalProperties
numvector< double, 2 >
Point2D
vInf
PhysicalProperties
numericalSchemes
NumericalSchemes
[см. легенду]
Открытые члены
Passport (const std::string &_dir, const std::string &_filePassport, const std::string &_defaults, const std::string &_switchers, const std::vector< std::string > &vars, bool _print)
Конструктор Подробнее...
virtual ~Passport ()
Деструктор Подробнее...

Some VM2D options — I

Flow around immovable airfoil

Flow around rotating airfiol

Some VM2D options — II

FSI problems

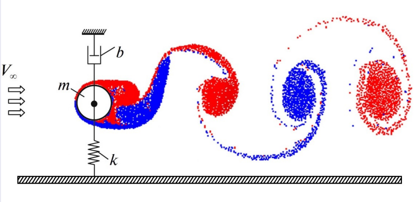
Flow around system of airfoils

Main computational block of the algorithm

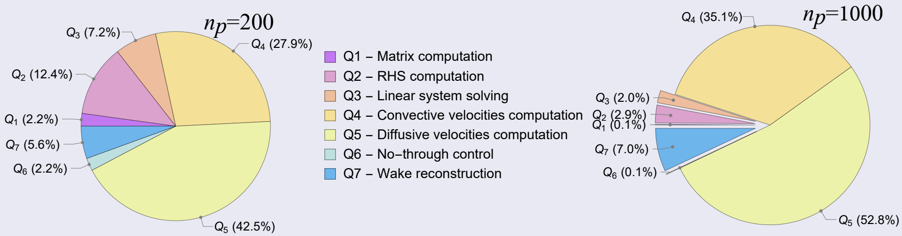
1. **Vorticity generation (solution of boundary integral equation)**
    - Calculation of matrix coefficients according to choosen numerical scheme.
    - Rigth-hand side calculation.
    - Solution of the system of linear equations by Gauss exclusion principle of iterative algorithms.
2. **Vortex elements velocities calculation**
    - Convective velocities calculation.
    - Diffusive velocities calculation.
3. **Hydrodynamic loads calculation**
    - Hydrodynamic force and the momentum calculation.
    - Viscous drag.
    - Pressure and velocities calculation in some fixed media points.
4. **Vortex wake evolution**
    - Solution of ODE system.
    - No-through control.
    - Vortex wake reconstruction.

Computational complexities of different operations

Problem 1

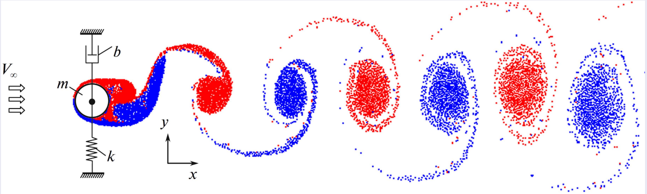


The results for different number of panels at the airfoil surface line

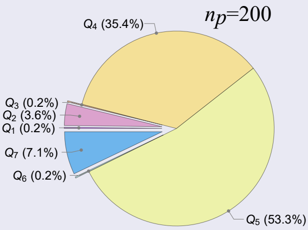


Computational complexities of different operations

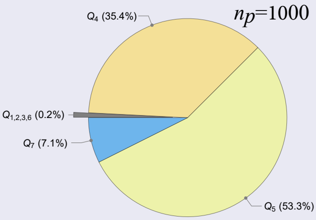
Problem 2



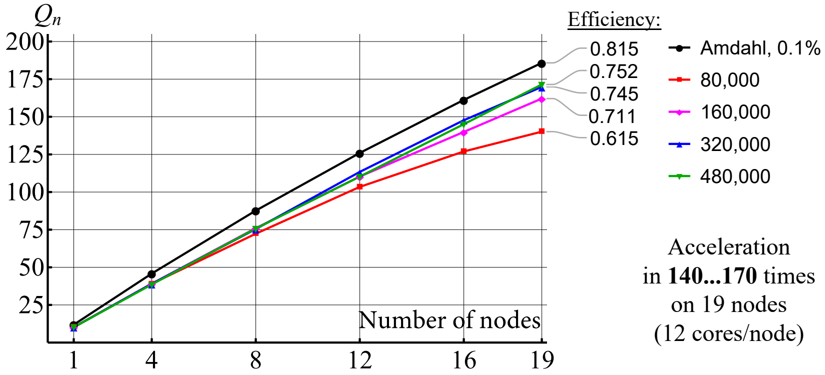
The results for different number of panels at the airfoil surface line



- Q1 – Matrix computation
- Q2 – RHS computation
- Q3 – Linear system solving
- Q4 – Convective velocities computation
- Q5 – Diffusive velocities computation
- Q6 – No–through control
- Q7 – Wake reconstruction



ISP RAS cluster BL2x220c G7, Infiniband QDR
19 nodes: 2 x Intel Xeon X5670 (6 cores), 2.93 GHz



Acceleration
in **140...170** times
on 19 nodes
(12 cores/node)

Efficiency

228 cores (19 cores)	**0.62 . . . 0.75**
96 cores (8 cores)	**0.75 . . . 0.79**

Operations implemented for GPU

1. Right-hand side calculation (Op.1).
2. Convective velocities calculation (Op.2).
3. Diffusive velocities calculation (Op.3).
4. Vortex wake reconstruction (Op.4).

- Calculation time (sec)

	Op.1	Op.2	Op.3	Op.4	Step time
Sequential	3.227	32.074	4.787	8.214	48.768
GeForce GTX 970	0.074	0.427	0.171	0.037	1.406
Quadro P4000	0.107	0.511	0.197	0.039	0.910
GTX Titan	0.053	0.262	0.138	0.047	0.549
Tesla V100	0.011	0.016	0.015	0.018	0.098

- Acceleration in comparison with sequential code

	Op.1	Op.2	Op.3	Op.4	Total
GeForce GTX 970	43.61	75.11	27.99	222.00	34.69
Quadro P4000	30.16	62.77	24.30	210.62	53.59
GTX Titan	60.89	122.42	34.69	174.77	88.83
Tesla V100	293.36	2004.63	319.13	456.33	497.63

Acceleration of whole the algorithm on GPU

- Numerical experiment for different GPU:

	Quadro P4000	GTX 970	Titan	Tesla K40c	Tesla V100
CUDA cores	256	1664	2688	2880	5120
Memory (Gb)	2	4 (3.5)	6	12	16
GFlops (doub.)	243	120	1400	1500	7000

- Comparison with cluster BL220 (ISP RAS)

Accelerations in comparison with sequential code

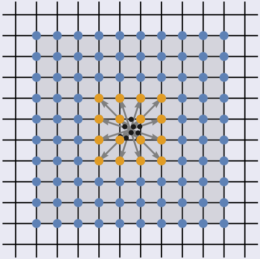
	4 Node	P4000	G970	8 Node	Titan	19 Nodes	K40	V100
‘80 000’	39.1	**49.3**	**58.7**	72.3	**78.8**	140.1	**158.9**	**856.1**
‘480 000’	38.6	**54.2**	**66.1**	75.8	**80.3**	171.5	**162.6**	**913.1**

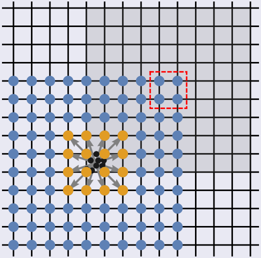
2 MPI nodes + CUDA $\rightarrow$ x 1.6 times
3 MPI nodes + CUDA $\rightarrow$ x 2.2 times

Fast method for vortex influence computation ($O(N \log N)$)

Main ideas

- Convolution integral calculation:
  $\vec{V} = \int\limits_S \vec{Q}(\vec{r}-\vec{\xi})\Omega(\vec{\xi},t)dS_\xi, \ \left(\vec{Q}(\vec{r}-\vec{\xi}) = \dfrac{\vec{k}\times(\vec{r}-\vec{\xi})}{2\pi|\vec{r}-\vec{\xi}|^2}\right) \to$ **FFT** technique $\to$
  nodal velocities values (on coarse rectangular mesh $M \times M$).
- Significant error in neighboring zone $\to$ correction procedure.
- Main idea: linear dependency between the velocity and nodal circulations.
- Exclusion of incorrect influence from neighboring zone of each cell.
- Addition the correct influence according to Biot — Savart law.





Parallel implementation (CUDA)

Model problem: $N = 1\,000\,000$ vortex elements.

Direct calculation (Tesla V100): 10 seconds.

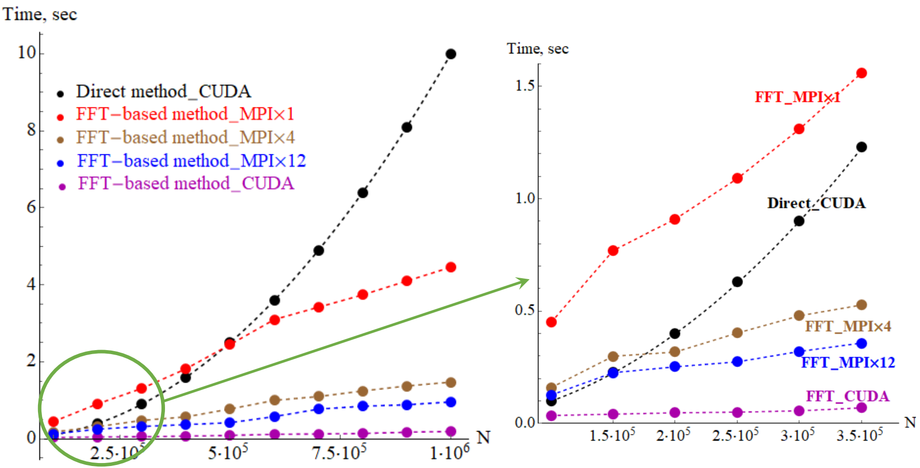
GPU: Tesla V100 (5120 CUDA cores)

Neighboring zone size	Mesh size in one direction		
	$M = 128$	$M = 256$	$M = 512$
3	0.166	0.214	0.464
4	0.173	0.213	0.482
5	0.212	0.228	0.514
6	0.215	0.236	0.546

GPU: GeForce GTX970 (1664 CUDA cores)

Neighboring zone size	Mesh size in one direction		
	$M = 128$	$M = 256$	$M = 512$
3	1.46	1.14	2.00
4	1.96	1.45	2.62
5	2.63	1.85	3.38
6	3.41	2.30	4.26

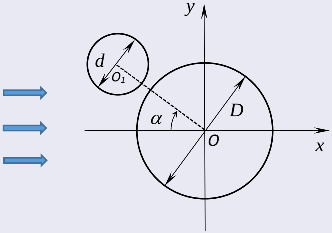
Comparison with the direct method



• Intel i7-920 (2008); • • Intel i9-7980XE (2018); • • GPU Tesla V100 (2018)

Comparison with OpenFOAM

Problem statement

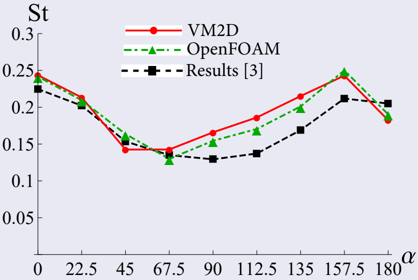


Parameters:

$D = 1.0; \quad d = 0.4;$
$\rho(O, O_1) = 1;$
$\alpha = 0, \ldots, 180$
$\mathrm{Re} = 1000; \quad N_p = 1000.$

3. Thapa J., Zhao M., Cheng L., Zhou T. Three-dimensional flow around two circular cylinders of different diameters in a close proximity, Physics Of Fluids 27, 2015.

Vortex shedding frequency



Computational time

VM2D

n_p	N	Steps	Time GTX970	Time V100
250	13500	18750	50 min	11 min
500	29000	37500	300 min	41 min

n_p — number of panels;
N — average number of vortex elements

OpenFOAM

N	Time
45000	58 min
125000	340 min

N — number of mesh cells.
12 MPI nodes

Conclusions

- VM2D code is an efficient tool for flow simulation around airfoils and solution of FSI problems.
- Parallel implementation allows to use modern multiprocessors, cluster systems and graphic accelerators.
- Usage of graphic accelerators is the most promising way: it was obtain that one powerful graphic accelerator as Tesla V100 replace tens or even hundreds CPU cores.
- The resulting hydrodynamic loads are in good agreement with results obtained with known mesh methods.
- The calculation time of VM2D running on GPU (Tesla V100) is significantly less than the OpenFOAM calculation time of same problem running on cluster system.
- Fast method for vortex influence computation allows to reduce significantly the computational complexity of the most time-consuming operation in vortex method.
- GPU implementation of the fast method provides 100-times acceleration in comparsion with the direct method running on the same GPU.

Thank you for attention!