Computers & Fluids 88 (2013) 210–224

Contents lists available at ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

Parallel computing strategy for a flow solver based on immersed boundary method and discrete stream-function formulation

Shizhao Wang, Guowei He, Xing Zhang *

LNM, Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China

ARTICLE INFO

Article history:
Received 3 February 2012
Received in revised form 26 August 2013
Accepted 2 September 2013
Available online 11 September 2013

Keywords:
Parallelization
Domain decomposition
Massage passing interface
Immersed boundary method
Discrete stream-function formulation

ABSTRACT

The development of a parallel immersed boundary solver for flows with complex geometries is presented. The numerical method for incompressible Navier–Stokes equations is based on the discrete stream-function formulation and unstructured Cartesian grid framework. The code parallelization is achieved by using the domain decomposition (DD) approach and Single Program Multiple Data (SPMD) programming paradigm, with the data communication among processes via the MPI protocol. A 'gathering and scattering' strategy is used to handle the force computing on the immersed boundaries. Three tests, 3D lid-driven cavity flow, sedimentation of spheres in a container and flow through and around a circular array of cylinders are performed to evaluate the parallel efficiency of the code. The speedups obtained in these tests on a workstation cluster are reasonably good for the problem size up to 10 million and the number of processes in the range of 16–2048.

1. Introduction

In recent years, there has been an increasing popularity of immersed boundary (IB) methods [1,2]. The reason behind this trend is that the meshes are not required to conform to the body surfaces. In the IB methods, appropriately defined forcing terms are added to the governing equations to mimic the effect of the immersed body on the motion of the fluid. The use of non-body-fitted meshes can significantly reduce the time and labor involved in meshing complex geometries. IB methods have now become a powerful tool for simulating flow involving complex, moving (or morphing) bodies. Comparing with the body-fitted-grid methods, IB methods usually require larger number of mesh points to achieve a proper resolution near the boundaries. This noticeable demerit is due to the use of simple Cartesian grid in the majority of IB methods. Some measures have already been taken to mitigate the situation, such as the use of stretched mesh, locally-refined mesh [3] or curvilinear mesh[4,5]. Although these strategies aforementioned can reduce the total mesh number to some extent, several million grid-points are still required in some high-fidelity 3D simulations (even for laminar flows at moderate Reynolds numbers).

With the continued rapid growth in computational power, larger and larger simulations are now conducted to study the phenomena in complex flow configurations. At the same time, massively parallel distributed-memory platforms have prompted new programming paradigms. To facilitate more efficient use of these computational resources in performing high-fidelity simulations, we need to investigate the parallelization strategy in addition to the discretization schemes and solution algorithms. Here, 'efficient' could refer to the ability to solve a problem of given size as fast as possible, but it could also mean that the overall time to solve the problem remains (nearly) constant when the problem size and the number of processors are increased at the same rate. The former definition relates to the strong scalability of a parallel implementation, whereas the latter requires weak scalability. In the present work, both types of scalability will be evaluated but the focus is put on the strong scalability property of the code.

The parallelization strategies under investigation in this paper include the following two aspects: (a) parallelization of the basic Navier–Stokes solver; (b) parallelization of the forcing computation near the immersed boundary. For the basic flow solver, most of the existing numerical models for the solution of the Navier–Stokes equations are based on Finite Difference Method (FDM), Finite Element Method (FEM) or Finite Volume Method (FVM). The space can be discretized with structured or unstructured grids, and the time with explicit or implicit techniques. The parallelization strategy differs greatly among different data structures and time advancing schemes. Numerical methods which depend on structured grids (such as FDM) and an explicit time discretization have certain advantages concerning the parallelization. The parallel implementation is easier and higher parallel efficiencies can be expected when compared with other candidates.

In the present work, we first describe the parallelization a sequential Navier–Stokes solver which is based on the discrete stream function formulation for incompressible flows. This

* Corresponding author. Tel.: +86 10 82543929; fax: +86 10 82543977.
E-mail address: zhangx@lnm.imech.ac.cn (X. Zhang).

0045-7930/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compfluid.2013.09.001

algorithm is essentially implicit and is currently implemented on an unstructured Cartesian grid. Although structured grid possesses the advantage of coding simplicity, fully unstructured data structure allows an easy treatment of anisotropic local mesh refinement (esp. hanging-nodes) [6]. Of course, the unstructured data management unavoidably raises the complexity of parallelization. Many references on parallelization of implicit unstructured solvers can be found in the published literatures, such as [7–10] on FVM, [11] on FEM and [12–15] on Control Volume Finite Element Method (CVFEM) (which is a mixture of FVM and FEM), just to list a few. The implicit parallel solution algorithm for Navier–Stokes equations in this work is based on the Domain Decomposition (DD) approach, which is preferable on a computer system with distributed memory. When applying the DD strategy, the sequential algorithm of discretization is mainly kept and each processor computes part of the basic tasks such as matrix–vector multiplication with its assigned data and the data exchange only occurs at the boundaries among sub-domains.

The existence of immersed boundaries (esp. moving ones) further complicates the parallel implementation. Intuitively, it is most reasonable to have a given processor deal with the 'markers' (Lagrangian points) which are currently located within its local sub-domain. However there are some open questions related to this strategy of parallelization, such as the load-balancing issue due to the unequal distribution of 'marker' points across processors; communication overhead produced by shared processing and handing-over of points among processes. These issues are seldom addressed in the literatures. In a report by Uhlmann [16], a 'master and slave' strategy is proposed for the simulation of freely-moving particles in fluid using the IB method. In this strategy, one 'master' processor is assigned to each particle for the general handling of it. If necessary, there will be a number of additional processors ('slaves') assigned to the particle to help the 'master'. Recently, Wang et al. [17] proposed a different method to handle the situation when particle crosses the boundaries of sub-domains. Preliminary 2D computations have demonstrated that the parallel efficiency and speedup in these two studies above are acceptable. In the present work, we employ a 'gathering-and-scattering' strategy. At each time step, a master processor is exclusively responsible for computing the force and then the result is scattered the slave processors on which the Navier–Stokes equations are solved in parallel. This strategy circumvents some difficulties aforementioned and is very easy to program.

The organization of the paper is as follows. Numerical method and data structures are presented in Section 2. Basic strategies employed in the parallelization will be discussed in Section 3. Validations and parallel performance tests will be presented in Section 4. Finally, the discussion and conclusions are given in Section 5.

2. Algorithm description

A brief introduction to the numerical methodology and data structure is given here to make the paper as self-contained as possible. For a more complete description, please refer to [18].

2.1. Immersed boundary method in discrete stream-function formulation

The three-dimensional incompressible viscous flow is considered in the present work. The governing equations of the flow can be written as:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = -\nabla \mathrm{p} + \frac{1}{\mathbf{Re}}\nabla^2\mathbf{u} + \mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{2}$$

where $\mathbf{u}$ and p are the velocity vector and pressure respectively. $\mathbf{f}$ is the force representing the effect of the immersed body on the flow. The Reynolds number is defined as $Re = UL/\nu$, where U and L are the reference velocity and length respectively, and ν is the kinematic viscosity of the fluid. The discretized form of Eqs. (1), (2) can be expressed by a matrix form as

$$\begin{bmatrix}\mathbf{A} & \mathbf{G}\\ \mathbf{D} & \mathbf{0}\end{bmatrix}\begin{bmatrix}q^{n+1}\\ p\end{bmatrix} = \begin{bmatrix}r^n\\ 0\end{bmatrix} - \begin{bmatrix}bc_1\\ bc_2\end{bmatrix} + \begin{bmatrix}\tilde{f}\\ 0\end{bmatrix}, \tag{3}$$

where q, p, and $\tilde{f}$ are the discrete velocity flux, pressure, and body force, respectively. The discrete velocity u, is related to q by multiplying the cell face area. $\mathbf{A}$, $\mathbf{G}$ and $\mathbf{D}$ are the implicit operator, gradient operator and divergence operator respectively. In addition, the negative transpose of the divergence operator is the gradient operator, i.e. $\mathbf{G} = -\mathbf{D}^T$. r^n is the explicit right-hand-side term of the momentum equation. bc_1 and bc_2 are the boundary condition vectors for the momentum and continuity equation respectively.

The discrete stream function (null-space) approach is a numerical method for solving Eq. (3) proposed by Chang et al. [19]. Unlike the classic fractional step method, in this method the divergence-free condition is satisfied to machine precision and there are no splitting errors associated with it. In the discrete stream function approach, a discrete stream-function s is defined, such that $q = Cs$, where $\mathbf{C}$ is the *curl* operator (which is a non-square matrix). This matrix is constructed in such a way that $\mathbf{D}$ and $\mathbf{C}$ enjoy the relation $\boldsymbol{DC} = 0$, thus the incompressibility condition is automatically satisfied.

In this approach, another type of curl operator $\mathbf{R}$, which is called the *rotation* operator, is also defined. The matrix $\mathbf{R}$ and matrix $\mathbf{C}$ enjoy the relation $\mathbf{R} = \mathbf{C}^T$. By pre-multiplying the momentum equation with $\mathbf{R}$, the pressure can be eliminated and the system of Eq. (3) is reduced to a single equation for s at each time step

$$\mathbf{C}^T\mathbf{A}\mathbf{C}s^{n+1} = \mathbf{R}(r^n - bc_1) + \mathbf{R}\tilde{f}. \tag{4}$$

The matrix $\mathbf{C}^T\mathbf{A}\mathbf{C}$ is symmetric, positive-definite and thus can be solved using the Conjugate Gradient (CG) method.

The forcing term $\mathbf{f}$ on the right hand side of Eq. (1) is computed in an implicit way. Within one step of time advancing (from n to n + 1), this procedure can be summarized as the following four sub-steps.

(i) A 'predicted' stream function is computed with the forcing at time step n and the velocity vectors are reconstructed using the 'predicted' stream function.

(ii) A 'force correction' is applied to achieve the desired velocity on the boundary. In this paper, we follow a similar procedure as that in [21] in computing the force. Mathematically, the 'force corrections' at the Lagrangian points are computed using the formula

$$\sum_{j=1}^{M}\left(\sum_{x}\delta_h(\mathbf{x}-\mathbf{X}_j)\delta_h(\mathbf{x}-\mathbf{X}_k)\Delta s^2 h^3\right)\Delta\mathbf{F}(\mathbf{X}_j) = \frac{\vec{U}^{n+1}(\mathbf{X}_k) - \vec{U}^{*}(\mathbf{X}_k)}{\Delta t}, \tag{5}$$

where $\vec{U}^{n+1}$ and $\vec{U}^{*}$ are the desired and 'predicted' velocities at the Lagrangian points respectively; δ_h is the regularized Delta function; h and Δs are the grid sizes of the Euler and Lagrangian points respectively; Δt is the time step. $\vec{U}^{*}$ is evaluated by

$$\vec{U}^{*}(\mathbf{X}_k) = \sum_{x}\vec{u}^{*}(\mathbf{x})\delta_h(\mathbf{x}-\mathbf{X}_k)h^3, \tag{6}$$

where $\vec{u}^{*}$ is the 'predicted' velocity computed in step (i). By definition, the force correction $\Delta\mathbf{f}$ at the Eulerian grid points can be computed using the transformation