A Graph-Partitioned Sharp-Interface Immersed Boundary Solver for Efficient Solution of Internal Flows

Chi Zhu[a], Jung-Hee Seo[a], Rajat Mittal[a,*]

[a]*Department of Mechanical Engineering, Johns Hopkins University, Baltimore, MD, 21218, United States*

ARTICLE INFO

Article history:

ABSTRACT

In this Short Note, a graph-partitioning framework for a sharp-interface immersed boundary method is proposed so as to increase its computational efficiency for simulating internal flows on large-scale parallel computers. Immersed boundary methods are generally inefficient for internal flows with complex geometries due to the larger proportion of grid points that fall outside the fluid domain for such configurations. The graph-partitioning framework proposed here enables the solver to effectively ignore these points and focus the computation on the active points inside the fluid domain. A novel coarsening-partitioning process is proposed to ensure that sufficient overlapping layers are available at the sub-domain interfaces to accommodate computational stencils associated with the discretization as well as the sharp-interface boundary conditions. The benchmark test shows that the adoption of the graph topology reduces the computational cost (wall-time and memory cost) substantially. Moreover, the computational cost is shown to only scale with the number of computationally active grid points. The capability of the graph-partitioned solver is further demonstrated by simulating the flow inside an arterial network, a configuration which would otherwise be out of reach for most immersed boundary methods.

1. Introduction

Immersed boundary (IB) methods enable simulations of flow past bodies with complex shapes as well as moving/deforming bodies on stationary Cartesian grids. IB methods avoid the computational cost and complexities associated with the use of body-conformal grids, which can be particularly severe for boundaries that are moving and/or deforming [1]. Moreover, IB method solvers can be parallelized relatively easily by employing a Cartesian domain decomposition topology [2].

While the above features of IB methods have made them popular for simulations of external flows, their application to internal flows leads to some problems. Internal flow with geometrically complex boundaries are encountered in

*Corresponding author: E-mail address: mittal@jhu.edu (R. Mittal).

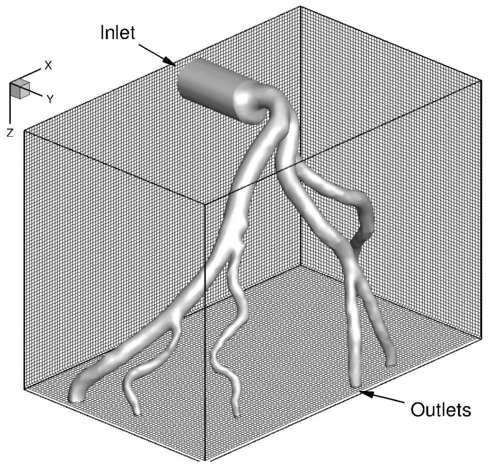

Fig. 1: Schematic of the computation domain for a patient-specific coronary artery network. Every seventh grid point in each direction is plotted here.

a variety of application areas, including but not limited to, physiology (e.g. blood flow in arteries and airflow in bronchial networks), chemical engineering (e.g. micromixing, bioreactors) and other configurations involving flow in curved or branching pipes and channels. IB simulations typically employ a single rectangular (or cuboidal in 3D) domain with a boundary non-conformal Cartesian grid covering this domain. A consequence of this is that only a subset of the total points of the grid lie in the fluid domain, while the rest fall outside in what we refer to here as the 'solid' domain. These grid points within the solid are in-principle associated with wasted computational cost. For external flows, the volume of the solid body is generally much smaller than the volume occupied by the flow and therefore, the proportion of these wasted solid points is small. In contrast, for internal flows, the points within the solid might account for a very large fraction (sometimes even the majority) of the total grid points, and represent a significant source of computational inefficiency for an IB method based solution. For example, consider the flow through the arterial network shown in Fig. 1, which is based on the computed tomography (CT) scan of a patient [3]. A uniform, isotropic grid with $650 \times 410 \times 468$ (i.e. about 125 million) points provides sufficient resolution (roughly 20×20 points across the narrowest branch) to resolve the details of the flow in the smallest artery of the network, but this leads to 98.22% of the total points being in the solid.

For diffuse interface methods [4], there is no particular distinction between the fluid and solid points since the same governing flow equations are solved for all the points in the computational domain, and a suitably localized body force is used to represent the effect of the solid on the fluid. Thus, for diffuse interface methods there is no simple way to reduce the collateral expense associated with these solid points, and the simulation of a flow such as in Fig. 1 would not be feasible using these methods without using strategies such as local-grid refinement [5]. On the other hand, for sharp-interface immersed boundary methods [6], except for, in some cases, a thin layer of points immediately adjacent to the solid boundary, the grid points within the solid are 'inactive' in that they have no bearing on the computed flow field. Thus, there is a possibility of designing algorithms that reduce or eliminate the overhead cost associated with these inactive points and make simulations such as those in Fig. 1 tractable using IB methods.

Most sharp-interface IB method solvers impose a trivial equation at these inactive points (such as $(\vec{u}, p) = (0, 0)$) and assemble this equation with the flow equations for the fluid points into one large sparse system of coupled equations for the entire set of grid points. While these inactive grid points do not affect the computed flow field, they do have a significant impact on the solution procedure and the computational cost. This is firstly because the sparse matrix solvers usually carry out the same operations for the grid points inside the solid as they do for points within the fluid region. Thus, the inactive solid points have a non-trivial contribution to the total floating-point operation (FLOP) count as well as the memory requirement whether the solver is serial or parallel. The situation further deteriorates when parallelization is taken into consideration. A Cartesian domain decomposition strategy is the obvious choice for parallelization (see Fig. 2(a)). However, even though each Cartesian sub-domain/process superficially contains nearly equal number of grid points, the number of active grid points (effective load) can vary significantly depending on the sub-domain topology and the geometry of the immersed body. For complicated geometries, there might be cases where many sub-domains consist entirely of inactive grid points leading to large imbalance of the effective load.

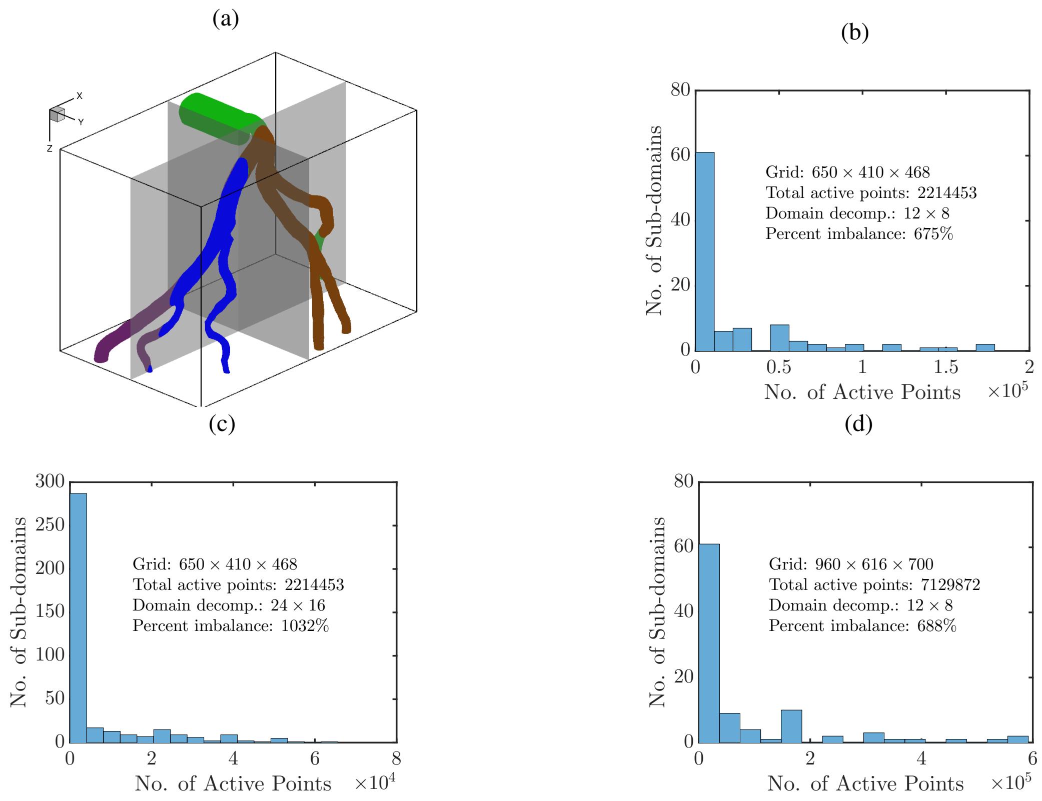

Fig. 2: (a) Example of a 2×2 (x direction $\times$ y direction) 2D domain decomposition using Cartesian topology. The gray planes depict the boundary of the parallel sub-domains; the active grid points in each sub-domain is shown in different colors. (b) Number density of active grid points in 96 sub-domains. (c) Number density of active grid points in 384 sub-domains. (d) Number density of active grid points in 96 sub-domains with increased grid resolution.

In order to further examine this issue, we consider again the arterial network geometry in Fig. 2(a). The 125 million point grid is decomposed into 96 (12×8) Cartesian domains with equal number of total grid points, but this results in a wide distribution of active grid points per domain (see Fig. 2(b)). As a matter of fact, 47 of the 96 sub-domains (i.e. 49%) do not contain any active grid points at all. We quantify the load imbalance by the parameter $\lambda = (L_{max}/\overline{L} - 1) \times 100\%$, where L_{max} and $\overline{L}$ are the maximum and mean load across the processes[7]. In the current context, the true computational load for each process is assumed to be proportional to the number of active grid points in this process. For the Cartesian decomposition in Fig. 2(b), the percent imbalance is 675%, indicating severe load imbalance. Moreover, as shown in Fig. 2(c), increasing the number of sub-domains to 384 (24×16) makes the problem worse, with 251 sub-domains (i.e. 65%) now containing no active grid points and $\lambda = 1032\%$. This issue is also not alleviated by increasing the grid resolution as shown in Fig. 2(d). Lastly, even in many of the domains that contain active points, a significant proportion of boundary communication between adjacent sub-domains involves 'useless' communication to and from inactive points, thereby further increasing the computational overhead.

Users of IB methods are aware of this problem and have come up with various strategies to circumvent this issue. Frankel and colleagues [8] developed a multiblock method and applied it to the flow inside a thoracic aortic aneurysm. Instead of using a single computational domain to cover the whole geometry, they employed a large number of smaller domains (blocks) to follow the overall shape of the geometry more closely. Each block was discretized using a uniform Cartesian mesh with the same resolution. However, the block sizes and locations had to be determined manually, although in Ref. [9], this process has been automated. Also, the number of inactive grid points is significantly reduced only when a relatively large number of blocks are used. For instance, Ref. [9] found it difficult to obtain high grid usage efficiency in a patient-specific cavopulmonary connection and the grid usage was only 40% when 1400 sub-domains were used. In a different approach, Zelicourt et al [10] discarded all of the inactive grid points by re-indexing the active grid points into a one-dimensional array, essentially converting the Cartesian mesh into an unstructured mesh. The advantage of this strategy was successfully demonstrated in a single processor

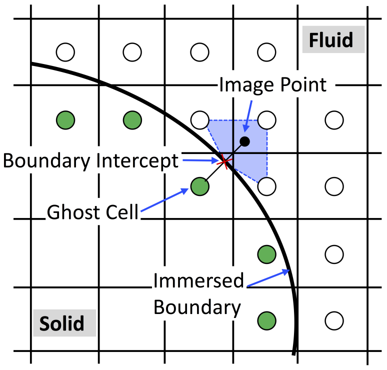

Fig. 3: Two dimensional schematic illustrating the ghost-cell method adopted in the current sharp-interface immersed boundary method.

implementation. However, this approach requires significant modification of any existing solver and furthermore, an efficient framework is still required to implement this method on large-scale parallel computing systems.

In this paper, we present an easy-to-implement, graph-partitioning approach for sharp-interface IB methods that can solve internal flow problems with complex geometries more efficiently on large-scale parallel computers. Compared with other methods, this method still operates on a single-domain Cartesian mesh and employs a well-established open-source tool to partition the active grid points for effective load balancing. Moreover, an effective precoarsening process is proposed to ensure that sufficient overlapping layers are available at the sub-domain interfaces to implement the discretization. The effectiveness of the method is demonstrated by simulating a small set of cases.

2. Methodology

The current approach can be viewed as a parallel framework to efficiently handle the underlying data structure for certain genre of immersed boundary methods. The proposed method is implemented with the ghost-cell method based sharp-interface immersed boundary method (ViCar3D) developed by Mittal et al. [6, 11], wherein the flow is governed by the incompressible Navier-Stokes equations. Though the solver is applied to 3D problems, the description in this section is mostly for a 2D configuration. Fig. 3 is an illustration of the procedure used in ViCar3D to implement the boundary conditions on the immersed boundary. First, all the grid points are separated into solid points and fluid points based on their location relative to the immersed boundary. Second, the ghost cells, which are points in the solid that have at least one neighbor in the fluid, are identified. Then, a normal probe is extended from the ghost cell into the fluid region, and an image point is defined such that the boundary intercept is the midpoint of the probe. Finally, a bilinear (or trilinear in 3D) interpolation is performed in the shaded area to obtain the value at the image point, and the boundary conditions are applied along the probe.

The key to solving internal flow problems more efficiently is to operate only on the active grid points, which, in the aforementioned sharp-interface IB method, are composed of the fluid points and the ghost points. The distribution of the active grid points closely shadows the shape of the immersed boundary, which may be highly irregular. To handle these irregular geometries, graph-partitioning with the open-source software package METIS [12] is adopted. METIS is widely used in the finite-element modeling community and is designed to deal with unstructured grids. It takes as input a graph constructed from the grid points and puts out partitions of these grid points that can be used in parallel computations. These partitions can be optimized to be well-balanced in total volume and have minimal communication interfaces. However, implementation of graph-partitioning within the context of the sharp-interface IB method requires some additional considerations, and these are described here.

The basic elements of a graph are the vertices and the edges. In this case, the vertices are the set of active points (i.e., the fluid and ghost points) while the edges represent connectivities between these active points. The existence of an edge between two vertices implies a coupling between these vertices. For a Cartesian mesh, the most common coupling is caused by the discretization of the derivatives. For instance, a second-order central scheme for the second derivatives in a two-dimensional grid couples the grid point with its neighbors along two axial directions. However, due to the ghost-cell method depicted in Fig. 3, the coupling can also occur along the diagonal directions. In order

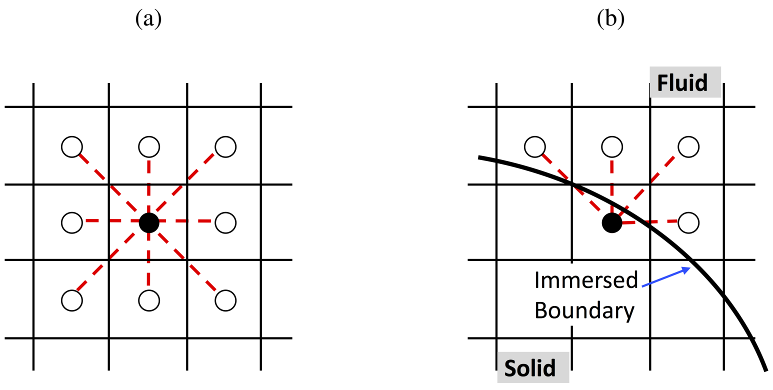

Fig. 4: Vertex (solid circle) and the edges (dash lines) it forms with its neighbors (circles). (a) Inside the fluid region; (b) near the immersed boundary.

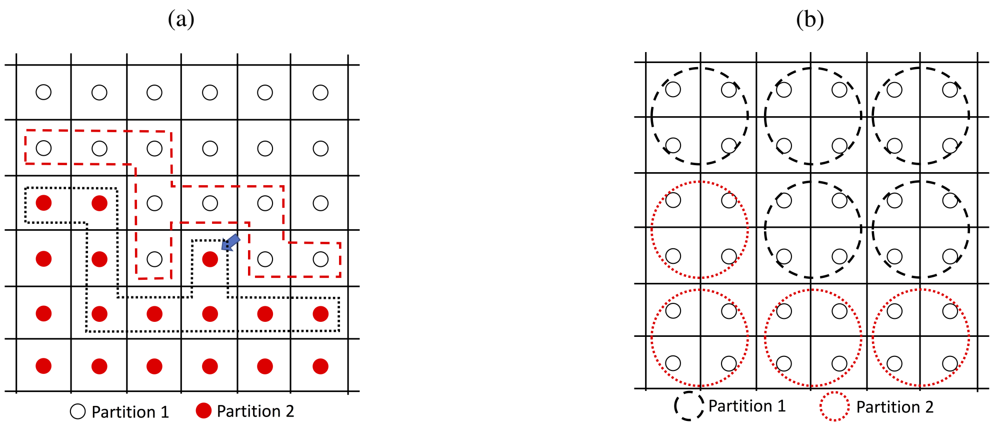

Fig. 5: (a) A zoom-in view at the interface between two partitions. Only two partitions are shown here and all the grid points are active grid points. The grid points inside the dash line are the first sub-domain overlapping layer of partition 2 from partition 1, and the grid points inside the dot line are the first sub-domain overlapping layer of partition 1 from partition 2. (b) Illustration of the coarsening-partitioning process when two sub-domain overlapping layers are required.

to apply the ghost-cell method correctly, the coupling (or edges) along the diagonal direction should also be included when present (see Fig. 4).

Once the graph constructed from the active grid points is supplied, METIS generates a map where each vertex is assigned a partition number. Fig. 5(a) shows an example of partition in 2D. Active grid points with the same partition number are grouped into one sub-domain, which is assigned to one MPI process. Layers of vertices that overlap adjacent partitions are usually required to ensure the accuracy and smoothness of the solution at the sub-domain interface. As shown in Fig. 5(a), a one-layer overlap between sub-domains is always well-defined via comparison of partition numbers of two edge-sharing vertices. However, multi-layer overlaps are required for reasons such as larger spatial discretization stencils [13], ghost-cell discretization when the grid cells near the interface have large aspect ratios, and cell merging in cut-cell type methods [14, 11], but these may not always be correctly defined in the current situation. For example in Fig. 5(a), a two-cell thick sub-domain overlapping layers would be ill-defined for the highlighted point in partition-2, since a second overlap layer for this point would intrude back into partition-2. Although such points constitute a very small proportion of the total number of interface points, they lead to large numerical errors and even numerical instability, if not treated properly. More complicated situations will arise in 3D problems or when the number of required overlapping layers increases, and this significantly complicates the implementation process.

To solve this problem, we have devised a coarsening process which is carried out before the partitioning step. Assuming that sub-domain overlapping layers of thickness N_{ol} are required, the original grid is first coarsened by a factor of N_{ol} in each direction; Fig. 5(b) shows an example of coarsening in 2D for $N_{ol} = 2$. Following this, the graph-

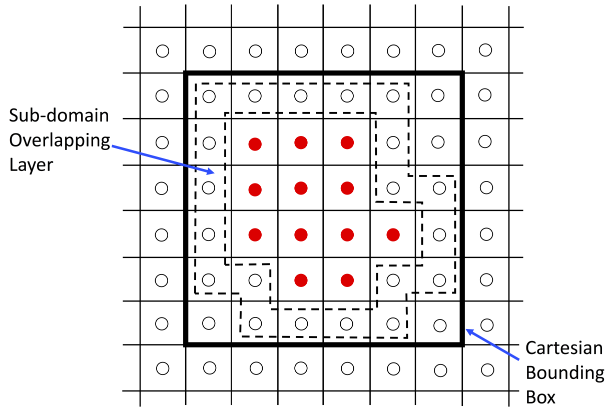

Fig. 6: An example of a Cartesian bounding box enclosing the partition and one sub-domain overlapping layer. The red solid circles represent the active points in the current partition, and the black circles are the active grid points from surrounding partitions.

partitioning is carried out on the coarsened mesh and the resulting partitioning is then simply applied to the original (uncoarsened) mesh. By following this procedure, ill-defined situations such as in Fig. 5(a) can be avoided, and well-defined and consistent sub-domain overlapping layers with sufficient thickness are guaranteed for all interface points. The grid points inside the sub-domain overlapping layers are assembled into a vector, which is then communicated to the neighboring partitions.

The graph partitioning leads to complex shaped sub-domains, and the usual practice is to employ an unstructured grid topology to address the point inside the sub-domain [15]. In the current graph-partitioned solver we avoid the complexities arising from this by retaining a structured (i.e. (i, j, k)) topology for each sub-domain. This is accomplished by defining a Cartesian bounding box that encloses each partition and its sub-domain overlapping layers (see Fig. 6 for example). The following pseudo code shows how the Cartesian bounding box is implemented with the graph topology. Here, (n_x, n_y, n_z) is the size of the Cartesian bounding box. Two benefits are provided through

```
1:  for i = 1 to $n_x$ do
2:      for j = 1 to $n_y$ do
3:          for k = 1 to $n_z$ do
4:              if point $(i, j, k)$ does not belong to current partition then
5:                  Skip
6:              else
7:                  Computation on point $(i, j, k)$
8:              end if
9:          end for
10:     end for
11: end for
```

this practice: first, matrix type data structure provides faster data access than the vector type in calculating multi-direction derivatives. Second, for legacy IB method solvers that employ Cartesian parallel topology, this practice greatly simplifies the transition from Cartesian to graph topology. However, these benefits come at the price of some extra memory usage, since 'dummy' points have to be added to fill the locations in the Cartesian bounding box that are not occupied by the active points associated with this sub-domain. Fortunately, METIS usually provides very high quality, compact partitions, and the extra memory usage is quite manageable in practice. Furthermore, since these dummy points carry no useful information, no computational operations are carried out for them.

It is noted that the adoption of Cartesian bounding box makes the current method very similar to the multiblock method developed by Frankel and colleagues [8, 9]. But there are several noteworthy differences. First, due to the irregular shape of the partitions generated by METIS in our approach, the sizes of the Cartesian bounding boxes would vary. Also, these boxes could partially overlap with their neighbors. Second, each block in the method developed by Frankel and colleagues [8, 9] behaves as a single Cartesian domain, which means computations are carried out on all grid points (active and inactive) in the block and data on the entire face are passed to neighbors of the block

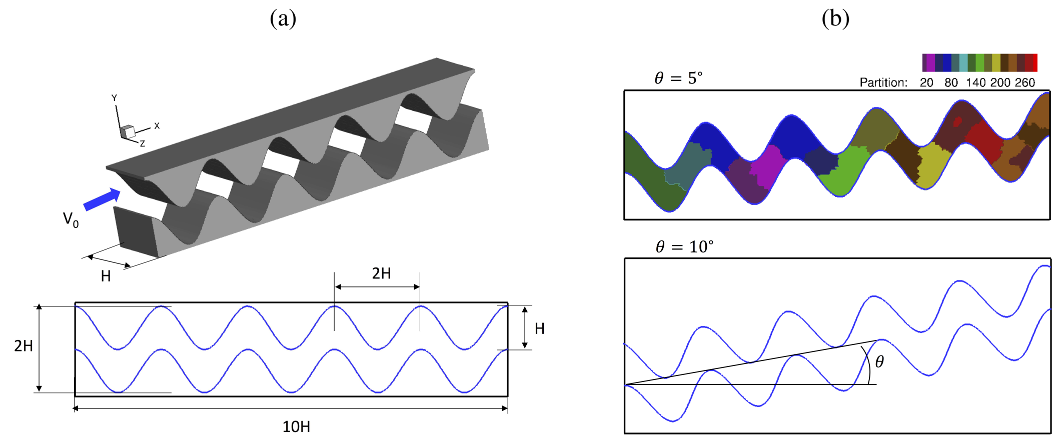

Fig. 7: (a) Schematics and dimensions of the baseline configuration of the wavy channel. (b) Schematics of the computational domain of the wavy channel rotated by 5° and 10°.

for parallel computing. Consequently, their method only achieves high grid usage efficiency when a large number of blocks are employed[9]. On the other hand, the current method remains efficient even when a small number of partitions are used, since the computation and communication are focused only on the active grid points.

In ViCar3D, the Navier-Stokes equation is solved by the fractional step method [16]. The advection-diffusion equation is solved with a point successive-over-relaxation (SOR) method, and the pressure Poisson equation is solved with a biconjugate gradient stabilized method (BiCGSTAB) that is tailored for the current ghost-cell method [17].

Though the proposed method is implemented in the sharp-interface IB method developed by Mittal et al. [6], it is worth emphasizing that it can be effectively applied in any flavor of IB method that has a separation between the solid and fluid points. The definition of the active grid points would vary depending on how the boundary conditions at the immersed boundary are enforced. Moreover, due to the irregular shape of the partitions, solvers designed for general sparse matrix systems (e.g. BiCGSTAB, GMRES) should be adopted. Other than this, the proposed method should require very limited modification of the solution procedure and code structure for any existing IB method solver.

3. Benchmark Test

The accuracy and versatility of the original sharp-interface IB method have been demonstrated previously in [6, 11]. In this section, we focus on the performance benefits the proposed framework provide for internal flow problems. The first test is flow through a wavy channel, the baseline geometry and dimensions of which are illustrated in Fig. 7(a). The wavy wall has a sinusoidal profile ($y = A\cos(2\pi x/\lambda)$) with wavelength $\lambda = 2H$ and amplitude $A = 0.5H$. The top and bottom walls have no phase shift and are separated to form a channel with height H. The channel size is half wavelength in the spanwise and five wavelengths in the streamwise direction. A uniform steady inflow V_0 is prescribed at the inlet, and a Neumann boundary condition is applied at the outlet. No slip and no penetration boundary condition is prescribed in the spanwise direction and on the top and bottom walls. The Reynolds number based on the channel height H and inlet velocity V_0 is 400, and this Reynolds number leads to a complex, fully three-dimensional unsteady flow inside the channel. This type of channel is a good test for the current method since it naturally introduces large regions with inactive points. In addition, to further demonstrate the efficiency of the current solver in handling even larger proportion of inactive points, the entire channel is rotated with angle θ to introduce even more inactive points while maintaining the fluid volume and appropriate inflow velocity condition. As shown in Fig. 7(b), two additional cases with θ equals 5° and 10° are simulated. Cartesian meshes with the same resolution are used to solve these three flows and the time-step for all the cases are $0.001H/V_0$. Thus, we generate a range of cases where the fluid domain and the flow field are almost the same but the sub-domain topology and the number of inactive points are quite different. Hence, these latter two factors are the only ones that can affect the comparative computational performance for these cases.

Further details about the simulations are listed in Table 1, where N_x, N_y and N_z represent the number of grid points in x, y and z direction, respectively. These three channels are solved using grids with the same resolution. It is apparent that as the angle increases, the total number of grid points as well as the number of inactive grid points

Table 1: Summary of the simulation and performance details for carrying out 10,000 time-steps for the different wavy channel configurations.

θ	(N_x, N_y, N_z)	Total Grid Points	Active Grid Points (Proportion of Total Grid Points)	Wall Time Per Active Grid Point (seconds)	Communication Cost	Total Allocated Memory (GB)	Percent Imbalance (λ)
0°	(640, 128, 80)	6,553,600	3,094,080 (47%)	2.61E-2	1.42%	252	6%
5°	(638, 176, 80)	8,983,040	3,084,560 (34%)	2.54E-2	1.48%	252	6%
10°	(630, 232, 80)	11,692,800	3,088,400 (26%)	2.56E-2	1.45%	252	6%
10°*	(630, 232, 80)	11,692,800	3,088,400 (26%)	4.86E-2	8.28%	342	289%

*Cartesian parallel topology with $36 \times 8 = 288$ decomposition along x and y direction

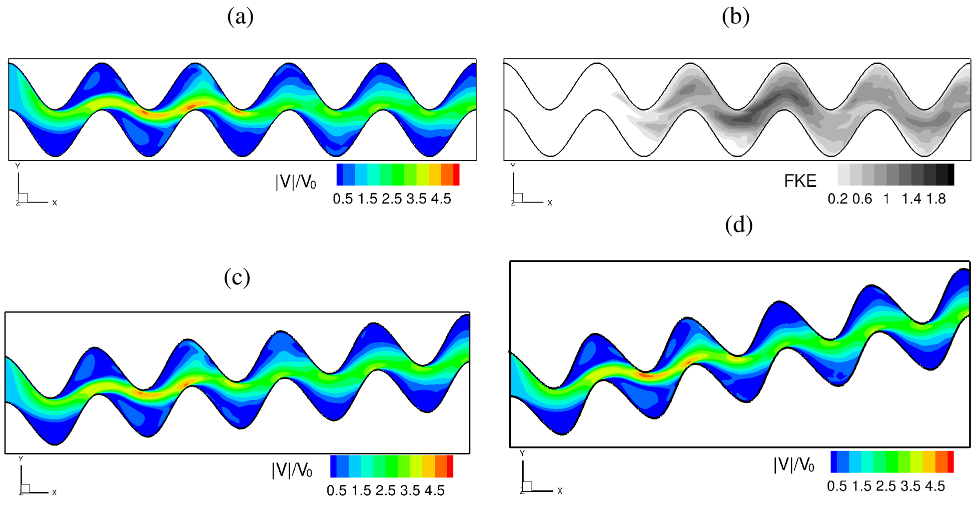

Fig. 8: Compute flow on the spanwise center-plane of the channel using the graph parallel topology. (a) Velocity magnitude based on the mean flow and (b) fluctuation kinetic energy (FKE) for $\theta = 0°$ channel. (c) Velocity magnitude based on the mean flow for the $\theta = 5°$ channel. (d) Velocity magnitude based on the mean flow for $\theta = 10°$.

increase rapidly, but the number of active grid points is nearly the same. The active grid points are partitioned into 288 sub-domains using METIS, and the simulations are carried out on TACC-Stampede2 with 288 Intel®Skylake Cores. The flow generated in these channels is transitional in nature, and all results shown here are accumulated over 10,000 time steps after the simulations become statistically stationary. Fig. 8(a,b) plot the velocity magnitude and the fluctuation kinetic energy at the center plane of the baseline case. The complex, unsteady nature of the flow ensures that the solver performs non-trivial computations at each time-step, thereby making this comparison a reasonable test of the solver efficiency. Fig. 8(c,d) plot the flow field for $\theta = 5°$ and $10°$. As can be seen here, the flow physics are very similar among the three cases and the flow field should therefore not be a factor in the performance comparison.

It is clear in Table 1 that the wall time per active grid point is quite similar among the first three cases, with the maximum difference being lower than 3% of the baseline case. This is a direct indicator that the inactive grid points have negligible effect on the speed of the simulation, indicating that they are effectively 'invisible' to the solver. The communication cost, which is measured as the percentage of total wall time that is spent on pure MPI communication (send/receive operations), is also maintained around 1.45% for all three cases. Moreover, the memory usage monitored through Intel®Vtune™Amplifier also shows negligible differences between the various cases that employ the graph partitioning. This further demonstrates that the solver is very efficient in ignoring the inactive grid points. The load is also well balanced when graph topology is adopted, with only 6% imbalance for all three cases. Also included in the table is the solver performance of the 10° case with the original Cartesian domain decomposition. It is seen that the allocated memory for this case goes up by 35% compared to the baseline case. It is noted that the memory consumption for the last two cases does not scale with the ratio of the active to total grid points. The reasons for this are that firstly, each processor stores a copy of the surface mesh that represents the immersed body, and secondly, there are some variables that are stored even for the 'dummy' points inside each Cartesian bounding box. It is also noted

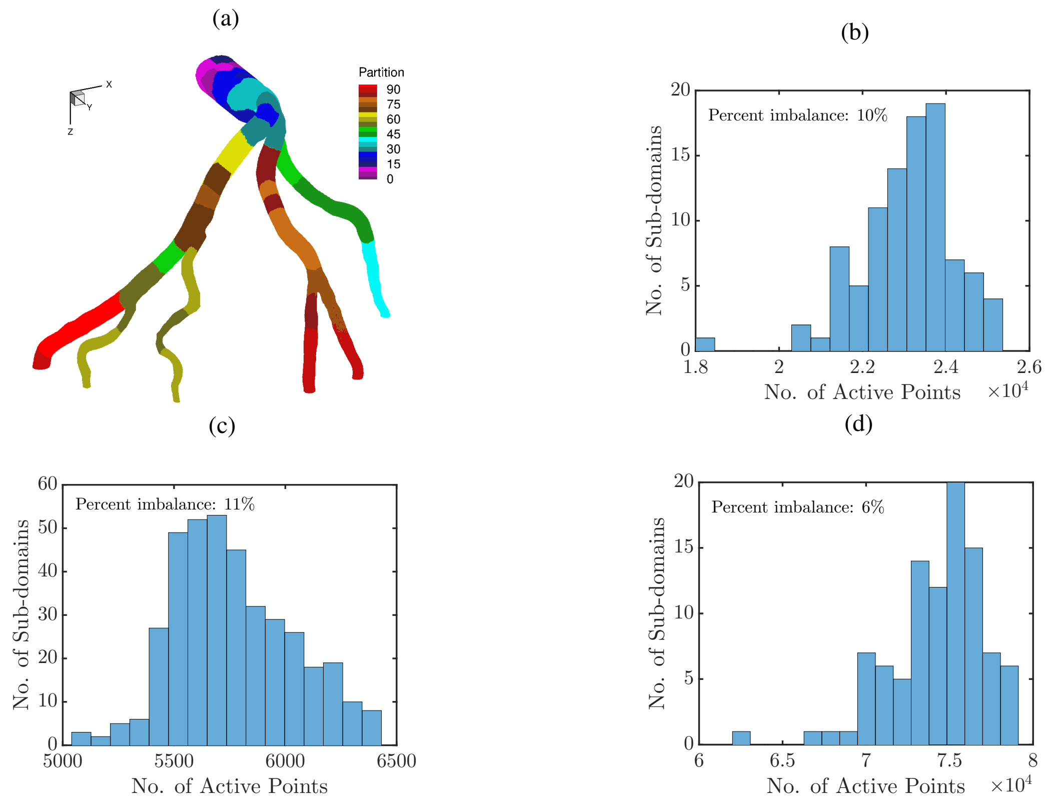

Fig. 9: (a) Results of the graph partitioning for the coronary arteries with 96 partitions and 2 sub-domain overlapping layers. (b) Number density of active grid points in 96 sub-domains. (c) Number density of active grid points in 384 sub-domains. (d) Number density of active grid points in 96 sub-domains with increased grid resolution. The grid resolution, number of active points and number of domains in (b-d) are the same as those in Fig. 2(b-d).

that the wall time per active grid point goes up by 86%, and the communication cost increases by 984%. The percent imbalance of the Cartesian topology is as high as 289%, representing a highly imbalanced computing load. Thus, even for this relatively simple, nominally 2D internal flow case, the current method provides significant advantages.

It is worth noting that the graph partitioning here is performed as a pre-processing procedure, and the partition information is supplied to flow solver as an input. Hence, the computational cost of the graph partitioning procedure is not factored into the overall computational cost. The partitioning process is however very fast, and the wall time required for partitioning this specific problem is equivalent to $O(100)$ time-steps for the flow solver.

4. Flow through an Arterial Network

One of the major applications for the upgraded solver is the modeling of cardiovascular flows. Fig 1 shows a section of the coronary artery reconstructed from the computed tomography (CT) scan of a patient[3]. This arterial network shows several small arteries branching from the same primary vessel. As noted earlier, such a geometry is virtually out of reach for all immersed boundary type methods and here, we use this configuration to demonstrate the feasibility of simulating such flows with the current graph-partitioned immersed boundary method.

In the current simulation, a constant pressure drop of 3mmHg (399.97Pa) is prescribed between the inlet and all the outlets. This is within the range of physiological conditions for the coronary artery and results in a Reynolds number of about 450 for the largest artery [3]. In order to provide reasonable resolution of the flow in the smallest arteries of this network (approximately 20×20 grid across the cross-section of the smallest artery), the computational domain is discretized by 650, 410 and 468 grid points in x, y and z directions, respectively. This leads to around 125 million grid points in total, which make this problem very challenging for typical IB methods that employ Cartesian parallel topology. However, only around 2.2 million (or 1.78%) of the total grid points here are active, making this an excellent case to apply the proposed methodology.

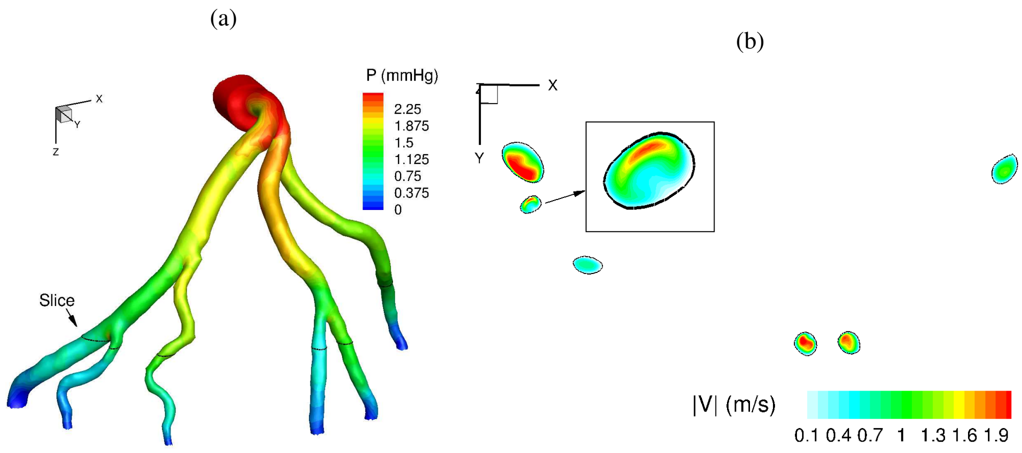

Fig. 10: (a) Wall pressure distribution; (b) velocity magnitude in the slice shown in (a).

Following the procedure described earlier, 96 partitions are created and 23,067 grid points are assigned to each partition on average (see Fig. 9(a)). From Fig. 9(b), it can be noted that the load is reasonably well-balanced among sub-domains with $\lambda = 10\%$. Moreover, compared to the Cartesian topology shown in Fig. 2(c,d), the load balance is well maintained with either increased number of partitions or increased grid resolution as demonstrated in Fig. 9(c,d). During the simulation, an adaptive time-step is used to maintain a maximum CFL number of 0.75 while reducing the wall time needed to reach stationary state (which requires about 2500 time-steps). Fig. 10(a) shows the wall pressure distribution and a monotonic pressure drop is observed from the inlet to the outlet. The distribution of total velocity on the cross-section of all branches is plotted in Fig. 10(b), and we note that there are a number of small-scale features and sharp gradients in the contours of the narrowest branches, confirming the need for the relatively high grid resolution employed here.

5. Conclusions

A graph-partitioning framework for efficient simulation of internal flow problems with a sharp-interface IB method on parallel computers is presented in this Short Note. The solver is subjected to a carefully designed benchmark test, which clearly shows that the computational cost, including wall-time and memory usage, only scales with the number of active grid points. The effectiveness of the method is also demonstrated by successfully simulating flow in a network of branching arteries on a Cartesian grid that has a total of $O(10^8)$ points. This mesh size would effectively be out of reach of most immersed boundary methods, but the proposed method can reduce the computation effectively to the $O(10^6)$ active grid points, which is considerably more manageable.

Acknowledgments

The authors acknowledge support from NSF Grants IIS-1344772 and CBET-1511200, AFOSR Grant FA9550-16-1-0404, and computational resource by XSEDE TACC-Stampede through NSF Grant TG-CTS100002. The first author would like to thank Drs. Kourosh Shoele and Vijay Vedula for their valuable suggestions.

References

[1] R. Mittal, G. Iaccarino, Immersed Boundary Methods, Annual Review of Fluid Mechanics 37 (2005) 239–261.
[2] W. Gropp, E. Lusk, A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, The MIT Press, 2014.
[3] P. Eslami, J. H. Seo, A. A. Rahsepar, R. George, A. C. Lardo, R. Mittal, Computational Study of Computed Tomography Contrast Gradients in Models of Stenosed Coronary Arteries, Journal of Biomechanical Engineering 137 (2015) 091002.
[4] C. Peskin, Flow patterns around heart valves: a numerical method, Journal of Computational Physics 271 (1972) 252–271.
[5] B. E. Griffith, R. D. Hornung, D. M. McQueen, C. S. Peskin, An adaptive, formally second order accurate version of the immersed boundary method, Journal of Computational Physics 223 (2007) 10–49.
[6] R. Mittal, H. Dong, M. Bozkurttas, F. M. Najjar, A. Vargas, A. von Loebbecke, A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries, Journal of Computational Physics 227 (2008) 4825–4852.

[7] O. Pearce, T. Gamblin, B. R. De Supinski, M. Schulz, N. M. Amato, Quantifying the effectiveness of load balance algorithms, in: Proceedings of the 26th ACM international conference on Supercomputing, ACM, 2012, pp. 185–194.

[8] K. Anupindi, Y. Delorme, D. A. Shetty, S. H. Frankel, A novel multiblock immersed boundary method for large eddy simulation of complex arterial hemodynamics, Journal of Computational Physics 254 (2013) 200–218.

[9] Y. T. Delorme, M. D. Rodefeld, S. H. Frankel, Multiblock high order large eddy simulation of powered fontan hemodynamics: Towards computational surgery, Computers & Fluids 143 (2017) 16 – 31.

[10] D. de Zélicourt, L. Ge, C. Wang, F. Sotiropoulos, A. Gilmanov, A. Yoganathan, Flow simulations in arbitrarily complex cardiovascular anatomies - An unstructured Cartesian grid approach, Computers and Fluids 38 (2009) 1749–1762.

[11] J. H. Seo, R. Mittal, A Sharp-Interface Immersed Boundary Method with Improved Mass Conservation and Reduced Spurious Pressure Oscillations., Journal of Computational Physics 230 (2011) 7347–7363.

[12] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1998) 359–392.

[13] C. K. Tam, J. C. Webb, Dispersion-relation-preserving finite difference schemes for computational acoustics, Journal of Computational Physics 107 (1993) 262 – 281.

[14] T. Ye, R. Mittal, H. Udaykumar, W. Shyy, An Accurate Cartesian Grid Method for Viscous Incompressible Flows with Complex Immersed Boundaries, Journal of Computational Physics 156 (1999) 209–240.

[15] I. M. Smith, D. V. Griffiths, L. Margetts, Programming the finite element method, John Wiley & Sons, 2013.

[16] Y. Zang, R. L. Street, J. R. Koseff, A Non-staggered Grid, Fractional Step Method for Time-Dependent Incompressible Navier-Stokes Equations in Curvilinear Coordinates 114 (1994) 18–33.

[17] C. Zhu, J. H. Seo, V. Vedula, R. Mittal, A highly scalable sharp-interface immersed boundary method for large-scale parallel computers, in: 23rd AIAA Computational Fluid Dynamics Conference, 2017, p. 3622.