An Efficient Parallel Immersed Boundary Algorithm using a Pseudo-Compressible Fluid Solver☆

Jeffrey K. Wiens[a,*], John M. Stockie[a]

[a]*Department of Mathematics, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada, V5A 1S6*

Abstract

We propose an efficient algorithm for the immersed boundary method on distributed-memory architectures, with the computational complexity of a completely explicit method and excellent parallel scaling. The algorithm utilizes the pseudo-compressibility method recently proposed by Guermond and Minev that uses a directional splitting strategy to discretize the incompressible Navier-Stokes equations, thereby reducing the linear systems to a series of one-dimensional tridiagonal systems. We perform numerical simulations of several fluid-structure interaction problems in two and three dimensions and study the accuracy and convergence rates of the proposed algorithm. For these problems, we compare the proposed algorithm against other second-order projection-based fluid solvers. Lastly, the strong and weak scaling properties of the proposed algorithm are investigated.

Keywords: immersed boundary method, fluid-structure interaction, fractional step method, pseudo-compressibility method, domain decomposition, parallel algorithm
2010 MSC: 74F10, 76M12, 76D27, 65Y05

1. Introduction

The immersed boundary (IB) method is a mathematical framework for studying fluid-structure interaction that was originally developed by Peskin to simulate the flow of blood through a heart valve [42]. The IB method has been used in a wide variety of biofluids applications including blood flow through heart valves [19, 42], aerodynamics of the vocal cords [11], sperm motility [9], insect flight [34], and jellyfish feeding dynamics [25]. The method is also increasingly being applied in non-biological applications [36].

☆We acknowledge support from the Natural Sciences and Engineering Research Council of Canada (NSERC) through a Postgraduate Scholarship (JKW) and a Discovery Grant (JMS). The numerical simulations in this paper were performed using computing resources provided by WestGrid and Compute Canada.

*Corresponding author.
Email addresses: jwiens@sfu.ca (Jeffrey K. Wiens), jstockie@sfu.ca (John M. Stockie)
URL: http://www.jkwiens.com/ (Jeffrey K. Wiens), http://www.math.sfu.ca/~stockie (John M. Stockie)

The immersed boundary equations capture the dynamics of both fluid and immersed elastic structure using a mixture of Eulerian and Lagrangian variables: the fluid is represented by Eulerian coordinates that are fixed in space, and the immersed boundary is described by a set of moving Lagrangian coordinates. An essential component of the model is the Dirac delta function that mediates interactions between fluid and IB quantities in two ways. First of all, the immersed boundary exerts a elastic force (possibly singular) on the fluid through an external force term in the Navier-Stokes equations that is calculated using the current IB configuration. Secondly, the immersed boundary is constrained to move at the same velocity as the surrounding fluid, which is just the no-slip condition. The greatest advantage of this approach is that when the governing equations are discretized, no boundary-fitted coordinates are required to handle the solid structure and the influence of the immersed boundary on the fluid is captured solely through an external body force.

When devising a numerical method for solving the IB equations, a common approach is to use a fractional-step scheme in which the fluid is decoupled from the immersed boundary, thereby reducing the overall complexity of the method. Typically, these fractional-step schemes employ some permutation of the following steps:

- *Velocity interpolation:* wherein the fluid velocity is interpolated onto the immersed boundary.
- *IB evolution:* the immersed boundary is evolved in time using the interpolated velocity field.
- *Force spreading:* which calculates the force exerted by the immersed boundary and spreads it onto the nearby fluid grid points, with the resulting force appearing as an external forcing term in the Navier-Stokes equations.
- *Fluid solve:* which evolves the fluid variables in time using the external force calculated in the force spreading step.

Algorithms that fall into this category include Peskin's original method [42] as well as algorithms developed by Lai and Peskin [29], Griffith and Peskin [20], and many others.

A popular recent implementation of fractional-step type is the IBAMR code [28] that supports distributed-memory parallelism and adaptive mesh refinement. This project grew out of Griffith's doctoral thesis [15] and was outlined in the papers [18, 20]. In the original IBAMR algorithm, the incompressible Navier-Stokes equations are solved using a second-order accurate projection scheme in which the viscous term is handled with an L-stable discretization [33, 53] while an explicit second-order Godunov scheme [8, 35] is applied to the nonlinear advection terms. The IB evolution equation is then integrated in time using a strong stability-preserving Runge-Kutta method [14]. Since IBAMR's conception, drastic improvements have been made that increase both the accuracy and generality of the software [17, 19].

Fractional-step schemes often suffer from a severe time step restriction due to numerical stiffness that arises from an explicit treatment of the immersed boundary in the most commonly used splitting approaches [50]. Because of this limitation, many researchers have proposed new algorithms that couple the fluid and immersed boundary together in an implicit fashion, for example [5, 27, 30, 37, 40]. These methods alleviate the severe

time step restriction, but do so at the expense of solving large nonlinear systems of algebraic equations in each time step. Although these implicit schemes have been shown in some cases to be competitive with their explicit counterparts [41], there is not yet sufficient evidence to prefer one approach over the other, especially when considering parallel implementations.

A common class of fractional-step schemes for solving the incompressible Navier-Stokes equations makes use of a projection method, and is divided into two steps. First, the discretized momentum equations are integrated in time to obtain an intermediate velocity field that in general is not divergence-free. In the second step, the intermediate velocity is projected onto the space of divergence-free fields using the Hodge decomposition. The projection step typically requires the solution of large linear systems in each time step that are computationally costly and exhibit poor parallel scaling. This cost is increased even more when a small time step is required for explicit implementations. Note that even though some researchers make use of unsplit discretizations of the Navier-Stokes equations [17, 41], there is nonetheless significant benefit to be had by using a split-step projection method as a preconditioner [16]. Therefore, any improvements made to a multi-step fluid solver can reasonably be incorporated into unsplit schemes as well.

In this paper, we develop a fractional-step IB method that has the computational complexity of a completely explicit method and exhibits excellent parallel scaling on distributed-memory architectures. This is achieved by abandoning the projection method paradigm and instead adopting the pseudo-compressible fluid solver developed by Guermond and Minev [21, 22]. Pseudo-compressibility methods relax the incompressibility constraint by perturbing it in an appropriate manner. Familiar methods include Temam's penalty method [51], the artificial compressibility method [6], and Chorin's projection method [7, 45]. Guermond and Minev's algorithm differentiates itself by employing a directional-splitting strategy, thereby permitting the linear systems of size $N^d \times N^d$ typically arising in projection methods (where $d = 2$ or 3 is the problem dimension) to be replaced with a set of one-dimensional tridiagonal systems of size $N \times N$. These tridiagonal systems can be solved efficiently on a distributed-memory computing architectures by combining Thomas's algorithm with a Schur-complement technique. This allows the proposed IB algorithm to exhibit near-ideal scaling. The only serious limitation of the IB algorithm is that it is restricted to simple (rectangular) geometries and boundary conditions due to the directional-splitting strategy adopted by Guermond and Minev. However, since IB practitioners often use a rectangular fluid domain with periodic boundary conditions, this is not a serious limitation. Instead, the IB method provides a natural setting to leverage the strengths of Guermond and Minev's algorithm allowing complex geometries to be incorporated into the domain through an immersed boundary. This is a simple alternative to the fictitious domain procedure proposed by Angot et al. [1].

In section 2, we begin by stating the governing equations for the immersed boundary method. We continue by describing the proposed numerical scheme in section 3 where we incorporate the higher-order rotational form of Guermond and Minev's algorithm which discretizes an $\mathcal{O}\left(\Delta t^2\right)$ perturbation of the Navier-Stokes equations to yield a formally $\mathcal{O}\left(\Delta t^{3/2}\right)$ accurate method. As a result, the proposed method has convergence properties similar to a fully second-order projection method, while maintaining the computational complexity of a completely explicit method. In section 4, we discuss implementation

details of the algorithm for distributed-memory architectures. Finally, in section 5, we demonstrate the accuracy, efficiency and parallel scaling properties of our method through several test problems in 2D and 3D.

2. Immersed Boundary Equations

In this paper, we consider a d-dimensional Newtonian, incompressible fluid that fills a periodic box $\Omega = [0, H]^d$ having side length H and dimension $d = 2$ or 3. The fluid is specified using Eulerian coordinates, $\boldsymbol{x} = (x, y)$ in 2D or (x, y, z) in 3D. Immersed within the fluid is a neutrally-buoyant, elastic structure $\Gamma \subset \Omega$ that we assume is either a single one-dimensional elastic fiber, or else is constructed from a collection of such fibers. In other words, Γ can be a curve, surface or region. The immersed boundary can therefore be described using a fiber-based Lagrangian parameterization, in which the position along any fiber is described by a single parameter s. If there are multiple fibers making up Γ (for example, for a "thick" elastic region in 2D, or a surface in 3D) then a second parameter r is introduced to identify individual fibers. The Lagrangian parameters are assumed to be dimensionless and lie in the interval $s, r \in [0, 1]$.

In the following derivation, we state the governing equations for a single elastic fiber in dimension $d = 2$, and the extension to the three-dimensional case or for multiple fibers is straightforward. The fluid velocity $\boldsymbol{u}(\boldsymbol{x}, t) = (u(\boldsymbol{x}, t), v(\boldsymbol{x}, t))$ and pressure $p(\boldsymbol{x}, t)$ at location $\boldsymbol{x}$ and time t are governed by the incompressible Navier-Stokes equations

$$\rho \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) + \nabla p = \mu \nabla^2 \boldsymbol{u} + \boldsymbol{f}, \tag{1}$$

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

where ρ is the fluid density and μ is the dynamic viscosity (both constants). The term $\boldsymbol{f}$ appearing on the right hand side of (1) is an elastic force arising from the immersed boundary that is given by

$$\boldsymbol{f}(\boldsymbol{x}, t) = \int_\Gamma \boldsymbol{F}(s, t)\, \delta(\boldsymbol{x} - \boldsymbol{X}(s, t))\, ds, \tag{3}$$

where $\boldsymbol{x} = \boldsymbol{X}(s, t) = (X(s, t), Y(s, t))$ represents the IB configuration and $\boldsymbol{F}(s, t)$ is the elastic force density. The delta function $\delta(\boldsymbol{x}) = \delta(x)\delta(y)$ is a Cartesian product of one-dimensional Dirac delta functions, and acts to "spread" the IB force from Γ onto adjacent fluid particles. In general, the force density $\boldsymbol{F}$ is a functional of the current IB configuration

$$\boldsymbol{F}(s, t) = \boldsymbol{\mathcal{F}}\left[\boldsymbol{X}(s, t)\right]. \tag{4}$$

For example, the force density

$$\boldsymbol{\mathcal{F}}[\boldsymbol{X}(s, t)] = \sigma \frac{\partial}{\partial s} \left(\frac{\partial \boldsymbol{X}}{\partial s} \left(1 - \frac{L}{|\frac{\partial \boldsymbol{X}}{\partial s}|} \right) \right) \tag{5}$$

corresponds to a single elastic fiber having "spring constant" σ and an equilibrium state in which the elastic strain $|\partial \boldsymbol{X}/\partial s| \equiv L$.

The final equation needed to close the system is an evolution equation for the immersed boundary, which comes from the simple requirement that Γ must move at the local fluid velocity:

$$\frac{\partial \boldsymbol{X}(s,t)}{\partial t} = \boldsymbol{u}(\boldsymbol{X}(s,t),t) = \int_{\Omega} \boldsymbol{u}(\boldsymbol{x},t)\,\delta(\boldsymbol{x}-\boldsymbol{X}(s,t))\,d\boldsymbol{x}. \tag{6}$$

This last equation is nothing other than the no-slip condition, with the second delta function convolution form being more convenient for numerical computations because of its resemblance to the IB forcing term (3). Periodic boundary conditions are imposed on both the fluid and the immersed structure and appropriate initial values are prescribed for the fluid velocity $\boldsymbol{u}(\boldsymbol{x},0)$ and IB position $\boldsymbol{X}(s,0)$. Further details on the mathematical formulation of the immersed boundary problem and its extension to three dimensions can be found in [43].

3. Algorithm

We now provide a detailed description of our algorithm for solving the immersed boundary problem. The novelty in our approach derives first from the use of a pseudo-compressibility method for solving the incompressible Navier-Stokes equations, which is new in the IB context and is described in this section. The second novel aspect of our algorithm is in the parallelization, which is detailed in section 4.

3.1. Pseudo-Compressibility Methods

Pseudo-compressibility methods [45, 48] are a general class of numerical schemes for approximating the incompressible Navier-Stokes equations by appropriately relaxing the incompressibility constraint. An $\mathcal{O}(\epsilon)$ perturbation of the governing equations is introduced in the following manner

$$\rho\left(\frac{\partial \boldsymbol{u}_\epsilon}{\partial t} + \boldsymbol{u}_\epsilon \cdot \nabla \boldsymbol{u}_\epsilon\right) + \nabla p_\epsilon = \mu \nabla^2 \boldsymbol{u}_\epsilon + \boldsymbol{f}, \tag{7}$$

$$\frac{\epsilon}{\rho}\mathbf{A} p_\epsilon + \nabla \cdot \boldsymbol{u}_\epsilon = 0, \tag{8}$$

where various choices of the generic operator $\mathbf{A}$ lead to a number of familiar numerical schemes. For example, choosing $\mathbf{A} = \mathbf{1}$ (the identity) corresponds to the penalty method of Temam [51], $\mathbf{A} = \partial_t$ yields the artificial compressibility method [6], $\mathbf{A} = -\nabla^2$ is equivalent to Chorin's projection scheme [7, 45] (as long as the perturbation parameter is set equal to the time step, $\epsilon = \Delta t$), and $\mathbf{A} = -\nabla^2 \partial_t$ yields Shen's method [47] (when $\epsilon = \beta\rho(\Delta t)^2$ for some positive constant β).

Recently, Guermond and Minev [21, 22] proposed a new pseudo-compressibility method that exhibits remarkable parallel scaling properties. The first-order version of their method can be cast in the form of an $\mathcal{O}(\epsilon)$-perturbation such as that shown in equations (7)–(8) with $\epsilon = \Delta t$ and

$$\mathbf{A} = \begin{cases} (1-\partial_{xx})(1-\partial_{yy}) & \text{in 2D,} \\ (1-\partial_{xx})(1-\partial_{yy})(1-\partial_{zz}) & \text{in 3D.} \end{cases}$$

They also proposed an $\mathcal{O}\left(\epsilon^2\right)$ (second-order in time) variant that corresponds to the three-stage scheme

$$\rho\left(\frac{\partial \boldsymbol{u}_\epsilon}{\partial t} + \boldsymbol{u}_\epsilon \cdot \nabla \boldsymbol{u}_\epsilon\right) + \nabla p_\epsilon = \mu \nabla^2 \boldsymbol{u}_\epsilon + \boldsymbol{f}, \tag{9}$$

$$\frac{\epsilon}{\rho}\mathbf{A}\psi_\epsilon + \nabla \cdot \boldsymbol{u}_\epsilon = 0, \tag{10}$$

$$\epsilon \frac{\partial p_\epsilon}{\partial t} = \psi_\epsilon - \chi \mu \nabla \cdot \boldsymbol{u}_\epsilon, \tag{11}$$

where ψ_ϵ is an intermediate variable and $\chi \in (0, 1]$ is an adjustable parameter (Guermond and Minev [21] suggest using $\chi = \frac{1}{2}$).

For both variants of the method, corresponding to either (9)–(10) or (9)–(11), the momentum equation is discretized in time using a Crank-Nicolson step and the viscous term is directionally-split using the technique proposed by Douglas [10]. The perturbed incompressibility constraint is solved using a straightforward discretization of the direction-split factors in the operator $\mathbf{A}$ that reduces to a set of one-dimensional tridiagonal systems. These simple linear systems can be solved very efficiently on a distributed-memory machine by combining Thomas's algorithm with a Schur-complement technique. This is achieved by expressing each tridiagonal system using block matrices and manipulating the original system into a set of block-structured systems and a Schur complement system. By solving these block-structured systems in parallel, the domain decomposition can be effectively parallelized.

It is important to note that Guermond and Minev's fluid solver cannot be recast as a pressure projection algorithm; nevertheless, it has been demonstrated both analytically [24] and computationally [23] to have comparable convergence properties to related projection methods. More precisely, the higher-order algorithm we apply here yields a formally $\mathcal{O}\left(\Delta t^{3/2}\right)$ accurate method for 2D flows, although in practice higher convergence rates are observed in both 2D and 3D computations.

The main disadvantage of the algorithm is that it is limited to simple (rectangular) geometries because of the use of directional-splitting. However, this is not a real disadvantage in the immersed boundary context because complex solid boundaries can be introduced by using immersed boundary points (attached to fixed "tether points") that are embedded within a regular computational domain. In this way, the IB method provides a simple and efficient alternative to the fictitious domain approach [1] and related methods that could be used to incorporate complex geometries into Guermond and Minev's fluid solver.

3.2. Discretization of Fluid and IB Domains

When discretizing the governing equations, we require two separate computational grids, one each for the Eulerian and Lagrangian variables. For simplicity, we state our discrete scheme for a two-dimensional fluid ($d = 2$) and a fiber consisting of a single one-dimensional closed curve. The immersed structure is discretized using N_s uniformly-spaced points $s_k = kh_s$ in the interval $[0, 1]$, with mesh spacing $h_s = 1/N_s$ and $k = 0, 1, \ldots, N_s - 1$. As a short-hand, we denote discrete approximations of the IB position at time $t_n = n\Delta t$ by

$$\boldsymbol{X}_k^n \approx \left(X(kh_s, t_n),\ Y(kh_s, t_n)\right),$$

where $n = 0, 1, 2, \ldots$. Similarly, the fluid domain $\Omega = [0, H]^2$ is divided into an $N \times N$, uniform, rectangular mesh in which each cell has side length $h = H/N$. We employ a *marker-and-cell* (MAC) discretization [26] as illustrated in Figure 1, in which the pressure

$$p_{i,j}^n \approx p(\boldsymbol{x}_{i,j}, t_n)$$

is approximated at the cell center points

$$\boldsymbol{x}_{i,j} = ((i + 1/2)h,\ (j + 1/2)h)\,,$$

for $i, j = 0, 1, \ldots, N-1$. The velocities on the other hand are approximated at the edges of cells

$$\boldsymbol{u}_{i,j}^{\mathrm{E},n} = \left(u_{i,j}^{\mathrm{E},n},\ v_{i,j}^{\mathrm{E},n}\right),$$

where

$$u_{i,j}^{\mathrm{E},n} \approx u(ih, (j + 1/2)h, t_n) \qquad \text{and} \qquad v_{i,j}^{\mathrm{E},n} \approx v((i + 1/2)h, jh, t_n).$$

The x-component of the fluid velocity is defined on the east and west cell edges, while the y-component is located on the north and south edges.

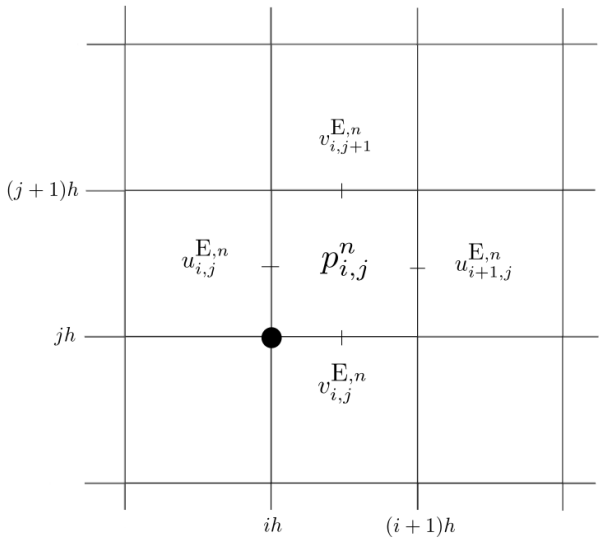


Figure 1: Location of fluid velocity and pressure variables on the staggered marker-and-cell (MAC) grid.

3.3. Spatial Finite Difference Operators

Next, we introduce the discrete difference operators that are used for approximating spatial derivatives. The second derivatives of a scalar Eulerian variable are replaced using

the second-order centered difference stencils

$$\mathbb{D}_{xx}p_{i,j} = \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{h^2}$$

$$\text{and} \qquad \mathbb{D}_{yy}p_{i,j} = \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{h^2}.$$

The same operators may be applied to the vector velocity, so that for example

$$\mathbb{D}_{xx}\boldsymbol{u}^{\mathrm{E}}_{i,j} = \begin{bmatrix} \mathbb{D}_{xx}u^{\mathrm{E}}_{i,j} \\ \mathbb{D}_{xx}v^{\mathrm{E}}_{i,j} \end{bmatrix}.$$

Since the fluid pressure and velocity variables are defined at different locations (i.e., cell centers and edges respectively), we also require difference operators whose input and output are at different locations, and for this purpose we indicate explicitly the locations of the input and output using a superscript of the form $^{Input \to Output}$. For example, an operator with the superscript $^{\mathrm{C}\to\mathrm{E}}$ takes a cell-centered input variable (denoted "C") and returns an output value located on a cell edge (denoted "E"). Using this notation, we may then define the discrete gradient operator $\mathbb{G}^{\mathrm{C}\to\mathrm{E}}$ as

$$\mathbb{G}^{\mathrm{C}\to\mathrm{E}}p_{i,j} = \left(\frac{p_{i,j} - p_{i-1,j}}{h}, \ \frac{p_{i,j} - p_{i,j-1}}{h} \right),$$

which acts on the cell-centered pressure variable and returns a vector-valued quantity on the edges of a cell. Likewise, the discrete divergence of the edge-valued velocity

$$\mathbb{D}^{\mathrm{E}\to\mathrm{C}} \cdot \boldsymbol{u}^{\mathrm{E}}_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{h} + \frac{v_{i,j+1} - v_{i,j}}{h},$$

which returns a cell-centered value.

Difference formulas are also required for Lagrangian variables such as $\boldsymbol{X}_k$, for which we use the first-order one-sided difference approximations:

$$\mathbb{D}^{+}_{s}\boldsymbol{X}_k = \frac{\boldsymbol{X}_{k+1} - \boldsymbol{X}_k}{h_s}$$

$$\text{and} \quad \mathbb{D}^{-}_{s}\boldsymbol{X}_k = \frac{\boldsymbol{X}_k - \boldsymbol{X}_{k-1}}{h_s}.$$

Finally, when discretizing the integrals appearing in (3) and (6), we require a discrete approximation to the Dirac delta function. Here, we make use of the following approximation

$$\delta_h(\boldsymbol{x}) = \frac{1}{h^2}\phi\left(\frac{x}{h}\right)\phi\left(\frac{y}{h}\right)$$

where

$$\phi(r) = \begin{cases} \frac{1}{8}(3 - 2|r| + \sqrt{1 + 4|r| - 4r^2}) & \text{if } 0 \le |r| < 1, \\ \frac{1}{8}(5 - 2|r| - \sqrt{-7 + 12|r| - 4r^2}) & \text{if } 1 \le |r| < 2, \\ 0 & \text{if } 2 \le |r|, \end{cases} \tag{12}$$

which is a popular choice in the immersed boundary literature [20, 29, 37]. Peskin [43] describes a number of desirable properties that the discrete delta function should satisfy, and several approximate delta functions that satisfy various subsets of the properties (including equation (12)) have been derived and used in practice [3, 20, 49].

3.4. IB-GM Algorithm

We are now prepared to describe our algorithm for the IB problem based on the fluid solver of Guermond and Minev [22], which we abbreviate "IB-GM". The fluid is evolved in time in two main stages, both of which reduce to solving one-dimensional tridiagonal linear systems. In the first stage, the diffusion terms in the momentum equations are integrated in time using the directional-splitting technique proposed by Douglas [10]. The nonlinear advection term on the other hand is dealt with explicitly using the second-order Adams-Bashforth extrapolation

$$N^{n+1/2} = \frac{3}{2}\mathbb{N}(\boldsymbol{u}_{i,j}^{\mathrm{E},n}) + \frac{1}{2}\mathbb{N}(\boldsymbol{u}_{i,j}^{\mathrm{E},n-1}), \tag{13}$$

where $\mathbb{N}(\bullet)$ is an approximation of the advection term $\boldsymbol{u}\cdot\nabla\boldsymbol{u}$. In this paper, we write the advection term in skew-symmetric form

$$\mathbb{N}(\boldsymbol{u}) \approx \frac{1}{2}\boldsymbol{u}\cdot\nabla\boldsymbol{u} + \frac{1}{2}\nabla\cdot(\boldsymbol{u}\boldsymbol{u}), \tag{14}$$

and then discretize the resulting expression using the second-order centered difference scheme studied by Morinishi et al. [38].

In the second stage, the correction term ψ is calculated using Guermond and Minev's splitting operator [22], and the actual pressure variable is updated using the higher-order variant of their algorithm corresponding to (9)–(11). For all simulations, we use the same parameter value $\chi = \frac{1}{2}$ as suggested in [22].

For the remaining force spreading and velocity interpolation steps, we apply standard techniques. The integrals appearing in equations (3) and (6) are approximated to second order using the trapezoidal quadrature rule and the fiber evolution equation (6) is integrated using the second-order Adams-Bashforth extrapolation.

Assuming that the state variables are known at the $(n-1)$th and nth time steps, the IB-GM algorithm proceeds as follows.

Step 1. Evolve the IB position to time $t_{n+1/2} = (n+1/2)\Delta t$:

1a. Interpolate the fluid velocity onto immersed boundary points:

$$\boldsymbol{U}_k^n = \sum_{i,j} \boldsymbol{u}_{i,j}^{\mathrm{E},n}\delta_h(\boldsymbol{x}_{i,j}^{\mathrm{E}} - \boldsymbol{X}_k^n)\,h^2.$$

1b. Evolve the IB position to time t_{n+1} using an Adams-Bashforth discretization of (6):

$$\frac{\boldsymbol{X}_k^{n+1} - \boldsymbol{X}_k^n}{\Delta t} = \frac{3}{2}\boldsymbol{U}_k^n - \frac{1}{2}\boldsymbol{U}_k^{n-1}.$$

1c. Approximate the IB position at time $t_{n+1/2}$ using the arithmetic average:

$$\boldsymbol{X}_k^{n+1/2} = \frac{1}{2}\left(\boldsymbol{X}_k^{n+1} + \boldsymbol{X}_k^n\right).$$

Step 2. Calculate the fluid forcing term:

2a. Approximate the IB force density at time $t_{n+1/2}$ using (5):

$$\boldsymbol{F}_k^{n+1/2} = \sigma \mathbb{D}_s^- \left(\mathbb{D}_s^+ \boldsymbol{X}_k^{n+1/2} \left(\mathbb{1} - \frac{L}{\left| \mathbb{D}_s^+ \boldsymbol{X}_k^{n+1/2} \right|} \right) \right).$$

2b. Spread the IB force density onto fluid grid points:

$$\boldsymbol{f}_{i,j}^{\mathrm{E},n+1/2} = \sum_k \boldsymbol{F}_k^{n+1/2} \, \delta_h(\boldsymbol{x}_{i,j}^{\mathrm{E}} - \boldsymbol{X}_k^{n+1/2}) \, h_s.$$

Step 3. Solve the incompressible Navier–Stokes equations:

3a. Predict the fluid pressure at time $t_{n+1/2}$:

$$p_{i,j}^{*,n+1/2} = p_{i,j}^{n-1/2} + \psi_{i,j}^{n-1/2}.$$

3b. Compute the first intermediate velocity field $\boldsymbol{u}^{\mathrm{E},*}$ by integrating the momentum equations explicitly:

$$\rho \left(\frac{\boldsymbol{u}_{i,j}^{\mathrm{E},*} - \boldsymbol{u}_{i,j}^{\mathrm{E},n}}{\Delta t} + N^{n+1/2} \right) = \mu \left(\mathbb{D}_{xx} + \mathbb{D}_{yy} \right) \boldsymbol{u}_{i,j}^{\mathrm{E},n} - \mathbb{G}^{\mathrm{C}\to\mathrm{E}} p_{i,j}^{*,n+1/2} + \boldsymbol{f}_{i,j}^{\mathrm{E},n+1/2}.$$

3c. Determine the second intermediate velocity $\boldsymbol{u}^{\mathrm{E},*}$ by solving the tridiagonal systems corresponding to the x-derivative piece of the directional-split Laplacian:

$$\rho \left(\frac{\boldsymbol{u}_{i,j}^{\mathrm{E},**} - \boldsymbol{u}_{i,j}^{\mathrm{E},*}}{\Delta t} \right) = \frac{\mu}{2} \mathbb{D}_{xx} \left(\boldsymbol{u}_{i,j}^{\mathrm{E},**} - \boldsymbol{u}_{i,j}^{\mathrm{E},n} \right).$$

3d. Obtain the final velocity approximation at time t_{n+1} by solving the following tridiagonal systems corresponding to the y-derivative piece of the directional-split Laplacian:

$$\rho \left(\frac{\boldsymbol{u}_{i,j}^{\mathrm{E},n+1} - \boldsymbol{u}_{i,j}^{\mathrm{E},**}}{\Delta t} \right) = \frac{\mu}{2} \mathbb{D}_{yy} \left(\boldsymbol{u}_{i,j}^{\mathrm{E},n+1} - \boldsymbol{u}_{i,j}^{\mathrm{E},n} \right).$$

3e. Determine the pressure correction term $\psi_{i,j}^{n+1/2}$ by solving

$$\left(\mathbb{1} - \mathbb{D}_{xx} \right) \left(\mathbb{1} - \mathbb{D}_{yy} \right) \psi_{i,j}^{n+1/2} = -\frac{\rho}{\Delta t} \mathbb{D}^{\mathrm{E}\to\mathrm{C}} \cdot \boldsymbol{u}_{i,j}^{\mathrm{E},n+1}.$$

3f. Calculate the pressure at time $t_{n+1/2}$ using

$$p_{i,j}^{n+1/2} = p_{i,j}^{n-1/2} + \psi_{i,j}^{n+1/2} - \chi \mu \mathbb{D}^{\mathrm{E}\to\mathrm{C}} \cdot \left(\frac{1}{2} (\boldsymbol{u}_{i,j}^{\mathrm{E},n+1} + \boldsymbol{u}_{i,j}^{\mathrm{E},n}) \right).$$

Note that in the first step of the algorithm with $n = 0$, we do not yet have an approximation of the solution at the previous time time step, and therefore we make the following replacements:

- In Step 1b, approximate the fiber evolution equation using a first-order forward Euler approximation $\boldsymbol{X}_k^1 = \boldsymbol{X}_k^0 + \Delta t \boldsymbol{U}_k^0$.
- In Step 3a, set $p_{i,j}^{*,1/2} = 0$.
- In Step 3b, the nonlinear term from equation (13) is replaced with $N^{1/2} = \mathbb{N}(\boldsymbol{u}_{i,j}^{\mathrm{E},0})$.

4. Parallel Implementation

Here we outline the details of the algorithm that relate specifically to the parallel implementation. Since the scaling properties of our algorithm is a primary feature, it is important to discuss our implementation in order to understand the parallel characteristics of the method.

4.1. Partitioning of the Eulerian and Lagrangian Grids

Suppose that the algorithm in section 3.4 is implemented on a distributed-memory computing machine with $P = P_x \cdot P_y$ processing nodes. The parallelization is performed by subdividing the rectangular domain Ω into equally-sized rectangular partitions $\{\Omega_{\ell,m}\}$, with $\ell = 1, 2, \ldots, P_x$ and $m = 1, 2, \ldots, P_y$, where P_x and P_y refer to the number of subdivisions in the x– and y–directions respectively. Each node is allocated a single domain partition $\Omega_{\ell,m}$, along with the values of the Eulerian and Lagrangian variables contained within it. For example, the (ℓ, m) node would contain in its memory the fluid variables $\boldsymbol{u}_{i,j}^{\mathrm{E}}$ and $p_{i,j}$ for all $\boldsymbol{x}_{i,j} \in \Omega_{\ell,m}$, along with all immersed boundary data $\boldsymbol{X}_k$ and $\boldsymbol{F}_k$ such that $\boldsymbol{X}_k \in \Omega_{\ell,m}$. This partitioning is illustrated for a simple 3×3 subdivision in Figure 2(a).

In this way, the computational work required in each time step is effectively divided between processing nodes by requiring that each node update only those state variables located within its assigned subdomain. Nonetheless, some inter-node communication is still required and because we are working on a distributed memory architecture this exchange of information can be very costly and should therefore be minimized. We now describe our approach for implementing the data partitioning and communication, which makes use of infrastructure provided by Open MPI [12] and PETSc [2].

Since the fluid and immersed boundary are discretized on two different grids, the data partitioning between nodes must be handled differently in each case. The partitioning of Eulerian variables is much simpler because the spatial locations remain fixed in time and remain associated with the same node for the entire computation. In contrast, Lagrangian variables are free to move throughout the fluid and so a given IB point may move between two adjacent subdomains in the course of a single time step. As a result, the data structure and communication patterns for the Lagrangian variables are more complex.

Consider the communication required for the update of fluid variables in each time step, for which the algorithm in section 3.4 requires the explicit computation of several discrete difference operators. For points located inside a subdomain $\Omega_{\ell,m}$, these discrete

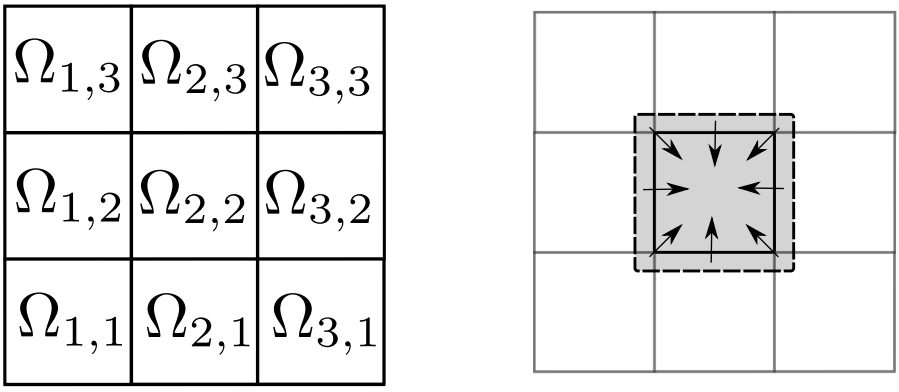


Figure 2: (a) Parallel domain decomposition with $P_x = P_y = 3$. (b) Communication required to update ghost cell regions for the subdomain $\Omega_{2,2}$.

operators are easily computed; however for points on the edge of a domain partition, a difference operator may require data that is not contained in the current node's local memory. For example, when calculating the discrete Laplacian (using the 5-point stencil) data at points adjacent to the given state variable are required. As a result, when an adjacent variable does not reside in $\Omega_{\ell,m}$, communication is required with a neighbouring node to obtain the required value. This communication is aggregated together using *ghost cells* that lie inside a strip surrounding the boundary of each $\Omega_{\ell,m}$ as illustrated in Figure 2(b). The width of the ghost region is set equal to the support of the discrete delta function used in the velocity interpolation and force spreading steps; that is, two grid points in the case of the delta-approximation (12)). When a difference operator is applied to a state variable stored in the (ℓ, m) node, the neighbouring nodes communicate the data contained in the ghost cells adjacent to $\Omega_{\ell,m}$. After the ghost region is filled, the discrete difference operators may then be calculated for all points in $\Omega_{\ell,m}$. When combined with the parallel linear solver discussed later in section 4.2, this parallel communication technique permits the fluid variables to be evolved in time.

As the IB points move through the fluid, the number of IB points residing in any particular subdomain may vary from one time step to the next. Therefore, the memory required to store the local IB data structure changes with time, as does the communication load. These complications are dealt with by splitting the data structure defining the immersed boundary into two separate components corresponding to IB points and force connections. The IB point (`IB`) data structure contains the position and velocity of all IB points resident in a given subdomain, whereas the force connection (`FC`) data structure keeps track of all force-generating connections between these points. The force density calculations depend on spatial information and so the `IB` data structure requires a globally unique index (which we call the "primary key") that is referenced by the `FC` data structure (the "foreign key"). This relationship is illustrated in Figure 3, where the

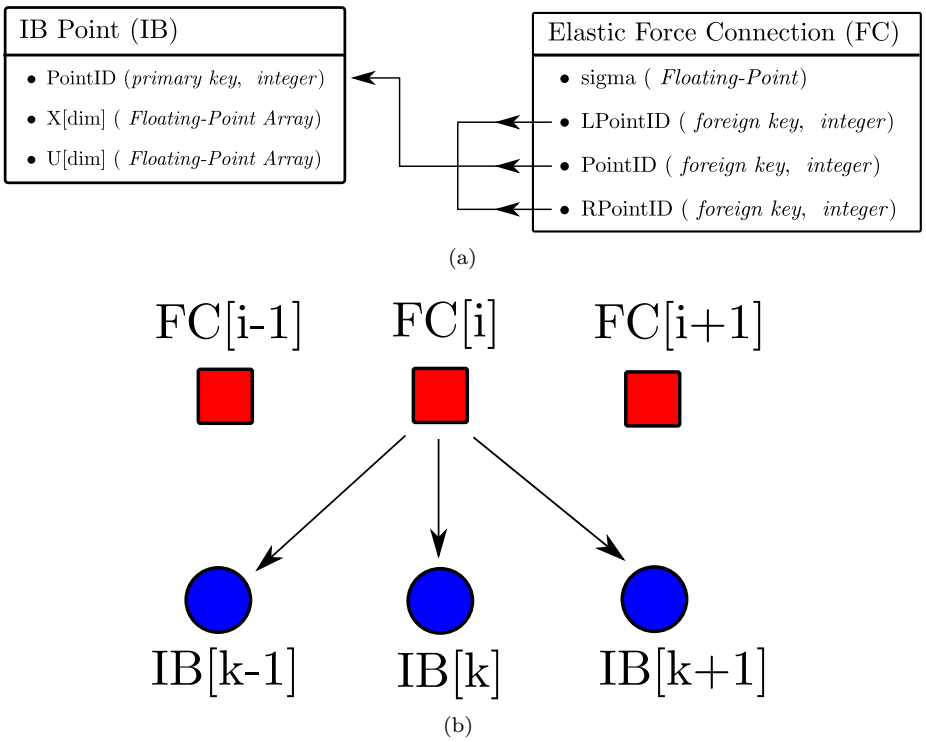


Figure 3: (a) Relationship between the data structures for the IB points (`IB`) and elastic force connections (`FC`). (b) References from a chosen force connection to the corresponding IB points.

force connections shown are consistent with the elastic force function (5). If the `IB` data structure is represented as an associative array using `PointID` as the key (and referenced as `IB[PointID]`) and `FC[i]` represents a specific element of the force connection array, then the force density calculation may be written as

$$\texttt{FC[i].Fdens} = \frac{\texttt{FC[i].sigma}}{h_s^2}\big(\texttt{IB[FC[i].LPointID].X} + \texttt{IB[FC[i].RPointID].X} - 2 * \texttt{IB[FC[i].PointID].X}\big),$$

where we have assumed here that the force parameter $L = 0$.

We are now prepared to summarize the complete parallel procedure that is used to evolve the fluid and immersed boundary. Keep in mind that at the beginning of each time step, a processing node contains only those IB points and force connections that reside in the corresponding subdomain. The individual solution steps are:

- *Velocity interpolation:* Interpolate the fluid velocity onto the IB points and store the result in `IB[•].U`. This step requires fluid velocity data from the ghost region.
- *Immersed boundary evolution:* Evolve the IB points in time by updating `IB[•].X` =

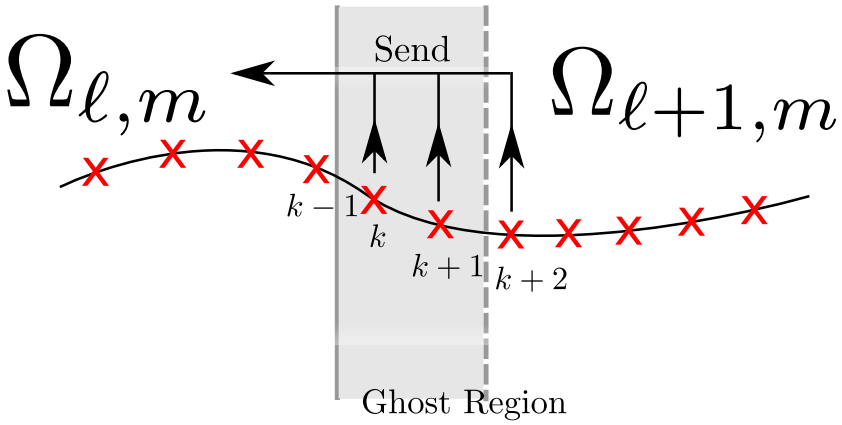


Figure 4: IB points inside the ghost region surrounding $\Omega_{\ell,m}$ are communicated from $\Omega_{\ell+1,m}$.

$\boldsymbol{X}_{\bullet}^{n+1}$. Note that the IB point position at the half time step ($\texttt{IB[•].Xh} = \boldsymbol{X}_{\bullet}^{n+1/2}$) also needs to be stored for the force spreading step.

- *Immersed boundary communication:* Send the data from IB points lying within the ghost region to the neighbouring processing nodes. Figure 4 illustrates how the IB points residing in the ghost region corresponding to $\Omega_{\ell,m}$ are copied from $\Omega_{\ell+1,m}$ (for both the full time step $n+1$ and the half-step $n+1/2$). In this example, three IB points (corresponding to $\texttt{PointID} = k, k+1, k+2$) and two force connections (with $\texttt{FC[i].PointID} = k, k+1$) are communicated to $\Omega_{\ell,m}$. The additional IB point is required to calculate the force density for $\texttt{FC[i].PointID} = k+1$. Because the IB point $k-1$ already resides in $\Omega_{\ell,m}$, the force density can be computed for $\texttt{FC[i].PointID} = k$ without any additional communication.

- *Force spreading:* Calculate the force density for all IB points in $\Omega_{\ell,m}$ and the surrounding ghost region at the time step $n+1/2$. Then spread the force density onto the Eulerian grid points residing in $\Omega_{\ell,m}$.

- *Immersed boundary cleanup:* Remove all IB points and corresponding force connections that do not reside in $\Omega_{\ell,m}$ at time step $n+1$.

- *Evolve fluid:* Evolve the fluid variables in time using the parallel techniques discussed above. This requires communication with the neighbouring processing nodes to update the ghost cell region, and further communication is needed while solving the linear systems.

Using the approach outlined above, each processing node only needs to communicate with its neighbouring nodes, with the only exception being the linear solver which we address in the next section. Since communication is often the primary bottleneck in the performance of a parallel algorithm, this is the property that allows our method to scale so well. For example, if the problem size and number of processing nodes are doubled,

then we would ideally want the execution time per time step to remain unchanged. Any algorithm that requires global communication cannot have this ideal scaling property because communication costs will increase as the number of nodes increases. Fortunately, no global communication is required in our algorithm so far, and so the linear solver is the only remaining obstacle to achieving the ideal parallel scaling.

4.2. Linear Solver

A key remaining component of the algorithm outlined in section 3.4 is the solution of the tridiagonal linear systems arising in the fluid solver. When solving the momentum equations the following linear systems arise:

$$\left(\mathbb{1} - \frac{\mu\Delta t}{2\rho}\mathbb{D}_{xx}\right)\boldsymbol{u}_{i,j}^{\mathrm{E},**} = \boldsymbol{u}_{i,j}^{\mathrm{E},*} - \frac{\mu\Delta t}{2\rho}\mathbb{D}_{xx}\boldsymbol{u}_{i,j}^{\mathrm{E},n}, \tag{15}$$

$$\left(\mathbb{1} - \frac{\mu\Delta t}{2\rho}\mathbb{D}_{yy}\right)\boldsymbol{u}_{i,j}^{\mathrm{E},n+1} = \boldsymbol{u}_{i,j}^{\mathrm{E},**} - \frac{\mu\Delta t}{2\rho}\mathbb{D}_{yy}\boldsymbol{u}_{i,j}^{\mathrm{E},n}, \tag{16}$$

while the pressure update step requires solving

$$\left(\mathbb{1} - \mathbb{D}_{xx}\right)\left(\mathbb{1} - \mathbb{D}_{yy}\right)\psi_{i,j}^{n+1/2} = -\frac{\rho}{\Delta t}\mathbb{D}^{\mathrm{E}\to\mathrm{C}} \cdot \boldsymbol{u}_{i,j}^{\mathrm{E},n+1}.$$

This last equation can be split into two steps

$$\left(\mathbb{1} - \mathbb{D}_{xx}\right)\psi_{i,j}^{*,n+1/2} = -\frac{\rho}{\Delta t}\mathbb{D}^{\mathrm{E}\to\mathrm{C}} \cdot \boldsymbol{u}_{i,j}^{\mathrm{E},n+1}, \tag{17}$$

$$\text{and} \qquad \left(\mathbb{1} - \mathbb{D}_{yy}\right)\psi_{i,j}^{n+1/2} = \psi_{i,j}^{*,n+1/2}, \tag{18}$$

where $\psi_{i,j}^{*}$ is an intermediate variable. Note that each linear system in (15)–(18) involves a difference operator that acts in one spatial dimension only and decouples into a set of one-dimensional periodic (or cyclic) tridiagonal systems. For example, equations (15) and (17) consist of N tridiagonal systems of size $N \times N$ having the general form

$$\mathbb{A}^{(j)}\Psi_{i,j} = b_{i,j}, \tag{19}$$

for each $j = 0, 1, \ldots, N-1$.

Because the processing node (ℓ, m) contains only fluid data residing in subdomain $\Omega_{\ell,m}$, these tridiagonal linear systems divide naturally between nodes. Each node solves those linear systems for which it has the corresponding data $b_{i,j} \in \Omega_{\ell,m}$ as illustrated in Figure 5. For example, when solving (19) along the x-direction, each processing node solves N/P_y linear systems and the total work is spread over P_x nodes. Similarly, when solving the corresponding systems along the y-direction, each processing node solves N/P_x systems spread over P_y nodes.

Each periodic tridiagonal system is solved directly using a Schur-complement technique [46, sec. 14.2.1]. This is achieved by rewriting the linear equations as a block-structured system where the interfaces between blocks correspond to those for the subdomains. To illustrate, let us consider an example with $P = 2$ processors only, for which

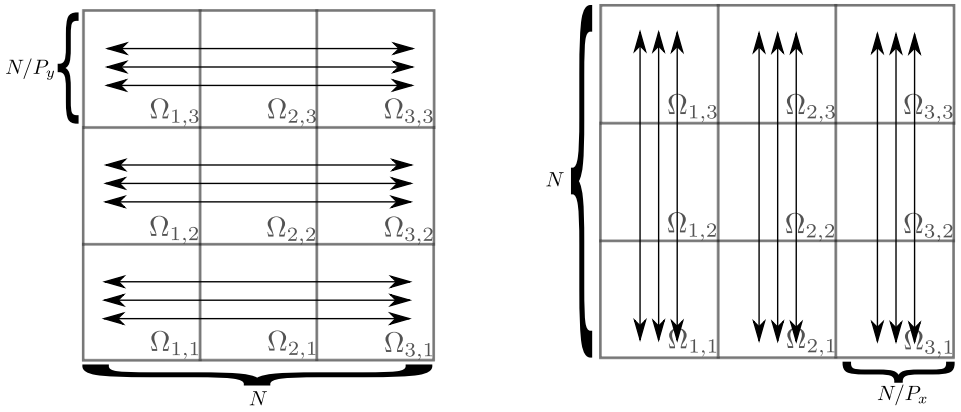


Figure 5: (a) Coupling direction for linear systems (15) and (18). (b) Coupling direction for linear systems (16) and (17). Each processing node participates in solving $N/P_{x,y}$ tridiagonal systems and requires communication in the direction of the arrows.

the periodic tridiagonal system

$$\left[\begin{array}{ccccccccccc}
a_1 & b_1 & & & & & & & & & c_1 \\
c_2 & a_2 & b_2 & & & & & & & & \\
 & \ddots & \ddots & & & & & & & & \\
 & & & \ddots & & & & & & & \\
 & & & c_{M-1} & a_{M-1} & b_{M-1} & & & & & \\
\hline
 & & & & c_M & a_M & b_M & & & & \\
 & & & & & c_{M+1} & a_{M+1} & b_{M+1} & & & \\
 & & & & & & \ddots & \ddots & & & \\
 & & & & & & & & \ddots & & \\
b_N & & & & & & & & c_N & a_N &
\end{array}\right]
\left[\begin{array}{c} y_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{M-1} \\ \hline y_2 \\ x_{M+1} \\ \vdots \\ \vdots \\ x_N \end{array}\right]
=
\left[\begin{array}{c} g_1 \\ f_2 \\ \vdots \\ \vdots \\ f_{M-1} \\ \hline g_2 \\ f_{M+1} \\ \vdots \\ \vdots \\ f_N \end{array}\right]$$

arises from a single row of unknowns in Figure 5(a) (or a single column in Figure 5(b)). In this example, the indices $M-1$ and M refer to the subdomain boundary points (denoted with a vertical line in the matrix above) so that the data (y_1, x_2, …, x_{M-1}, g_1, f_2, …, f_{M-1}) reside on processor 1 and (y_2, x_{M+1}, …, x_N, g_2, f_{M+1}, …, f_N) on processor 2. To isolate the coupling between subdomains, the rows in the matrix are reordered to shift the unknowns at periodic subdomain boundaries (y_1 and y_2) to the last two rows, and then the columns are reordered to keep the diagonal entries on the main diagonal.

This yields the equivalent linear system

$$\left[\begin{array}{ccccc|ccc|cc} a_2 & b_2 & & & & & & & c_2 & \\ & \ddots & & & & & & & & \\ & & \ddots & & & & & & & \\ & & c_{M-1} & a_{M-1} & & & & & & b_{M-1} \\ \hline & & & & & a_{M+1} & b_{M+1} & & & c_{M+1} \\ & & & & & & \ddots & & & \\ & & & & & & & \ddots & & \\ & & & & & & & c_N & a_N & b_N \\ \hline b_1 & & & & & & & & c_1 & a_1 \\ & & & c_M & & b_M & & & & a_M \end{array}\right] \left[\begin{array}{c} x_2 \\ \vdots \\ \vdots \\ x_{M-1} \\ \hline x_{M+1} \\ \vdots \\ \vdots \\ x_N \\ \hline y_1 \\ y_2 \end{array}\right] = \left[\begin{array}{c} f_2 \\ \vdots \\ \vdots \\ f_{M-1} \\ \hline f_{M+1} \\ \vdots \\ \vdots \\ f_N \\ \hline g_1 \\ g_2 \end{array}\right],$$

which has the block structure

$$\left[\begin{array}{ccc} \boldsymbol{B}_1 & & \boldsymbol{E}_1 \\ & \boldsymbol{B}_2 & \boldsymbol{E}_2 \\ \boldsymbol{F}_1 & \boldsymbol{F}_2 & \boldsymbol{C} \end{array}\right] \left[\begin{array}{c} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{y} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{g} \end{array}\right].$$

In the more general situation with P subdomains, the block structure becomes

$$\left[\begin{array}{cccc} \boldsymbol{B}_1 & & & \boldsymbol{E}_1 \\ & \boldsymbol{B}_2 & & \boldsymbol{E}_2 \\ & & \ddots & \vdots \\ & & \boldsymbol{B}_P & \boldsymbol{E}_P \\ \boldsymbol{F}_1 & \boldsymbol{F}_2 & \cdots & \boldsymbol{F}_P \quad \boldsymbol{C} \end{array}\right] \left[\begin{array}{c} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_P \\ \boldsymbol{y} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \vdots \\ \boldsymbol{f}_P \\ \boldsymbol{g} \end{array}\right],$$

or more compactly

$$\left[\begin{array}{cc} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{F} & \boldsymbol{C} \end{array}\right] \left[\begin{array}{c} \boldsymbol{x} \\ \boldsymbol{y} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{f} \\ \boldsymbol{g} \end{array}\right], \tag{20}$$

where $\boldsymbol{C} \in \mathbb{R}^{P\times P}$, $\boldsymbol{B} \in \mathbb{R}^{(N-P)\times(N-P)}$, $\boldsymbol{E} \in \mathbb{R}^{(N-P)\times P}$, and $\boldsymbol{F} \in \mathbb{R}^{P\times(N-P)}$. Here, $\boldsymbol{x}$ and $\boldsymbol{f}$ denote the data located in the interior of a subdomain while $\boldsymbol{y}$ and $\boldsymbol{g}$ denote the data residing on the interface between subdomains. Next, we use the LU factorization to rewrite the block matrix from (20) as

$$\left[\begin{array}{cc} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{F} & \boldsymbol{C} \end{array}\right] = \left[\begin{array}{cc} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{F}\boldsymbol{B}^{-1} & \boldsymbol{I} \end{array}\right] \left[\begin{array}{cc} \boldsymbol{B} & \boldsymbol{E} \\ \boldsymbol{0} & \boldsymbol{S} \end{array}\right],$$

where $\boldsymbol{S} = \boldsymbol{C} - \boldsymbol{F}\boldsymbol{B}^{-1}\boldsymbol{E}$ is the Schur complement. Using this factorized form, we can decompose the block system into the following three smaller problems:

$$\boldsymbol{B}\boldsymbol{f}^* = \boldsymbol{f}, \tag{21}$$

$$\boldsymbol{S}\boldsymbol{y} = \boldsymbol{g} - \boldsymbol{F}\boldsymbol{f}^*, \tag{22}$$

$$\boldsymbol{B}\boldsymbol{x} = \boldsymbol{B}\boldsymbol{f}^* - \boldsymbol{E}\boldsymbol{y}. \tag{23}$$

Based on this decomposition, we can now summarize the solution procedure as follows:

- *Local tridiagonal solver:* Each processor solves a local non-periodic tridiagonal system

$$\boldsymbol{B}_p \boldsymbol{f}_p^* = \boldsymbol{f}_p,$$

  which can be solved efficiently using Thomas's algorithm. The matrices $\boldsymbol{B}_p$ are the non-periodic tridiagonal blocks in the block diagonal matrix $\boldsymbol{B}$.

- *Gather data to master node:* Each processor sends three scalar values to the master node corresponding to the first and last entries of the vector $\boldsymbol{f}_p^*$, as well as the scalar g_p. Because $\boldsymbol{F}_p$ is sparse, only a few values are required to construct the right hand side of the Schur complement system.

- *Solve Schur complement system:* On the master node, solve the reduced $P \times P$ Schur complement system (22). Based on the sparsity patterns of $\boldsymbol{F}$ and $\boldsymbol{E}$, the Schur complement matrix $\boldsymbol{S}$ is periodic and tridiagonal and therefore can be inverted efficiently using Thomas's algorithm.

- *Scatter data from master node:* The master node scatters two scalar values from $\boldsymbol{y}$ to each processor. Because of the sparsity of $\boldsymbol{B}^{-1}\boldsymbol{E}$, only a few values of $\boldsymbol{y}$ are required in the next step. Therefore, the pth processor only requires the entries of $\boldsymbol{y}$ numbered p and $\mathrm{mod}(p+1, P)$.

- *Correct local solution:* Each processor corrects its local solution

$$\boldsymbol{x}_p = \boldsymbol{f}_p^* - \boldsymbol{B}_p^{-1}\boldsymbol{E}_p\boldsymbol{y},$$

  using the local values $\boldsymbol{f}_p^*$ computed in the first step.

The tridiagonal systems above can be parallelized very efficiently. As already indicated earlier, this procedure only requires two collective communications – scatter and gather – and because global communication only occurs along one spatial direction the communication overhead increases only marginally with the number of processors. A further cost savings derives from the fact that the tridiagonal systems do not change from one time step to the next, and so the matrices $\boldsymbol{S}$ and $\boldsymbol{B}^{-1}\boldsymbol{E}$ can be precomputed.

The only potential bottleneck in this procedure is in solving the reduced Schur complement system (22). Since the reduced system is solved only on the master node, the clock cycles on the remaining idle nodes are wasted at this time. Furthermore, this wasted time increases as the number of processors increase since the Schur complement system grows with P. Fortunately, the IB algorithm never solves just a single tridiagonal system. For example, when solving (19) along the x-direction, $P = P_x$ processing nodes work together to solve N/P_y tridiagonal systems. Therefore, the N/P_y systems when solved together require solving N/P_y different Schur complement systems. This workload can be spread out evenly between the P processors keeping all the processors occupied.

5. Numerical Results

To test the accuracy and parallel performance of our algorithm, we consider the following four model problems:

- *Thin ellipse:* an idealized one-dimensional elliptical membrane with zero thickness immersed in a 2D fluid. This is a standard test problem in the IB literature.
- *Thick elliptical shell:* a generalization of the first example to a thick immersed boundary, made up of multiple fibers with an elastic stiffness that is reduced smoothly to zero at the edges. This example demonstrates the second-order spatial accuracy of the numerical method for smooth problems.
- *Multiple thin ellipses:* that simulates multiple copies of the "thin ellipse" and demonstrates the parallel scaling of the algorithm in 2D.
- *Cylindrical shell:* a natural extension of the "thin ellipse" problem to 3D that demonstrates the effectiveness of our algorithm for 3D problems.

These test examples are chosen in order to show the ability of our algorithm to solve a wide range of immersed boundary problems.

5.1. Thin Ellipse

For our first 2D model problem, the initial configuration is an elliptical membrane with semi-axes r_1 and r_2, parameterized by

$$\boldsymbol{X}(s,0) = \left(\frac{1}{2} + r_1\cos(2\pi s),\ \frac{1}{2} + r_2\sin(2\pi s)\right),$$

with $s \in [0,1]$. The ellipse is placed in a unit square containing fluid that is initially stationary with $\boldsymbol{u}(\boldsymbol{x},0) = 0$. We see from the solution snapshots in Figure 6 that the elastic membrane undergoes a damped periodic motion, oscillating back and forth between elliptical shapes having a semi-major axis aligned with the x- and y-directions. The amplitude of the oscillations decreases over time, and the membrane tends ultimately toward a circular equilibrium state with radius approximately equal to $\sqrt{r_1 r_2}$ (which has the same area as the initial ellipse).

For this problem, we actually computed results for two immersed boundary algorithms corresponding to different fluid solvers. The first algorithm, denoted GM-IB, is the same one described in section 3.4 that uses Guermond and Minev's fluid solver. The second algorithm, denoted BCM-IB, is identical to the first except that the fluid solver is replaced with a second-order projection method described by Brown, Cortez, and Minion [4]. We take values of the parameters from Griffith [17], who used $\mu = 0.01$, $\rho = 1$, $r_1 = \frac{5}{28}$, $r_2 = \frac{7}{20}$ and $N_s = \frac{19}{4}N$. We then compare our numerical results for different choices of the membrane elastic stiffness (σ) and spatial discretization (N). Unless stated otherwise, the time step is chosen so that the simulation is stable on the finest spatial grid (with $N = 512$). This is a conservative choice for the time step that attempts to avoid any unreasonable accumulation of errors in time, but it also provides limited information regarding the time step restrictions for the two methods; however, we observe in practice that there is little difference between the time step restrictions for the GM-IB and BCM-IB algorithms.

Because the fluid contained within the immersed boundary cannot escape, the area of the oscillating ellipse should remain constant in time. However, many other IB computations for this thin ellipse problem exhibit poor volume conservation which manifests

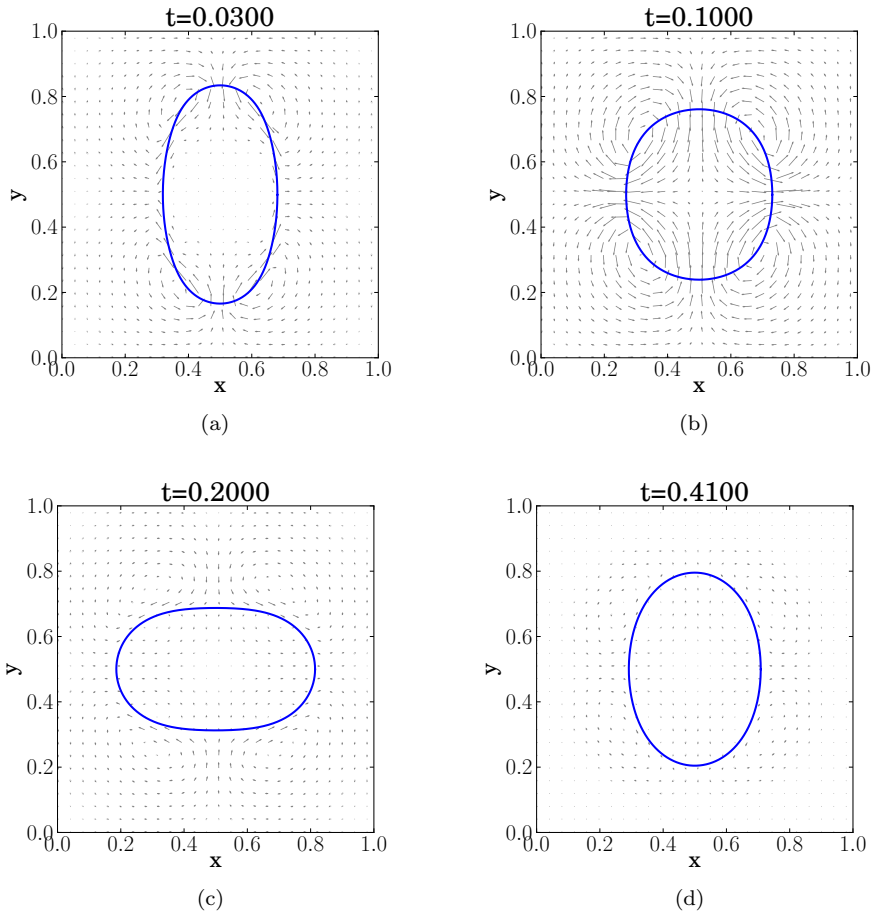
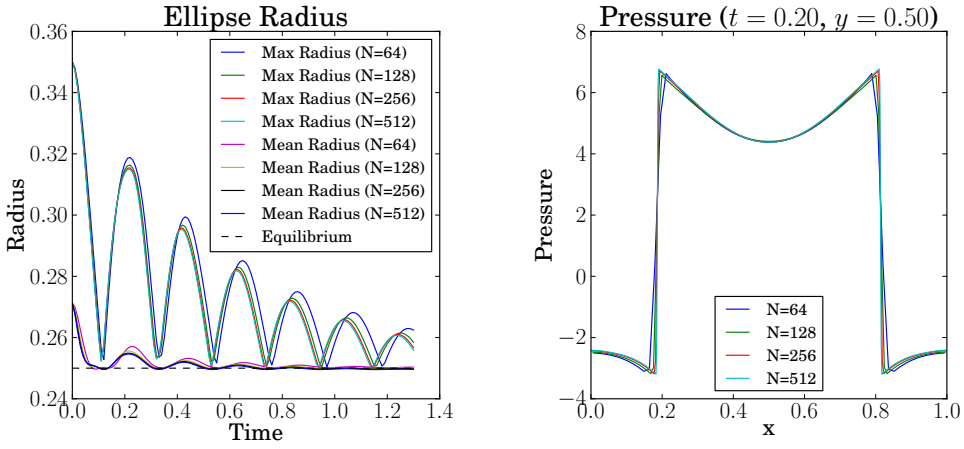


Figure 6: Snapshots of a thin oscillating ellipse using the GM-IB method, with parameters $\sigma = 1$, $N = 256$ and $\Delta t = 0.04/512$.



(a) (b)

Figure 7: Results for the thin ellipse problem using the GM-IB method with $\sigma = 1$ and $\Delta t = 0.04/512$. (a) Maximum and mean radii. (b) Pressure slices across the x–axis with $y = 0.5$ and $t = 0.2$.

itself as an apparent "leakage" of fluid out of the immersed boundary. The source of this volume conservation error is numerical error in the discrete divergence-free condition for the interpolated velocity field located on immersed boundary points, which can be non-zero even when the fluid solver guarantees that the velocity is discretely divergence-free on the Eulerian fluid grid [39, 44]. Griffith [17] observed that volume conservation can be improved by using a pressure-increment fluid solver instead of a pressure-free solver, and furthermore that fluid solvers based on a staggered grid tended to perform better than those using a collocated grid. We have employed both of these ideas in our proposed method and so we expect to see significant improvement in volume conservation relative to other IB methods.

We begin by plotting the maximum and mean radii of the ellipse versus time in Figure 7(a), from which it is clear that the immersed boundary converges to a circular steady state having radius $\sqrt{r_1 r_2} = \frac{1}{4}$. The BCM-IB results are indistinguishable from those using GM-IB, and so only the latter are depicted in this figure. The low rate of volume loss observed in both algorithms is consistent with the numerical experiments of Griffith [17]. Owing to the relatively high Reynolds number for this flow ($Re \approx 150$) there exists a noticeable error in the oscillation frequency for coarser discretizations, although we note that this error is much smaller for lower Re flows. We suspect that this frequency error could be reduced significantly by employing higher-order approximations in the nonlinear advection term and the IB evolution equation (6), such as has been done by Griffith [18]. Finally, we note that Figure 7(b) shows that the GM-IB algorithm captures the discontinuity in pressure without any visible oscillations.

We next estimate the error and convergence rate for both algorithms. Because the thin ellipse problem is characterized by a singular IB force, there is a discontinuity in velocity derivatives and pressure and so our numerical scheme is limited to first order

accuracy. We note that improvements in the convergence rate could be achieved by explicitly incorporating these discontinuities into the difference scheme, for example as is done in the immersed interface method [31, 32].

When reporting the error in a discrete variable q_N that is approximated on a grid at refinement level N, we use the notation

$$\mathcal{E}\left[q; N\right] = \|q_N - q_{\text{exact}}\|_2. \tag{24}$$

Because the exact solution for the thin ellipse problem is not known, we estimate q_{exact} by using the approximate solution on the finest mesh corresponding to $N_f = 512$, and then take $q_{\text{exact}} = \mathcal{I}^{N_f \to N} q_{N_f}$, where $\mathcal{I}^{N_f \to N}$ is an operator that interpolates the finest mesh solution q_{N_f} onto the current coarse mesh with N points. We use the discrete ℓ^2 norm to estimate errors, which is calculated for an Eulerian quantity such as the pressure using

$$\|p_{i,j}\|_2 = \left(h^2 \sum_{i,j} |p_{i,j}|^2 \right)^{1/2}, \tag{25}$$

and similarly for a Lagrangian quantity such as the IB position using

$$\|\boldsymbol{X}_k\|_2 = \left(h_s \sum_{k} |\boldsymbol{X}_k|^2 \right)^{1/2}, \tag{26}$$

where $|\cdot|$ represents the absolute value in the first formula and the Euclidean distance in the second. The convergence rate can then be estimated using solutions q_N, q_{2N} and q_{4N} on successively finer grids as

$$\mathcal{R}\left[q; N\right] = \log_2 \left(\frac{\|q_N - \mathcal{I}^{2N \to N} q_{2N}\|_2}{\|q_{2N} - \mathcal{I}^{4N \to 2N} q_{4N}\|_2} \right). \tag{27}$$

A summary of convergence rates and errors is given in Tables 1 and 2 for both the GM-IB and BCM-IB algorithms, taking different values of the elastic stiffness parameter σ. The error in all cases is measured at a time three-quarters through the ellipse's first oscillation, when the membrane is roughly circular in shape. Table 1 clearly shows that the two algorithms exhibit similar convergence rates for all state variables. First-order convergence is seen in both the fluid velocity and membrane position, while the pressure shows the expected reduction in accuracy to $\mathcal{O}\left(h^{1/2}\right)$ owing to the pressure discontinuity. The errors in Table 2 show that GM-IB and BCM-IB are virtually indistinguishable from each other except for the error in the divergence-free condition, $\mathcal{E}\left[\nabla \cdot \boldsymbol{u}; N\right]$, where the BCM-IB algorithm appears to enforce the incompressibility constraint better than GM-IB. Because Guermond and Minev's fluid solver does not project the velocity field onto the space of divergence-free velocity fields (even approximately), it is not surprising that BCM-IB performs better in this regard. This difference will be explored further in the following sections. Lastly, we remark that the magnitude of the fluid variables increases with the stiffness σ, so that the error increases as well (since $\mathcal{E}$ is defined as an absolute error measure); however, the relative error and convergence rates remain comparable as σ varies over several orders of magnitude.

Table 1: Estimated ℓ^2 convergence rates for the thin ellipse problem with three different parameter sets: ($\sigma = 0.1$, $t = 1.06$, $\Delta t = 0.08/512$), ($\sigma = 1$, $t = 0.31$, $\Delta t = 0.04/512$), ($\sigma = 10$, $t = 0.0975$, $\Delta t = 0.01/512$).

		$\mathcal{R}[\boldsymbol{u}; N]$		$\mathcal{R}[p; N]$		$\mathcal{R}[\boldsymbol{X}; N]$	
σ	N	GM	BCM	GM	BCM	GM	BCM
0.1	64	1.02	1.02	0.55	0.55	1.46	1.46
	128	1.05	1.06	0.53	0.53	1.28	1.29
1	64	1.48	1.51	0.72	0.73	1.34	1.35
	128	0.96	1.03	0.57	0.58	1.31	1.37
10	64	1.27	1.33	0.88	0.84	1.35	1.33
	128	0.89	1.03	0.68	0.82	1.32	1.71

Table 2: Estimated ℓ^2 errors the thin ellipse problem with three different parameter sets: ($\sigma = 0.1$, $t = 1.06$, $\Delta t = 0.08/512$), ($\sigma = 1$, $t = 0.31$, $\Delta t = 0.04/512$), ($\sigma = 10$, $t = 0.0975$, $\Delta t = 0.01/512$).

		$\mathcal{E}[\boldsymbol{u}; N]$		$\mathcal{E}[p; N]$		$\mathcal{E}[\boldsymbol{X}; N]$		$\mathcal{E}[\nabla \cdot \boldsymbol{u}; N]$	
σ	N	GM	BCM	GM	BCM	GM	BCM	GM	BCM
0.1	64	7.41e−3	7.44e−3	4.13e−2	4.13e−2	2.81e−4	2.82e−4	4.77e−3	3.67e−16
	128	3.10e−3	3.16e−3	2.36e−2	2.36e−2	6.69e−5	6.75e−5	1.05e−2	7.22e−16
	256	9.64e−4	1.03e−3	1.03e−2	1.04e−2	1.36e−5	1.39e−5	1.86e−2	1.41e−15
	512	1.91e−4	–	8.84e−4	–	1.80e−6	–	2.91e−2	2.83e−15
1	64	5.61e−2	5.69e−2	4.56e−1	4.57e−1	4.32e−4	4.36e−4	1.04e−1	1.86e−15
	128	1.88e−2	1.98e−2	2.56e−1	2.57e−1	1.06e−4	1.09e−4	2.50e−1	3.60e−15
	256	6.32e−3	6.65e−3	1.13e−1	1.11e−1	2.08e−5	2.15e−5	4.55e−1	7.04e−15
	512	5.46e−3	–	4.77e−2	–	9.21e−6	–	7.23e−1	1.40e−14
10	64	3.37e−1	3.38e−1	5.88e+0	5.89e+0	6.33e−4	6.38e−4	6.12e−1	7.01e−15
	128	1.61e−1	1.62e−1	3.12e+0	3.04e+0	1.56e−4	1.57e−4	2.12e+0	1.40e−14
	256	7.06e−2	5.48e−2	1.42e+0	1.20e+0	3.08e−5	2.60e−5	3.94e+0	2.72e−14
	512	6.24e−2	–	1.12e+0	–	2.01e−5	–	6.34e+0	5.39e−14

5.2. Thick Elliptical Shell

Our second test problem involves the thick elastic shell pictured in Figure 8 that has been studied before by Griffith and Peskin [20]. This is a natural generalization of the thin ellipse problem, wherein the shell is treated using a nested sequence of elliptical immersed fibers. The purpose of this example is not only to illustrate the application of our algorithm to more general solid elastic structures, but also to illustrate the genuine second-order accuracy of our numerical method for problems that are sufficiently smooth.

To this end, we take an elliptical elastic shell with thickness γ using two independent Lagrangian parameters $s, r \in [0, 1]$, and specify the initial configuration by

$$\boldsymbol{X}(s, r, 0) = \left(\frac{1}{2} + (r_1 + \gamma(r - 1/2)) \cos(2\pi s),\ \frac{1}{2} + (r_2 + \gamma(r - 1/2)) \sin(2\pi s) \right).$$

The shell is composed of circumferential fibers having an elastic stiffness that varies in the radial direction according to

$$\sigma(r) = 1 - \cos(2\pi r).$$

Because the elastic stiffness drops to zero at the inner and outer edges of the shell, the corresponding Eulerian force $\boldsymbol{f}$ is a continuous function of $\boldsymbol{x}$; this should be contrasted with the "thin ellipse" example in which the fluid force is singular, since it consists of a 1D delta distribution in the tangential direction along the membrane. As a result, we expect in this example to observe higher order convergence because the solution does not contain the discontinuities in pressure and velocity derivatives that were present in the thin ellipse problem. Unless otherwise indicated, we take the parameter values $\rho = 1$, $r_1 = 0.2$, $r_2 = 0.25$, $\gamma = 0.0625$, $N_s = (75/16)N$, $N_r = (3/8)N$ and $\Delta t = 0.08/512$ that are consistent with the computations in [20].

The dynamics of the thick ellipse problem illustrated in Figure 8 are qualitatively similar to those in the previous section, in that the elastic shell undergoes a damped oscillation. In Table 3, we present the ℓ^2 convergence rates in the solution for different values of fluid viscosity μ. We also include the corresponding results computed by Griffith and Peskin [20] and observe that the GM-IB, BCM-IB, and Griffith-Peskin algorithms all exhibit remarkably similar convergence rates. The ℓ^2 errors for the GM-IB and BCM-IB methods are almost identical to Griffith and Peskin's, and so we have not reported them for this example.

It is only when the viscosity is taken very small ($\mu = 0.0005$) that the Griffith-Peskin algorithm begins to demonstrate superior results. Because this improvement corresponds to a higher Reynolds number, we attribute it to differences in the treatment of the nonlinear advection term and the IB evolution equation. Indeed, Griffith and Peskin approximate the nonlinear advection term using a high-order Godunov method [8, 35] and integrate the IB equation (6) using a strong stability preserving Runge-Kutta method [14]. We have made no attempt to incorporate these modifications into our algorithm because Godunov's method requires solving a Poisson problem, whereas a Runge-Kutta time integration would require an additional velocity interpolation step; consequently, employing either of these changes would drastically reduce the parallel efficiency of our algorithm. Recall that one of our primary aims is to avoid making use of Poisson solvers that are used in so many other IB methods. With this in mind, we restrict our attention to lower Reynolds number flow corresponding roughly to $Re \lessapprox 1000$.

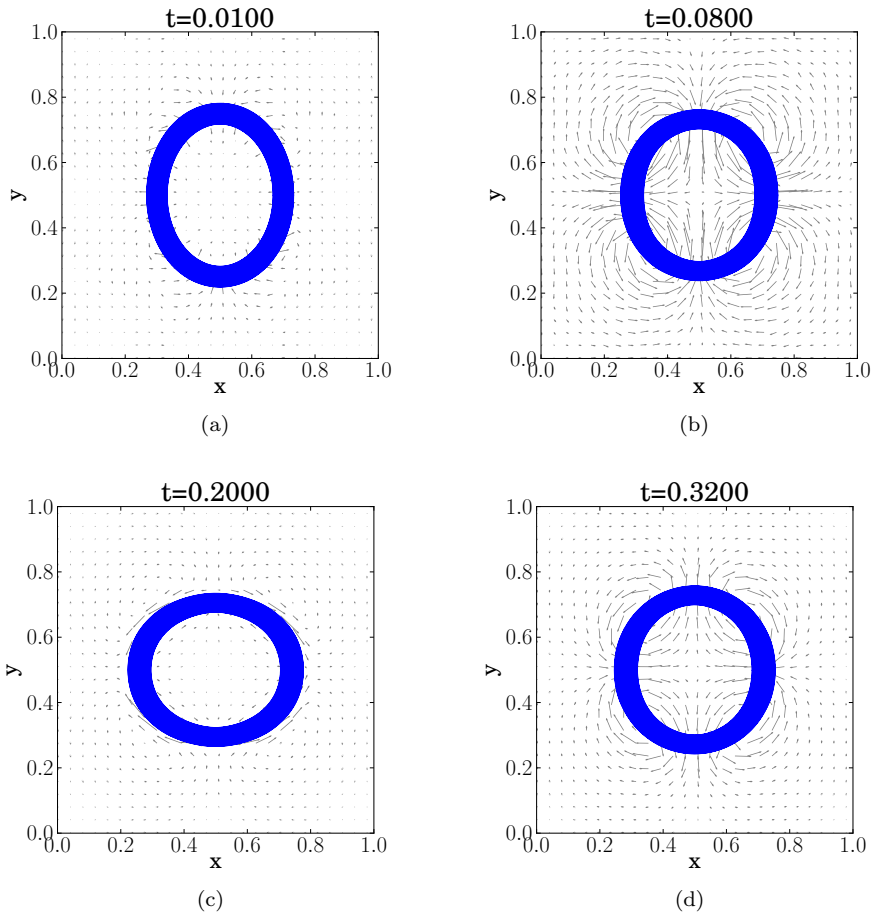


Figure 8: Snapshots of a thick oscillating ellipse using GM-IB method, with parameters $\mu = 0.005$, $N = 256$ and $\Delta t = 0.08/512$.

Table 3: Estimated ℓ^2 convergence rates $\mathcal{R}[q; 128]$ for the thick ellipse problem at time $t = 0.4$. For comparison, Griffith's results [20] are reported in the final row. Since Griffith reports the component-wise convergence rate of the velocity field, we approximate $\mathcal{R}[\boldsymbol{u}; 128] \approx \max(\mathcal{R}[u; 128], \mathcal{R}[v; 128])$.

	$\mu = 0.05$			$\mu = 0.01$			$\mu = 0.005$		
	$\boldsymbol{u}$	p	$\boldsymbol{X}$	$\boldsymbol{u}$	p	$\boldsymbol{X}$	$\boldsymbol{u}$	p	$\boldsymbol{X}$
GM-IB	2.10	1.88	1.69	2.12	1.88	1.76	2.11	1.88	1.99
BCM-IB	2.11	1.88	1.69	2.09	1.87	1.74	2.09	1.87	1.99
Griffith [20]	2.16*	1.89	1.98	–	–	–	2.20*	1.86	1.74

Lastly, we investigate the accuracy with which our discrete solution satisfies the discrete divergence-free condition for a variety of time steps and spatial discretizations. Our aim in this instance is to determine how well the fluid solver of Guermond and Minev approximates the incompressibility constraint, which is related to the volume conservation issue discussed in the thin ellipse example. Table 4 lists values of the error in the discrete divergence of velocity, $\mathcal{E}\left[\nabla \cdot \boldsymbol{u}; N\right]$, measured at time $t = 0.4$ and estimated using equation (24). Observe that $\mathcal{E}\left[\nabla \cdot \boldsymbol{u}; N\right]$ increases slightly as the spatial discretization is refined, but decreases when a smaller time step is used. This last result is to be expected because Guermond and Minev use a $\mathcal{O}\left(\Delta t\right)$ perturbation of the incompressibility constraint (8).

Table 4: Error in the divergence-free condition $\mathcal{E}\left[\nabla \cdot \boldsymbol{u}; N\right]$ for the thick ellipse problem using the GM-IB method and $\mu = 0.01$.

	$\Delta t = \frac{0.08}{512}$	$\Delta t = \frac{0.04}{512}$	$\Delta t = \frac{0.02}{512}$	$\Delta t = \frac{0.01}{512}$	$\Delta t = \frac{0.005}{512}$	$\Delta t = \frac{0.0025}{512}$
$N = 64$	6.34e−3	2.37e−3	8.93e−4	3.20e−4	1.08e−4	3.42e−5
$N = 128$	8.94e−3	3.68e−3	1.49e−3	5.75e−4	2.11e−4	7.36e−5
$N = 256$	1.00e−2	4.19e−3	1.73e−3	6.79e−4	2.55e−4	9.06e−5
$N = 512$	1.04e−2	4.35e−3	1.80e−3	7.11e−4	2.68e−4	9.59e−5

5.3. Multiple Thin Ellipses

The next example is designed to explore in more detail the parallel scaling properties of our proposed GM-IB algorithm, by computing a variation of the thin ellipse problem from section 5.1. Because our 2D computations are performed on a doubly-periodic fluid domain, the thin ellipse geometry is actually equivalent to an infinite array of identical elliptical membranes. This periodicity in the solution provides a simple mechanism for increasing the computational complexity of a simulation by explicitly adding multiple periodic copies while technically solving a problem with an identical solution. Each copy of the original domain (see section 5.1) may then be handled by a different processing node, which allows us to explore the parallel scaling of the GM-IB algorithm.

Suppose that we would like to perform a parallel simulation using $P = P_x \cdot P_y$ processing nodes. On such a cluster, we can simulate a rectangular $P_x \times P_y$ array of identical ellipses, situated on the fluid domain $\Omega = [0, P_x] \times [0, P_y]$. We subdivide the domain into equal partitions so that each processor handles the unit-square subdomain $\Omega_{\ell,m} = [\ell - 1, \ell] \times [m - 1, m]$, for $\ell = 1, 2, \ldots, P_x$ and $m = 1, 2, \ldots P_y$. If we denote by $(x_{\ell,m}, y_{\ell,m})$ the centroid of $\Omega_{\ell,m}$, then each subdomain contains a single ellipse having the initial configuration

$$\boldsymbol{X}_{\ell,m}(s, 0) = \left(x_{\ell,m} + r_1 \cos(2\pi s),\ y_{\ell,m} + r_2 \sin(2\pi s)\right),$$

using the same Lagrangian parameter $s \in [0, 1]$ as before. In order to make the flow slightly more interesting, and to test the ability of our parallel algorithm to handle immersed boundaries that move between nodes, we impose an initial background fluid velocity field $\boldsymbol{u}(\boldsymbol{x}, 0) = \frac{1}{2}\left(1, \sqrt{3}\right)$ instead of the zero initial velocity used in section 5.1. Snapshots of the solution for a 2×2 array of ellipses are illustrated in Figure 9.

To investigate the parallel performance of our algorithm, we simulate arrays of thin ellipses corresponding to different values of P_x and P_y in the range $[1, 12]$. For each

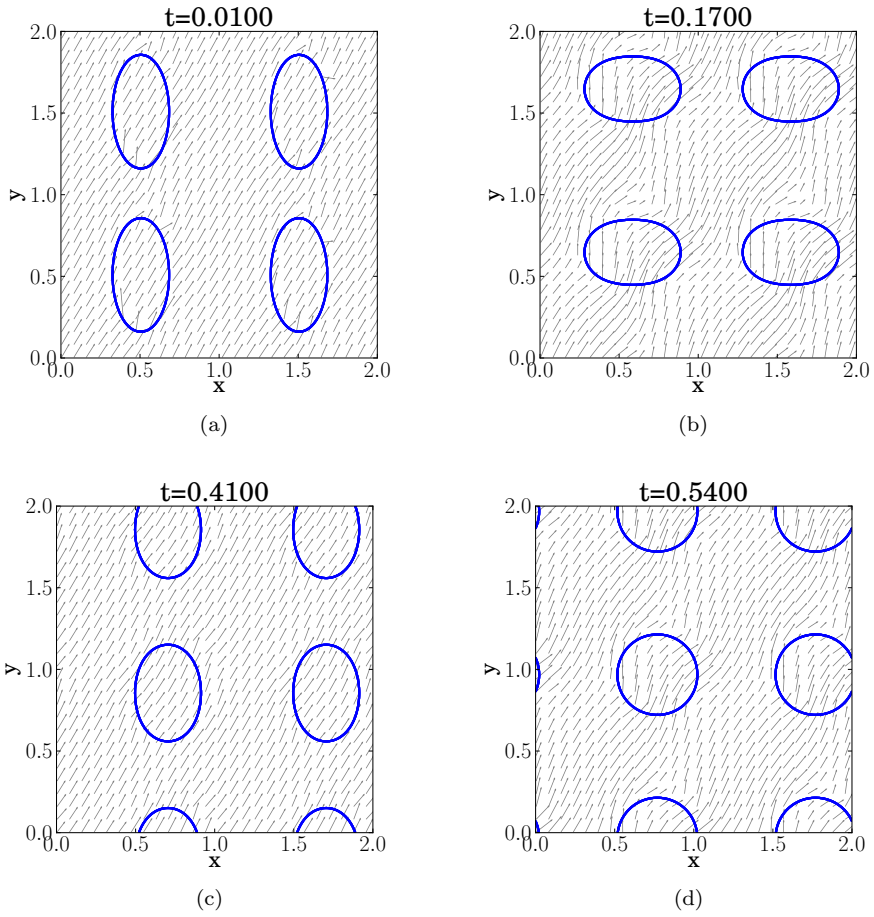


Figure 9: Simulation of a 2×2 array of thin ellipses.

simulation, we use parameters $\mu = 0.01$, $\rho = 1$, $\sigma = 1$, $r_1 = \frac{5}{28}$, $r_2 = \frac{7}{20}$, $h = \frac{1}{128}$, $h_s = \frac{4}{19}h$ and $\Delta t = \frac{0.01}{128}$ and compute up to time $t = 1.00$. In the case of perfect parallel scaling the execution time should remain constant between simulations, since for example if the problem size is doubled the number of nodes P is also doubled. Therefore, the problem represents a weak scalability test for the proposed algorithm since the workload per processor node remains constant as the number of nodes increase.

Computations for this problem were performed on the Bugaboo cluster that is managed by WestGrid [54], a member of the high-performance computing consortium Compute Canada. This cluster consists of 12-core blades, each containing two Intel Xeon X5650 6-core processors (2.66 GHz) that are connected by Infiniband using a 288-port QLogic switch.

The execution times for the various array sizes (P_x, P_y) are summarized in Table 5. We observe that while there is a slight increase in execution time with P, the differences are minimal considering the large variation in problem size. For example, the difference in execution time between simulations with $P = 1$ and $P = 64$ is only 31%! This slight degradation in performance can be attributed primarily to the increase in communication costs from the tridiagonal solver, which we recall involves scatter/gather operations before/after solving the Schur complement. As P increases, the time required for this communication increases. However, even though the size of the Schur complement systems also increases with P, these systems are distributed between more processors and so there is not a significant hit on the parallel performance owing to the Schur complement solves.

The degradation in performance caused by communication is occasionally exacerbated by high network loads experienced on the Bugaboo cluster, where we observed significant variation in execution times between otherwise identical runs. Therefore our results must be viewed as demonstrating the algorithm performance in a less than ideal setting, which is in reality a more realistic environment for most applications.

Finally, we report in Table 5 values of the parallel "efficiency" which is the ratio between the execution time of a multi-node parallel simulation with its serial counterpart ($P = 1$). In other words, the efficiency of a P-node simulation is

$$E_P = \frac{T_1}{PT_P},$$

where P is the number of processors and T_P is the corresponding execution time. This efficiency represents the strong scalability of our algorithm. An efficiency of $E_P = 1$ is "ideal" and smaller values of E_P indicate reduced parallel efficiency. Note that the serial computation involves no Schur complement systems because the tridiagonal systems can be computed directly, and therefore a reduction efficiency is to be expected. According to Table 5, we see that the parallel efficiency remains close to 1 for all simulations, and it is only for much larger problems that any significant reduction in efficiency is observed. Indeed, for some simulations we even observe super-linear speed-up ($E_P > 1$) which might be attributed to more efficient memory utilization in these simulations owing perhaps to a reduced number of cache misses. This same super-linear speed-up has also been observed by Ganzha et al. [13], who studied the parallel performance of the Guermond-Minev algorithm for the fluid-only case. We remark in closing that these results could most likely be improved by improving the underlying network topology.

Table 5: Execution time (in seconds) and parallel efficiency for the multiple thin ellipse problem with P ellipses on P processors, and $P = P_x \times P_y$.

P_x	P_y	Wall Time	Efficiency	P_x	P_y	Wall Time	Efficiency
1	1	107.60	1.00	7	7	137.78	1.03
1	2	110.28	0.99	7	8	140.55	0.93
2	2	111.44	1.03	8	8	141.36	0.91
2	3	112.42	1.05	8	9	153.00	0.83
3	3	116.67	1.05	9	9	156.18	0.85
3	4	116.87	1.08	9	10	157.25	0.86
4	4	119.63	1.06	10	10	164.55	0.81
4	5	128.97	0.99	10	11	164.54	0.77
5	5	130.78	0.97	11	11	164.59	0.78
5	6	132.72	0.96	11	12	171.04	0.76
6	6	133.96	1.07	12	12	176.45	0.76
6	7	136.35	1.04				

5.4. Cylindrical Shell in 3D

For our final test case, we consider a three-dimensional example in which the immersed boundary is a cylindrical elastic shell. The initial configuration of the shell is an elliptical cylinder having semi-axes r_1 and r_2 that is parameterized by

$$\boldsymbol{X}(s,r,0) = \left(r,\ \frac{1}{2} + r_1 \cos(2\pi s),\ \frac{1}{2} + r_2 \sin(2\pi s) \right),$$

using the two Lagrangian parameters $s, r \in [0,1]$. The force density exerted by the shell is

$$\boldsymbol{\mathcal{F}}[\boldsymbol{X}(s,r,t)] = \sigma_s \frac{\partial^2 \boldsymbol{X}}{\partial s^2} + \sigma_r \frac{\partial}{\partial r} \left(\frac{\partial \boldsymbol{X}}{\partial r} \left(1 - \frac{L}{\left| \frac{\partial \boldsymbol{X}}{\partial r} \right|} \right) \right),$$

which treats the elastic shell as an interwoven mesh of one-dimensional fibers. The s parameterization describes fibers running around the elliptical cross-section of the cylinder, that have a zero resting length and elastic stiffness σ_s. On the other hand, the r parameterization describes fibers running axially along the length of the cylinder, and here we impose a non-zero resting-length L and elastic stiffness σ_r. Since the domain is periodic in all directions, the ends of the cylinder are connected to their periodic copies so that there are no "cuts" along the fibers. This problem is essentially equivalent to the two-dimensional thin ellipse problem considered in section 5.1, with the only difference being that the 2D problem does not have any fibers running along the non-existing third dimension. The 2D thin ellipse and 3D cylinder problems are only strictly equivalent when $\sigma_r = 0$. However, we take $\sigma_r = \sigma_s = 1$ and $L = 1$ in order to maintain the integrity of the elastic shell and avoid any drifting of elliptical cross-sections in the x-direction.

The elastic shell is discretized using an interwoven mesh of 1D fibers corresponding to equally-spaced values of the parameters s and r. In the simulations to follow, we use the parameter values $\mu = 0.01$, $\rho = 1$, $r_1 = \frac{5}{28}$, $r_2 = \frac{7}{20}$, $N = 128$, $N_s = \frac{19}{4}N$, $N_r = 3N$ and $\Delta t = 0.04/N$.

The solution dynamics are illustrated by the snapshots pictured in Figure 10, which show that the elastic shell oscillates at the same frequency as the thin ellipse shown in Figure 6. Although the geometry of this problem may seem somewhat of a special case because of the alignment of axial fibers along the x-coordinate direction, this feature has no noticeable impact on parallel performance measurements. Indeed, the reason that fiber alignment doesn't affect communication cost is because all IB points and force connections residing in the ghost region are communicated regardless of whether or not they actually cross subdomain boundaries. In Table 6, we present measurements of execution time and efficiency that illustrate the parallel scaling over the first 100 time steps with the number of processors P varying between 1 and 128. In all runs, the domain is partitioned evenly between the P processing nodes using rectangular boxes. We note that the IB points in this example are not evenly distributed between domain partitions so that the computational work is not shared equally among processing nodes. This is reflected in a reduced parallel efficiency for the "Entire Computation" when P is taken larger; however, the efficiency of the fluid component of the algorithm remains near the ideal efficiency value of 1. Once again, we observe for several runs that the efficiency exceeds 1 which we ascribe to more efficient memory utilization (i.e., fewer cache misses).

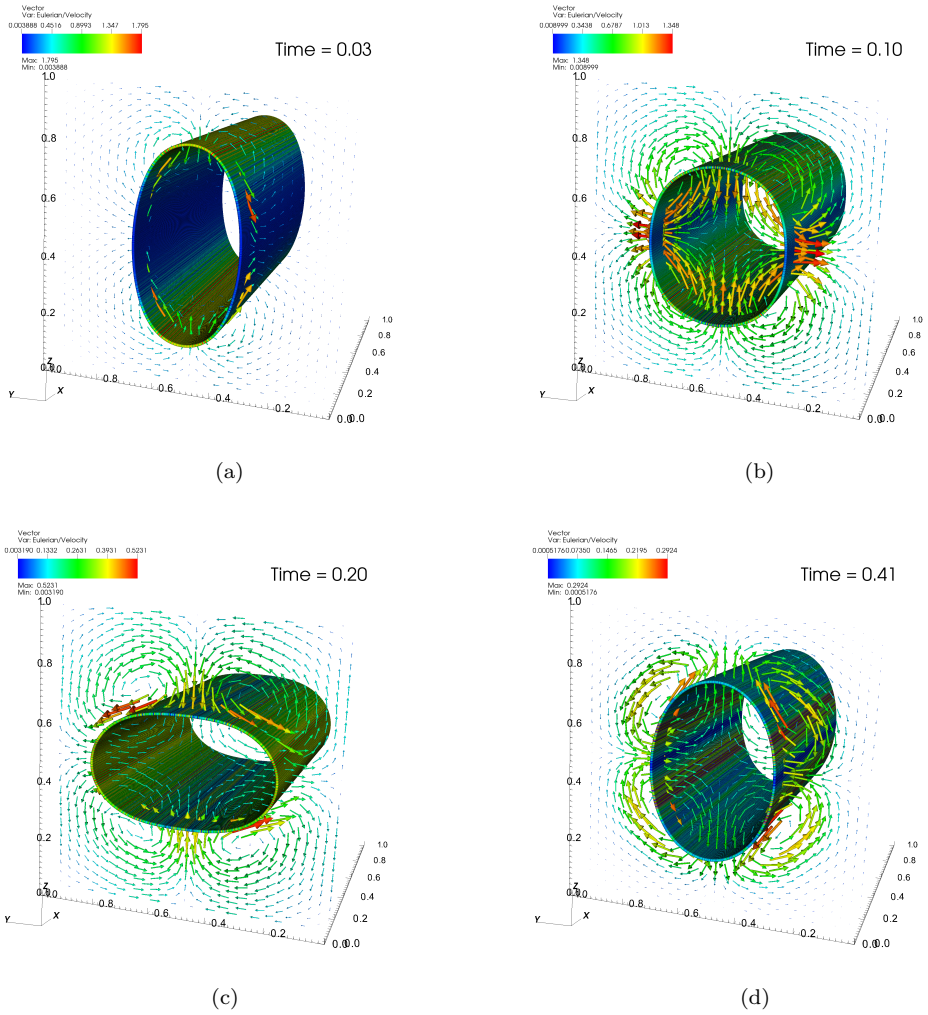
Table 6: Execution time (in seconds) and efficiency for the 3D cylindrical shell problem for a fixed fluid discretization ($N = 128$) and varying the number of processing nodes (P).

	Entire Computation		Fluid Computation	
P	Wall Time	Efficiency	Wall Time	Efficiency
1	1041.50	1.00	203.59	1.00
2	522.10	1.00	98.00	1.04
4	267.56	0.97	52.42	0.97
8	138.80	0.94	24.49	1.04
16	76.64	0.85	12.14	1.05
32	40.39	0.81	6.39	1.00
64	37.97	0.43	3.48	0.91
128	31.74	0.26	1.88	0.84

6. Conclusions

We have developed a new algorithm for solving the immersed boundary problem on distributed-memory parallel computers that is based on the pseudo-compressibility method of Guermond and Minev for solving the incompressible Navier-Stokes equations. The fundamental advantage of this fluid solver is the direction-splitting strategy applied to the incompressibility constraint, which reduces to solving a series of tridiagonal linear systems that have an extremely efficient parallel implementation. We have extended Guermond and Minev's approach by incorporating periodic boundary conditions and using a slightly different parallel implementation.

Numerical computations demonstrate the ability of our method to simulate a wide range of immersed boundary problems that includes not only 2D flows containing isolated fibers and thick membranes constructed of multiple nested fibers, but also 3D flows containing immersed elastic surfaces. The strong and weak scalability of our algorithm



(a) (b)

(c) (d)

Figure 10: Snapshots of an oscillating cylindrical shell that is initially stretched outward along the z–direction.

is demonstrated in tests with up to 144 distributed processors, where excellent speedups are observed. It is only for larger simulations that our implementation exhibits sub-linear scaling for the fluid portion of the computation. Lastly, since our implementation does not apply any load balancing strategy, some degradation in the parallel efficiency is observed in immersed boundary portion of the computation when the elastic membrane is not equally divided between processors.

We believe that our computational approach is a very promising one for solving fluid-structure interaction problems in which the solid elastic component takes up a large portion of the fluid domain, such as occur with dense particle suspensions [52] or very complex elastic structures that are distributed throughout the fluid. These are problems where local adaptive mesh refinement is less likely to offer any advantage because of the need to use a nearly-uniform fine mesh over the entire domain in order to resolve the immersed boundary. It is for this class of problems that we expect our approach to offer advantages over methods such as that of Griffith et al. [18].

We plan in future to focus on implementing modifications to our algorithm that will improve the parallel scaling, and particularly on improving memory access patterns for the Lagrangian portion of the calculation related to force spreading and velocity interpolation. We will also investigate code optimizations that aim to reduce cache misses and exploit on-chip parallelism.

References

[1] P. Angot, J. Keating, and P. D. Minev. A direction splitting algorithm for incompressible flow in complex geometries. *Computer Methods in Applied Mechanics and Engineering*, 217:111–120, 2012.

[2] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang. PETSc Users Manual, Revision 3.3. Report ANL-95/11, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, June 2012. Retrieved April 28, 2013 from `http://www.mcs.anl.gov/petsc/documentation/index.html`.

[3] T. T. Bringley and C. S. Peskin. Validation of a simple method for representing spheres and slender bodies in an immersed boundary method for Stokes flow on an unbounded domain. *Journal of Computational Physics*, 227:5397–5425, 2008.

[4] D. L. Brown, R. Cortez, and M. L. Minion. Accurate projection methods for the incompressible Navier–Stokes equations. *Journal of Computational Physics*, 168(2):464–499, 2001.

[5] H. D. Ceniceros, J. E. Fisher, and A. M. Roma. Efficient solutions to robust, semi-implicit discretizations of the immersed boundary method. *Journal of Computational Physics*, 228(19):7137–7158, 2009.

[6] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2(1):12–26, 1967.

[7] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Mathematics of Computation*, 22(104):745–762, 1968.

[8] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *Journal of Computational Physics*, 87(1):171–200, 1990.

[9] R. H. Dillon, L. J. Fauci, C. Omoto, and X. Yang. Fluid dynamic models of flagellar and ciliary beating. *Annals of the New York Academy of Sciences*, 1101(1):494–505, 2007.

[10] J. Douglas. Alternating direction methods for three space variables. *Numerische Mathematik*, 4:41–63, 1962.

[11] C. Duncan, G. Zhai, and R. Scherer. Modeling coupled aerodynamics and vocal fold dynamics using immersed boundary methods. *Acoustical Society of America Journal*, 120(5):2859–2871, 2006.

[12] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings,*

11th European PVM/MPI Users' Group Meeting, pages 97–104, Budapest, Hungary, September 2004.

[13] M. Ganzha, K. Georgiev, I. Lirkov, S. Margenov, and M. Paprzycki. Highly parallel alternating directions algorithm for time dependent problems. In *Third Conference on Application of Mathematics in Technical and Natural Sciences*, volume 1404 of *AIP Conference Proceedings*, pages 210–217, Albena, Bulgaria, June 20-25, 2011.

[14] S. Gottlieb, C. W. Shu, and E. Tadmor. Strong stability-preserving high-order time discretization methods. *SIAM Review*, 43(1):89–112, 2001.

[15] B. E. Griffith. *Simulating the blood-muscle-valve mechanics of the heart by an adaptive and parallel version of the immersed boundary method.* PhD thesis, New York University, 2005.

[16] B. E. Griffith. An accurate and efficient method for the incompressible Navier-Stokes equations using the projection method as a preconditioner. *Journal of Computational Physics*, 228(20):7565–7595, 2009.

[17] B. E. Griffith. On the volume conservation of the immersed boundary method. *Communications in Computational Physics*, 12:401–432, 2012.

[18] B. E. Griffith, R. D. Hornung, D. M. McQueen, and C. S. Peskin. An adaptive, formally second order accurate version of the immersed boundary method. *Journal of Computational Physics*, 223(1):10–49, 2007.

[19] B. E. Griffith, X. Y. Luo, D. M. McQueen, and C. S. Peskin. Simulating the fluid dynamics of natural and prosthetic heart valves using the immersed boundary method. *International Journal of Applied Mechanics*, 1(1):137–177, 2009.

[20] B. E. Griffith and C. S. Peskin. On the order of accuracy of the immersed boundary method: Higher order convergence rates for sufficiently smooth problems. *Journal of Computational Physics*, 208(1):75–105, 2005.

[21] J. L. Guermond and P. D. Minev. A new class of fractional step techniques for the incompressible Navier-Stokes equations using direction splitting. *Comptes Rendus Mathematique*, 348:581–585, 2010.

[22] J. L. Guermond and P. D. Minev. A new class of massively parallel direction splitting for the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 200(23-24):2083–2093, 2011.

[23] J. L. Guermond and P. D. Minev. Start-up flow in a three-dimensional lid-driven cavity by means of a massively parallel direction splitting algorithm. *International Journal for Numerical Methods in Fluids*, 68(7):856–871, 2011.

[24] J. L. Guermond, P. D. Minev, and A. J. Salgado. Convergence analysis of a class of massively parallel direction splitting algorithms for the Navier-Stokes equations in simple domains. *Mathematics of Computation*, 81(280):1951, 2012.

[25] C. Hamlet, A. Santhanakrishnan, and L. A. Miller. A numerical study of the effects of bell pulsation dynamics and oral arms on the exchange currents generated by the upside-down jellyfish *Cassiopea xamachana*. *Journal of Experimental Biology*, 214:1911–1921, 2011.

[26] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182–2189, 1965.

[27] T. Y. Hou and Z. Shi. An efficient semi-implicit immersed boundary method for the Navier-Stokes equations. *Journal of Computational Physics*, 227(20):8968–8991, 2008.

[28] IBAMR: An adaptive and distributed-memory parallel implementation of the immersed boundary (IB) method. Retrieved April 28, 2013 from `http://ibamr.googlecode.com`.

[29] M. C. Lai and C. S. Peskin. An immersed boundary method with formal second-order accuracy and reduced numerical viscosity. *Journal of Computational Physics*, 160(2):705–719, 2000.

[30] D. V. Le, J. White, J. Peraire, K. M. Lim, and B. C. Khoo. An implicit immersed boundary method for three-dimensional fluid-membrane interactions. *Journal of Computational Physics*, 228(22):8427–8445, 2009.

[31] L. Lee and R. J. LeVeque. An immersed interface method for incompressible Navier–Stokes equations. *SIAM Journal on Scientific Computing*, 25(3):832–856, 2003.

[32] R. J. LeVeque and Z. Li. Immersed interface methods for Stokes flow with elastic boundaries or surface tension. *SIAM Journal on Scientific Computing*, 18(3):709–735, 1997.

[33] P. McCorquodale, P. Colella, and H. Johansen. A Cartesian grid embedded boundary method for the heat equation on irregular domains. *Journal of Computational Physics*, 173(2):620–635, 2001.

[34] L. A. Miller and C. S. Peskin. Computational fluid dynamics of 'clap and fling' in the smallest insects. *Journal of Experimental Biology*, 208:195–212, 2005.

[35] M. L. Minion. On the stability of Godunov-projection methods for incompressible flow. *Journal of*

Computational Physics, 123(2):435–449, 1996.

[36] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annual Review of Fluid Mechanics*, 37:239–261, 2005.

[37] Y. Mori and C. S. Peskin. Implicit second-order immersed boundary methods with boundary mass. *Computer Methods in Applied Mechanics and Engineering*, 197(2528):2049–2067, 2008.

[38] Y. Morinishi, T. Lund, O. Vasilyev, and P. Moin. Fully conservative higher order finite difference schemes for incompressible flow. *Journal of Computational Physics*, 143(1):90–124, 1998.

[39] E. P. Newren. *Enhancing the immersed boundary method: Stability, volume conservation, and implicit solvers*. PhD thesis, Department of Mathematics, University of Utah, Salt Lake City, UT, May 2007.

[40] E. P. Newren, A. L. Fogelson, R. D. Guy, and R. M. Kirby. Unconditionally stable discretizations of the immersed boundary equations. *Journal of Computational Physics*, 222(2):702–719, 2007.

[41] E. P. Newren, A. L. Fogelson, R. D. Guy, and R. M. Kirby. A comparison of implicit solvers for the immersed boundary equations. *Computer Methods in Applied Mechanics and Engineering*, 197(25-28):2290–2304, 2008.

[42] C. S. Peskin. Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10:252–271, 1972.

[43] C. S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002.

[44] C. S. Peskin and B. F. Printz. Improved volume conservation in the computation of flows with immersed elastic boundaries. *Journal of Computational Physics*, 105:33–46, 1993.

[45] R. Rannacher. On Chorin's projection method for the incompressible Navier-Stokes equations. In J. Heywood, K. Masuda, R. Rautmann, and V. Solonnikov, editors, *The Navier-Stokes Equations II – Theory and Numerical Methods*, volume 1530 of *Lecture Notes in Mathematics*, pages 167–183. Springer, Berlin/Heidelberg, 1992.

[46] Y. Saad. *Iterative Methods for Sparse Linear Systems*, volume 620. PWS Publishing Company, Boston, 1996.

[47] J. Shen. On a new pseudo-compressibility method for the incompressible Navier-Stokes equations. *Applied Numerical Mathematics*, 21:71–90, 1996.

[48] J. Shen. Pseudo-compressibility methods for the unsteady incompressible Navier-Stokes equations. In *Proceedings of the 1994 Beijing Symposium on Nonlinear Evolution Equations and Infinite Dynamical Systems*, pages 68–78, 1997.

[49] J. M. Stockie. *Analysis and Computation of Immersed Boundaries, with Application to Pulp Fibres*. PhD thesis, Institute of Applied Mathematics, University of British Columbia, Vancouver, Canada, 1997. Available from `https://circle.ubc.ca/handle/2429/7346`.

[50] J. M. Stockie and B. R. Wetton. Analysis of stiffness in the immersed boundary method and implications for time-stepping schemes. *Journal of Computational Physics*, 154(1):41–64, 1999.

[51] R. Temam. Une méthode d'approximation de la solution des équations de Navier-Stokes. *Bulletin de la Société Mathématique de France*, 98(4):115–152, 1968.

[52] A.-K. Tornberg and M. J. Shelley. Simulating the dynamics and interactions of flexible fibers in Stokes flows. *Journal of Computational Physics*, 196:8–40, 2004.

[53] E. Twizell, A. Gumel, and M. Arigu. Second-order, L0-stable methods for the heat equation with time-dependent boundary conditions. *Advances in Computational Mathematics*, 6:333–352, 1996.

[54] WestGrid. QuickStart Guide to Bugaboo. Retrieved April 11, 2013 from `http://www.westgrid.ca/support/quickstart/bugaboo`.