GPU-Based Fluid-Structure Interaction using Immersed Boundary Methods

Christopher Minar[a], Kyle E. Niemeyer[a,*]

[a]*School of Mechanical, Industrial, and Manufacturing Engineering Oregon State University, Corvallis, OR 97331, USA*

Abstract

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.

Keywords: GPU, computational fluid dynamics, immersed boundary, fluid-structure interaction

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 generate 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 relatively immobile base 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. This paper will discuss the development and characterization cuIBM-FSI [1], a fast fluid flow solver for handling fluid structure interaction with complex bodies.

Graphics processing units (GPUs) have a large theoretical performance for highly parallel problems. The extensive linear algebra required in computational fluid dynamics presents an enticing opportunity for GPUs. To achieve a speedup over commercial solvers, cuIBM-FSI performs most calculations using GPUs.

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 [2, 3, 4]. This is sometimes referred to as GPU acceleration. When there is only one node, such as in a workstation, data transfer times can prohibit efficient simultaneous use of both CPU and GPU. cuIBM-FSI was forked from cuIBM [5], a GPU-based immersed boundary solver primarily based on the method developed by Taira and Colonius [6]. cuIBM can not handle coupled fluid-structure interaction and produces numerical oscillations when working with moving bodies [7]. Improvements to cuIBM in-

*Corresponding author

Email address: Kyle.Niemeyer@oregonstate.edu (Kyle E. Niemeyer) clude adding fluid-structure interaction and suppressing numerical oscillations by implementing a new fluid solver based on the method presented by Luo et al. [8].

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. Immersed boundary method techniques are applicable to a wide range of problems, as reviewed by Mittal and Iaccarino [9] and more recently Sotiropoulos and Yang [10]. These techniques are well-suited for simulating flow involving complex, moving bodies because they do not require re-meshing between time steps. We have implemented two immersed boundary method solvers for operating on, and investigating, graphics processing units designed to handle incompressible fluid–structure interaction problems with rigid bodies. The methods focused on in this work are both from a popular subgroup of immersed boundary methods known as direct forcing methods [11, 12]. 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.

Using a direct forcing method with a moving body can cause numerical oscillations [10, 13, 8]. This effect is manageable for preset motion, but with a freely moving body the solver can poorly predict body position, fluid values, 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. [8] suppress the numerical oscillations with a weighting function to transition between the two schemes, removing the discontinuity where the regimes transition.

2. Numerical methods and Implementation

The cuIBM-FSI is comprised of two methods based off the work of Luo et al. [8]. For now, it will suffice to refer to these as the external, and embedded methods, leaving their explanations for later.

Both methods start with the two-dimensional, incompressible form of the Navier–Stokes momentum and mass continuity equations:

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

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

where $\mathbf{u}$ is velocity, p is pressure, and ν is the (constant) kinematic viscosity. The fractional-step approach [14] is used to solve Equations (1) and (2), which breaks up the discretization of the Navier–Stokes equations into three sub-steps. In the first sub-step, the momentum equations are solved without pressure to calculate an intermediate velocity:

$$\hat{\mathbf{u}} - \frac{\Delta t}{2} L(\hat{\mathbf{u}}) = \mathbf{u}^n + \Delta t(\mathbf{RHS}^n) \ , \tag{3}$$

where $\hat{\mathbf{u}}$ is the intermediate velocity, Δt is the time step, L is the Laplacian operator, and **RHS** contains the discretized advection and diffusion terms. The superscript n represents the current 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.

In the second sub step, the continuity equation is imposed to approximate the pressure, p, resulting in a Poisson equation:

$$\nabla^2 p^{n+1} = -\frac{\nabla \hat{\mathbf{u}}}{\Delta t} \ , \tag{4}$$

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

$$\mathbf{u}^{n+1} = \hat{\mathbf{u}} - \Delta t \nabla p^{n+1} \ . \tag{5}$$

2.1. 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. Although efforts are typically taken to avoid thread divergence when writing code for a GPU, it was found to be 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. In the cases tested, transferring data back and forth took significantly more time than would be saved by running some operations on a CPU.

Secondly, kernel performance is evaluated compared to the Poisson equation. 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 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 only occurs in cells near the body which is generally a low percentage of the overall cell count.

The third strategy comes from Layton et al. [5]. 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. [5] 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.

2.2. Luo et al. method

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. [8] and is similar to Mittal et al. [9]. There are four types of nodes. Nodes immediately outside and inside the body are called hybrid nodes and ghost nodes, respectively. Everything outside of the body that is not a hybrid node is a fluid node. Everything else (i.e., all nodes inside the body that are not ghost nodes) are solid nodes. Figure 1 shows an example body and grid with labeled nodes.

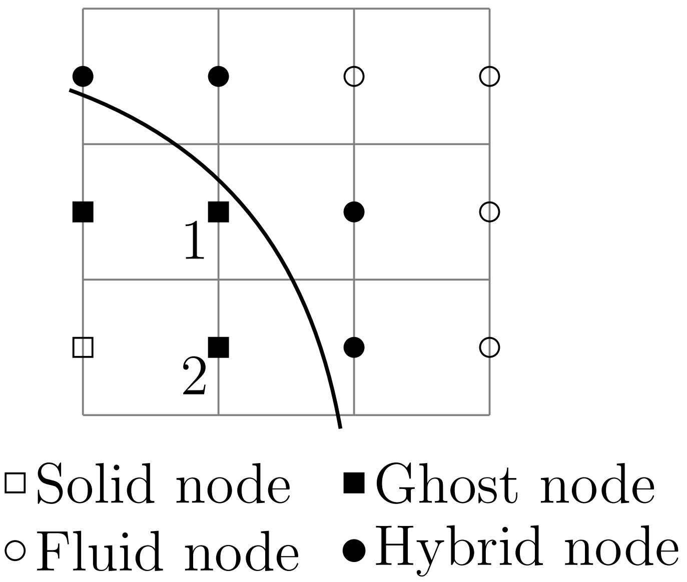


Figure 1: An example grid with u-velocity nodes labeled. Squares mark nodes inside the immersed boundary and circles mark nodes outside the boundary. Solid markers indicate the nearest nodes to the boundary.

Hybrid nodes are interpolated for using values at the body and nearby fluid nodes. Ghost nodes have data extrapolated across the body using nearby fluid nodes,in a way that enforces the body's boundary conditions. Fluid nodes are solved with the Navier–Stokes equations. 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.

At the beginning of each time step, nodes are identified using a ray-tracing algorithm (described in O'Rourke [15]). The implementation is based on the CPU implementation from cuIBM [16]. 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 2). The node is then tested for proximity to the body in the x direction. To

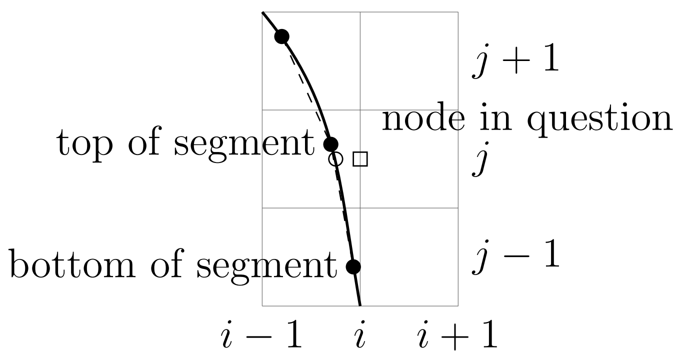


Figure 2: To find the hybrid and ghost nodes the position of the body must be translated from the Lagrangian body nodes to the Eulerian grid. The solid line is the real body which is represented by the Lagrangian body nodes (solid circles). The dashed lines are drawn between the body nodes and represent the discretized body.

determine what the node is, five x-coordinates are used: x_{i-1}, x_i, x_{i+1}, x_{BI}, and $x_{\text{body center}}$. Using Figure 2 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.

2.2.1. Weighting

As the body moves across the background grid, nodes change how they calculate values. For example a bulk-fluid u-velocity node at time step t^n could become hybrid node at time step t^{n+1}. As the node transitions, the intermediate velocity is first calculated by Navier–Stokes (Equation (3)), then with interpolation (Equation (8)). Both equations are correct representations for the intermediate velocity, but have different errors, causing the calculated velocities to be slightly different. This discontinuous change is responsible for numerical oscillations.

In the method proposed by Luo et al. [8], hybrid node values are calculated via both Navier–Stokes and interpolation, and a weighted combination smooths 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.

The weighting scalar α is introduced to the solutions for intermediate velocity and the pressure:

$$\theta = (1-\alpha)\,\theta_{\text{Navier-Stokes}} + \alpha\theta_{\text{Interpolated}}, \tag{6}$$

where θ 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 α, it is assumed that each hybrid node has at most two neighboring ghost nodes (this will not be true for sharp corners):

$$\alpha = \sqrt{\left(\frac{\Delta_1}{\Delta x}\right)^2 + \left(\frac{\Delta_2}{\Delta y}\right)^2}, \tag{7}$$

where Δ_1 and Δ_2 correlate to GN_1 and GN_2, respectively, and Δx and Δy are the grid spacing in the x and y directions, respectively. As described by Luo et al. [8] and shown in Figure 3, Δ_1 and Δ_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.

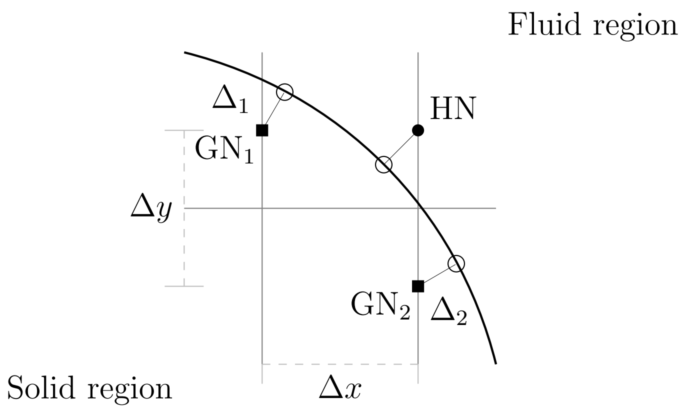


Figure 3: An example grid labeling values needed to calculate α for a u-velocity hybrid node. The body intercepts (open circles) are found by projecting from the hybrid or ghost node along the surface normal. Δ_1 and Δ_2 are the distance between the ghost nodes and their respective body intercepts.

2.2.2. *Field extrapolation to ghost nodes*

To use the Navier–Stokes equations at hybrid nodes, velocity and pressure must be extrapolated across the body to the ghost nodes using boundary conditions such that the presence of the immersed body is imposed. No-slip is used for velocity and the pressure boundary condition is approximated by assuming a constant change in pressure along the surface normal. 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.

Luo et al. [8] extrapolated field values across the body using 2D bilinear interpolation. cuIBM-FSI uses a staggered grid, causing the extrapolation process to have three variations. The process will be described and shown for u-velocity nodes but is the same for v and pressure. Values are first interpolated for at an image point, represented by triangles in Figure 4, outside the body using the surrounding velocities and the boundary condition. The 2D bilinear interpolation approximates the velocity between four nodes surrounding the image point with Equation (8). Each node surrounding the image point is used to set up a system of four equations to solve for the coefficients of Equation (8), which can then be used to solve for the velocities. Some of the interpolation nodes are moved to account for the presence of the body. In Figure 4 the bottom left corner of each interpolation region is moved to the body intercept point.

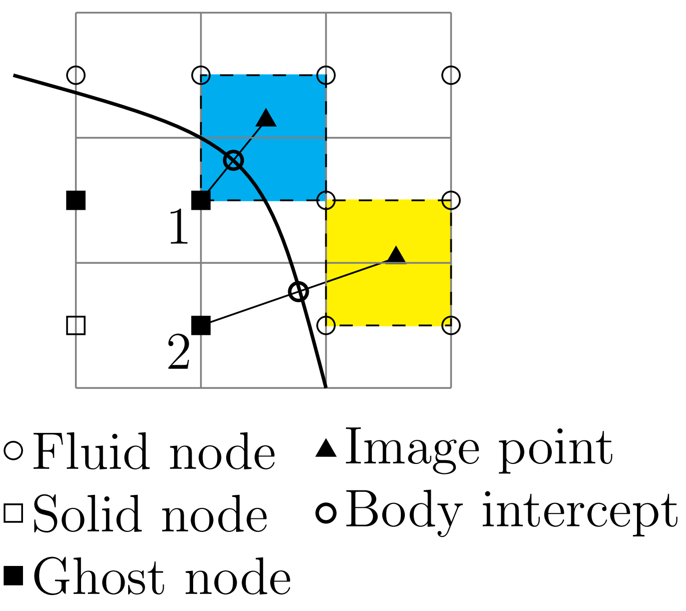


Figure 4: An example grid showing the relevant nodes for extrapolation of two different types of u-velocity ghost nodes.

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 4, 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 velocities 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 (10). Using the relocated corners, a system of equations is set up to solve for the coefficients in Equation (8). Pressure nodes on the body use Equation (9) in the system, which is simply the derivative of Equation (8):

$$\phi(x,y) = a_0 + a_1 x + a_2 y + a_3 xy \tag{8}$$

$$\phi(x,y) = a_1 + a_2 + a_3(x+y) \tag{9}$$

$$\left.\frac{\partial p}{\partial \mathbf{n}}\right|_{BI} = -\rho \frac{D\mathbf{u}}{Dt} \cdot \hat{\mathbf{n}}\Big|_{BI}, \; . \tag{10}$$

where ϕ 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; ρ is the density; $D\mathbf{u}/Dt$ is the material derivative; and $\mathbf{n}$ is the unit vector normal to the body. Solving this system on the GPU requires direct inversions of the 4×4 matrices. Once the a coefficients have been found, the velocity at the image point can be calculated with Equation (8) and extrapolated across the surface using Equation (11) for velocity or (12) for pressure:

$$\mathbf{u}_{GN} = 2\mathbf{u}_{BI} - \mathbf{u}_{IP} \tag{11}$$

$$p_{GN} = p_{IP} - \Delta l \left.\frac{\partial p}{\partial \mathbf{n}}\right|_{BI}, \tag{12}$$

where the subscripts GN, BI, and IP indicate ghost nodes, body intercept, and image point, and Δl is the distance from ghost node to image point.

2.2.3. Interpolation to hybrid nodes

Interpolating for the field value at the hybrid node is largely the same as the ghost nodes procedure 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 5. For a hybrid node, an image points purpose is only to locate 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 (8) is solved, the hybrid node is interpolated for.

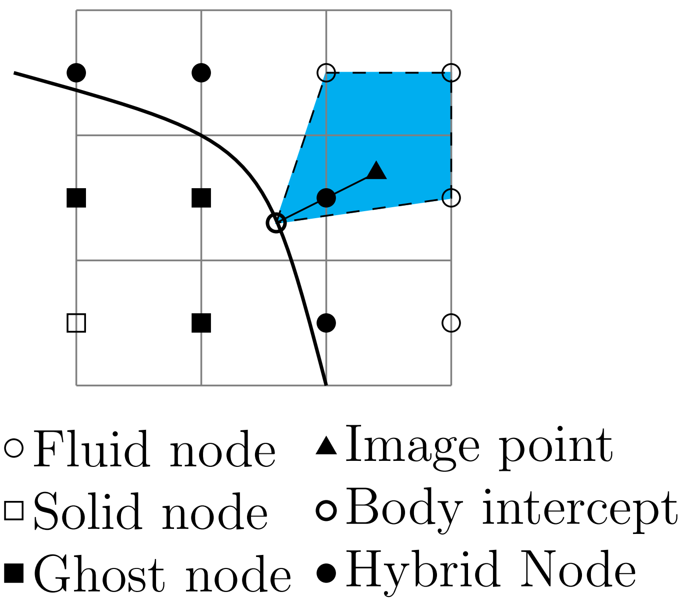


Figure 5: An example grid showing the relevant nodes for interpolation to a u-velocity hybrid node.

2.2.4. External method

cuIBM-FSI has two methods based off the approach given by Luo et al. [8]. The key difference between the two approaches is when the extrapolation and interpolation takes place. 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 (3) or

(4) and (8) into Equation (6). The discretization of these can be seen in Section 2.2.5. 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.

Two key implementation details distinguish cuIBM-FSI from Luo et al. [8]. Firstly, Luo et al. did not use a staggered grid. Secondly, Luo et al. used Crank–Nicholson to discretize both the advection and diffusion terms, whereas cuIBM-FSI uses Adams–Bashforth for the advection terms.

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 (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.
14. Advance time.

2.2.5. Embedded interpolation method

In the intermediate velocity sub-step, 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 as they are only dependent on the geometry, not velocity or pressure. The equations for bilinear interpolation, Equation (8), and interpolating to the ghost node, Equations (11) or (12), 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 ϕ in Equation (17):

$$u(x,y) = a_0 + a_1x + a_2y + a_3xy. \tag{13}$$

Each bilinear interpolation requires four data points to solve as discussed above. They will be labeled roughly according to their cardinal direction to the interpolation location. The resulting system of equations is laid out below in (**??**).

$$\underbrace{\begin{bmatrix}1 & x_{sw} & y_{sw} & x_{sw}y_{sw}\\ 1 & x_{se} & y_{se} & x_{se}y_{se}\\ 1 & x_{nw} & y_{nw} & x_{nw}y_{nw}\\ 1 & x_{ne} & y_{ne} & x_{ne}y_{ne}\end{bmatrix}}_{\mathbf{A}}\underbrace{\begin{bmatrix}a_0\\ a_1\\ a_2\\ a_3\end{bmatrix}}_{\mathbf{a}} = \underbrace{\begin{bmatrix}u_{sw}\\ u_{se}\\ u_{nw}\\ u_{ne}\end{bmatrix}}_{\mathbf{u}}. \tag{14}$$

Equation (15) gives the exact inverse of matrix $\mathbf{A}$:

$$\mathbf{A}^{-1} = \frac{\mathbf{B}}{\det \mathbf{A}} \tag{15}$$

$$\mathbf{a} = \mathbf{A}^{-1}\mathbf{u}. \tag{16}$$

Matrix $\mathbf{B}$ is based on the values of $\mathbf{A}$. The inverse is used to solve for the $\mathbf{a}$ coefficients, Equation (16). 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 (17):

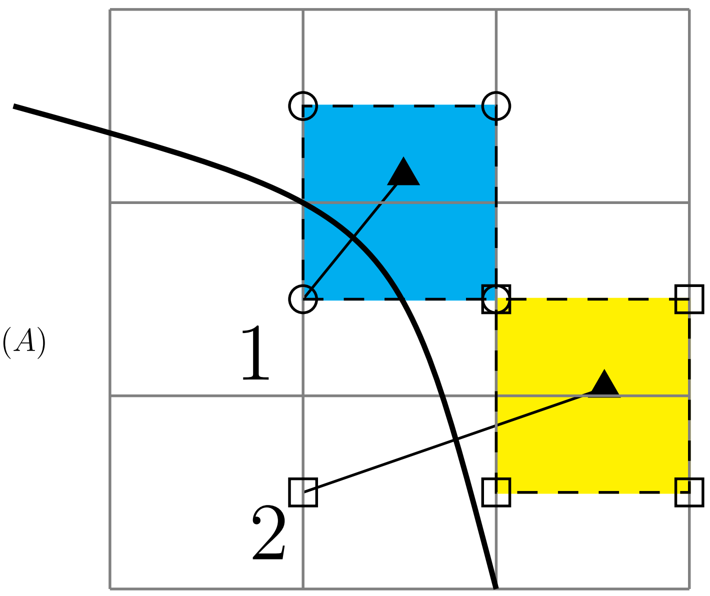
$$u_{GN} = 2u_{BI} + C_{nw}u_{nw} + C_{ne}u_{ne} + C_{sw}u_{sw} + C_{se}u_{se} \tag{17}$$

The array C holds the velocity coefficients resulting from (16) being manipulated. Subscripts indicate the row and column of the matrix. The calculation of C, Equation (18), is the portion of the bilinear interpolation that is performed before the linear algebra step:

$$\begin{bmatrix} C_{sw} \\ C_{se} \\ C_{nw} \\ C_{ne} \end{bmatrix} = \begin{bmatrix} b_{11} + b_{21}x_{IP} + b_{31}y_{IP} + b_{41}x_{IP}y_{IP} \\ b_{12} + b_{22}x_{IP} + b_{32}y_{IP} + b_{42}x_{IP}y_{IP} \\ b_{13} + b_{23}x_{IP} + b_{33}y_{IP} + b_{43}x_{IP}y_{IP} \\ b_{14} + b_{24}x_{IP} + b_{34}y_{IP} + b_{44}x_{IP}y_{IP} \end{bmatrix} / \det(A) \tag{18}$$

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 6a 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. Ghost node two, which depends on the nodes marked with open squares, does not need any special treatment because it uses five nodes.

Figure 6 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.



(a) Stencil modification for ghost nodes. Ghost node 1 has a stencil that only uses four nodes and ghost node 2 has a stencil that uses five.

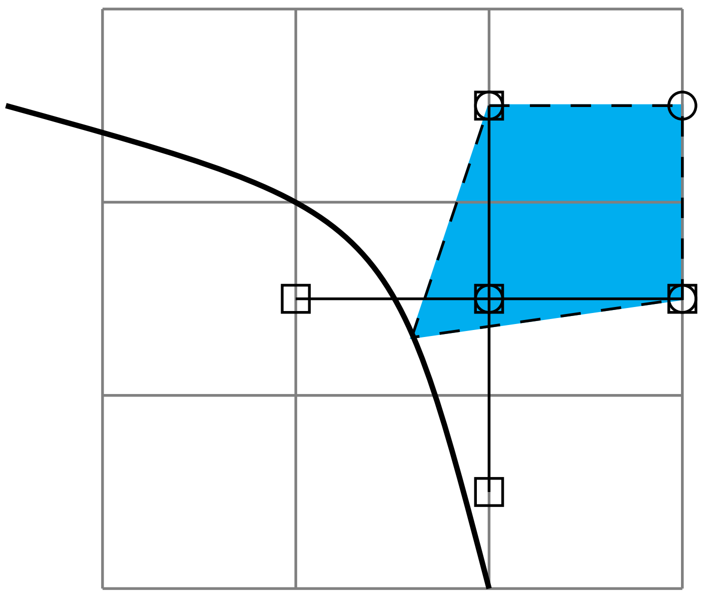

(b) The combination of Navier–Stokes (bold lines) and interpolation(shaded section) stencils.

Figure 6: The stencil of the left hand side matrices change when doing the interpolations inside the linear algebra step.

After the left-hand side matrix is made, it is sorted by row and column using a built-in CUSP function and 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, $\hat{\mathbf{u}}$ is calculated using CUSP.

The implementation of the embedded Poisson equation is the same as outlined above.

3. Verification and validation

Both methods are tested using several well studies simulations. The first simulation tests the solvers ability to handle a body with prescribed motion. In this test, a cylinder has a set sinusoidal motion in an initially stagnant flow. The second simulation features an impulsively started cylinder, with a set sinusoidal motion. Both of these tests 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 solvers' ability to accurately predict body location, velocity, and force when the body motion is not prescribed.

3.1. In-line oscillating cylinder driven flow

An in-line oscillating cylinder with initial velocities and Dirichlet boundary conditions of zero is used by Liao et al. [13] to demonstrate numerical oscillation suppression. The results of Liao et al. and cuIBM-FSI are validated against the experimental data of Dütsch et al. [17]. The equations that govern movement, (19) and (20), are the same form as above but with a period, maximum velocity, and position of 1, 1, and $1/2\pi$:

$$x(t) = \frac{-1}{2\pi}\sin 2\pi t \tag{19}$$

$$u(t) = -cos 2\pi t. \tag{20}$$

The diameter is matched to the Liao et al. [13] simulation, which is 0.2. These values give the simulation a Reynolds number of 100 and a KeuleganCarpenter 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(21):

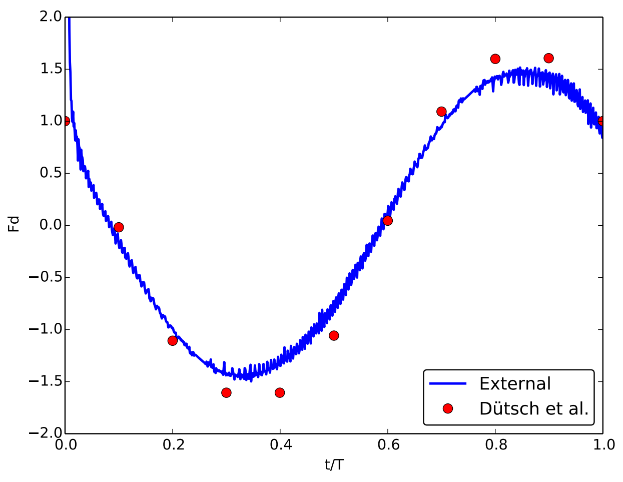
$$KC = \frac{U_{\max}}{fD}, \tag{21}$$

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 ± 5.5 in x to ± 3.5 in y. Figure 7 shows the drag during the first period versus the steady state drag from Dütsch et al. [17]. Figure 8 shows the steady state drag. In both figures, (a) is the external method and (b) is the embedded method. 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.

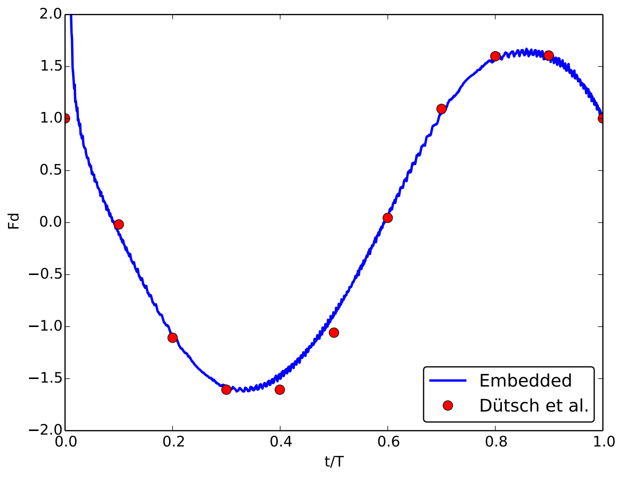
Figure 9 shows the validation against Dütsch et al. for a Reynolds number of 200 and a KC of 10. Note that this is comparing the drag dating during the 14th period. Dütsch et al. show force comparisons for various cycles into the low 100s. Re and KC can both be doubled by doubling the maximum velocity of the body which can be done by multiplying Equation (20) 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.

3.2. Impulsively started oscillating cylinder

The impulsively started oscillating cylinder is a simulation used by Luo et al. [8] to demonstrate and compare the magnitude numerical oscillations between different immersed boundary methods. An "impulsively started" simulation sets the initial conditions to mimic a body being 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 as the boundary layer develops. To simulate an impulsively started cylinder, 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

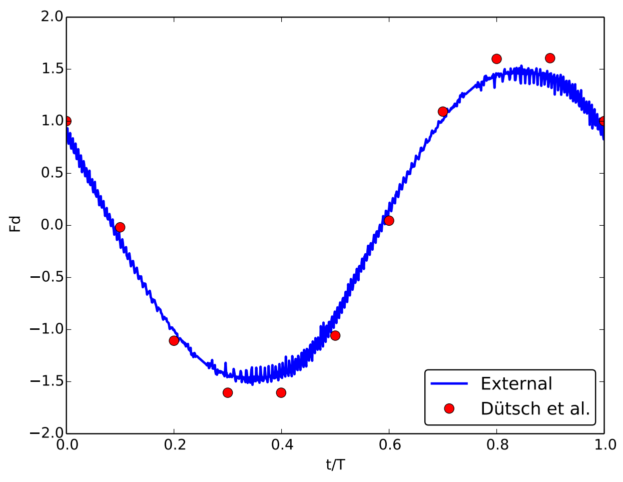


(a) External interpolation method.

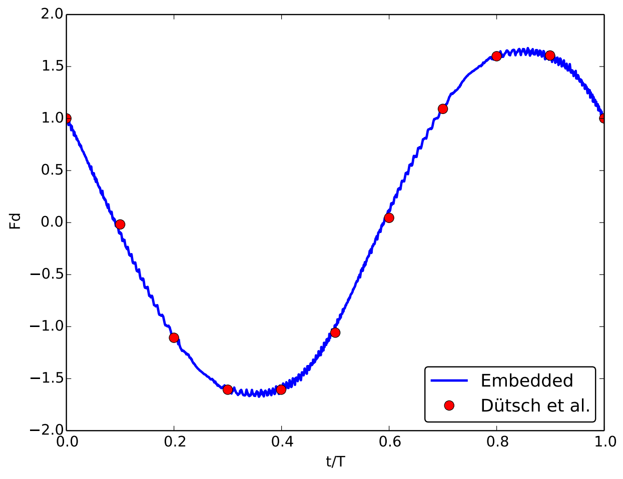


(b) Embedded interpolation method.

Figure 7: Drag force over the first period for an in-line oscillating cylinder flow compared to steady state Dütsch et al. [17], KC=5, Re=100.



(a) External interpolation method.



(b) Embedded interpolation method.

Figure 8: Drag force over a steady state period for in-line oscillating cylinder flow compared to Dütsch et al. [17], KC=5, Re=100.

forming an inlet and two free stream boundaries, respectively. A convective boundary condition is used at the exit. The grid has a uniform spacing in the area immediately around the immersed body which spans (-2,-2) to (2,2). The cells stretch as they moves away from the center until it reaches the domain's edge at $\pm$ 15.

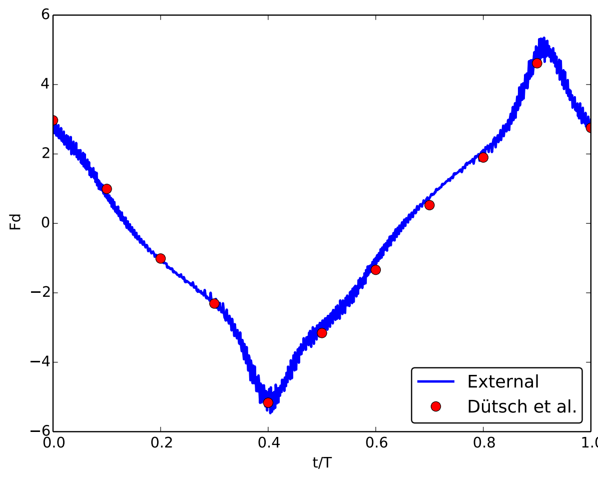
The cylinder oscillates along the horizontal centerline according to Equations (22) and (23):

$$x = -0.25\cos 0.4\pi t \tag{22}$$

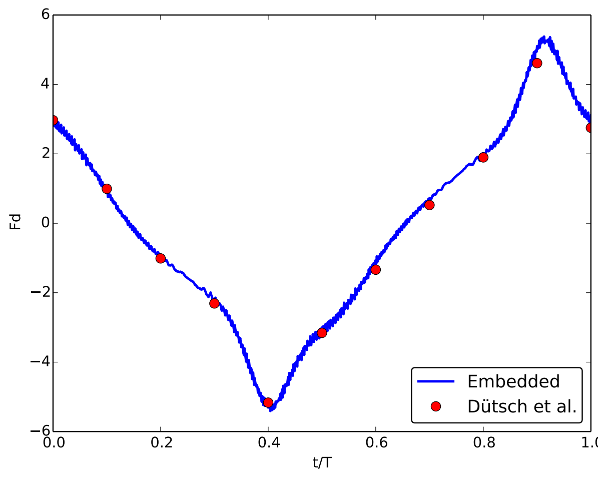
$$u = 0.1\pi \sin 0.4\pi t. \tag{23}$$

The resulting period, maximum velocity, and position are 5, $\pi/10$, and 0.25, respectively. Figure 10 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 64×64 grid for the first row and 128×128 grid for last three rows row 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.

Drag over the 14th period, KC=10, Re=200



(a) External interpolation method.



(b) Embedded interpolation method.

Figure 9: Drag force over the 14th period for in-line oscillating cylinder flow compared to Dütsch et al. [17], KC=10, Re=200.

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 more 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, 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.

3.3. 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 vortices—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. [18] the mass-spring damper system (24) that governs the body's movement is:

$$M\frac{\partial^2 Y}{\partial t^2} + C\frac{\partial Y}{\partial t} + KY = F_y, \tag{24}$$

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 (25) and (26):

$$\omega = 2\pi f = \sqrt{\frac{K}{M}} \tag{25}$$

$$C_{cr} = 2\sqrt{MK} = 2K\omega \tag{26}$$

Then Equation (24) can non-dimensionalized to become Equation (27):

$$\frac{\partial^2 Y}{\partial t^2} + 4\pi\zeta\frac{1}{U_{red}}\frac{\partial Y}{\partial t} + 4\pi^2\frac{1}{U_{red}^2}Y = \frac{1}{2M_{red}}C_Y, \tag{27}$$

where the non-dimensional coefficients are:
Damping coefficient

$$\zeta = \frac{C}{C_{cr}} \tag{28}$$

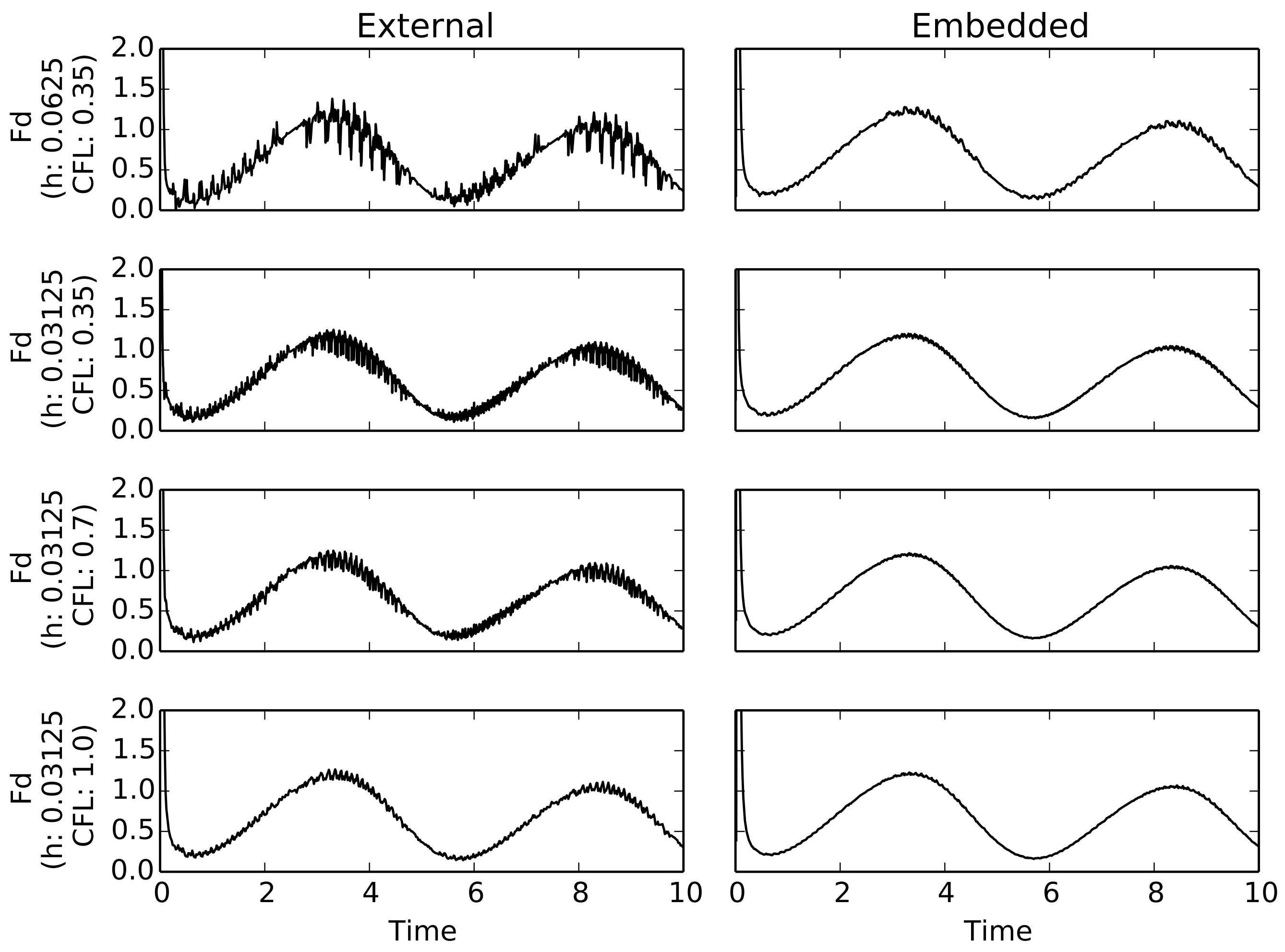


Figure 10: Drag force for flow over and in-line oscillating cylinder. The left column shows external interpolation and the right column shows the embedded interpolation. Rows one though four have maximum CFL 0.35, 0.35, 0.7 and 1.0 and a middle grid spacing of 0.0625, 0.0313, 0.0313 and 0.0313.

Reduced velocity

$$U_{red} = \frac{U}{fD} \tag{29}$$

Reduced mass

$$M_{red} = \frac{M}{\rho D^2} \tag{30}$$

Force coefficient

$$C_Y = \frac{2F_Y}{\rho U^2 D} \tag{31}$$

Varying the parameters Re, ζ, 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, ζ=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 (27). 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 (32) and (33):

$$v^{n+1} = v^n - \frac{\Delta t \pi^2 4}{U_{red}^2}\left(y^n + \frac{\Delta t v^n}{2}\right) + \frac{\Delta t C_Y}{(2M_{red})\left(1 + \frac{4\Delta t^2 \pi^2}{U_{red}^2}\right)} \quad (32)$$

$$y^{n+1} = y^n + \frac{\Delta t\left(v^{n+1} + v^n\right)}{2} \quad (33)$$

Equation (27) 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 (32) and (34), where α is some relaxation value.
4. If $|Y^{k+1} - Y^k| \approx 0$, advance time; otherwise advance sub step.

$$y^{k+1} = \alpha\left(y^n + \frac{\Delta t\left(v^{k+1} + v^n\right)}{2}\right) + (1-\alpha)\, y^k \quad (34)$$

Figure 11 shows 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. Results are verified against the curvilinear immersed boundary method of Borazjani et al. [18] and the unstructured finite-element approach of Ahn and Kallinderis [19]. 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 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.

4. Solver characterization

This section will evaluate the methods used in terms of error norms, order of accuracy, and performance scaling.

4.1. Impulsively started cylinder flow error norms

Figure 12 shows the L_1, L_2, and L_∞ error norms for the both immersed boundary methods tested using an immobile cylinder. The L_1, L_2 and L_∞ error norms are calculated using Equations (35), (36), (37), respectively.

$$L_1 = \sum_i \left|u_{\text{fine},i} - u_{\text{coarse},i}\right| \quad (35)$$

$$L_2 = \sqrt{\sum_i \left(u_{\text{fine},i} - u_{\text{coarse},i}\right)^2} \quad (36)$$

$$L_\infty = \max_i \left(u_{\text{fine},i} - u_{\text{coarse},i}\right). \quad (37)$$

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_∞ is the most intuitive of the error norms: it is simply the largest difference between the fine and coarse grids. An L_∞ 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. The external method has the lower error norms across the board.

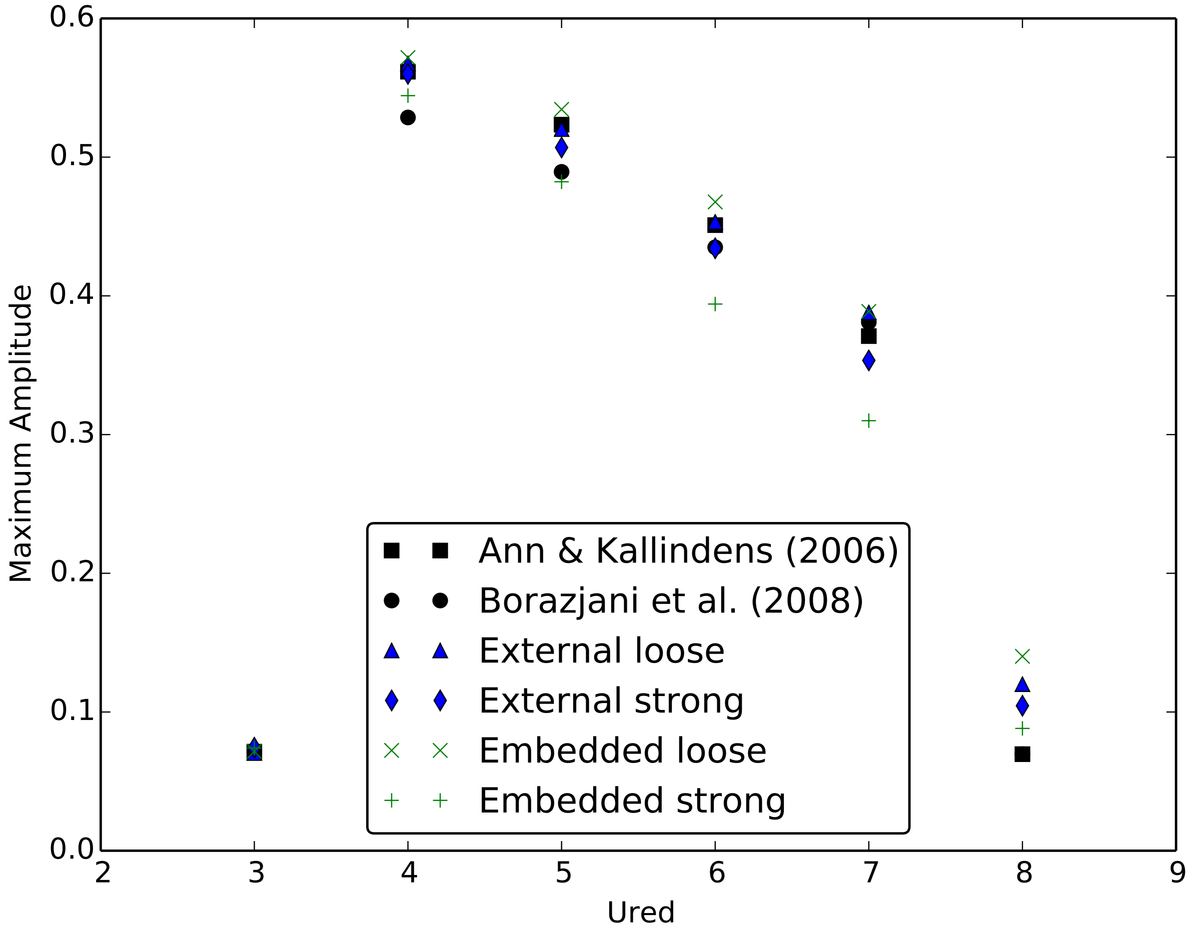
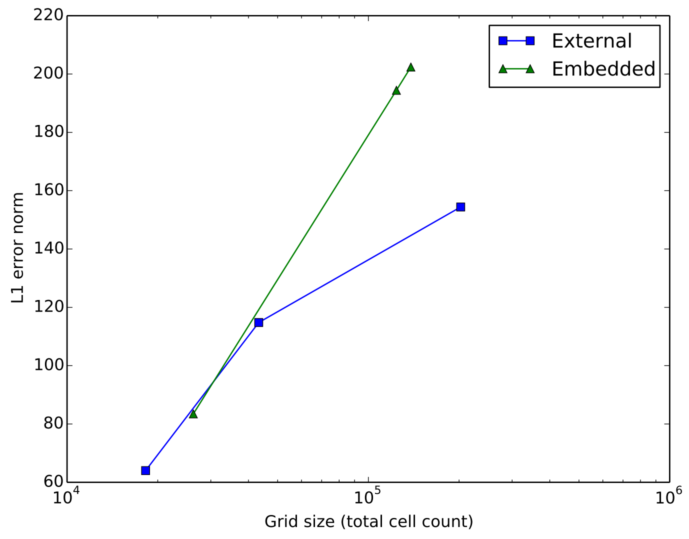
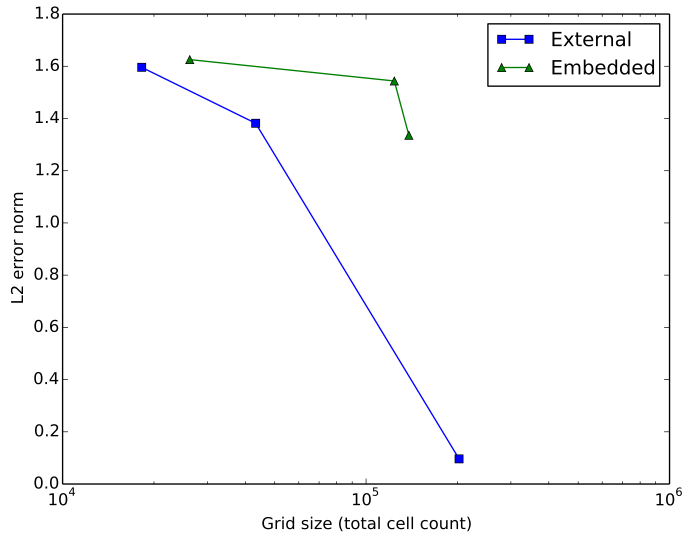


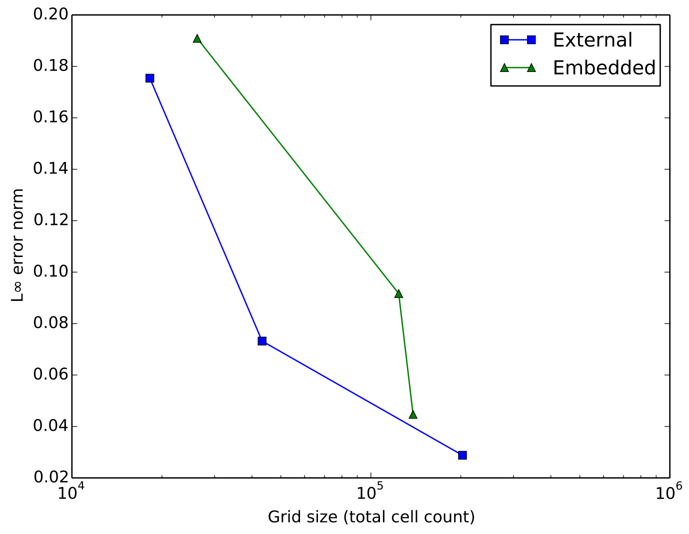
Figure 11: Maximum cylinder amplitude in the lock-in region as computed with the external and embedded methods using loose and strong coupling.



(a) L1 error norm.



(b) L2 error norm.



(c) L$_\infty$ error norm.

Figure 12: L_1, L_2, and L_∞ error norms for an impulsively started cylinder.

4.2. *Impulsively started oscillating cylinder order of accuracy*

The embedded and external methods also have their order of accuracy tested using flow over an in-line oscillating cylinder. Figure 13 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 3.2).

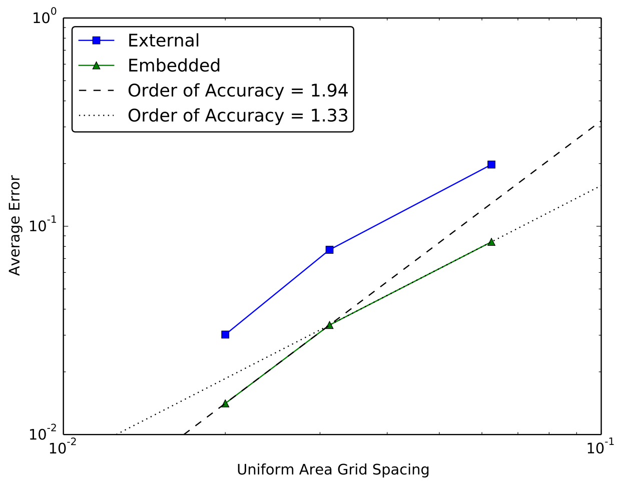


Figure 13: Error plotted against grid spacing shows the two solvers' order of accuracy for an oscillating cylinder in flow.

4.3. *Performance*

Performance is evaluated by comparing the total number of cells and the computational time. Most of these simulations are run twice on different GPUs. The NVIDIA K20 is a dedicated computational graphics card with nearly 2500 processor cores and the NVIDIA 750 ti is a less powerful card with 640 cores.

Figure 14 shows the performance of an immobile impulsively started cylinder. Computational time for both methods, across every type of simulation, is dominated by the Poisson equation linear algebra. The question then becomes: why does the Poisson solver take longer for the embedded method? Answering this question requires a look at the linear algebra multi-grid method. The

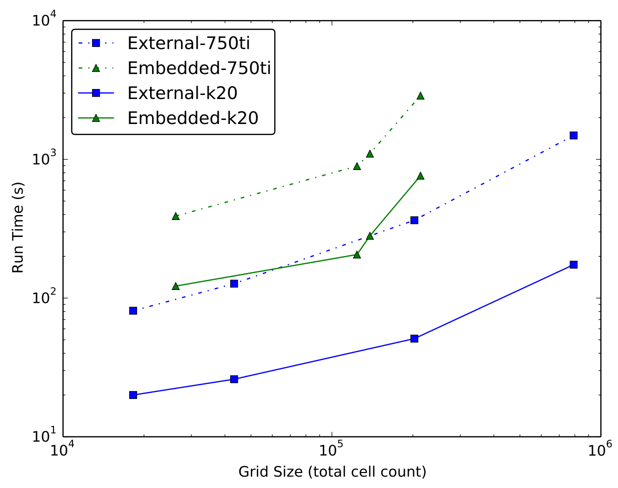


Figure 14: Impulsively started cylinder performance.

modified embedded methods use non-standard stencils near the body because it enforces boundary conditions inside the Poisson solver. 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 ends up being much faster because it uses the standard Poisson stencil for the entire domain.

It is worth noting the difference in slope between the solvers' performance curves. The external method is more shallow than the embedded method. This means that means that the embedded method is scaling poorly as the cell count increases.

Figure 15 the performance of an impulsively started 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. For the oscillating cylinder, smaller cell counts achieve a low (negligible) 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 more speedup.

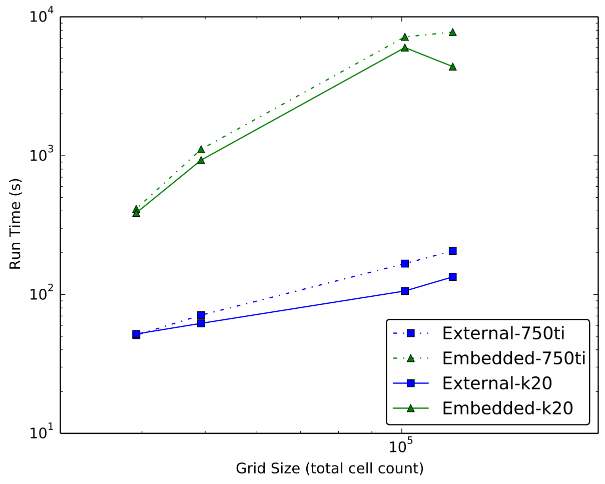


Figure 15: Performace for flow over an in-line oscillating cylinder.

Figure 15 is slightly misleading because 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. To verify that the computational time is dependent on center grid spacing the impulsively started oscillating cylinder simulation is run using three grid spacings of 0.0625, 0.05, and 0.03125 while keeping the overall cell count constant at 224×224 ($\approx$60k total) by modifying the stretch ratios. Figure 16 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%.

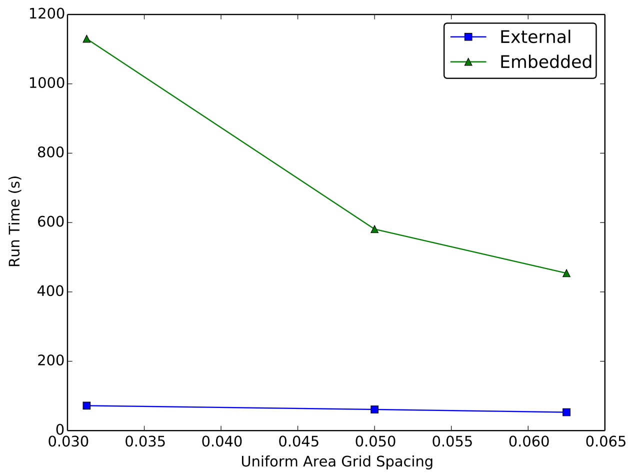


Figure 16: Computational time for flow over an in-line oscillating cylinder. The total cell count and uniform grid section size are held constant.

5. 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. [12]. 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. [8]. 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.

Acknowledgments

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 the cuIBM code.

appendix

highlights file for journal

References

[1] C. Minar, O. Mesnard, A. Krishnan, A. Schfer, L. Barba, K. E. Niemeyer, cuIBM: v0.1 release (Aug. 2016).

[2] C. Xu, X. Deng, L. Zhang, J. Fang, G. Wang, Y. Jiang, W. Cao, Y. Che, Y. Wang, Z. Wang, et al., Collaborating cpu and gpu for large-scale high-order cfd simulations with complex grids on the tianhe-1a supercomputer, Journal of Computational Physics 278 (2014) 275–297.

[3] X. Liu, Z. Zhong, K. Xu, A hybrid solution method for cfd applications on gpu-accelerated hybrid hpc platforms, Future Generation Computer Systems 56 (2016) 759–765.

[4] H. Norouzi, R. Zarghami, N. Mostoufi, New hybrid cpu-gpu solver for cfd-dem simulation of fluidized beds, Powder Technology.

[5] S. K. Layton, A. Krishnan, L. A. Barba, cuIBM–a GPU-accelerated immersed boundary method, arXiv preprint arXiv:1109.3524.

[6] K. Taira, T. Colonius, The immersed boundary method: A projection approach, J. Comput. Phys. 225 (2) (2007) 2118–2137. doi:10.1016/j.jcp.2007.03.005.

[7] A. Krishnan, L. A. Barba, Validation of the cuIBM code for Navier–Stokes equations with immersed boundary methods (2012).

[8] H. Luo, H. Dai, P. J. S. A. F. de Sousa, B. Yin, On the numerical oscillation of the direct-forcing immersed-boundary method for moving boundaries, Comput. Fluids 56 (2012) 61–76. doi:10.1016/j.compfluid.2011.11.015.

[9] R. Mittal, G. Iaccarino, Immersed boundary methods, Annu. Rev. Fluid Mech. 37 (1) (2005) 239–261. doi:10.1146/annurev.fluid.37.061903.175743.

[10] F. Sotiropoulos, X. Yang, Immersed boundary methods for simulating fluid–structure interaction, Prog. Aerosp. Sci. 65 (2014) 1–21. doi:10.1016/j.paerosci.2013.09.003.

[11] J. Mohd-Yusof, Combined immersed-boundary/B-spline methods for simulations of flow in complex geometries, Center for Turbulence Research Annual Research Briefs (1997) 317–327.

[12] E. A. Fadlun, R. Verzicco, P. Orlandi, J. Mohd-Yusof, Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations, J. Comput. Phys. 161 (1) (2000) 35–60. doi:10.1006/jcph.2000.6484.

[13] C.-C. Liao, Y.-W. Chang, C.-A. Lin, J. McDonough, Simulating flows with moving rigid boundary using immersed-boundary method, Computers & Fluids 39 (1) (2010) 152–167.

[14] J. B. Perot, An analysis of the fractional step method, J. Comp. Phys. 108 (1) (1993) 51–58. doi:10.1006/jcph.1993.1162.

[15] J. o'Rourke, Computational geometry in C, Cambridge university press, 1998.

[16] A. Krishnan, L. Barba, Validation of the cuIBM code for Navier–Stokes equations with immersed boundary methods, Tech. rep., figshare, July (2012).

[17] H. Dütsch, F. Durst, S. Becker, H. Lienhart, Low-Reynolds-number flow around an oscillating circular cylinder at low keulegan–carpenter numbers, Journal of Fluid Mechanics 360 (1998) 249–271.

[18] I. Borazjani, L. Ge, F. Sotiropoulos, Curvilinear immersed boundary method for simulating fluid structure interaction with complex 3d rigid bodies, Journal of Computational physics 227 (16) (2008) 7587–7620.

[19] H. T. Ahn, Y. Kallinderis, Strongly coupled flow/structure interactions with a geometrically conservative ALE scheme on general hybrid meshes, Journal of Computational Physics 219 (2) (2006) 671–696.