A TensorFlow-based new high-performance computational framework for CFD

Article *in* Journal of Hydrodynamics · August 2020

DOI: 10.1007/s42241-020-0050-0

4 authors, including:

Weijie Liu
Zhejiang University
13 PUBLICATIONS **43** CITATIONS

SEE PROFILE

Springer

Available online at https://link.springer.com/journal/42241
http://www.jhydrodynamics.com
Journal of Hydrodynamics, 2020, 32(4): 735-746
https://doi.org/10.1007/s42241-020-0050-0

A TensorFlow-based new high-performance computational framework for CFD *

Xi-zeng Zhao[1, 2], Tian-yu Xu[1], Zhou-teng Ye[1], Wei-jie Liu[1]
1. *Ocean College, Zhejiang University, Zhoushan 316021, China*
2. *The Engineering Research Center of Oceanic Sensing Technology and Equipment, Ministry of Education, Zhejiang University, Zhoushan 316021, China*

(Received March 14, 2019, Revised June 23, 2019, Accepted July 22, 2019, Published online August 26, 2020)

Abstract: In this study, a computational framework in the field of artificial intelligence was applied in computational fluid dynamics (CFD) field. This Framework, which was initially proposed by Google AI department, is called "TensorFlow". An improved CFD model based on this framework was developed with a high-order difference method, which is a constrained interpolation profile (CIP) scheme for the base flow solver of the advection term in the Navier-Stokes equations, and preconditioned conjugate gradient (PCG) method was implemented in the model to solve the Poisson equation. Some new features including the convolution, vectorization, and graphics processing unit (GPU) acceleration were implemented to raise the computational efficiency. The model was tested with several benchmark cases and shows good performance. Compared with our former CIP-based model, the present TensorFlow-based model also shows significantly higher computational efficiency in large-scale computation. The results indicate TensorFlow could be a promising framework for CFD models due to its ability in the computational acceleration and convenience for programming.

Key words: TensorFlow, vectorization, Navier-Stokes equations, graphics processing unit (GPU) acceleration, constrained interpolation profile (CIP) method, preconditioned conjugate gradient (PCG) method

Introduction

Computational fluid dynamics (CFD) models directly solving Navier-Stokes (N-S) equations are convenient tools to study problems involving water flows. It is well known that the computational efficiency is the main limitation of a CFD model when it comes to complex physical problems . Therefore, researchers have devoted their efforts to accelerating the computation of CFD models. Among various computational acceleration methods, graphics processing unit (GPU) acceleration is one of the useful methods. Despite the rapid development of central processing units (CPUs), GPUs usually have the advantage of higher computational efficiency especially when the grid number is very large. A number of researchers have successfully applied GPU acceleration for CFD model calculations[1-2]. However, in order to take advantage of GPUs, researchers are supposed to get knowledge of how to program based on compute unified device architecture (CUDA)[3] or Open Computing Language (OpenCL)[4] while some researchers may not be familiar with CUDA or OpenCL frameworks. This might hinder the wide application of GPU acceleration in the CFD field. In this paper, a new framework called TensorFlow is introduced as an optional solution to the problem mentioned above. TensorFlow is a popular open-source framework for high-performance computation, initially developed by Google AI Department[5] for computational acceleration in machine learning or deep learning. Nowadays, there are also other frameworks or open-source libraries available for the CFD field, such as OpenFOAM and Reef 3D. Many practical problems have been studied using these open-source tools, such as turbulent flow past a square cylinder[6], wave simulations[7], bubble drag reduction[8], inner flow in a pump[9], etc.. Self-defined algorithms can also be developed based on OpenFOAM[10] or Reef 3D[11]. Compared with these frameworks, TensorFlow has the following advantages. Firstly, TensorFlow is operational in all platforms including Windows, Mac OS, Linux (even Android and iOS) while OpenFOAM and Reef 3D are not recommended to be installed on the Windows system due to the numerical instability. Secondly, OpenFOAM and Reef 3D only adopt C++ as the

* Project supported by the National Natural Science Foundation of China (Grant No. 51679212, 51979245).
Biography: Xi-zeng Zhao (1979-), Male, Ph. D., Professor,
E-mail: xizengzhao@zju.edu.cn
Corresponding author: Wei-jie Liu,
E-mail: weijieliu@zju.edu.cn

coding language, while TensorFlow offers multiple choices including Python, C++, Java, etc.. Thirdly, TensorFlow APIs are optimized specially for large-scale computation, and also helps to realize distributed computation and GPU acceleration easily. Those researchers who know little about GPUs may also be able to apply them in the computational acceleration. Fourthly, a lot of TensorFlow application programming interfaces (APIs) have strong links to artificial intelligence, which may be extended to CFD models in future work.

Considering these advantages, a new CFD model with high-performance computational ability was developed based on the TensorFlow framework. The model works on the Cartesian grid system and finite difference methods are applied. In this model, the constrained interpolation profile (CIP) method is used to discretize the convection term of N-S equations. The CIP method, which is a third order finite difference scheme to solve the hyperbolic partial differential equations, was used to solve hyperbolic equations[12]. Our former model, which is called CIP-based model, has been successfully applied in many cases such as dam break[13] and Fluid-structure interaction[14-15]. Besides, the preconditioned conjugate gradient (PCG) method, which is a widely used iterative method for solving sparse matrices, is implemented in the present model to solve the Poisson equation. The Conjugate Gradient method was popularized by Reid[16] as an iterative solver for large and sparse matrices. In linear algebra, preconditioning is A universal method to obtain a more rapid convergence and the PCG method is more commonly used rather than the conjugate gradient method. Nowadays many preconditioners are available such as the Jacobi preconditioner, incomplete Cholesky factorization, sparse approximate inverse (SPAI) preconditioner, successive over-relaxation precondi- tioner, etc.. In the present work, we packaged the CIP scheme and the PCG method as optimized modules in a vectorized version, which were specially designed for the computational efficiency of our model. The present model was tested with various benchmark cases. The first case is Zalesak's advection test selected to verify the advection solver in our TensorFlow-based model with the CIP scheme. The second case is selected to verify the Poisson equation solver with the PCG method. The other two cases of lid-driven cavity flow and flow over a circular cylinder are selected to demonstrate the accuracy and computational efficiency of the model. As a preliminary test to prove the high-performance computational ability, the free surface is not included and left for future work.

1. Mathematical method

1.1 *Governing equations*

The governing equations adopted in the present model are N-S equations for the incompressible viscous fluid. The continuity and momentum equations are expressed as:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{1}{\rho}\frac{\partial}{\partial x_j}\left(\mu \frac{\partial u_i}{\partial x_j}\right) + F_i \tag{2}$$

where $x_i\,(i=1,2)$ are the coordinates in a Cartesian coordinate system, $u_i\,(i=1,2)$ are the velocity components, t is time, ρ is the fluid density, μ is the dynamic viscosity, F_i is the body force and p is the pressure.

Here the projection method[17] is applied, and the momentum equations are solved by the following three steps:

$$\frac{\partial u_i^*}{\partial t} + u_j{}^* \frac{\partial u_i^*}{\partial x_j} = 0 \tag{3}$$

$$\frac{u_i^{**} - u_i^*}{\Delta t} = \frac{1}{\rho}\frac{\partial}{\partial x_j}\left(\mu \frac{\partial u_i^*}{\partial x_j}\right) + F_i \tag{4}$$

$$\frac{\partial}{\partial x_j}\left(\frac{1}{\rho}\frac{\partial p}{\partial x_i}\right) = \frac{1}{\Delta t}\frac{\partial u_i^{**}}{\partial x_i} \tag{5}$$

In the first step, we neglect the diffusion term and pressure term as seen in Eq. (3). The CIP method is used to solve this equation and the solution is u^*, which is considered as an intermediate velocity. In the second step, a central difference scheme is used to calculate the gradient of the non-advection term of Eq. (4) and the solution is u^{**}, which is a renewed intermediate velocity. In the third step, the pressure coupling with the velocity field is calculated using a PCG method. Then the flow field data are updated and these steps continue as a circle until the end of the simulation. The details of the flow solver can be found in previous studies[13].

1.2 *Fluid and structure interactions*

In order to deal with the fluid-structure interaction, an immersed boundary method[18] is applied. The velocities in the computational domain can be obtained by

$$U = \varphi U_b + (1-\varphi)u \tag{6}$$

where u is the flow velocity of the fluid, U_b is the velocity of the solid object, φ stands for the solid phase and $1-\varphi$ stands for the liquid phase.

1.3 *The CIP scheme*

The main idea of the CIP scheme is that when calculating the advection Eq. (7), the transportation equation of f, as well as its spatial gradient $g = \partial f / \partial x$ are both applied.

$$\frac{\partial f}{\partial t} + u\frac{\partial f}{\partial x} = 0 \tag{7}$$

For simplicity, we take the 1-D advection with a constant advection velocity of $u > 0$ as an example. Figure 1(a) shows the initial profile and the exact solution after advection at discretized points as shown in Fig. 1(b). In the CIP scheme, we approximate a profile for f^n inside the upwind cell $[x_{i-1}, x_i]$ as:

$$F_i = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{8}$$

$$G_i = a_i(x - x_i)^2 + b_i(x - x_i) + c_i \tag{9}$$

Then the value of the next time step can be obtained with the semi-Lagrangian method:

$$f_i^{n+1} = F_i(x_i - u \cdot \Delta t) \tag{10}$$

$$g_i^{n+1} = G_i(x_i - u \cdot \Delta t) \tag{11}$$

The coefficients in Eq. (8) can be expressed as follows[19]:

$$a_i = \frac{g_i^n + g_{i-1}^n}{\Delta x_i^2} - \frac{2(f_i^n - f_{i-1}^n)}{\Delta x_i^3} \tag{12}$$

$$b_i = \frac{3(f_i^n - f_{i-1}^n)}{\Delta x_i^2} - \frac{2g_i^n + g_{i-1}^n}{\Delta x_i} \tag{13}$$

$$c_i = g_i^n \tag{14}$$

$$d_i = f_i^n \tag{15}$$

Figures 1(c) and 1(d) shows the corresponding solutions from the linear interpolation scheme and CIP scheme respectively.

1.4 *The preconditioned conjugate gradient method*

The conjugate gradient method[20] is used in machine learning and deep learning to solve the linear system of equations with sparse matrices and it's also applicable in the CFD field. It has the advantages of fast convergence and high calculation stability. Moreover, the precondition is usually used to increase the convergence speed.

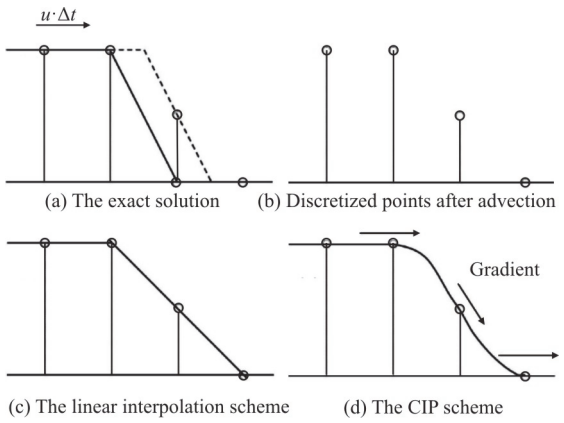

Fig. 1 The introduction of CIP method

The linear equations system can be written as $Ax = b$. When using the preconditioned conjugate gradient (PCG) method to solve this linear equations system, A should be a symmetrical positive determined matrix with the size of $N \times N$, where N is a positive integer. Assuming the exact solution of x being x_0, we define the error as $e_i = x_0 - x_i$, and define the residual as $r_i = b - Ax_i$ (where x_i stands for the numerical result solved in an iterative way, and i is the number of iterations), which both have the size of $N \times 1$. The iteration step goes as:

$$x_{i+1} = x_i + \alpha_i r_i \tag{16}$$

$$\alpha_i = \frac{d_i^{\mathrm{T}} e_i}{d_i^{\mathrm{T}} d_i} = \frac{d_i^{\mathrm{T}} A e_i}{d_i^{\mathrm{T}} A d_i} = \frac{d_i^{\mathrm{T}} r_i}{d_i^{\mathrm{T}} A d_i} \tag{17}$$

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i \tag{18}$$

where $\beta_{i+1} = (r_{i+1}^{\mathrm{T}} r_{i+1}) / r_i^{\mathrm{T}} r_i$. β_i is the step length of each iteration. After certain finite iterations, an approximate numerical solution can be obtained. As the iteration step increases, the residual can be reduced as small as possible. A precondition is usually implemented to achieve a high convergence speed. The precondition is a technique which can improve the condition number of a matrix and make it easier to be solved. In our TensorFlow-based model, one of the widely used pre-conditioners, which is called Jacobi pre-conditioner, is chosen to solve the Poisson equation because of its simplicity and time-saving characteristics in the calculation.

1.5 *Convolution and vectorization tricks*

Different from our former CIP-based model, convolution and vectorization are applied in the present model to accelerate the calculation.

The idea of using convolution kernels to capture flow field features is inspired by the convolutional neural networks (CNN), which is specially designed for computer vision and image processing. As image pixels are similar to grids, convolution can also be applied in CFD models. It should be pointed out that convolution is one of the built-in functions in TensorFlow, which not only can be directly used but also specially optimized for large-scale computation. Figure 2 is shown to illustrate how it works.

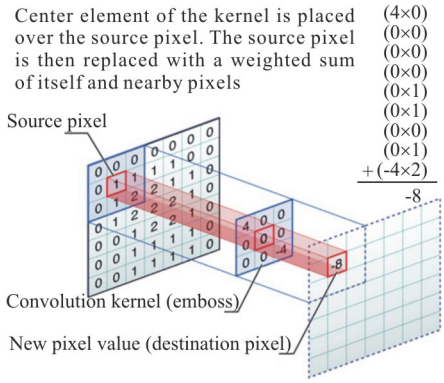

Fig. 2 (Color online) Convolution process

As illustrated in Fig. 2, we avoid the loop but make a convolution kernel to realize the finite difference scheme in TensorFlow.

For example, when calculating $\Delta f/\Delta x = (f_{i+1,j} - f_{i-1,j})/(x_{i+1,j} - x_{i-1,j})$, the designed convolution kernel and the corresponding value of f are shown in Fig. 3.

0	0	0
-1	0	1
0	0	0

Fig. 3(a) Convolution kernel

$f_{i-1,j+1}$	$f_{i,j+1}$	$f_{i+1,j+1}$
$f_{i-1,j}$	$f_{i,j}$	$f_{i+1,j}$
$f_{i-1,j-1}$	$f_{i,j-1}$	$f_{i+1,j-1}$

Fig. 3(b) (Color online) The corresponding value of f

The numbers of 0, −1 and 1 in this convolution kernel stand for weight coefficients, which will multiply the corresponding value of f and the weighted sum will be calculated as the result after this convolution process. The same convolution kernel will be applied to the numerator and denominator respectively, which means:

$$\begin{aligned} f = {} & 0\times f_{i-1,j+1} + 0\times f_{i,j+1} + 0\times f_{i+1,j+1} + \\ & (-1)\times f_{i-1,j} + 0\times f_{i,j} + 1\times f_{i+1,j} + \\ & 0\times f_{i-1,j-1} + 0\times f_{i,j-1} + 0\times f_{i+1,j-1} = \\ & f_{i+1,j} - f_{i-1,j} \end{aligned} \tag{19}$$

$$\begin{aligned} x = {} & 0\times x_{i-1,j+1} + 0\times x_{i,j+1} + 0\times x_{i+1,j+1} + \\ & (-1)\times x_{i-1,j} + 0\times x_{i,j} + 1\times x_{i+1,j} + \\ & 0\times x_{i-1,j-1} + 0\times x_{i,j-1} + 0\times x_{i+1,j-1} = \\ & x_{i+1,j} - x_{i-1,j} \end{aligned} \tag{20}$$

In this way, the finite difference scheme in the model can be realized with different convolution kernels. And it should be pointed out that after the convolution process, the boundary values are supposed to be corrected according to the boundary conditions.

Vectorization is the process of reconstructing a loop using certain instruction sets such as single instruction multiple data (SIMD) or multiple instruction multiple data (MIMD). Instead of processing individual elements in an array or matrix for multiple times, vectorization will process them entirely.

Here is the example code to illustrate the differences between the traditional form and the vectorization form. A and B are both data arrays with the same type of $1\times N$, where N is an integer.

The traditional form

$$(i = 0, i < N, i{+}{+}) : \{\mathrm{A}[i] += \mathrm{B}[i]\}$$

The vectorization form

$$\mathrm{A} += \mathrm{B}$$

In this way, the computational resources can be fully used and thus a lot of computational time can be saved. Generally speaking, vectorization may

accelerate several times compared with using loops in the calculation. As a result, most loops have been replaced by vectorization in the present TensorFlow-based model, in which case the computational efficiency is greatly improved.

1.6 *Implementation of the CFD solver*

The details of solving a CFD problem in TensorFlow are introduced in this section. TensorFlow executes the calculation in the form of computation graphs (Fig. 4), with every single node standing for a calculation operation, and each operation may obtain zero or several inputs and outputs. All nodes are designed previously in operation libraries using the advanced language, Python. The simulation runs in the form of a session, which is executed in C++ to increase computational speed. Besides, TensorFlow has its own computational logic system and unique way to process data. All data is calculated and stored as a special data type called "tensor", which is specially optimized for the large-scale computation.

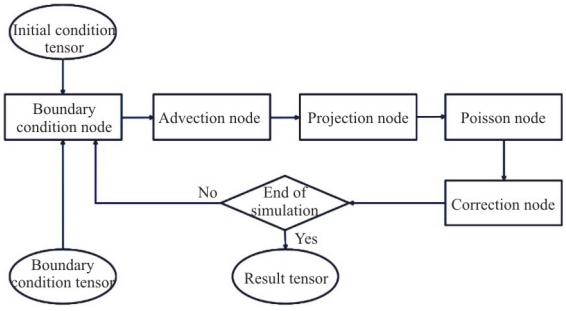

Fig. 4 The computation graph

As seen in Fig. 4, at the very beginning, initial conditions, boundary conditions, and convolution kernels are supposed to be specified. Assuming that the current time step is n, and the velocity and pressure can be marked as u^n and p^n respectively. Then the advection term of Eq. (3) is firstly solved and the intermediate velocity u^* is obtained. After projection operation, a renewed intermediate velocity of u^{**} in Eq. (5) is obtained. Due to the fact that the pressure and velocities are highly coupled, in the Poisson Node, the pressure of the next time step, which is marked as p^{n+1} is solved in an iterative way using the PCG method. At last in the Correction Node, the velocity of u^{n+1} is rectified. In this way, the loop of updating the velocities and pressure is completed. In addition, if the current time step is not the end step, the loop will continue to be executed, otherwise, the total simulation will come to the end.

Details of the CIP module and the PCG module are explained in this paragraph. The CIP module consists of two parts: the first part works for calculating the necessary coefficients of the CIP solver (e.g., for 2-D cases, totally 7 coefficients marked as A1-A7), and the second part works for calculating results in a vectorized way as follows:

$$gx\_vec+=[3.0\times A_1\times XX+2.0\times(A_2\times YY+A_3)]\times XX+(A_4+A_6\times YY)\times YY \tag{21}$$

$$gy\_vec+=[3.0\times A_5\times YY+2.0\times(A_6\times XX+A_7)]\times YY+(A_4+A_2\times XX)\times XX \tag{22}$$

$$f\_vec+=[(A_1\times XX+A_2\times YY+A_3)\times XX+A_4\times YY+gx\_vec]\times XX+[(A_5\times YY+A_6\times XX+A_7)\times YY+gy\_vec]\times YY \tag{23}$$

where $XX=-u\times \mathrm{d}t$, $YY=-v\times \mathrm{d}t$ (u and v are velocities). The result of f_vec is what we needed, and gx_vec, gy_vec are its gradient values in x, y directions, respectively. The PCG module also consists of two parts: the first part is to apply the preconditioner, and the second part is to solve the matrix with CG. In the second part, two factors including the step length, step direction are supposed to be specified. The iteration goes in the following way: $x\_vec+=alpha\times r$, where $alpha$ is the step length, r stands for the iterative direction. This iteration also works in a vectorized way in order to optimize the computational speed.

2. Tests and validations

In this section, several tests are performed to prove the validity of our TensorFlow-based model. All tests in this paper are conducted using the same computational devices, whose GPU is NVIDIA GeForce GTX Titan Black with 6G memory and CPU is Intel (R) Core (TM) i7-6700K CPU@4.00GHz.

2.1 *CIP scheme validation*

To evaluate the performance of the CIP method based on the Tensorflow framework, Zalesak's test[21] was simulated. Zalesak's test, also called solid body rotational problem, is a classical test for advection flow. The motion of Zalesak's solid body is governed by the advection equation

$$\frac{\partial f_i}{\partial t}+u_j\frac{\partial f_i}{\partial x_j}=0 \tag{24}$$

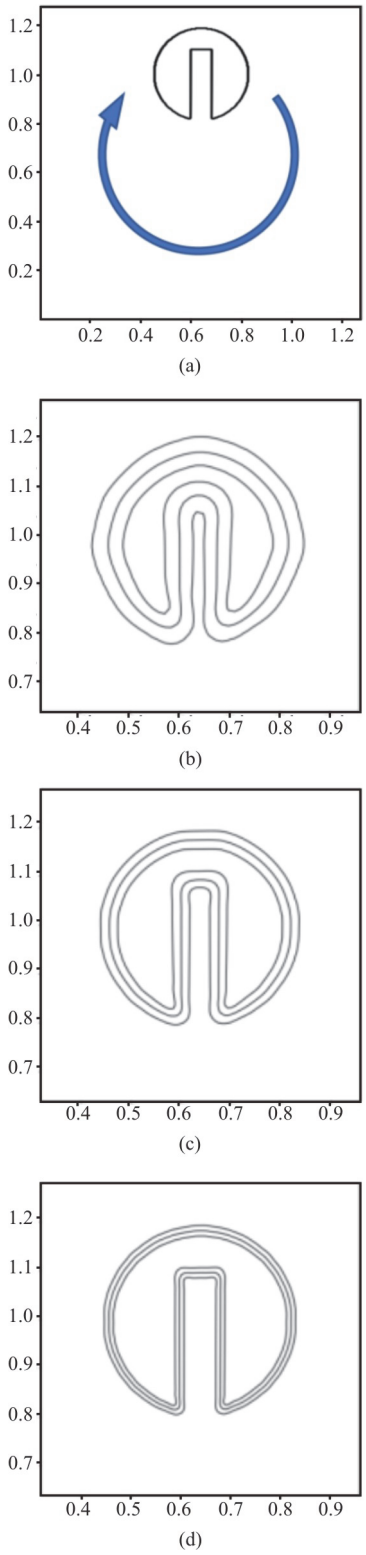

Fig. 5 (Color online) (a) The initial state of the Zalesak's solid body (b)-(d) The result after a period of advection with the grid number being 100×100, 200×200 and 400×400. Three contour lines stand for the values of 0.05, 0.50 and 0.95, respectively

The velocities are given as follows: $u=\omega(y-0.64)$ and $v=\omega(0.64-y)$, where ω is 1.0, standing for the angular velocity. The value of f inside the cut-out circle is 1.0, and outside is 0. The computational domain is $[0\text{ m},1.28\text{ m}]\times[0\text{ m},1.28\text{ m}]$, and the grid space is 0.0064 m. Time step is 0.00314 s. The number of total steps is 2 000 and thus the total simulation time is 6.28 s. Figure 5 shows the results of this problem after one complete revolution, with the grid number being 100×100, 200×200 and 400×400 respectively.

To evaluate the model performance, numerical errors are calculated by the following definition[22]: $E_0=\sqrt{\sum_{i,j}(f_{i,j}{}^{n}-f_{i,j}{}^{ex})^2/\sum_{i,j}(f_{i,j}{}^{ex})^2}$. Different values of E_0 are listed in Table 1 when the domain is uniformly divided into 100×100, 200×200 and 400×400 grids. The numerical error of Fukumitsu et al.'s result[19] with grid number being 10 000 is also listed for comparison.

Table 1 Numerical errors of Zalesak's test

Grid number	Numerical error
10 000	0.352
10 000[19]	0.390
40 000	0.261
160 000	0.189

2.2 *Poisson equation solver*

In this section, the performance of the Poisson equation solver in the TensorFlow-based model is tested. Equation (5) is a typical expression of the Poisson equation. As we assume no free surface in the computational field, the fluid density can be treated as a constant and therefore Eq. (5) can also be written as $\nabla^2 p=\rho/\Delta t(\partial u/\partial x+\partial v/\partial y)$. In this equation, the velocities and pressure are fully coupled, which means pressure can be obtained from velocities through iterations. In other words, we are solving a linear equation system $Ax=b$ iteratively, where A is the coefficient matrix, x is pressure (p) in the Poisson equation above and b is the term on the right side, which is equal to $\rho/\Delta t(\partial u/\partial x+\partial v/\partial y)$ solved in a numerical way.

The velocities are given as:

$$u=\pi\times\sin(2\pi y)\sin^2(\pi x) \tag{25}$$

$$v=-\pi\times\sin(2\pi x)\sin^2(\pi y) \tag{26}$$

Preconditioned conjugate gradient method is applied to solve the linear equation system and Jacobi

pre-conditioner is taken as the default pre-conditioner. Here the residual, which can be expressed as $\nabla^2 p - \rho / \Delta t(\partial u / \partial x + \partial v / \partial y)$, is defined as the difference between the numerical and analytical solution. The total residual of the Poisson equation is defined as L2 Norm$(r) = \sum r_{i,j}{}^2$, where $r_{i,j}$ is the residual of each grid point. The computation domain is $[0\ \mathrm{m}, 2\ \mathrm{m}] \times [0\ \mathrm{m}, 2\ \mathrm{m}]$, divided into 500×500 grids. The value of fluid density is assumed as 1.0 kg/m^3 and the time step is set to 0.01 s. We defined a parameter named "tolerance", the value of which can be given by users to adjust the convergence precision. The results of the Poisson equation with different tolerances are listed in Table 2, which shows that the PCG Poisson solver used in the present model is quite reliable. Figure 6 shows the numerical solution of pressure, which is noted as p in Eq. (5). Figure 7(a) is the analytical solution of $\nabla^2 p$ and Fig. 7(b) is the numerical solution of $\nabla^2 p$, in which case the tolerance is set as 10^{-4}. Qualitatively, they show good agreement. Figure 8 illustrates residuals of the Poisson equation when different tolerance values are given for quantitative comparison. The results show that our Poisson solver is highly precise and is workable for our model.

Table 2 Results of the Poisson equation with different tolerances

Tolerance	Iterations	Max residual	Time/s
10^{-4}	294	2.598×10^{-4}	0.283
10^{-8}	789	2.590×10^{-8}	0.541
10^{-12}	1 393	1.748×10^{-12}	0.869
10^{-16}	2 202	1.662×10^{-16}	1.289

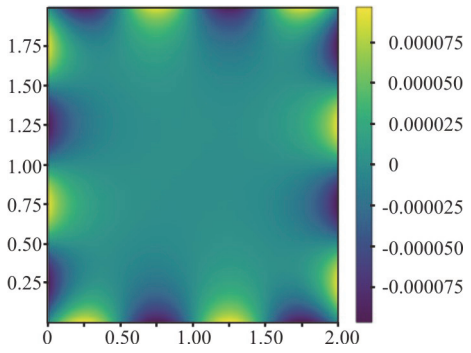

Fig. 6 (Color online) The numerical results of pressure distribution

2.3 *Lid-driven cavity flow*

The lid-driven cavity flow is considered as the classical test problem for the validation of numerical methods and codes for solving N-S equations. The

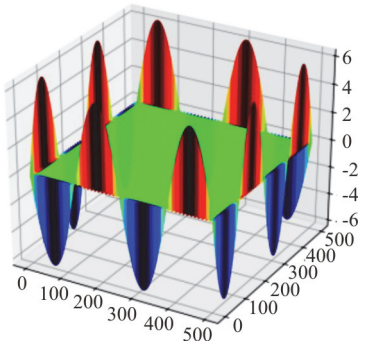

Fig. 7(a) (Color online) Analytical solution of $\nabla^2 p$

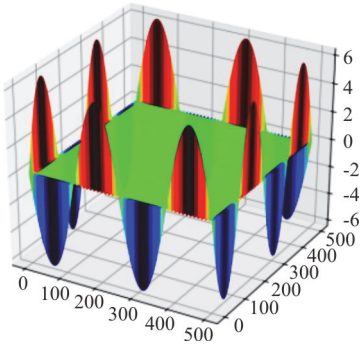

Fig. 7(b) (Color online) Numerical solution of $\nabla^2 p$

case was simulated to prove the validity of the solver in our TensorFlow-based model as well as its high-performance computational ability. In the simulation of this case, the total simulation time is 100 s with the time step, 0.0005 s. The computational domain is $[0\ \mathrm{m}, 2.56\ \mathrm{m}] \times [0\ \mathrm{m}, 2.56\ \mathrm{m}]$ with the grid number, 160×160, and no-slip boundary conditions are specified. The horizontal velocity u on the top side where $y = 2.56$ is set to −1 m/s, and on the other three sides the values of u are 0. Besides, the vertical velocity v is 0 on each side of the boundaries. The fluid density is 1.0 kg/m^3, and the Reynolds number is 1000. It has been verified that the solution obtained at *Re*=1000 is quite close from one author to another[23]. Thus, our results are compared with Ghia's classical results[24] as well as CIP-based model's results to prove the validity of the present model as shown in Fig. 9. The lengths of the cavity edges are marked as *xm* and *ym* in Fig. 9.

It should be pointed out that different Poisson equation solvers are implemented in the present model and CIP-based model, which call for different convergence criterions. In our TensorFlow-based model, the PCG solver is applied and the criterion is restricting residuals of the whole linear equations system, while in the CIP-based model, the SOR method

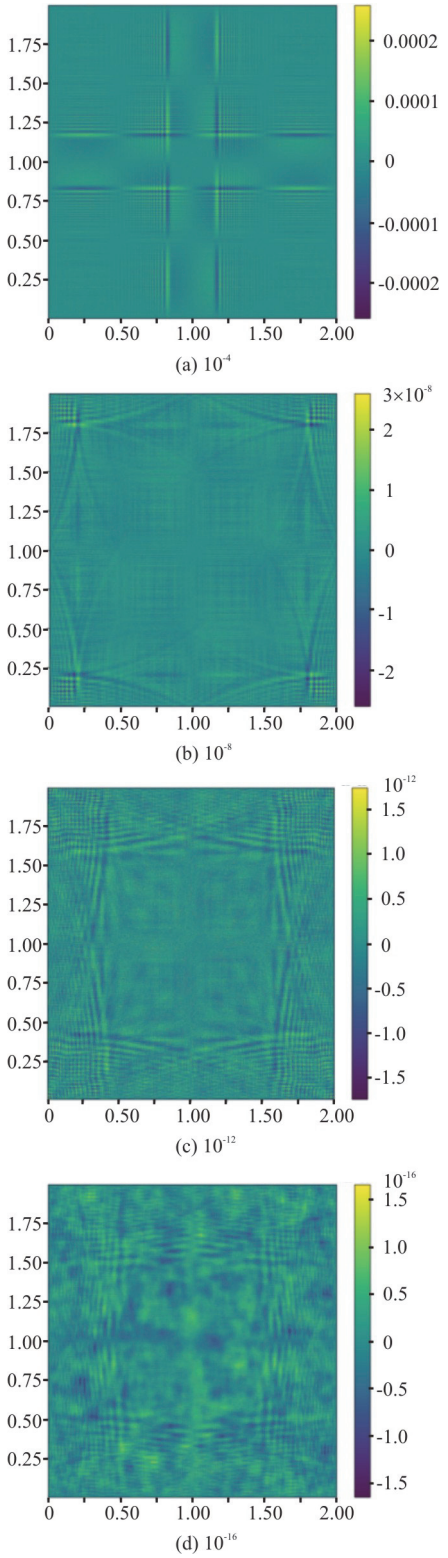

Fig. 8 (Color online) Residuals of the Poisson equation with different tolerances

is used and the criterion is restricting max differences between two consecutive iterations. Theoretically, it is hard to determine which criterion is better, while it can be seen from Fig. 9 that the TensorFlow-based model performs better with the same tolerance value assigned $(\text{tolerance} = 1 \times 10^{-3})$.

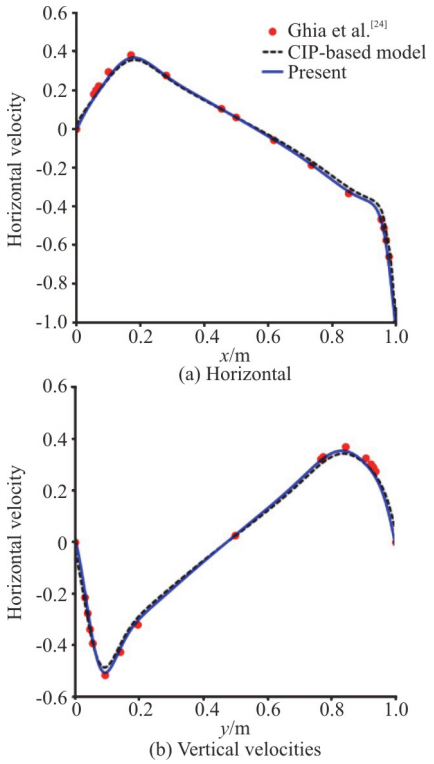

Fig. 9 (Color online) Comparisons of the results from the present model, CIP-based model and Ghia[24]

The streamline, pressure, and vortex distribution are also plotted in Fig. 10. As seen in Fig. 10(a), there are three vortices. One is the primary vortex and the other two are secondary vortices. Figures 10(b) and 10(c) show the vortex distribution and the pressure distribution respectively. Bruneau et al.'s results[23] are also displayed in Fig. 11 for comparison. It can be noticed that predicted results from the present model agree qualitatively well with their results.

Table 3 and Fig. 12 show the computational time of the TensorFlow-based model and CIP-based model with different grid numbers. When the grid number is small, the time of data transport and exchange on the GPU can not be neglected compared to the total computational time so that the efficiency of GPUs is relatively lower than CPUs. However in the large-scale computation, this cost can be negligible and as the grid number increases, the computational time of the CIP-based model increases much faster than the TensorFlow-based model. With the help of the convolution, vectorization and GPU acceleration skills, our TensorFlow-based model may work more than 10 times faster than the CIP-based model. It should be noticed that, due to different Poisson solvers

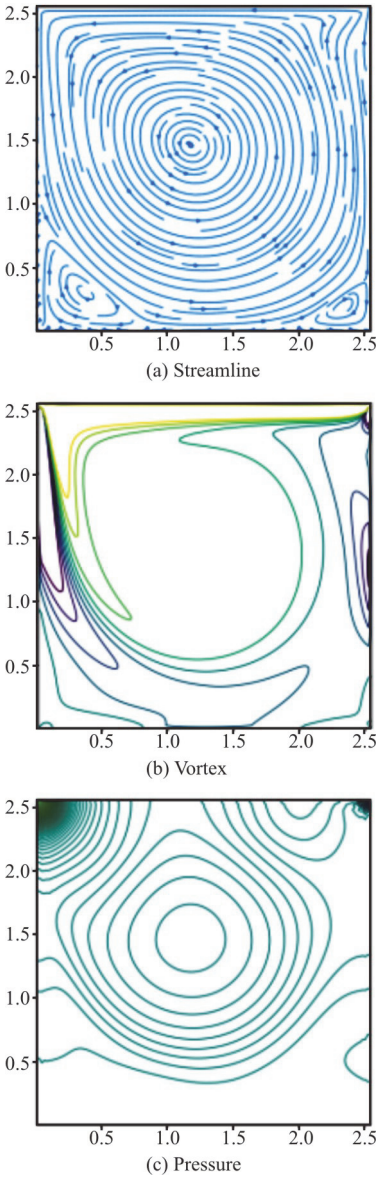

Fig. 10 (Color online) The streamlines, vortex, and pressure distribution of the lid-driven cavity flow test's steady solution at $Re = 1\,000$

Fig. 11 Bruneau et al.'s results of the lid-driven cavity flow test's steady solution at $Re = 1\,000$ [23]

and convergence criterions used in these two models, it's very hard to produce results with the same precision and our idea is to prove that the TensorFlow-based model works not only better but also faster than the CIP-based model with the same tolerance value assigned, which is 1×10^{-3} in this case.

2.4 *Flow past a circular cylinder*

The 2-D flow past a stationary cylinder was also simulated using the immersed boundary method. This case is a test about the model performance for fluid-structure interactions as well as computational efficiency. The simulation was performed in a rectangular domain

Table 3 The computational time of the TensorFlow-based model and CIP-based model with different grid numbers

Grid number	Present/s	CIP-based model/s
6 400	1 587	768
25 600	1 804	3 233
102 400	2 869	18 014
409 600	8 213	106 137

of $[30D, 20D]$, where $D = 0.4$ m is the radius of the cylinder. The circular cylinder's center is located at $x = 10D$, $y = 10D$ in the computational domain. The Newman boundary condition is specified on the

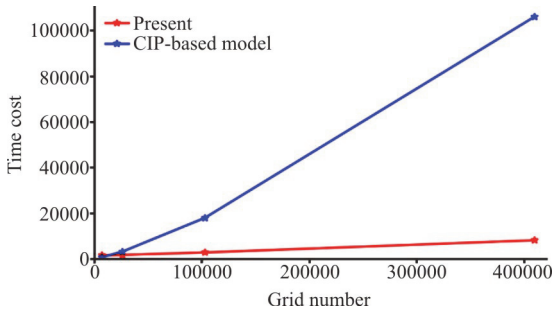

Fig. 12 (Color online) The computational time of the TensorFlow-based model and CIP-based model with different grid numbers

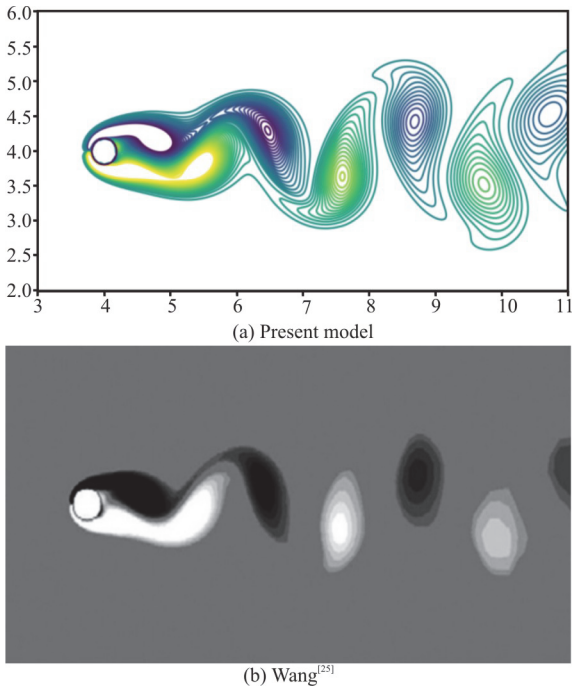

Fig. 13 (Color online) Vortex shedding simulated by the (a) present model and (b)Wang[25]

top, bottom and outlet boundaries. The grid space is set as $D/25$ and the time step is 0.0005 s. A constant velocity of $u=1$ m/s is given at the domain entrance. The liquid density is 1.0 kg/m^3, and the Reynolds number is 100. It has been found in this case that the vortices begin shedding regularly when the simulation time is longer than 80s. Therefore the total simulation time is set as 120 s. Figure 13(a) is the vortex distribution at $t=120$ s. Figure 13(b) shows Wang's results[25] with the same Reynolds number and our result shows similar locations and shapes of the shedding vortices.

In order to make a quantitative comparison, the drag and lift coefficients are generally calculated as:

$$C_D = \frac{F_x}{0.5\rho u d^2} \tag{27}$$

$$C_L = \frac{F_y}{0.5\rho u d^2} \tag{28}$$

The Strouhal number is defined as the dimensionless frequency with which the vortices are shed behind the body:

$$S_t = \frac{fd}{u} \tag{29}$$

The results of the Tensorflow-based model and CIP-based model with different grid resolutions are listed in Table 4. In this case, the tolerance value is set as 1×10^{-4}. The coarse grid space is $D/20$, the middle one is $D/25$, and the fine one is $D/32$. As seen in Table 4, the results of two models show good agreement. Predicted results from the Tensorflow-based model with the fine grid size and existing literature data are listed in Table 5. As seen in Table 5, our predicted result of the drag coefficient (C_D) is 1.37, which falls in the reference interval $[1.33,1.4473]$. Our predicted result of the lift coefficient (C_L) is ±0.32, which falls in the reference interval $[\pm0.29,\pm0.3299]$ and our result of the Strouhal number is 0.165, which is equal to Lai and Peskin's as well as Kim et al.'s data.

Table 4 Comparison of results from two models with different grid resolutions

Case	C_D	C_L	St
Present (coarse grid)	1.31	±0.29	0.163
Present (middle grid)	1.36	±0.32	0.165
Present (fine grid)	1.37	±0.32	0.165
CIP-based (coarse grid)	1.31	±0.28	0.163
CIP-based (middle grid)	1.35	±0.31	0.165
CIP-based (fine grid)	1.36	±0.32	0.165

Table 5 Comparison of results from the present model with the fine grid size and existing literature data

Case	C_D	C_L	St
Present (fine grid)	1.37	±0.32	0.165
Tseng and Ferziger[26]	1.42	±0.29	0.164
Kim et al.[27]	1.33	-	0.165
Lai and Peskin[28]	1.4473	±0.3299	0.165
Rajani et al.[29]	1.3353	-	-

Table 6 and Fig. 14 show the comparison of the computational time between the Tensorflow-based model and CIP-based model. It is clearly illustrated that the computational time of the TensorFlow-based model increases much more slowly than that of the CIP-based model, which is similar to the lid-driven cavity flow test above. When the grid number increases to 614 400, the CIP-based model takes more than 92 h while our TensorFlow-based model only takes about 6.77 h, in which case the computational efficiency is raised by 13.68 times. It should also be noted that with the same tolerance assigned, the present TensorFlow model is supposed to produce more precise results. Considering that different programming languages and different Poisson solvers are used, the acceleration ratio is only taken as a reference here.

Table 6 Comparisons of the computational time of the TensorFlow-based model and CIP-based model

Grid space	Grid number	Present/s	CIP-based model
$D/20$	240 000	17 523.6	136 230.7
$D/25$	375 000	19 042.1	208 432.3
$D/32$	614 400	24 370.8	333 491.2

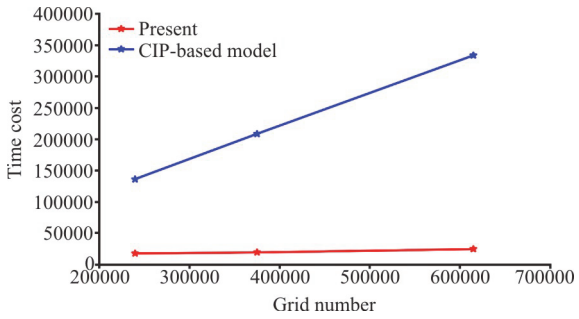

Fig. 14 (Color online) Comparisons of the computational time between the TensorFlow-based model and CIP-based model

3. Conclusion

In this paper, a new efficient CFD model is developed based on the TensorFlow framework. The CIP scheme is adopted as the base flow solver for the advection term in the N-S equations and the PCG method is used to solve the Poisson equation. Some new features including the convolution, vectorization, and GPU acceleration were implemented to raise the computational efficiency. The model was validated by several numerical tests and the results agree well with exact solutions or previous numerical results. The presented results indicate that the TensorFlow framework not only helps researchers to reduce programming work with Python, but also helps to achieve GPU acceleration easily to promote computational efficiency. Compared with our former CIP-based model, which uses the same discrete scheme but is written in Fortran, the computational time of the present model can be greatly shortened in time-consuming simulations, and the acceleration ratio increases as the grid number grows. Due to its high-performance computational ability and simplicity for programming, the TensorFlow framework might be a new choice for CFD models.

Acknowledgements

This work was supported by the Natural Science Foundation of Zhejiang Provincial (Grant No. LR16E090002), the Fundamental Research Funds for the Central Universities (Grant No. 2018QNA4041), Blue Bay Renovation Project of Pingtan Comprehensive Pilot Zone, the Bureau of Science and Technology of Zhoushan (Grant No. 2018C81040), the HPC Center OF ZJU (Zhoushan Campus) and the Tang scholar.

References

[1] Sasan T., Patrick L. Celeris: A GPU-accelerated open source software with a Boussinesq-type wave solver for real-time interactive simulation and visualization [J]. *Computer Physics Communications*, 2017, 217: 117-127.

[2] Xu C., Deng X., Zhang L. et al. Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the Tianhe-1A supercomputer [J]. *Journal of Computational Physics*, 2014, 278: 275-297.

[3] Thibault J. C., Senocak I. CUDA implementation of a Navier–Stokes solver on multi-GPU desktop platforms for incompressible flows [C]. *Proceedings of the 47th AIAA Aerospace Sciences Meeting*, Florida, USA, 2009.

[4] Farshid M., Riccardo R., Pooyan D. et al. OpenCL-based implementation of an unstructured edge-based finite element convection-diffusion solver on graphics hardware [J]. *International Journal for Numerical Methods in Engineering*, 2012, 89(13): 1635-1651.

[5] Abadi M. Tensorflow: Learning functions at scale [C]. *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, New York, USA, 2016.

[6] Aminian J. A. Scale adaptive simulation of vortex structures past a square cylinder [J]. *Journal of Hydrodynamics*, 2018, 30(4): 657-671.

[7] Wang D. X., Sun J. W., Gui J. S. et al. A numerical piston-type wave-maker toolbox for the open-source library OpenFOAM [J]. *Journal of Hydrodynamics*, 2019, 31(4): 800-813.

[8] Zhao X. J., Zong Z., Jiang Y. C. et al., Numerical simulation of micro-bubble drag reduction of an axisymmetric body using OpenFOAM [J]. *Journal of Hydrodynamics*, 2019, 31(5): 900-910.

[9] Liu H. L., Ren Y., Wang K. et al. Research of inner flow in a double blades pump based on OpenFOAM [J]. *Journal of Hydrodynamics*, 2012, 24(2): 226-234.

[10] Wang J. H., Zhao W. W., Wan D. C. Development of

naoe-FOAM-SJTU solver based on OpenFOAM for marine hydrodynamics [J]. *Journal of Hydrodynamics*, 2019, 31(1): 1-20.
[11] Hans B., Arun K. Combined level set/ghost cell immersed boundary representation for floating body simulations [J]. *International Journal For Numerical Methods In Fluids*, 2017, 83: 905-916.
[12] Yabe T., Aoki T., Sakaguchi G. et al. The compact CIP (cubic-interpolated pseudo-particle) method as a general hyperbolic solver [J]. *Computers and Fluids*, 1991, 19(3-4): 421 - 431.
[13] Ye Z., Zhao X. Investigation of water-water interface in dam break flow with a wet bed [J]. *Journal of Hydrology*, 2017, 548: 104-120.
[14] Li M., Zhao X., Ye Z. et al. Generation of regular and focused waves by using an internal wave maker in a CIP-based model [J]. *Ocean Engineering*, 2018, 167: 334-347.
[15] Fu Y. N., Zhao X. Z., Cao F. F. et al. Numerical simulation of viscous flow past an oscillating square cylinder using a CIP-based model [J]. *Journal of Hydrodynamics*, 2017, 29(1): 96-108.
[16] Reid J. K. On the method of conjugate gradients for the solution of large sparse systems of linear equations [C]. *Proceedings of the Conference on Large Sparse Sets of Linear Equations*, 1971.
[17] Guermond J., Minev P., Shen J. An overview of projection methods for incompressible flows [J]. *Computer Methods in Applied Mechanics and Engineering*, 2006, 195(44-47): 6011-6045.
[18] Peskin C. S. Numerical analysis of blood flow in the heart [J]. *Journal of Computational Physics*, 1977, 25(3): 220-252.
[19] Fujimatsu N., Suzuki K. New interpolation technique for the CIP method on curvilinear coordinates [J]. *Journal of Computational Physics*, 2010, 229(16): 5573-5596.
[20] Jeffrey B., Ian F., Eitan G. et al. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid [J]. *ACM Transactions Graphics*, 2003, 22(3): 917-924.
[21] Zalesak S. T. Fully multidimensional flux-corrected transport algorithms for fluids [J]. *Journal of Computational Physics*, 1979, 31(3): 335-362.
[22] Fukumitsu K., Yabe T., Ogata Y. et al. A new directional-splitting CIP interpolation with high accuracy and low memory consumption [J]. *Journal of Computational Physics*, 2015, 286: 62-69.
[23] Bruneau C. H., Saad M. The 2D lid-driven cavity problem revisited [J]. *Computers and Fluids*, 2006, 35(3): 326-348.
[24] Ghia U., Ghia K., Shin C. High-*Re* solutions for incompressible flow using the Navier-Stokes equations and a multigrid method [J]. *Journal of Computational Physics*, 1982, 48(3): 387-411.
[25] Wang S., Zhang X. An immersed boundary method based on discrete stream function formulation for two- and three-dimensional incompressible flows [J]. *Journal of Computational Physics*, 2011, 230(9): 3479-3499.
[26] Tseng Y. H., Ferziger J. H. A ghost-cell immersed boundary method for flow in complex geometry [J]. *Journal of Computational Physics*, 2003, 192(2): 593-623.
[27] Kim J., Kim D., Choi H. An immersed-boundary finite-volume method for simulations of flow in complex geometries [J]. *Journal of Computational Physics*, 2001, 171(1): 132-150.
[28] Lai M., Peskin C. An immersed boundary method with formal second order accuracy and reduced numerical viscosity [J]. *Journal of Computational Physics*, 2000, 160(2): 705-719.
[29] Rajani B., Kandasamy A., Majumdar S. Numerical simulation of laminar flow past a circular cylinder [J]. *Applied Mathematical Modelling*, 2009, 33(3): 1228-1247.