AN ABSTRACT OF THE THESIS OF

Christopher Minar for the degree of Masters of Science in Mechanical Engineering presented on December 5, 2016.

Title: GPU-Based Fluid-Structure Interaction using Immersed Boundary Methods

Abstract approved: Kyle Niemeyer

Engineering applications often require fast, accurate solutions of fluid flow around freely moving bodies. The massive parallelism enabled by graphics processing unit (GPU) architecture enables high performance, offering a promising alternative to traditional solver acceleration via multicore central processing units (CPU). However, fully harnessing GPU parallelism requires specialized algorithms and computing strategies. This work modifies direct-forcing immersed boundary methods to model fluid-structure interaction and investigates this behavior on GPUs. I performed solver verification using lid-driven cavity flow, impulsively started flow over a cylinder, flow over a forced oscillating cylinder and vortex-induced vibration.
GPU-Based Fluid-Structure Interaction using Immersed Boundary Methods

by

Christopher Minar

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Masters of Science

Presented December 5, 2016
Commencement June 2017
Masters of Science thesis of Christopher Minar presented on December 5, 2016.

APPROVED:

____________________________
Major Professor, representing Mechanical Engineering

____________________________
Head of the School of Mechanical Industrial and Manufacturing Engineering

____________________________
Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

____________________________
Christopher Minar, Author
ACKNOWLEDGEMENTS

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K20 GPU used for this research. We also thank the Barba Group for developing, maintaining, and distributing their cuIBM code. Thank you to my committee, everyone in the Niemeyer research group, and the NNMREC lab.
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Numerical methods</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Navier–Stokes fractional-step</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Node identification</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Fadlun method</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Modified Fadlun method</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>Luo method</td>
<td>7</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Field extrapolation to ghost nodes</td>
<td>8</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Interpolation to hybrid nodes</td>
<td>11</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Weighting at hybrid nodes</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Implementation and discretization</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>Sparse matrices</td>
<td>14</td>
</tr>
<tr>
<td>3.2</td>
<td>GPU implementation strategy</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>Grid staggering</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Node identification</td>
<td>18</td>
</tr>
<tr>
<td>3.5</td>
<td>Force calculation</td>
<td>18</td>
</tr>
<tr>
<td>3.6</td>
<td>Linear algebra</td>
<td>20</td>
</tr>
<tr>
<td>3.7</td>
<td>Navier–Stokes equations</td>
<td>20</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Intermediate velocity</td>
<td>21</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Poisson equation</td>
<td>27</td>
</tr>
<tr>
<td>3.8</td>
<td>Modified Fadlun immersed boundary method</td>
<td>27</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Intermediate velocity</td>
<td>27</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Poisson equation</td>
<td>29</td>
</tr>
<tr>
<td>3.9</td>
<td>Luo immersed boundary method</td>
<td>29</td>
</tr>
<tr>
<td>3.9.1</td>
<td>External interpolation method</td>
<td>30</td>
</tr>
<tr>
<td>3.9.2</td>
<td>Embedded interpolation method</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>Verification and validation</td>
<td>35</td>
</tr>
<tr>
<td>4.1</td>
<td>Lid-driven cavity flow</td>
<td>36</td>
</tr>
<tr>
<td>4.2</td>
<td>Impulsively started cylinder flow</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Flow past an in-line oscillating cylinder</td>
<td>40</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>4.4</td>
<td>In-line oscillating cylinder</td>
<td>41</td>
</tr>
<tr>
<td>4.5</td>
<td>Vortex-induced vibration</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>Solver characterization</td>
<td>51</td>
</tr>
<tr>
<td>5.1</td>
<td>Order of accuracy</td>
<td>51</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Impulsively started cylinder flow</td>
<td>51</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Oscillating cylinder in flow</td>
<td>58</td>
</tr>
<tr>
<td>5.2</td>
<td>Performance</td>
<td>58</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Impulsively started cylinder</td>
<td>59</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Oscillating cylinder</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>Summary and recommendations for future work</td>
<td>65</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>67</td>
</tr>
<tr>
<td>Appendices</td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>A</td>
<td>Solving a system of four equations on the GPU</td>
<td>72</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>10</td>
</tr>
<tr>
<td>2.6</td>
<td>12</td>
</tr>
<tr>
<td>2.7</td>
<td>13</td>
</tr>
<tr>
<td>3.1</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>22</td>
</tr>
<tr>
<td>3.5</td>
<td>23</td>
</tr>
<tr>
<td>3.6</td>
<td>24</td>
</tr>
<tr>
<td>3.7</td>
<td>26</td>
</tr>
<tr>
<td>3.8</td>
<td>28</td>
</tr>
<tr>
<td>3.9</td>
<td>29</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES (Continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.10</td>
<td>33</td>
</tr>
</tbody>
</table>
| The stencil of the left hand side matrices change when doing the interpo-
| lations inside the linear algebra step.                                      |
| 4.1    | 36   |
| The domain of the lid-driven cavity flow simulation with boundary condi-
| tions and centerlines shown.                                                 |
| 4.2    | 37   |
| Verification for lid-driven cavity flow. The top row shows Re=100 and the 
<p>| bottom row shows Re=1000. The left column shows the $v$ velocity at the    |
| horizontal centerline and the right column has the $u$ velocity at the     |
| vertical centerline.                                                        |
| 4.3    | 38   |
| The domain and boundary conditions for the impulsively started cylinder   |
| simulation with Re=40. The embedded method is excluded because its         |
| results are the same as the external method.                                |
| 4.4    | 39   |
| Drag force over time for two solvers compared to Koumoutsakos and Leonard |
| [1] for flow over an impulsively started cylinder. The embedded method is   |
| excluded because its results are the same as the external method.           |
| 4.5    | 39   |
| Drag force over time for two solvers compared to Koumoutsakos and Leonard |
| [1] for flow over an impulsively started cylinder. The embedded method is   |
| excluded because its results are the same as the external method.           |
| 4.6    | 40   |
| Drag force over time for two solvers compared to Koumoutsakos and Leonard |
| [1] for flow over an impulsively started cylinder.                          |
| 4.7    | 42   |
| Drag force for flow over and in-line oscillating cylinder. The left column |
| shows external interpolation and the right column shows the embedded      |
| interpolation. Rows one though four have maximum CFL 0.35, 0.35, 0.7, and  |
| 1.0 and a middle grid spacing of 0.0625, 0.0313, 0.0313, and 0.0313.       |
| 4.8    | 43   |
| Drag force vs. time for the first period compared with steady state Dütsch |
| 4.9    | 43   |
| Drag force verses time for steady state oscillating cylinder in now flow   |
| compared to Dütsch et al. [2].                                               |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.10</td>
<td>Drag force over the 14th period for in-line oscillating cylinder flow with Re=200, KC=10 compared to Dütsch et al. [2]</td>
</tr>
<tr>
<td>4.11</td>
<td>External method loose coupling. The maximum amplitude vs. reduced velocity for the lock-in region.</td>
</tr>
<tr>
<td>4.12</td>
<td>External method strong coupling. The maximum amplitude vs. reduced velocity for the lock-in region.</td>
</tr>
<tr>
<td>4.13</td>
<td>Embedded method loose coupling. The maximum amplitude vs. reduced velocity for the lock-in region.</td>
</tr>
<tr>
<td>4.14</td>
<td>Embedded method strong coupling. The maximum amplitude vs. reduced velocity for the lock-in region.</td>
</tr>
<tr>
<td>5.1</td>
<td>Average error in drag force plotted against grid spacing, showing the order of accuracy for the three solvers.</td>
</tr>
<tr>
<td>5.2</td>
<td>Drag force vs. time for different grid spacings using the embedded solver.</td>
</tr>
<tr>
<td>5.3</td>
<td>Error in drag force vs. time using the embedded method with a center grid spacing of 0.01.</td>
</tr>
<tr>
<td>5.4</td>
<td>Drag force versus time with different grid sizes using the modified Fadlun method.</td>
</tr>
<tr>
<td>5.5</td>
<td>Order of accuracy calculated at multiple time steps before the discontinuities.</td>
</tr>
<tr>
<td>5.6</td>
<td>Impulsively started cylinder error norms.</td>
</tr>
<tr>
<td>5.7</td>
<td>Error plotted against grid spacing shows the two solvers’ order of accuracy for an oscillating cylinder in flow.</td>
</tr>
<tr>
<td>5.8</td>
<td>Poisson iterations taken each time step. From left to right: Impulsively started cylinder with external method, in-line oscillating cylinder in flow with external method, in-line oscillating cylinder in flow with embedded method.</td>
</tr>
<tr>
<td>5.9</td>
<td>Impulsively started cylinder performance.</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>5.10</td>
<td>Grid reduction that takes place inside of the linear multi-grid solver.</td>
</tr>
<tr>
<td>5.11</td>
<td>Performance for flow over an in-line oscillating cylinder.</td>
</tr>
<tr>
<td>5.12</td>
<td>Computational time for flow over an in-line oscillating cylinder. The total cell count and uniform grid section size are held constant.</td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>4.1 Which immersed boundary methods are tested with what simulations.</td>
<td>35</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction

Wave energy converters are seafaring devices that harvest waves to create electrical energy. In the Atlantic, the sea floor can be shallow enough to use pylons to support devices on the surface. A wave energy converter can use the relative motion of a buoy rolling in the waves and the anchored pylon to make electricity. In the Pacific, the sea floor is too deep to make use of the sea bed. Instead, some wave energy converters use a large underwater disk called a “heave plate”. The purpose of a heave plate is to make as much drag as possible when the buoy tries to pull it up through the water, creating an immobile base as possible to substitute for an anchor. To make wave energy converters cost competitive with established renewable and fossil fuels, they must be optimized to be as efficient as possible. Doing this requires a fast, accurate fluid flow solver capable of handling fluid structure interaction for complex bodies.

Modern CPUs typically have four to eight cores. Graphics processing units (GPUs) can have hundreds to thousands of simpler cores. The key to achieving high performance with a GPU is using them for highly parallel problems. A highly parallel problem is one where large quantities of data can be independently operated on at the same time. Many operations in computer graphics are parallel because each pixel can be calculated independently. Computational fluid dynamics presents another opportunity for GPU acceleration because when solving the fluid flow the Navier–Stokes equations must be discretized and solved simultaneously at many finite grid points. Most of the calculations required can assign one grid point to its own core.

A lot of recent work has focused on writing algorithms to work on GPU-accelerated hybrid clusters, or traditional supercomputers with GPUs available at each node [3, 4, 5]. This is sometimes referred to as GPU acceleration. On a cluster it makes sense to use both CPU and GPU because data passed between nodes using message passing interface must be transferred to the CPU. When there is only one node, such as in a workstation, the time it takes to transfer data between the CPU and GPU is often prohibitive. In addition, modern GPUs are approaching 10 TeraFLOPs compared to CPUs, which typically achieve 500 GigaFLOPs without over-clocking. The goal of this project is to develop a
fluid flow solver that only uses the GPU, called cuIBM-FSI [6]: CUda (Compute Unified Device Architecture, a programming language for NVIDIA GPUs) Immersed Boundary Method for Fluid-Structure Interaction. The eventual purpose of cuIBM-FSI is to have a highly efficient solver to study wave energy converters. cuIBM-FSI was forked from cuIBM, which was developed by the Barba group [7]. cuIBM is a GPU-based immersed boundary method solver primarily based on the method developed by Taira and Colonius [8]. It can not handle coupled fluid-structure interaction and produces numerical oscillations when working with moving bodies [9]. Since forking the original cuIBM, most of the discretization done in cuIBM has been rewritten for the new solvers but the code structure and i/o is largely the same. This thesis will showcase cuIBM-FSI and the development of simulating two-dimensional, incompressible flow over a freely moving body using a GPU.

Immersed boundary methods (IBMs) refer to a group of approaches used to simulate fluid flow over complex bodies, representing a body without using a body-fitted mesh. These techniques are well-suited for simulating flow involving complex, moving bodies because they do not require re-meshing between time steps. We have developed several immersed boundary method solvers for operating on, and investigating, graphics processing units (GPUs,) designed to handle incompressible fluid–structure interaction problems with rigid bodies.

Peskin [10] created the original immersed boundary method in the 1970s to model biological flows. He used the immersed boundary method to more easily represent the moving, elastic artery walls. Peskin’s immersed boundary method adds a forcing term to the Navier–Stokes equations that represents the force a body would apply to the fluid if it was physically present. This forcing term is modeled as a spring via \( f = kx \), and a Dirac-delta function ensures the force only acts upon fluid nodes near the body. Modeling the force as a spring worked well for Peskin’s purposes but proved to have several shortcomings. First, the order of accuracy heavily depends on how it and the delta function are handled. Second, to apply the spring model to rigid bodies, a large spring coefficient must be used which inadvertently makes simulations stiff.

Immersed boundary method techniques have since evolved to be applicable to a wide range of problems, as reviewed by Mittal and Iaccarino [11] and more recently Sotiropoulos and Yang [12]. The methods focused on in this work are all from a popular subgroup of immersed boundary method called direct forcing methods. Mohd-Yusof [13]
and Fadlun et al. [14] developed two early versions of direct forcing. Direct forcing methods interpolate for velocity at the nodes nearest to the body using the body velocity, eliminating the need to solve for a forcing term while simultaneously enforcing the no-slip condition. The early direct forcing methods did not include a method of modifying the Poisson equation for the presence of a body. Not accounting for the body in the Poisson equation violates conservation of mass at the body. Kim et al. [15] developed a direct forcing method that adds mass source or sink terms at nodes intersected by the boundary to satisfy the conservation equation.

Using the direct forcing method with a moving body can cause numerical oscillations [16, 17]. This effect can be manageable for preset motion, but with a freely moving body the solver will poorly predict body position, velocity, and forces. The numerical oscillations are caused by having different solution regimes in the domain, e.g., interpolation near the body and Navier–Stokes everywhere else. Luo et al. [17] deal with the numerical oscillations with a weighting function to transition between the two schemes, removing the discontinuity where the regimes transition.

The rest of thesis is structure as follows. Chapter 2 discusses the theory of direct forcing methods and their applicability to freely moving bodies. Chapter 3 covers the discretization of these methods as well as the challenges, strategies, and implementation for the GPU. Chapter 4 shows the verification and validation of different aspects of the solver using five different test cases: lid-driven cavity flow, flow over an impulsively started cylinder, forced motion of a cylinder in static flow, forced motion of a cylinder in flow, and vortex-induced vibrations (VIV). Chapter 5 characterizes the order of accuracy and performance scaling for the immersed boundary method approaches considered. The final chapter will summarize results, discuss issues encountered, and recommend future work.
Chapter 2: Numerical methods

The cuIBM-FSI solver is comprised of three separate methods: one modified from the work of Fadlun et al. [14], and two based off the work of Luo et al. [17]. For now, it will suffice to call these the modified Fadlun, external, and embedded methods and leave their explanations for later.

All of these methods, however, stem from the two-dimensional, incompressible form of the Navier–Stokes momentum and mass continuity equations:

\[
\frac{\partial \mathbf{u}}{\partial t} + \nabla (\mathbf{u} \cdot \mathbf{u}) = -\nabla p + \nu \nabla^2 \mathbf{u} \tag{2.1}
\]

\[
\nabla \cdot \mathbf{u} = 0, \tag{2.2}
\]

where \( \mathbf{u} \) is velocity, \( p \) is pressure, and \( \nu \) is the (constant) kinematic viscosity.

2.1 Navier–Stokes fractional-step

All three methods use the fractional-step approach [20] to solve Equations (2.1) and (2.2), which breaks up the discretization of the Navier–Stokes equations. Specifically, each discretized time step is comprised of three sub-steps. In the first sub-step, the momentum equations are solved without pressure to calculate an intermediate velocity:

\[
\hat{\mathbf{u}} - \frac{\Delta t}{2} L(\hat{\mathbf{u}}) = \mathbf{u}^n + \Delta t (\text{RHS}^n), \tag{2.3}
\]

where \( \hat{\mathbf{u}} \) is the intermediate velocity, \( \Delta t \) is the time step, \( L \) is the Laplacian operator, and \( \text{RHS} \) is the discretized advection and diffusion terms. The superscript \( n \) represents the time step corresponding to time \( t^n \). In time, second-order Adams–Bashforth and Crank–Nicolson methods are used to discretize the advection and diffusion terms, respectively. The implicit Crank–Nicolson method is used for the diffusion term because it is stable and the explicit Adams–Bashforth is used because it is difficult to discretize a non-linear term implicitly.
In the second sub step, the continuity equation is imposed to approximate the pressure, $p$, resulting in a Poisson equation:

$$\nabla^2 p^{n+1} = -\frac{\nabla \hat{u}}{\Delta t},$$  \hspace{1cm} (2.4)

where $n + 1$ indicates the value at the next time step. Velocity is updated in the last sub step with:

$$u^{n+1} = \hat{u} - \Delta t \nabla p^{n+1}.$$  \hspace{1cm} (2.5)

2.2 Node identification

To facilitate the discussion of immersed boundary methods, the nomenclature with regards to point identification will be explained here. The nomenclature is mostly adopted from the work of Luo et al. [17]. There are four types of nodes. Not all node types are used in each method, and the methods treat nodes differently. Nodes immediately outside the body are called hybrid nodes. In all three methods values at the hybrid nodes are interpolated for using values at the body and fluid nodes near the body. Everything outside of the body that is not a hybrid node is a fluid node. Fluid nodes are solved with the Navier–Stokes equations. Nodes immediately inside the body are ghost nodes and behave similar to Mittal et al. [11]. The Fadlun (and modified Fadlun) method does not use these, but the external and embedded methods both try to set the values at the nodes in a way that enforces the body’s boundary conditions; this will be discussed more in Section 2.5.1. Everything else (i.e., all nodes inside the body that are not ghost nodes) are solid nodes. The treatment of solid nodes does not affect the solution, but it does affect the computational time and force calculation. If solid nodes are solved, an arbitrary flow field will develop inside the body. Figure 2.1 shows an example body and grid with labeled nodes.

2.3 Fadlun method

Peskin’s original immersed boundary method [10] solved for a forcing term $f$ in Equation (2.1) at the hybrid nodes in a way that enforces the no-slip boundary condition. The
direct forcing methods developed by Fadlun et al. [14] and Yusof [13] do not solve the Navier–Stokes equation at the hybrid nodes. Instead, the velocity at the hybrid node is approximated by linearly interpolating between the second closest node and the body surface as seen in Figure 2.2. At this point a clever observer might notice a complication induced by the fractional step method: should no slip be enforced on the intermediate velocity in the first sub-step, or the velocity during the third sub-step? Fadlun et al. [14] found that when no-slip is applied during the intermediate time step, the error introduced
in the velocity correction step was always negligibly small compared to the velocity. RHS and \(1 - \frac{\Delta t}{2} L\) from Equation (2.3) can be set up as if there was no body, then modified at the hybrid nodes to enforce no slip with the linear interpolation. The Poisson Equation (2.4) and projection step in Equation (2.5) are left unmodified in the work of Fadlun et al. [14].

2.4 Modified Fadlun method

As will be described in more detail later (Section 4.2), the original Fadlun method proved mediocre at predicting force in the transient region (while the boundary layer was developing in the impulsively started cylinder simulation). This happens because the Poisson equation is not modified to account for the presence of a body, causing violations of the continuity equation. Depending on the type of simulation this can create significant creation or destruction of mass at the body, leading to poor solver accuracy. A number of methods have been proposed and tested to fix this issue. Kim et al. [15] added mass source or sink terms to the continuity equations at cells cut by the body surface. Mittal et al. [21] imposed no-slip by extrapolating velocity data outside the body to ghost nodes. Luo et al. [17] also extrapolated inwards in addition to enforcing an approximate pressure boundary condition. Here, cuIBM-FSI’s modified Fadlun method uses a cut-cell approach to fix the mass continuity issue. This is explained in more detail in Section 3.8.

2.5 Luo method

Direct forcing methods suffer from numerical oscillation when the immersed body is moving [12, 11, 16]. As the body moves, background nodes change how they calculate values. For example, Figure 2.3 depicts a bulk-fluid \(u\) velocity node at time step \(t^n\) becoming a hybrid node at time step \(t^{n+1}\). As the node transitions, the intermediate velocity will change from being calculated by Navier–Stokes (Equation (2.3)) to an interpolation (Equation (2.6)). Both equations are correct representations for the intermediate velocity, but they have different errors associated with them, which cause the calculated velocities to be slightly different. This discontinuous change in velocity is responsible for the numerical oscillations. This effect happens whenever any node transitions, not just
the intermediate $u$ velocity nodes.

![Diagram showing the interaction of an immersed body with the background grid causing numerical oscillations.](image)

**Figure 2.3:** A diagram illustrating the immersed body’s interaction with the background grid that causes numerical oscillations. The solid line represents the body at time $t^n$ and the dashed line represents the body at time $t^{n+1}$.

In the method proposed by Luo et al. [17], hybrid node values are calculated via both Navier–Stokes and interpolation. The final value at the hybrid node is a weighted combination of both solutions. The weighting is designed to smooth the transition between solutions. When the hybrid node is close to the body, it will be dominated by the interpolated solution, while a hybrid node far from the body will be dominated by the Navier–Stokes solution. This process is discussed more in Section 2.5.3. The field values (velocity and pressure) must be extrapolated across the body using appropriate boundary conditions to get a valid result at the hybrid nodes using the Navier–Stokes equations, discussed in Section 2.5.1.

### 2.5.1 Field extrapolation to ghost nodes

The Navier–Stokes equations can be used to calculate values at the hybrid nodes if the ghost nodes are modified to correctly represent the presence of the body. If the slope of velocity between the hybrid node and the body is assumed to be linear, then the slope can be extrapolated inwards to the ghost node. Figure 2.4 shows a simple example of this. The ghost node is set such that the velocity is zero via linear interpolation between
the ghost and hybrid nodes. When the Navier–Stokes equations are then solved, the fluid velocity at the body surface will be calculated as the velocity of the body, mimicking the no-slip boundary condition. In the original Luo et al. [17] method, field values are extrapolated across the body using 2D bilinear interpolation. Unlike the work of Luo et al. [17], which used a collocated grid, cuIBM-FSI uses a staggered grid, causing the extrapolation process to have three variations: one for pressure, one for \( u \) velocity nodes, and one for \( v \) velocity nodes. Values are first interpolated for at an image point, represented by triangles in Figure 2.5, outside the body using the surrounding field values and the boundary condition. The 2D bilinear interpolation approximates the field between four nodes surrounding the image point with Equation (2.6). Each node surrounding the image point is used to set up a system of four equations to solve for the coefficients of Equation (2.6), which can then be used to solve for the field values. Some of the interpolation nodes will be moved to account for the presence of the body. In Figure 2.5 the bottom left corner of each interpolation region is moved to the body intercept point.

The ghost nodes, indicated by solid squares, are projected onto the surface to find the body intercept, indicated by the thick open circle. Body nodes used to track the body’s position are not the same as the body intercept and the two are typically not coincident. If the body has curvature, the body intercept will not fall exactly on the body due to its discrete representation of the body. The line drawn between the body intercept and ghost node will be perpendicular to the tangent line at the body intercept, i.e., the ghost node is projected along the surface normal. That line is then mirrored over the surface to find the image point, indicated by the solid triangle.

The rules to determine which values are used to interpolate for the image point...
are as follows: If one of the four interpolation nodes is inside the body, as seen in the extrapolation for ghost node 1 in Figure 2.5, then that node is moved to the body intercept and the boundary condition is used. It is possible for multiple corners to be inside the body. Any corner inside the body will always be a ghost node and have its own, corresponding body intercept. All corners inside the body are moved to their corresponding body intercept such that there are always four corners in separate locations to use for the interpolation. If none of the four field values are inside the body as seen in the extrapolation for ghost node 2, then the node closest to the body intercept is moved to the body intercept. To satisfy the no slip condition at the body, a Dirichlet boundary condition equal to the body velocity is used when extrapolating for velocity ghost nodes. The pressure boundary condition is approximated by forcing the slope of pressure normal to the body surface, \( \frac{dp}{dn} \), as constant using Neumann boundary conditions, Equation (2.8). Using the relocated corners, a system of equations is set up to solve for the coefficients in Equation (2.6). Pressure nodes on the body use Equation (2.7) in the system, which
is simply the derivative of Equation (2.6):

\[
\phi(x, y) = a_0 + a_1 x + a_2 y + a_3 xy
\]  

(2.6)

\[
\phi(x, y) = a_1 + a_2 + a_3 (x + y)
\]  

(2.7)

\[
\frac{\partial p}{\partial n} \bigg|_{BI} = -\rho \frac{Du}{Dt} \cdot \hat{n} \bigg|_{BI},
\]  

(2.8)

where \( \phi \) is the variable being interpolated (\( u, v, \) or \( p \)); \( a_0, a_1, a_2 \) and \( a_3 \) are coefficients; \( x \) and \( y \) give the node location; \( \rho \) is the density; \( Du/Dt \) is the material derivative; and \( \hat{n} \) is the unit vector normal to the body. Solving this system on the GPU requires direct inversions of the 4 \( \times \) 4 matrices. Appendix A details this procedure. Once the \( a \) coefficients have been found, the field value at the image point can be calculated with Equation (2.6) and extrapolated across the surface using Equation (2.9) for velocity or (2.10) for pressure:

\[
u_{GN} = 2u_{BI} - u_{IP}
\]  

(2.9)

\[
p_{GN} = p_{IP} - \Delta l \frac{\partial p}{\partial n} \bigg|_{BI},
\]  

(2.10)

where the subscripts GN, BI, and IP indicate ghost nodes, body intercept, and image point, and \( \Delta l \) is the distance from ghost node to image point.

2.5.2 Interpolation to hybrid nodes

Interpolating for the field value at the hybrid node is largely the same as interpolating for the ghost nodes’ image point described previously. The body intercept is once again found by projecting the hybrid node along the surface normal. The image point is not used in the same way as the ghost node image point. If a given body intercept is located at \((x_{BI}, y_{BI})\) and its hybrid node is located at \((x_{BI} + \Delta x, y_{BI} + \Delta y)\), the corresponding image point is located at \((x_{BI} + 2\Delta x, y_{BI} + 2\Delta y)\) as shown in Figure 2.6. In the hybrid node case, an image points purpose is locating the four interpolation nodes, one of which will always be the corresponding hybrid node. The interpolation node coincident with the hybrid node is moved to the body intercept and the appropriate boundary condition for velocity or pressure is applied. Once Equation (2.6) is solved, the hybrid node is
interpolated for.

```
\[ \theta = (1 - \alpha) \theta_{\text{Navier-Stokes}} + \alpha \theta_{\text{Interpolated}}, \]  
\text{(2.11)}
```

where \(\theta\) represents either intermediate velocity or pressure. Transitioning between the Navier–Stokes and interpolated solutions should meet three criteria:

1. Hybrid nodes farther from the body should favor the Navier–Stokes solution (\(\alpha \to 0\) as distance \(\uparrow\)).
2. Hybrid nodes closer to the body should favor the interpolated solution (\(\alpha \to 1\) as distance \(\to 0\)).
3. The weighting function should be smooth and continuous as the hybrid node moves away from the body.

To solve for $\alpha$, it is assumed that each hybrid node has at most two neighboring ghost nodes (this will not be true for sharp corners):

$$\alpha = \sqrt{\left(\frac{\Delta_1}{\Delta x}\right)^2 + \left(\frac{\Delta_2}{\Delta y}\right)^2},$$

(2.12)

where $\Delta_1$ and $\Delta_2$ correlate to $\text{GN}_1$ and $\text{GN}_2$, respectively, and $\Delta x$ and $\Delta y$ are the grid spacing in the $x$ and $y$ directions, respectively. As described by Luo et al. [17] and shown in Figure 2.7, $\Delta_1$ and $\Delta_2$ are the closest distances between the body and the $x$ and $y$ ghost nodes, respectively. If the hybrid node only has one neighbor, then $\Delta = 0$ for the other direction.

![Figure 2.7: Schematic of the calculation of $\alpha$ for u velocity nodes.](image-url)
Chapter 3: Implementation and discretization

This chapter covers the discretization of the Navier–Stokes equations and all three immersed boundary methods. It also covers various details required to implement or understand the methods including sparse matrices, GPU strategies, grid staggering, node identification, force calculation, and linear algebra.

3.1 Sparse matrices

When using a dense matrix, every possible element of the matrix has a physical location in memory. The matrix that represents the coefficients for intermediate velocity and pressure are comprised of mostly zeros, i.e., it is sparse. It is impractical to store the entirety of a sparse matrix in memory. cuIBM-FSI stores sparse matrices in the COO (coordinate list) format. A COO matrix stores each entry as three values: row, column, and value. CUSP represents each COO matrix as three separate arrays. On a CPU each node is stepped through in order and its values are added to the next available location in the COO arrays. When adding values to a sparse matrix with a GPU, each node much have a predetermined location or a race condition would occur. Nodes at the corners, sides, and center of the domain have three, four, and five matrix entries, respectively. One of each neighboring node and one for the center node. The total number of entries is calculated before anything is added to the sparse matrix. The nodes are from left to right, top to bottom. In the kernel (a kernel is a fancy name for a GPU function) that generates the COO matrix, each node is handled by a single thread. When the GPU is adding values to the matrix, each thread has a predetermined location for its entries based on where the node is in the domain.

3.2 GPU implementation strategy

cuIBM-FSI is developed for full GPU operation to improve performance relative to CPU-based algorithms or partial GPU-accelerated approaches which involve significant CPU-
GPU communication. To this end, several overarching strategies have been implemented. First, all calculations are performed on the GPU, and all data is stored on the GPU. Typically, efforts are taken to avoid thread divergence when writing code for a GPU. I found it favorable to write kernels with occasionally poor thread-parallel performance, i.e., kernels with significant divergence, rather than copying data to the host to do calculations to reduce divergence. Increasing the parallel performance of kernels has little effect on the overall solver performance, because solving the Poisson equation typically takes more than 90% of the total computational time. In the cases that were tested, transferring data back and forth took significantly more time than would be saved by running some operations on a CPU. In addition to non-Poisson kernels being a low overall portion of the run time, the effect of the divergence turns out to be small. For most kernels, the divergence occurs in cells near the body because differentiating between hybrid, ghost, fluid, and solid nodes requires if statements. Divergence happens due to the hardware operating inside of a GPU. A GPU is made up of some number of logic units, each of which controls some number arithmetic units. The number of logic units is determined by the GPU model, and current NVIDIA GPU architecture has 32 arithmetic units. Ideally, the logic unit is able to send the exact same command to each arithmetic unit. Divergence occurs when the logic unit has to send a command to some of arithmetic units while making the other ones wait. This happens, for instance, during an if-then statement. Figure 3.1 shows an example body with hybrid nodes shaded light gray and ghost nodes shaded dark gray. Cells edges are represented by the dashed lines and the warps are represented by the solid lines. For simplification, each row is depicted as being part of a warp, with the remainder of the warps being made of cells to the right and left of the shown domain. This is the worst situation for divergence: the most possible differences in as few warps as possible. The fluid and solid nodes are all treated the same and each only takes one pass with the logic unit. Assuming the worst case scenario—each ghost and hybrid node requires its own pass—the total number of passes for each row is 1, 4, 6, 4, 4, 4, 6, 4, and 1. Each additional pass that each warp has to make can be considered the same as adding 32 more nodes to the simulation. In the example so far, the 320 cells of the ten warps partially shown would take approximately 1200 cells worth of computational time. A factor of four slowdown is bad but the number of cells near the body is very small compared to the overall simulation. In the first test discussed in Chapter 4, the cells near the body account for approximately 10% of the
Figure 3.1: Nodes near the body cause divergence. Dark gray represent ghost nodes and light gray represent hybrid nodes. Each row of cells comprises part of a warp.

Data transfer time is particularly relevant when moving from a stationary body to a moving body. When simulating a stationary body, several calculations only need to be performed once: the left-hand side matrices of Equations (2.3) and (2.4), node identification, and preconditioners. When simulating a moving body, these values need to be recalculated every time step. If arrays need to be transferred back and fourth between the CPU and GPU to recalculate those every time step a lot of time is wasted. The transferring can take as long as the Poisson equation solve (typically by far the longest step).

The third strategy comes from Layton et al. [7]. When using the CUSP library to perform the multigrid method, creating the preconditioner can take as much time as solving the Poisson equation. In a moving-body simulation, the preconditioner is normally recalculated each time step, but Layton et al. [7] found it possible to skip some time steps without loss of fidelity. Here, preconditioner is recalculated only if the solution
of the previous time step required more than 100 iterations. The external method does not need the preconditioner to be remade, even when the body is moving, because each node always depends on the same neighbors regardless of the position of the body.

3.3 Grid staggering

A collocated grid stores all variable information at the center of each cell. When discretized on a collocated grid, the Navier–Stokes equations will yield an odd-even decoupling of pressure and velocity resulting in a checkerboard pattern, unless special steps are taken to prevent it. Using a staggered grid is a relatively straightforward way to avoid odd-even decoupling. A staggered grid stores scalar values (pressure, temperature, density etc.) at the cell centers and velocities at the cell faces. Staggered grids also have the advantage of not requiring a pressure boundary condition. cuIBM-FSI is staggered in the positive $x$ and $y$ directions. That is, if the cell center is denoted as $i$ as shown in Figure 3.2, the top and right faces are also denoted as $i$ and the bottom and left faces are $i - n_x$ and $i - 1$ where $n_x$ is the number of cells that comprise the width of the domain. It is worth noting that Luo et al. [17] used a collocated grid. Due to the sensitivity of immersed boundary methods near the body, this may cause some differences.

![Figure 3.2: Grid staggering: The velocities are staggered in the positive $x$ and $y$ directions.](image-url)
3.4 Node identification

Nodes are identified using a ray-tracing algorithm described in O’Rourke [22]. The implementation is based on the CPU implementation of the original cuIBM [19]. One thread is generated for each node, be it \( u \), \( v \), or \( p \) and this thread performs the following actions to find rays in the \( x \) direction. First, the Lagrangian body nodes are looped through to make segments with their nearest neighbors. If the top of this segment is above the node and the bottom of the segment is below the node, then the \( x \) location of the body at the height of the node is found (open circle in Figure 3.3). The node is then tested for proximity to the body in the \( x \) direction. To determine what the node is, five \( x \)-coordinates are used: \( x_{i-1}, x_i, x_{i+1}, x_{BL}, \) and \( x_{body\ center} \). Using Figure 3.3 for example, the body intercept will test as being greater than the body center, greater than \( x_{i-1} \), and less than \( x_i \), which makes \( u_{i,j} \) a hybrid node.

It is possible for a hybrid or ghost node to be unable to find the body when searching in the \( x \) direction, e.g., if the node is above the highest body node. After the algorithm looks for intersections in the \( x \) direction, it searches in the \( y \) direction. In the case where a node is able to find the body in both the \( x \) and \( y \) directions, the \( x \) direction is always chosen. The \( v \) velocity and pressure nodes are found in the same way as described above, but using different coordinates.

After all the points have been tagged, the solid nodes are set to an arbitrary value so they can be easily found later. This step is not critical for the solution of the Navier–Stokes equations, but is needed for the force calculation, which disregards everything inside the body, and can be convenient when visualizing the solution.

3.5 Force calculation

The drag and lift on the body are calculated using the control volume approach detailed by Lai and Peskin [23]. The implementation of this approach is largely the same as the original cuIBM [7]. Lai and Peskin give Equation (3.1) to calculate the drag force:

\[
F_D = -\int_{\partial \Omega_0} \rho u_i (\mathbf{u} \cdot \mathbf{n}) ds - \int_{\partial \Omega_0} p n_i ds + \int_{\partial \Omega_0} \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) n_j ds - \frac{\partial}{\partial t} \int_{\Omega_0} \rho u_i dx, \quad (3.1)
\]
Figure 3.3: To find the hybrid and ghost nodes the position of the body must be translated from the Lagrangian body nodes to the Eulerian grid. The solid line is the real body which is represented by the Lagrangian body nodes (solid circles). The dashed lines are drawn between the body nodes and represent the discretized body.

where $\Omega_0$ is any square sub-domain place around the body and $n_i$ and $n_j$ are the normal vectors in the $x$ and $y$ directions. Once discretized, simplified and non-dimensionalized the terms can be rearranged into three groups: $F_{x,x}$, $F_{x,y}$ and $F_{x,u}$. The first term, $F_{x,x}$, represents the forces in the $x$ direction from the left and right of the control volume and is formed from the second and third integrals in Equation (3.1). $F_{x,y}$ is the forces in the $x$ direction from the top and bottom of the control volume; it comes from the first and third integral. The fourth integral becomes $F_{x,u}$, the $x$ force from the unsteady term and includes everything inside of the control volume. These terms are partially discretized in Equations (3.2), (3.3), and (3.4):

$$F_{x,x} = -\sum_{\Omega_y} \left( (p_e - p_w) + (u_e^2 - u_w^2) - \nu \left( \frac{du}{dx} \right)_{e} - \nu \left( \frac{dv}{dx} \right)_{w} \right) dy \quad (3.2)$$

$$F_{x,y} = -\sum_{\Omega_x} \left( 0.5(u_nv_n - u_sv_s) - 0.25\nu \left( \frac{du}{dy} \right)_{n} + \frac{dv}{dy} \right)_{n} - \nu \left( \frac{dv}{dy} \right)_{s} - \nu \left( \frac{du}{dy} \right)_{s} \right) \quad (3.3)$$

$$F_{x,u} = \frac{u_{i+1}^n - u_i^n}{dt} \quad (3.4)$$

The subscripts $n,e,s,w$ represent the north, east, south, and west faces, respectively, of the square domain $\Omega_0$. In practice, varying the size of $\Omega_0$ from $L = 4$ to $L = 10$ was not found to have a significant effect on the force or computable time. The body also does not have to be at the exact center of the domain. Given this, the control volume
is generally kept in the same location as the body moves around inside it. Typically, a square with sides $L = 10$ is used for a body of size $D = 1$ that moves about $D$ away from center.

If the solid and ghost nodes are not at steady state, Equation (3.1) will predict force poorly. To circumvent this effect over multiple solvers, all nodes inside the body are excluded from the unsteady term.

### 3.6 Linear algebra

CUSP (CUDA sparse) is the linear algebra library used by cuIBM-FSI. CUSP can handle many different types of sparse data structures and has several solvers. cuIBM-FSI uses a bi-conjugate gradient stabilized (BiCGSTAB) method for both the intermediate velocity and Poisson equation. Preconditioners are used to improve performance. A diagonal preconditioner is used for the intermediate velocity and a smoothed aggregation for the pressure step. Smoothed aggregation is a type of multi-grid approach to linear algebra that can drastically speed up convergence. In a multi-grid method, several iterations of BiCGSTAB (or any convergence method) are done on the full grid, then the solution is transferred to a coarser grid. This process is repeated until the grid is very coarse and then the solution is rebuilt on increasingly finer grids until a solution has been obtained for the full grid. Approximating the inverse of the matrix using the coarser grids saves computational time.

### 3.7 Navier–Stokes equations

The 2D, incompressible Navier–Stokes equations form the backbone of the immersed boundary method. The fractional step method is used to solve these equations. In the first step, Equation (3.5) drops the pressure term and is discretized to get an intermediate velocity [20]:

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla (\mathbf{u} \mathbf{u}) = -\nabla p + \nu \nabla^2 \mathbf{u}$$  \hspace{1cm} (3.5)

In the second sub step, Equation (2.4) is solved for pressure. In the third sub step, the final velocity is updated. This section goes over the discretization details of using fractional step for the standard Navier–Stokes equations. Sections 3.8, 3.9.1, and 3.9.2
cover the modifications required for the presence of an immersed boundary with the modified Fadlun, external, and embedded methods.

3.7.1 Intermediate velocity

The advection term, $\nabla (uu)$, is discretized in time using explicit, second-order Adams–Bashforth. The diffusion, $\nu \nabla^2 u$, is discretized using Crank–Nicolson. The result is an expanded form of Equation (2.3):

$$\hat{u} - \frac{\Delta t}{2} L(\hat{u}) = u^n + \Delta t \left( 0.5L(u^n) - 1.5N(u^n) + 0.5N(u^{n-1}) + BC \right),$$

(3.6)

where $N$ is the advection operator, $L$ is the diffusion Laplacian operator, and $BC$ is the boundary condition term (the leftover terms from applying the Laplacian to $\hat{u}$). Each term ($N$, $L$, $BC$, and $LHS$) is described in its own section below.

3.7.1.1 Advection terms

The advection term, $\nabla (uu)$, expands to become Equations (3.7) and (3.8), which are in the $x$-momentum and $y$-momentum equations, respectively. Due to the staggered grid, the discretizations of Equations (3.7) and (3.8) are slightly different, but for brevity only the discretization of $x$-momentum will be shown. Note that $u$ and $v$ represent scalar velocities but $u$ is a vector.

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y}$$

(3.7)

$$u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y}$$

(3.8)
Discretizing Equation (3.7) on a uniform grid using second order central differences results in Equation (3.9):

\[
\frac{u}{\partial x} + v \frac{\partial u}{\partial y} = \frac{u_{i+1,j} - u_{i-1,j}}{2dx} + \frac{v_{i,j} + v_{i+1,j} + v_{i,j-1} + v_{i+1,j-1}}{4} \frac{u_{i,j+1} - u_{i,j-1}}{2dy},
\]

(3.9)

\[
\frac{du}{dx}\bigg|_i \approx \frac{du}{dx}\bigg|_1 + \frac{du}{dx}\bigg|_2 - \frac{du}{dx}\bigg|_0 0.5\left(dx_i + dx_{i+1}\right) dx_i
\]

(3.10)

Figure 3.4a shows a section of uniform grid for reference with Equation (3.9). A non-uniform grid requires a more involved discretization, shown in (3.10) and (3.13). If \(\Delta x_{i+1}\) does not equal \(\Delta x\) then \(\frac{\partial u}{\partial x}\) must be calculated at the cell centers, indicated by the open circles marked 1 and 2 in Figure 3.4b, then interpolated to \(u_{i,j}\).

In addition, \(v\) must be bilinearly interpolated for at \(u_{i,j}\) on a non-uniform grid. Figure 3.5 shows a schematic of bilinear interpolation. First, \(v\) at the nodes marked 1 and 2 in Figure 3.4b are linearly interpolated for using Equations (3.11) and (3.12).
Then, $v_1$ and $v_2$ are used to linearly interpolate for $v$ at $u_{i,j}$ using Equation (3.13).

$$v_1 \approx \frac{x_2 - x}{x_2 - x_1} v_{i,j} + \frac{x - x_1}{x_2 - x_1} v_{i+1,j}$$  \hspace{1cm} (3.11)$$

$$v_2 \approx \frac{x_2 - x}{x_2 - x_1} v_{i,j-1} + \frac{x - x_1}{x_2 - x_1} v_{i+1,j-1}$$  \hspace{1cm} (3.12)$$

$$v_x \approx \frac{y_2 - y}{y_2 - y_1} v_1 + \frac{y - y_1}{y_2 - y_1} v_2$$ \hspace{1cm} (3.13)$$

Figure 3.5: Non-uniform bi linear interpolation. The dashed lines are reference lines and not indicative of cell faces.

### 3.7.1.2 Diffusion terms

The diffusion term, $\nu \nabla^2 u$, expands to become Equations (3.14) and (3.15) for the $x$ and $y$ momentum equations, respectively:

$$\frac{1}{Re} \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$  \hspace{1cm} (3.14)$$

$$\frac{1}{Re} \left( \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right),$$  \hspace{1cm} (3.15)$$
where $\nu$ can be simplified to the inverse of Reynolds number (Re) as long as the characteristic velocity is one. As with advection, the discretizations for Equations (3.14) and (3.15) are different because they are computed at the $u$ and $v$ nodes. The discretization is also more complex when done on a non-uniform grid. Equation (3.14) becomes (3.16) when discretized over a uniform grid:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{\Delta^2}$$

(3.16)

To discretize over a non-uniform grid like that of Figure 3.6, the terms in Equation (3.14) are expanded. For example, $\frac{\partial^2 u}{\partial x^2}$ will become $\frac{\partial}{\partial x} (\frac{\partial u}{\partial x})$. The partial derivative of $u$ with respect to $x$ is calculated at the cell centers adjacent to $u_{i,j}$ and the partial derivative of that is taken with respect to $x$ at $u_{i,j}$. Only the discretization of $\frac{\partial^2 u}{\partial x^2}$ is shown in Equation (3.17). Non-uniform discretization of $\frac{\partial^2 u}{\partial y^2}$ follows the same pattern.

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - u_{i,j}}{dx_i(dx_{i+1} + dx_i)0.5} + \frac{u_{i-1,j} - u_{i,j}}{dx_i(dx_{i+1} + dx_i)0.5}$$

(3.17)
3.7.1.3 Boundary terms

Special treatment of the advection and diffusion terms is required at the boundaries. Problems encountered at the boundary will be discussed using the discretization of the advection and diffusion terms from the $u$-momentum equation, Equations (3.7) and (3.14). Figure 3.7 shows the grid relevant to discretization at the north-east corner. The problems associated with being at the domain’s edge are all based on trying to access values that are out of bounds. This issue pops up with several flavors. The first is trying to access a velocity value on the boundary edge. For example, if $\frac{\partial u}{\partial x}$ is calculated at $u_{i,j}$ using the standard second-order central differencing, $u_{i+1,j}$ is required but it falls outside the domain. The solver uses Dirichlet and convective velocity boundary conditions. If an edge uses a convective boundary condition, that velocity gets calculated before the time step starts, so that the rest of the solver can treat it as a Dirichlet boundary condition. When the solver encounters a value it needs on the boundary edge, it pulls a velocity value for that edge from a separate array.

Boundary arrays store values at the edge of the domain. The second type of issue occurs when the required velocity value is outside the boundary. For example if $\frac{\partial^2 u}{\partial y^2}$ from the diffusion term is calculated at $u_{i,j}$, it needs a value from $u_{i,j+1}$, which is outside the boundary. Outlying velocities are approximated by assuming the derivative of velocity is constant at the boundary. The linear extrapolation in Equation (3.18) simplifies to (3.19), a relation for the outlying velocity in terms of known values:

\[
\begin{align*}
    u_{bc} &= \frac{u_{i,j+1} - u_{i,j}}{dy_j}0.5dy_j + u_{i,j} \\
    u_{i,j+1} &= 2u_{bc} - u_{i,j}
\end{align*}
\]

Uniform and stretching grids both place the outlying node $0.5dx$ or $0.5dy$ away from the domain edge to satisfy Equation (3.18).

3.7.1.4 Left-hand side matrix

The left hand side of the intermediate velocity Equation (2.3) is restated here for reference:

\[
\dot{u} - \frac{\Delta t}{2}L(\dot{u})
\]
The discretization of the Laplacian term is the same as described in the diffusion Section (3.7.1.2). Boundary terms are approximated by assuming $\hat{u} = u$. All the boundary terms become known values using this assumption, so they are moved to the right side of the equation. Because both the $u$ and $v$ velocities are stored in the same array, the left-hand side matrix will be size $(2((n_x - 1)n_y + (n_y - 1)n_x))^2$, where $n_x$ and $n_y$ represent the number of cells in the $x$ and $y$ directions, respectively. The “-1” is a by-product of staggering the grid. Each row in the matrix corresponds to one velocity node. Most of the rows will have five non-zero columns: one for the neighboring nodes to the north, east, south, and west and one for the center node. Rows that represent a velocity node on the edge of the domain will only have four non-zero columns, and rows that represent a corner velocity will only have three.
3.7.2 Poisson equation

The Poisson equation is the most computationally intensive step, typically taking more than 90% of the total run time. If Equation (2.4) is discretized, a pressure boundary condition is needed. Instead, the velocity reconstruction equation, (2.5), and the continuity equation, (2.2), are combined. At a non-boundary node the two equations simplify to become the Poisson equation. When at a boundary, the segment that would use a pressure term outside the domain is replaced by \( u^{n+1} \), eliminating the need for a pressure boundary condition.

The discretized continuity Equation (3.21) is shown alongside a discretized velocity reconstruction for \( \hat{u}_{i+1,j} \), Equation (3.22). The other velocities used in the continuity equation are discretized in the same way. In the reference Figure 3.8, \( u^{n+1}_{i,j} \) is not substituted for. Instead, the Dirichlet boundary condition velocity is used:

\[
\frac{u^{n+1}_{i,j} - u^{n+1}_{i-1,j}}{dx_i} + \frac{v^{n+1}_{i,j} - v^{n+1}_{i,j-1}}{dy_i} = 0
\]  

(3.21)

\[
u^{n+1}_{i,j} = \bar{u} - \frac{p^{n+1}_{i+1,j} - p^{n+1}_{i,j}}{dx_i + dx_{i+1}} 2\Delta t
\]

(3.22)

3.8 Modified Fadlun immersed boundary method

When at an immersed body, the Navier–Stokes equation needs some additional treatment to correctly represent a body. This section details the implementation of my modified Fadlun et al. [14] approach to dealing with an immersed boundary.

3.8.1 Intermediate velocity

The interpolation that enforces the no-slip condition happens in the intermediate velocity step. To solve for intermediate velocity, the right hand side is first computed as for no body as described in Section 3.7.1. Then, the hybrid nodes are replaced with interpolation values. Equation (3.23) gives an example interpolation that corresponds
Figure 3.8: A discretization of the Poisson equation at the eastern boundary
to Figure 3.9:

\[
[1] u_{i+1,j} + \left[ -\frac{b}{a} \right] u_{i,j} = \left( 1 - \frac{b}{a} \right) u_{\text{body}}
\]

(3.23)

The coefficients inside the brackets are the values that will be placed in the left-hand side matrix and the term on the right side of the equation will replace the value from Equation (3.6). The first term can be any of the hybrid node’s neighbors and is determined by the location of the body. The direction of interpolation is chosen by the point-identification algorithm before the time step. As a rule of thumb, the interpolation will be along the \(x\) direction if the body’s tangent line is vertical and be along the \(y\) direction if the tangent line is horizontal. The algorithm prefers to interpolate along the \(x\) direction; it only interpolates along \(y\) if it can not interpolate along the \(x\). Figure 3.9 shows an example of the linear interpolation at the boundary. Equation (3.23) shows the interpolation reworked to fit into established matrices used to solve for intermediate velocity. The terms inside the brackets represent the left-hand side matrix. It proved to be somewhat difficult to write a GPU kernel to edit the sparse left side matrix to incorporate the interpolation. Instead of creating the left-hand side matrix for Navier–Stokes
then editing it for the immersed boundary method as was done with the right side, a separate kernel was written.

Figure 3.9: Linear interpolation to enforce no-slip condition at the boundary.

3.8.2 Poisson equation

Fadlun et al. [14] did not modify the Poisson equation. When testing their method with an impulsively started cylinder, I found that it failed to accurately predict drag in the transient regime. By modifying the intermediate velocity at the boundary without compensating, the continuity equation is not satisfied. Having a mass source or sink at each hybrid node drastically affects the accuracy and stability. My modification to the Fadlun method handles this problem by adding a treatment to the Poisson equation in a manner akin to a sharp interface method [12]. At a hybrid node, if any term from Equation (3.22) is inside the boundary it is assumed that $u_{i,j}^{n+1}$ is close enough to the body to approximate it as the body velocity. If the body passes through the cell, $dx_i$ and/or $dy_j$ from Equation (3.21) are decreased to appropriately measure the new cell size.

3.9 Luo immersed boundary method

cuIBM-FSI has two methods based off the approach given by Luo et al. [17]. Luo et al. [17] extrapolates values to ghost nodes an interpolates values at hybrid nodes. They
left out details describing exactly where all this interpolating and extrapolating takes place. I tried doing it two ways. The key difference between the two approaches is when the extrapolation and interpolation takes place. The “external” method moves the interpolation and extrapolation outside of the linear algebra. Separating the extrapolation, interpolation, and linear algebra steps causes each one to not have access to the most updated information when being calculated because they all depend on each other. Simplifying the linear algebra this way reduces the computational time at the cost of reduced accuracy and increased numerical oscillation.

The “embedded” method performs interpolations, extrapolations, and weighting inside of the linear algebra solutions for the intermediate velocity and pressure. All of the extrapolation and interpolation happens inside of the linear algebra steps. Field values will solved by plugging Equations (2.3) or (2.4) and (2.6) into Equation (2.11). The discretization of these can be seen in Section 3.9.2.

Two key implementation details distinguish cuIBM-FSI from Luo et al. [17]. First, Luo et al. did not use a staggered grid. Second, Luo et al. used Crank–Nicholson to discretize both the advection and diffusion terms, whereas cuIBM-FSI uses Adams–Bashforth for the advection terms.

### 3.9.1 External interpolation method

The external interpolation method tries to resolve the time step in a non-iterative manner by calculating the interpolations, extrapolations and weighting outside of the linear algebra procedures. Before a value is used by the fractional-step calculations, it is calculated at the hybrid and ghost nodes. For example, the equation for intermediate velocity, Equations (2.3), depends on the velocity field, so $u$ and $v$ are interpolated for at the hybrid nodes and extrapolated for at the ghost nodes before the right-hand side matrix is calculated. The following sequence is set up such that a value is interpolated or extrapolated just before it is needed for the subsequent step:

1. Identify all nodes as hybrid, ghost, fluid, or solid.
2. Extrapolate $u$ and $v$ velocity values to ghost nodes.
3. Interpolate for $u$ and $v$ velocities at hybrid nodes.
4. Calculate the right-hand side terms for the intermediate velocity solution.

5. Solve for intermediate velocity.

6. Weight the two intermediate velocity solutions.

7. Extrapolate intermediate velocities to ghost nodes.

8. Calculate the right-hand side terms for the Poisson equation.

9. Solve the Poisson equation.

10. Interpolate for pressure at hybrid nodes.

11. Weight the two pressure solutions.

12. Extrapolate pressure to ghost nodes.

13. Update velocity.


3.9.2 Embedded interpolation method

In the embedded interpolation method, the Navier–Stokes and interpolated solutions for field values are calculated and weighted inside each linear-algebra solver. In the intermediate velocity substep, the right hand side terms are first calculated as if no body was present. The weighting coefficients and part of the interpolation can be calculated prior to the linear algebra because they are only dependent on the geometry, not the velocity or pressure. The equations for bilinear interpolation, Equation (2.6), and interpolating to the ghost node, Equations (2.9) or (2.10), are combined then rearranged to separate field value terms. A similar operation happens for the hybrid nodes. To illustrate the process of separating these variables, \( u \) is plugged in for \( \phi \) in Equation (3.29):

\[
u(x, y) = a_0 + a_1x + a_2y + a_3xy.
\]

(3.24)

Each bilinear interpolation requires four data points to solve. The method for placing these four points is discussed in Chapter 2. They will be labeled roughly according to
their cardinal direction to the interpolation location. The resulting system of equations is laid out below in Equation (3.25) and matrix form (3.26):

\[
Aa = u \quad \text{(3.25)}
\]

\[
\begin{bmatrix}
1 & x_{sw} & y_{sw} & x_{sw}y_{sw} \\
1 & x_{se} & y_{se} & x_{se}y_{se} \\
1 & x_{nw} & y_{nw} & x_{nw}y_{nw} \\
1 & x_{ne} & y_{ne} & x_{ne}y_{ne}
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3
\end{bmatrix}
= 
\begin{bmatrix}
u_{sw} \\
u_{se} \\
u_{nw} \\
u_{ne}
\end{bmatrix}.
\quad \text{(3.26)}
\]

Equation (3.27) gives the exact inverse of matrix \(A\):

\[
A^{-1} = \frac{B}{\det A} \quad \text{(3.27)}
\]

\[
a = A^{-1}u. \quad \text{(3.28)}
\]

Matrix \(B\) is based on the values of \(A\). The inverse is used to solve for the \(a\) coefficients, Equation (3.28). The external method stops here because all the \(u\) values are considered known. The embedded method moves, rearranges, and combines with the ghost node interpolation to become Equation (3.29):

\[
u_{GN} = 2u_{BI} + C_{nw}u_{nw} + C_{se}u_{se} + C_{sw}u_{sw} + C_{se}u_{se} \quad \text{(3.29)}
\]

The array \(C\) holds the velocity coefficients resulting from (3.28) being manipulated. Subscripts indicate the row and column of the matrix. The calculation of \(C\), Equation (3.30), is the portion of the bilinear interpolation that is performed before the linear algebra step:

\[
\begin{bmatrix}
C_{sw} \\
C_{se} \\
C_{nw} \\
C_{ne}
\end{bmatrix} = 
\begin{bmatrix}
b_{11} + b_{21}x_{IP} + b_{31}y_{IP} + b_{41}x_{IP}y_{IP} \\
b_{12} + b_{22}x_{IP} + b_{32}y_{IP} + b_{42}x_{IP}y_{IP} \\
b_{13} + b_{23}x_{IP} + b_{33}y_{IP} + b_{43}x_{IP}y_{IP} \\
b_{14} + b_{24}x_{IP} + b_{34}y_{IP} + b_{44}x_{IP}y_{IP}
\end{bmatrix} / \det(A) \quad \text{(3.30)}
\]

Before the left-hand side matrix can be calculated, the number of values in the sparse
matrix must be found because the hybrid nodes now have a larger stencil than normal. Figure 3.10a shows how the ghost node stencils can be modified. Ghost node 1, in the upper left, only uses four nodes indicated by the open circles. In a situation where a stencil is smaller than five nodes, an additional node that is not being used is added with a coefficient of zero (this is done because CUSP COO matrices are initialized with a specified number of non-zero nodes). When initialized, all the values are set at (0,0,0), where the zeros represent the row, column, and value of a given non-zero entry. If the CUSP multi-grid solver encounters multiple (0,0,0) entries, it will fail to converge upon a solution. It is easier to add a zero value than to restructure the matrix to not have a value. For example, let node 1 be the 1000th spot in the array, and the grid span $n_x$ cells in the $x$ direction. The nodes that represent the stencil will be $(1000,1000+n_x,u_{north})$, $(1000,1000+n_x+1,u_{north\ east})$, $(1000,1000,u_{center})$, and $(1000,1000+1,u_{east})$. The value, $(1000,1000-1,0)$ would be added to the sparse matrix to ensure that there are five total nodes that node 1000 is dependent on. Ghost node two, which depends on the nodes marked with open squares, does not need any special treatment because it uses five nodes.

Figure 3.10 shows the stencil for hybrid nodes, with two overlapping stencils. The
interpolation stencil is shown by the shaded region and depends on the nodes marked with circles. The Navier–Stokes stencil is marked with the solid black lines and open squares. Combining the two into a hybrid stencil results in six entries to the sparse matrix. As the left-hand side matrix is being built, the algorithm will place five of the six nodes in their allocated memory locations. To place the last node, each hybrid node is counted and labeled from left to right, top to bottom. Extra memory is allocated to the end of the sparse matrix arrays, with one additional spot for each counted hybrid node. The sixth entry to sparse matrix is placed at the end of the normal amount of entries to the sparse matrix plus the count of the hybrid nodes. For example, if before the immersed boundary is considered, the left-hand side matrix is expected to have twenty thousand entries and has 200 hybrid nodes then, hybrid node 50 will have its sixth value placed at location 20050 in the sparse matrix arrays.

After the left-hand side matrix is made, it is sorted by row and column using a built-in CUSP function and then the preconditioners are rebuilt. The CUSP solvers are optimized to operate on a sorted matrix and would sometimes fail to converge due to out of order values at the end of the arrays. During the calculation of the LHS matrix, the modifications to the right hand side matrix are stored in two values. The first one is a multiplier for the original RHS matrix and the second is added to that value. This makes it possible to multiply the original value by the Navier–Stokes, weight then add the interpolated weight. Finally, $\hat{u}$ is calculated using CUSP.

The implementation of the embedded Poisson equation is the same as outlined above.
Chapter 4: Verification and validation

The three immersed boundary methods have been compared against four well known simulations and experiments. The term “verification” will be used when comparing against other simulations and the term “validation” will be used when comparing against experimental data. Each simulation was chosen to test a specific capability of the solver and compound in difficulty. Table 4.1 shows which immersed boundary methods are tested with each simulation.

Lid-driven cavity flow is the classic validation case for my fluid dynamics solver. It tests discretization of the basic Navier–Stokes equations, which includes the advection and diffusion terms, linear algebra, and Dirichlet boundary conditions. As it does not have a body, no immersed boundary method is used. The second simulation is an impulsively started cylinder, which tests the immersed boundary method implementation and force calculation. The third simulation tests the solver’s ability to handle a body with prescribed motion. In this test, a cylinder has a set sinusoidal motion in an initially stagnant flow. The fourth simulation is another impulsively started cylinder, but with a set cylinder motion. Tests three and four look at the force calculation for a moving body and help quantify the numerical oscillation caused by direct forcing. The last simulation is called vortex-induced vibration (VIV) and tests the solver’s ability to accurately predict body location, velocity, and force when the body motion is not prescribed.

<table>
<thead>
<tr>
<th>#</th>
<th>Simulation name</th>
<th>Modified Fadlun</th>
<th>External</th>
<th>Embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lid-driven cavity flow</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>2</td>
<td>Impulsively started cylinder</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Oscillating cylinder</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Impulsively started oscillating cylinder</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Vortex induced vibration</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.1: Which immersed boundary methods are tested with what simulations.
4.1 Lid-driven cavity flow

Lid-driven cavity flow is a simulation that tests the discretization and implementation of the incompressible, two-dimensional Navier–Stokes Equations (2.1) on the GPU. In lid-driven cavity flow, fluid flow is “driven” by motion of the top boundary of a square domain; i.e., the lid of the square domain has a Dirichlet boundary condition of \( u = 1 \), and all of the other velocity boundary conditions are set to zero. Eventually the flow will achieve a steady-state swirling flow field that can be verified with results in literature. I chose to compare simulations with Reynolds numbers of 100 and 1000 to those of Ghia et al. [24]. A simple way to compare lid-driven cavity flow results is to plot the \( u \) and \( v \) velocities at the vertical and horizontal centerlines, respectively. Figure 4.1 shows the domain and aforementioned centerlines.

![Figure 4.1: The domain of the lid-driven cavity flow simulation with boundary conditions and centerlines shown.](image)

For a Reynolds number of 100 the domain was discretized into a \( 32 \times 32 \) grid. With a time step of 0.01, the maximum CFL number works out to approximately 0.3 using Equation (4.1):

\[
\text{CFL}_{\text{max}} = \Delta t \left( \frac{u}{\Delta x} + \frac{v}{\Delta y} \right)_{\text{max}}
\]  

(4.1)

For a Reynolds number of 1000 the domain was discretized into a \( 128 \times 128 \) grid. A time step of 0.004 keeps the maximum CFL at 0.3. Figures 4.2 (a) and (c) show the \( v \) velocity at the horizontal centerline and (b) and (d) show the \( u \) velocity at the vertical centerline,
with the top and bottom rows showing results for Re=100 and 1000, respectively.

Figure 4.2: Verification for lid-driven cavity flow. The top row shows Re=100 and the bottom row shows Re=1000. The left column shows the $v$ velocity at the horizontal centerline and the right column has the $u$ velocity at the vertical centerline.

4.2 Impulsively started cylinder flow

The impulsively started cylinder simulation tests the immersed boundary methods’ ability to handle an immersed body. It also tests the stretched grid and force calculation. This test mimics a body being “impulsively started”, as if it was struck by a hammer. In the first few time steps, a large sheer stress on the body will cause a high drag, which rapidly dissipates and then slowly reaches steady state as the boundary layer develops and stabilizes. The immersed boundary method results can be verified by computing the drag over time and comparing with the results of Koumoutsakos and Leonard [1]. To
simulate an impulsively started cylinder, a circular body is anchored to the origin and an initial $u$ velocity of one is set over the entire fluid domain. The left, top, and bottom boundaries have Dirichlet $u$ and $v$ values of one and zero forming an inlet and two free stream boundaries, respectively. A convective boundary condition, given in Equation (4.2) is used at right edge that uses one and zero for nominal $u$ and $v$ values.

$$u_{nx} = u_{nx} \left(1 - \frac{U_{\text{nominal}} \Delta t}{dx}\right) + \frac{U_{\text{nominal}} \Delta t}{dx} u_{nx-1}$$

(4.2)

The grids are made up of uniform spacing in the area immediately around the immersed body that stretches as it moves away from the center until it reaches the domain’s edge at ±15. Figure 4.3 shows the domain of the Re=40 simulation, the domains for Re=550 and 3000 have a slightly smaller uniform grid section in the middle.

![Figure 4.3: The domain and boundary conditions for the impulsively started cylinder simulation with Re=40. The embedded method is excluded because its results are the same as the external method.](image)

All lengths are normalized against the body diameter, i.e., the cylinder’s diameter is one. For a Re=40 a uniform grid with spacing 0.025 is used from (-0.6,-0.6) to (0.6,0.6) for a total of $186 \times 186$ (34596) cells are used. A time step of 0.005 is used with a maximum CFL of approximately 0.4. Figures 4.4(a) and 4.4(b) compare the computed drag force of the modified Fadlun and external Luo methods, respectively, with that of Koumoutsakos and Leonard [1]. The embedded method is not shown because the force is indistinguishable (in the graph) from the external method.

For Re=550, a uniform grid with spacing 0.01 is used from (-0.54,-0.54) to (0.54,0.54)
Figure 4.4: Drag force over time for two solvers compared to Koumoutsakos and Leonard [1] for flow over an impulsively started cylinder. The embedded method is excluded because its results are the same as the external method.

For a total of $450 \times 450 \ (202500)$ cells are used. A time step of 0.0025 is used for a maximum CFL number of 0.6. Figures 4.5(a) and 4.5(b) compare the computed drag force of the modified Fadlun and external Luo methods, respectively, with that of Koumoutsakos and Leonard [1].

Figure 4.5: Drag force over time for two solvers compared to Koumoutsakos and Leonard [1] for flow over an impulsively started cylinder. The embedded method is excluded because its results are the same as the external method.

For $Re=3000$, a uniform grid with spacing 0.004 is used from (-0.52,-0.52) to (0.52,0.52) for a total of $986 \times 986 \ (972196)$ cells are used. A time step of 0.001 is used for a maximum CFL of 0.65. Figures 4.6(a) and 4.6(b) compare the computed drag force of the
modified Fadlun external Luo methods, respectively, with that of Koumoutsakos and Leonard [1] for a Reynolds number of 3000.

![Graph](image)

(a) Modified Fadlun style solver.  
(b) External Luo style solver.

Figure 4.6: Drag force over time for two solvers compared to Koumoutsakos and Leonard [1] for flow over an impulsively started cylinder.

The large drag at the beginning is caused by the impulsive start of flow around the body. At the first time step, when all the fluid nodes have a $u$ velocity of one, the $\frac{du}{dx}$ term of shear force is large, as $du$ is one and $dx$ is the small grid spacing. It is interesting that neither method consistently over or under-predicts the correct drag. Instead, both of them change based on time and Re. Another result of note are the drag oscillations between 0.25 and 0.75 in the Re=3000 Fadlun results shown in Figure 4.6(a). The modified Fadlun method can develop numerical oscillations when at extreme Reynolds number, grid sizing, or time step.

4.3 Flow past an in-line oscillating cylinder

The impulsively started oscillating cylinder is a simulation used by Luo et al. [17] to demonstrate and compare the magnitude numerical oscillations between different immersed boundary methods. It does not a verify or validate as it does not compare the results with published work, but rather it is a useful tool to visualize the magnitude of numerical oscillations. A true moving-body verification is done in Section 4.4 This simulation has the same domain, initial, and boundary conditions as the stationary impulsively started cylinder, save for the uniform grid at the center which spans (-2,-2) to (2,2) to account for body’s movement. The cylinder oscillates along the horizontal
centerline according to Equations (4.3) and (4.4):

\begin{align}
  x &= -0.25 \cos 0.4\pi t \quad (4.3) \\
  u &= 0.1\pi \sin 0.4\pi t. 
\end{align}

The resulting period, maximum velocity, and position are 5, \pi/10, and 0.25, respectively. Figure 4.7 compares the numerical oscillations observed in drag vs. time. The left column shows drag force experienced with the external method, and the right column shows the drag force experienced by the embedded method. The uniform section is discretized into a 64 \times 64 grid for the first row and 128 \times 128 grid for last three rows for h (center grid spacing) values of 0.0625 and 0.03125. The maximum CFL numbers by row are approximately 0.35, 0.35, 0.7, and 1.0.

Both methods suppress the magnitude of the oscillations with decreasing grid size. The frequency of the oscillations is also reduced by increasing the maximum CFL number. This effect is most easily seen in the left column. The embedded method, as expected, is better at suppressing oscillations because it has access to updated information while doing its interpolations, extrapolations, and weighting. Examining the bottom row closely, it can be seen that both methods only suppress the oscillations, not eliminate them. It is interesting that, in spite of the oscillations, similar maximum force values could be obtained from any of the eight plots. The accuracy of the two methods is examined in the next section.

4.4 In-line oscillating cylinder

An in-line oscillating of a cylinder (with no inlet or exit and a free stream and initial velocity of zero) is used by Liao et al. [16] to demonstrate numerical oscillation suppression. The results of Liao et al. and cuIBM-FSI are validated against the experimental data of Dütsch et al. [2]. The equations that govern movement, (4.5) and (4.6), are the same form as above but with a period, maximum velocity, and position of 1, 1, and 1/2\pi:

\begin{align}
  x(t) &= -\frac{1}{2\pi} \sin 2\pi t \quad (4.5) \\
  u(t) &= -\cos 2\pi t. 
\end{align}
Figure 4.7: Drag force for flow over and in-line oscillating cylinder. The left column shows external interpolation and the right column shows the embedded interpolation. Rows one through four have maximum CFL 0.35, 0.35, 0.7, and 1.0 and a middle grid spacing of 0.0625, 0.0313, 0.0313, and 0.0313.

The diameter is matched to the Liao et al. [16] simulation, which is 0.2. These values give the simulation a Reynolds number of 100 and a Keulegan-Carpenter number (KC number) of 5. The KC number is a non-dimensional quantity representing the importance of drag and inertial forces on an oscillating body and can be found with Equation (4.7):

$$ KC = \frac{U_{\text{max}}}{fD}, $$

(4.7)
where $f$ is the oscillation frequency. The domain has a uniform section with grid spacing 0.005 from (-0.4,-0.2) to (0.4,0.2) ranging from $\pm 5.5$ in $x$ to $\pm 3.5$ in $y$. Figure 4.8 shows the drag during the first period versus the steady state drag from Dütsch et al. [2]. Figure 4.9 shows the steady state drag. In both figures, (a) is the external method and (b) is the embedded method.

Figure 4.8: Drag force vs. time for the first period compared with steady state Dütsch et al. [2] for an in-line oscillating cylinder.

Figure 4.9: Drag force verses time for steady state oscillating cylinder in now flow compared to Dütsch et al. [2]

Again, the embedded interpolation method suppresses the oscillations better than the external method. It also better predicts the maximum peak, which is about 15% off of the experimental data. Figure 4.10 shows the validation against Dütsch et al. for a
Reynolds number of 200 and a KC of 10. Note that this is comparing the drag dating during the 14th cycle. Dütsch et al. show force comparisons for various cycles into the low 100s. Re and KC can both be doubled by doubling the maximum velocity of the body which can be done by multiplying Equation (4.6) by two and integrating to get a new position equation. Once again, the embedded method suppresses oscillations better than the external method, but both do a good job at predicting the force.

![Graph](image)

(a) External interpolation method.  (b) Embedded interpolation method.

Figure 4.10: Drag force over the 14th period for in-line oscillating cylinder flow with Re=200, KC=10 compared to Dütsch et al. [2]

### 4.5 Vortex-induced vibration

The vortex-induced vibration simulation tests the solvers’ ability to predict the position of a freely moving body in coupled fluid structure interaction. The two-dimensional domain is the same as the impulsively started cylinder simulation except the uniform region ranges from (-1.5, -1.5) to (1.5, 1.5). An immersed circle starts at the origin with no initial velocity. The body is free to move in the y direction but fixed in the x and mounted to a spring. The Reynolds number is set to a range where the cylinder sheds oscillating vortice—in this case Re=150. This combined with the spring causes the body to establish a steady-state sinusoidal motion for which the amplitude can be verified with literature results. Following the work of Borazjani et al. [25] the mass-spring damper
system (4.8) that governs the body’s movement is:

\[ M \frac{\partial^2 Y}{\partial t^2} + C \frac{\partial Y}{\partial t} + KY = F_y, \]  

(4.8)

where \( Y \) is the center coordinate of the body, \( M \) is the mass, \( C \) is the damping factor, \( K \) is the spring stiffness, and \( F_y \) is the total fluid force on the body in the \( y \) direction. The natural frequency and critical damping are given by (4.9) and (4.10):

\[ \omega = 2\pi f = \sqrt{\frac{K}{M}} \]  

(4.9)

\[ C_{cr} = 2\sqrt{MK} = 2K\omega \]  

(4.10)

Then Equation (4.8) can non-dimensionalized to become Equation (4.11):

\[ \frac{\partial^2 Y}{\partial t^2} + 4\pi \zeta \frac{1}{U_{red}} \frac{\partial Y}{\partial t} + 4\pi^2 \frac{1}{U_{red}^2} Y = \frac{1}{2M_{red}} C_Y, \]  

(4.11)

where the non-dimensional coefficients are:

Damping coefficient

\[ \zeta = \frac{C}{C_{cr}} \]  

(4.12)

Reduced velocity

\[ U_{red} = \frac{U}{fD} \]  

(4.13)

Reduced mass

\[ M_{red} = \frac{M}{\rho D^2} \]  

(4.14)

Force coefficient

\[ C_Y = \frac{2F_y}{\rho U^2 D} \]  

(4.15)

Varying the parameters \( Re, \zeta, M_{red}, \) and \( U_{red} \) changes the resulting steady-state amplitude. In vortex-induced vibration there is a “lock-in” phenomenon that occurs when \( Re=150, \zeta=0, M_{red}=2, \) and \( U_{red} \) is varied between 3 and 8. At either extreme for \( U_{red}, \) the steady-state amplitude will be small, and for values in between the steady-state amplitude will be much larger.

There are two different ways to update body position: loose and strong coupling.
Loose coupling comes from the explicit derivation of Equation (4.11). A time step using loose coupling proceeds as follows:

1. Solve for field values at \( t^{n+1} \) using the Navier–Stokes equations.
2. Calculate body forces.
3. Move body using Equations (4.16) and (4.17):

\[
v^{n+1} = v^n - \frac{\Delta t \pi^2 4}{U_{red}^2} \left( y^n + \frac{\Delta t v^n}{2} \right) + \frac{\Delta t C_Y}{(2M_{red}) \left( 1 + \frac{4\Delta t \pi^2}{U_{red}^2} \right)}
\] (4.16)

\[
y^{n+1} = y^n + \frac{\Delta t (v^{n+1} + v^n)}{2}
\] (4.17)

Equation (4.11) can also be discretized implicitly to achieve strong coupling. A strong coupling time step repeatedly calculates the field values and body kinetics/kinematics until reaching a converged solution. As this causes the Poisson equation to be solved multiple times for each time step, strong coupling is a significantly more computationally expensive method:

1. Solve for field values at sub step \( t^{k+1} \) using the Navier–Stokes equations.
2. Calculate body forces.
3. Move body using Equations (4.16) and (4.18), where \( \alpha \) is some relaxation value.
4. If \(|Y^{k+1} - Y^k| \approx 0\), advance time; otherwise advance sub step.

\[
y^{k+1} = \alpha \left( y^n + \frac{\Delta t (v^{k+1} + v^n)}{2} \right) + (1 - \alpha) y^k
\] (4.18)

Figures 4.11, 4.12, 4.13, and 4.14 show the maximum amplitude of the six vortex-induced vibration simulations as calculated by external loose coupling, external strong coupling, embedded loose coupling and embedded strong coupling respectively. Results are verified against the curvilinear immersed boundary method of Borazjani et al. [25] and the unstructured finite-element approach of Ahn and Kallinderis [26]. All four configurations used to solve for vortex-induced vibration (embedded/external and loose/strong coupling) of the are able to predict the maximum steady state amplitude fairly accurately.
Figure 4.11: External method loose coupling. The maximum amplitude vs. reduced velocity for the lock-in region.

even though the external method exhibits noticeable oscillations of similar magnitude to those shown for the impulsively started oscillating cylinder. The main difference between the external and embedded method is the wall-clock time taken to run each, with the embedded method taking nearly an order of magnitude more than the external method.
Figure 4.12: External method strong coupling. The maximum amplitude vs. reduced velocity for the lock-in region.
Figure 4.13: Embedded method loose coupling. The maximum amplitude vs. reduced velocity for the lock-in region.
Figure 4.14: Embedded method strong coupling. The maximum amplitude vs. reduced velocity for the lock-in region.
Chapter 5: Solver characterization

This section will evaluate the different methods used in terms of order of accuracy and performance scaling. The modified Fadlun, external Luo, and embedded Luo methods will be evaluated using the impulsively started cylinder simulation and the external and embedded Luo methods will be evaluated again using an oscillating cylinder in flow.

5.1 Order of accuracy

Order of accuracy is evaluated by using a simulation evaluated with a fine grid as the “exact” solution. Coarser grids have their error calculated using the exact solution and are plotted on a log scale to show effects of grid size on accuracy.

5.1.1 Impulsively started cylinder flow

All grid sizes will refer to the grid spacing in the middle, which is uniform. Grid spacings of 0.005, 0.005, and 0.01 are used as the “exact” solution for the modified Fadlun, external and embedded impulsively started cylinder test. Using a different exact grid size causes the embedded results to not be directly comparable but using a finer grid proved to make the simulation unstable. The reason for this is discussed below. Figure 5.1 shows the error calculated based on drag coefficient for three coarser grids with center grid spacings ranging from 0.0625 to 0.01.
Figure 5.1: Average error in drag force plotted against grid spacing, showing the order of accuracy for the three solvers.
The external method shows an order of accuracy around two but the modified Fadlun and embedded methods appear to not become sufficiently more accurate with decreasing grid size. To explain this why they are not improving the individual results must be examined. Let’s start with embedded method. Figure 5.2 shows the drag force of the two finest grid spacings, those corresponding with the exact solution and the left grid point of Figure 5.1c. Using a grid spacing of 0.01 produces a small discontinuity at time 0.3 that none of the coarser grids have. The force error of grid size 0.015625 is plotted in Figure 5.3. The error is dominated by the discontinuity after time 0.3 which causes poor order of accuracy results for the embedded method. If the error had stayed around the same magnitude before and after time 0.3 the average error would have worked out to be around 0.005 instead of the 0.015 plotted above. It is possible that the discontinuity is a real physical effect that the finer grid is capturing, like a boundary layer reattaching to the body. It was not obvious what was causing the discontinuity by examining at velocity and pressure contours before and after the jump. This, coupled with the lack of a discontinuity present in the external method, suggests it could be a numerical nuance. However, the effect is also present in the modified Fadlun method, at a different time.

Figure 5.2: Drag force vs. time for different grid spacings using the embedded solver.

Figure 5.4 shows the drag force produced with the modified Fadlun method using grid sizes of 0.005 (finest) and 0.025 (coarsest). The finest grid size for the Fadlun method, like the embedded method, also picks up a discontinuity in the force, although at time 0.7. Unlike the embedded method, the Fadlun method has trouble appropriately
Figure 5.3: Error in drag force vs. time using the embedded method with a center grid spacing of 0.01.

handling the jump and ends up creating numerical oscillations. The coarsest grid does not pick up the discontinuity and returns a smooth drag curve.

Figure 5.4: Drag force versus time with different grid sizes using the modified Fadlun method.

Figure 5.5 presents a better calculation of order of accuracy. The order of accuracy here is calculated using the $u$ velocity field. Velocity values from the fine grid are bilin-
early interpolated for at the location of velocity values from the coarse grid then used to
calculate to error using Equation (5.1):

\[
Error_{\text{avg}} = \left( \frac{\sum_{n_x \cdot n_y} |u_{\text{fine}} - u_{\text{coarse}}|}{n_x \cdot n_y} \right) / \left( \frac{n_x \cdot n_y}{u_{\text{fine}}} \right)
\]  

(5.1)

The average error can then be used to calculate the order of accuracy using Equation
(5.2):

\[
\text{Order of Accuracy} = \frac{\log (Error_{\text{fine}}/Error_{\text{coarse}})}{\log (h_{\text{fine}}/h_{\text{coarse}})}
\]  

(5.2)

Figure 5.5: Order of accuracy calculated at multiple time steps before the discontinuities.

The order of accuracy for all three methods is around one and a half, with the external method having the highest and the embedded having the lowest. Although Crank–Nicolson and Adams–Bashforth discretization are both second order accurate, the treatment at the body to between first- and second- order accuracy.

Figure 5.6 shows the \( L_1 \), \( L_2 \), and \( L_\infty \) error norms for the three immersed boundary methods. The \( L_1 \), \( L_2 \) and \( L_\infty \) error norms are calculated using Equations (5.3), (5.4),
The $L_1$ error norm is the sum of all the error in every cell. $L_2$ is the square root of the sum of the squared errors. The error in each cell is less than one so the $L_2$ error norm is several orders of magnitude lower than the $L_1$. $L_\infty$ is the most intuitive of the error norms: it is simply the largest difference between the fine and coarse grids. An $L_\infty$ value of 0.2 obtained from the coarse grids is mediocre because the maximum velocity is around one and a half, giving about a 13% relative error compared to 3% for the fine grids. As the grid size decreases and the cell count increases, the most accurate solver will have the smallest slope. For the impulsively started cylinder the external method has the lowest error norms across the board. I believe this happens because the error norms are being calculated at the end of the simulation, and as shown above the modified Fadlun and embedded methods have discontinuities in the “exact” solutions. This effect is evident in the $L_2$ error norm where the slope of the modified Fadlun and embedded method are both relatively shallow compared to the external method.
Figure 5.6: Impulsively started cylinder error norms.
5.1.2 Oscillating cylinder in flow

The embedded and external methods also have their order of accuracy tested using flow over an in-line oscillating cylinder. The modified Fadlun method was not tested here because it proved too unstable to perform with varying grid sizes and time steps. Figure 5.7 shows the error with changing grid spacing for the external and embedded methods. Interestingly, the two methods have nearly the same order of accuracy even though the external method is known to have higher amplitude oscillations (as shown in Section 4.3).

![Graph showing error plotted against grid spacing for an oscillating cylinder in flow.](image)

Figure 5.7: Error plotted against grid spacing shows the two solvers’ order of accuracy for an oscillating cylinder in flow.

5.2 Performance

Performance is evaluated by comparing the total number of cells and the computational time. Typically, certain parts of the simulation take longer than others. The number of iterations taken to solve the Poisson equation gives a rough estimate of the overall wall-clock time because the Poisson step takes a large majority of the overall time. Figure 5.8 shows the number of iterations used for each Poisson solve. Figures 5.8(a), (b), and (c) show the iterations over time for an impulsively started cylinder simulation, and an
external and embedded oscillating cylinder in flow simulation, respectively. The iteration count is largest when the flow field is highly transient. For instance, in the impulsively started cylinder flow the iterations are largest at the beginning because the flow fields are going from unity velocity and no pressure gradient to a fully developed boundary layer. Note that 5.8(a) is solved using the external method which does not have any discontinuities or oscillations in the drag as seen in Figures 5.2 or 5.4. The oscillating cylinder simulation spans two periods with the cylinder starting at $x_{body} = -0.25$ and $u_{body} = 0$. With the embedded method, low numbers of Poisson iterations correspond with less numerical iterations which happen when the body velocity is low. The peaks in the external method occur when the oscillations go from low to high, although the iterations go back down while there is still high numerical oscillation. The average iteration count for the external method is around 30 and the embedded method is an order of magnitude more, around 500. If the above assertion that iteration count is a rough estimate of time is true, then the embedded method should take an order of magnitude more time for the oscillating cylinder simulation.

5.2.1 Impulsively started cylinder

The performance of the impulsively started cylinder simulation is evaluated using the same set of simulations as the order of accuracy. Figure 5.9 shows the performance of all three methods calculated on two different GPUs. The 750ti is less powerful than the K20 with 640 and 2496 processor cores, respectively. The external method performs faster than the modified Fadlun method, though it involves more complex equations. This trend surfaces because of the GPU. While the external method has complex additional steps, they all end up being trivial to solve on a GPU (from a computational time perspective). The time required to perform the interpolations and extrapolations is irrelevant when compared to the Poisson solution. Computational time for all three methods, across every type of simulation, is dominated by that of the Poisson solver. The question then becomes: why does the Poisson solver take longer to solve for the different methods? Answering this question requires another look at the linear multi-grid method. Figure 5.10 shows Poisson stencils on the original grid (left) and a grid reduced by the multi-grid method (right). Figure 5.10a shows the standard stencil that is used for most nodes. The modified Fadlun and embedded methods use non-standard
stencils near the body because they both enforce boundary conditions inside the Poisson solver. An example of an atypical stencil, one of the possible stencils for an embedded hybrid node, is shown in Figure 5.10(b). Linear algebra solvers commonly encounter tri-diagonal or banded tri-diagonal matrices that spawn from the discretization of one-, two-, and three- dimensional Poisson equations. As such, most linear algebra solvers are highly optimized algorithms to deal with those types of matrices. When non-standard stencils get thrown at the solver, like those found at ghost and hybrid nodes in the embedded method, the solver is unable to efficiently solve the system of equations—resulting in more iterations and longer computational time. The external solver is the fastest because it uses the standard Poisson stencil for the entire domain.

It is worth noting the difference in slope between the three solvers’ performance curves. If the grid spacing is cut in half the cell count over that domain quadruples. When this happens a non-multi-grid solution should take approximately four times as long to solve the resulting system of equations. Theoretically, a multi-grid method should be able to achieve better scaling because much of the computation is done on reduced-size grids. While the Fadlun and external methods have roughly the same slope, that of the embedded method is much steeper. This implies that the multi-grid method handles the different abnormal Poisson stencils differently. The clustered, non-standard stencils used at the hybrid and ghost nodes may result in the multi-grid solver being unable to sufficiently reduce the grid. This might explain the higher order of magnitude of computation time seen with increasing cell count.

5.2.2 Oscillating cylinder

Figure 5.11 the performance for the oscillating cylinder. Similar trends to the impulsively started cylinder test can be observed here. Most obvious is the order of magnitude difference in the time taken for each method. In addition, the external method scales better with cell count. Lastly, it is possible to see another effect caused by GPU scaling, which was not visible for the impulsively started cylinder. Previously, the more powerful GPU (specifically as measured by the amount of processor cores available) achieved roughly the same speedup regardless of cell count (speedup is a term used to measure the difference in performance between algorithms performing the same task, e.g., if one algorithm takes ten seconds and another algorithm takes five then the second algorithm
achieved a speedup of two). For the oscillating cylinder, smaller grid sizes have a very low speedup, which means that using a more powerful GPU for those simulations does not improve performance. This happens when the non-parallel sections of the solver take more time than the parallel sections. As cell count increases, the parallel portions become larger resulting in a larger speedup.

Figure 5.11 is slightly misleading. The simulations that comprise it use the same uniform domain size and the same stretching ratios. Total cell count is changed solely by changing the grid spacing in the uniform section. This graph implies that the overall computational time relies on the total cell count, which is true but the embedded method also relies on the center grid spacing h. In the previous section it was suggested that the multi-grid solver does not properly reduce the grid for the embedded method. If this is true than the embedded method should have a negative correlation between grid spacing and computational time (while keeping the overall cell count constant). To test this, three grid spacings of 0.0625, 0.05, and 0.03125 were tested. The uniform grid section was held constant, ranging from (-1.5,-1.5) to (1.5,1.5). The cell count was kept at 224×224 (≈60k total) by modifying the stretch ratios. Figure 5.12 shows the resulting computational time. Halving the grid size causes the embedded method’s computational time to increase by 150%, where the external method only increases by 33%.
Figure 5.8: Poisson iterations taken each time step. From left to right: Impulsively started cylinder with external method, in-line oscillating cylinder in flow with external method, in-line oscillating cylinder in flow with embedded method.
Figure 5.9: Impulsively started cylinder performance.

(a) Reduction of the standard Poisson stencil.

(b) Reduction of a hybrid node Poisson stencil.

Figure 5.10: Grid reduction that takes place inside of the linear multi-grid solver.
Figure 5.11: Performance for flow over an in-line oscillating cylinder.

Figure 5.12: Computational time for flow over an in-line oscillating cylinder. The total cell count and uniform grid section size are held constant.
Chapter 6: Summary and recommendations for future work

Immersed boundary methods are a powerful numerical approach that excel at simulating complex fluid structure interactions problems. The study and optimization of such a device requires an accurate and fast solver. This speed is obtained through the massive parallelism of a GPU, which is currently largely underutilized in commercial software. cuIBM-FSI was developed to test various immersed boundary methods on a GPU, with the intent of finding the best solver to study complex freely moving body in incompressible flows.

To this end, three direct forcing immersed boundary methods were considered. The modified Fadlun method is based off the work of Fadlun et al. [14]. Like the original method it does away with the force calculation at the hybrid nodes (nodes just outside the body) in favor of interpolation between the body and nearby non-hybrid fluid nodes during the intermediate velocity step. Unlike the original Fadlun it modifies the Poisson equation to maintain mass conservation. The external and embedded methods are based off the work of Luo et al. [17]. Both interpolate for the pressure and intermediate velocity at hybrid nodes. They also extrapolate intermediate velocities and pressure values across the body such that hybrid node values can be calculated using a standard stencil. These two solutions are weighted based on nodal distance from the body to suppress numerical oscillations caused by moving bodies in combination with direct forcing methods. The external method does the weighting, interpolation, and extrapolating before and after the linear algebra solves, where the embedded method does these steps all simultaneously inside the linear algebra calculations.

The three solvers were tested using five well-studied benchmarks. The underlying Crank–Nicholson, Adams–Bashforth, and fractional step methods used to discretize the underlying Navier–Stokes equation were tested using lid-driven cavity flow. An impulsively started cylinder test was used to test a stationary immersed body. This test confirmed that the original Fadlun method exhibits error in high shear situations, which led to the implementation of the modified Fadlun method. Two simulations were used to test the methods for an imposed body motion. Both simulations use a circular body
oscillating sinusoidally on the $x$-axis. The modified Fadlun method proved unstable with moving bodies, resulting from overwhelming numerical oscillations present in all direct forcing methods with a moving body. Flow in the $x$-direction over the oscillating cylinder was used to qualitatively assess the numerical oscillations in the method. This test shows that the embedded method is better at suppressing numerical oscillations than the external method. For both methods, decreasing the grid spacing, or increasing the CFL number reduces oscillations and increases accuracy. Oscillations in the $x$-direction in a static flow field is used to quantitatively verify the two methods. This simulation showed the external method to be slightly less accurate than the embedded method, and reinforced, the previously stated results of the impulsively started oscillating cylinder simulation.

The last test is vortex-induced vibration, which features bodies that are anchored at the origin but free to move based on forces imparted by the flow. Two methods for updating the fluid structure interaction were tested: loose and strong coupling. All four variations of loose and strong coupling, external and embedded were able to successfully predict the forces, position, and velocity of vortex-induced vibrations. The main difference between the embedded method and the external method is the time taken to solve each. Each of the six simulations ($U_{red}$ one through six) is simulated to time 60. The external loose coupling takes around eight hours where as the embedded loose coupling takes around 60. Interestingly the strong coupling does not take much longer, with embedded strong coupling taking about 72 hours. The main reason for this is that in each sub-step after the first, the Poisson solution has a nearly perfect field to start with which requires much less computational time.

So, which method is the “best”? The external method performs the best for stationary bodies, as it is the fastest while all methods produced the same results. For moving bodies, the external method is overwhelming faster than the embedded method, and scales much better—but more prone to oscillations and slightly less accurate. The external method is also more stable, but that seems to be mostly due to the linear algebra solver. During optimization or parameter sweeps, it may be prudent to use the external method to narrow down the field, and then use the embedded method afterwards for more accuracy.

To improve the embedded and external methods, improving the speed of the Poisson solve is still the most efficient (and only practical) approach. Replacing the library
that cuIBM-FSI uses to solve the linear algebra, CUSP, is the first step that should be taken. The most recent official release of CUSP was in 2014 and designed to work with CUDA 5.5. cuIBM-FSI currently uses CUDA 7.5, and at the time of writing CUDA 8.0 is available. Other libraries, such as AmgX are available that have more advanced multi-grid methods that may not suffer from the same scaling problems CUSP does. In addition, CUSP does not support multiple GPUs, meaning that this code can not be used on a cluster.
Bibliography


APPENDICES
Appendix A: Solving a system of four equations on the GPU

\[
A = \begin{bmatrix}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{bmatrix} \tag{A.1}
\]

If:

\[
\det A = a_{11}a_{22}a_{33}a_{44} + a_{11}a_{23}a_{34}a_{42} + a_{11}a_{24}a_{32}a_{43} + a_{12}a_{21}a_{34}a_{43} + a_{12}a_{23}a_{31}a_{44} + a_{12}a_{24}a_{33}a_{41} + a_{13}a_{21}a_{32}a_{44} + a_{13}a_{22}a_{34}a_{41} + a_{13}a_{24}a_{31}a_{42} + a_{14}a_{21}a_{33}a_{42} + a_{14}a_{22}a_{31}a_{43} + a_{14}a_{23}a_{32}a_{41} \neq 0 \tag{A.2}
\]

Then there is an inverse of A:

\[
A^{-1} = \frac{1}{\det A} \begin{bmatrix}
b_{11} & b_{12} & b_{13} & b_{14} \\
b_{21} & b_{22} & b_{23} & b_{24} \\
b_{31} & b_{32} & b_{33} & b_{34} \\
b_{41} & b_{42} & b_{43} & b_{44}
\end{bmatrix} \tag{A.3}
\]
where:

\[ b_{11} = a_{22}a_{33}a_{44} + a_{23}a_{34}a_{42} + a_{24}a_{32}a_{43} - a_{22}a_{34}a_{43} - a_{23}a_{32}a_{44} - a_{24}a_{33}a_{42} \]

\[ b_{12} = a_{12}a_{34}a_{43} + a_{13}a_{32}a_{44} + a_{14}a_{33}a_{42} - a_{12}a_{33}a_{44} - a_{13}a_{34}a_{42} - a_{14}a_{32}a_{43} \]

\[ b_{13} = a_{12}a_{23}a_{44} + a_{13}a_{24}a_{42} + a_{14}a_{22}a_{43} - a_{12}a_{24}a_{43} - a_{13}a_{22}a_{44} - a_{14}a_{23}a_{42} \]

\[ b_{14} = a_{12}a_{24}a_{33} + a_{13}a_{22}a_{34} + a_{14}a_{23}a_{32} - a_{12}a_{23}a_{34} - a_{13}a_{24}a_{32} - a_{14}a_{22}a_{33} \]

\[ b_{21} = a_{21}a_{34}a_{43} + a_{23}a_{31}a_{44} + a_{24}a_{33}a_{41} - a_{21}a_{33}a_{44} - a_{23}a_{31}a_{44} - a_{24}a_{33}a_{41} \]

\[ b_{22} = a_{11}a_{33}a_{44} + a_{13}a_{34}a_{41} + a_{14}a_{31}a_{43} - a_{11}a_{34}a_{43} - a_{13}a_{31}a_{44} - a_{14}a_{33}a_{41} \]

\[ b_{23} = a_{11}a_{24}a_{43} + a_{13}a_{21}a_{44} + a_{14}a_{23}a_{41} - a_{11}a_{23}a_{44} - a_{13}a_{21}a_{44} - a_{14}a_{24}a_{43} \]

\[ b_{24} = a_{11}a_{23}a_{34} + a_{13}a_{24}a_{31} + a_{14}a_{21}a_{33} - a_{11}a_{24}a_{33} - a_{13}a_{21}a_{34} - a_{14}a_{23}a_{31} \]

\[ b_{31} = a_{21}a_{32}a_{44} + a_{22}a_{31}a_{42} + a_{24}a_{33}a_{42} - a_{21}a_{34}a_{42} - a_{22}a_{31}a_{44} - a_{24}a_{33}a_{42} \]

\[ b_{32} = a_{11}a_{34}a_{42} + a_{12}a_{31}a_{44} + a_{14}a_{32}a_{41} - a_{11}a_{32}a_{44} - a_{12}a_{31}a_{44} - a_{14}a_{32}a_{41} \]

\[ b_{33} = a_{11}a_{22}a_{44} + a_{12}a_{24}a_{41} + a_{14}a_{21}a_{42} - a_{11}a_{24}a_{42} - a_{12}a_{21}a_{44} - a_{14}a_{22}a_{41} \]

\[ b_{34} = a_{11}a_{24}a_{32} + a_{12}a_{21}a_{34} + a_{14}a_{22}a_{31} - a_{11}a_{22}a_{34} - a_{12}a_{24}a_{31} - a_{14}a_{21}a_{32} \]

\[ b_{41} = a_{21}a_{33}a_{42} + a_{22}a_{31}a_{43} + a_{23}a_{32}a_{41} - a_{21}a_{32}a_{43} - a_{22}a_{33}a_{41} - a_{23}a_{31}a_{42} \]

\[ b_{42} = a_{11}a_{32}a_{43} + a_{12}a_{33}a_{41} + a_{13}a_{31}a_{42} - a_{11}a_{33}a_{42} - a_{12}a_{31}a_{43} - a_{13}a_{32}a_{41} \]

\[ b_{43} = a_{11}a_{23}a_{42} + a_{12}a_{21}a_{43} + a_{13}a_{24}a_{41} - a_{11}a_{24}a_{41} - a_{12}a_{23}a_{41} - a_{13}a_{21}a_{42} \]

\[ b_{44} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31} \]