

# Multi-physics simulations of particle tracking in arterial geometries with a scalable moving window algorithm

1<sup>st</sup> Gregory Herschlag

*Biomedical Engineering and Mathematics Computational Science and Engineering Division*  
Duke University  
Durham, NC, U.S.A.  
gjh@math.duke.edu

2<sup>nd</sup> John Gounley

*Computational Science and Engineering Division*  
Oak Ridge National Laboratory  
Oak Ridge, TN, U.S.A.  
gounleyjp@ornl.gov

3<sup>rd</sup> Sayan Roychowdhury

*Biomedical Engineering*  
Duke University  
Durham, NC, U.S.A.  
sayan.roychowdhury@duke.edu

4<sup>th</sup> Erik W. Draeger

*Center for Applied Scientific Computing*  
Lawrence Livermore National Laboratory  
Livermore, CA, U.S.A.  
draeger1@llnl.gov

5<sup>th</sup> Amanda Randles

*Biomedical Engineering*  
Duke University  
Durham, NC, U.S.A.  
amanda.randles@duke.edu

**Abstract**—In arterial systems, cancer cell trajectories determine metastatic cancer locations; similarly, particle trajectories determine drug delivery distribution. Predicting trajectories is challenging, as the dynamics are affected by local interactions with red blood cells, complex hemodynamic flow structure, and downstream factors such as stenoses or blockages. Direct simulation is not possible, as even a single simulation of a large arterial domain with explicit red blood cells is currently intractable on even the largest supercomputers. To overcome this limitation, we present a multi-physics adaptive window algorithm, in which individual red blood cells are explicitly modeled in a small region of interest moving through a coupled arterial fluid domain. We describe the coupling between the window and fluid domains, including automatic insertion and deletion of explicit cells and dynamic tracking of cells of interest by the window. We show that this algorithm scales efficiently on heterogeneous architectures and enables us to perform large, highly-resolved particle-tracking simulations that would otherwise be intractable.

**Index Terms**—Lattice Boltzmann, Immersed Boundary, Heterogeneous computing, Cellular trajectory

## I. CELL AND PARTICLE TRAJECTORIES IN ARTERIAL FLOWS

Cell and particle trajectories in arterial systems determine both metastatic cancer locations [1] and how particulates, such as drugs or toxins, distribute throughout the body [2]. However, understanding and predicting such trajectories remains an elusive goal, as it is not currently tractable to track cells or particles over time *in vivo* on a patient-specific basis. Therefore accurate computational models are needed.

Such models must account for several key length scales that influence the overall trajectories. First, a model must capture the cellular length scale by accounting for any given cells' elastic and deformation properties. Second, a model must consider inter-cellular interactions occurring either through fluid structure interactions or extracellular proteins; modeling for these dynamics will require a sufficient (perhaps large) number of cells to be modeled. Third, the model must accurately capture the complex hemodynamics that may occur at length scales from the size of a cell to the width of an artery. Fourth, models must account for the tortuous geometries of a patient's vasculature to include effects of downstream geometric features such as an abnormal narrowing or widening of the vessel.

Coupling these length scales presents a significant technical challenge: the largest length scale requires a large computational domain and the smallest length scale requires a fine resolution. For example, modeling trajectories in a human aorta would require computational domain volume of approximately  $3 \times 10^4 \text{mm}^3$ , which is roughly  $1.5 \times 10^{12}$  times larger than the volumetric scale of a single red blood cell. If one assumes a

Research reported in this publication was supported by the Office of the Director, National Institutes Of Health of the National Institutes of Health under Award Number DP5OD019876. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. Compute time on summit was provided through the ORNL Director's Discretionary Program. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

lower physiologically relevant hematocrit of 30% and a fairly coarse finite element model for a red blood cell with 162 vertices (see, for example [3]), the memory consumption of the cells alone would be 1.5 petabytes. If we assume the fluid is coarsely resolved at the order of the finite element mesh (say  $1\mu\text{m}$ ), a single array D3Q19 lattice Boltzmann method would take up over 4 petabytes. This is more than the total amount of volatile memory on all of Summit [4].

The majority of existing computational models either use some type of coarsening, or resolve simple domains with complex cellular models. In general, the larger the arterial region, the coarser the model. At one extreme, models may reconstruct large regions of patient-specific arterial geometries and assess fluid flow in the system; however, these models entirely neglect complex cellular-scale interactions. Such simulations may either capture complex fluid dynamics through three dimensional computational fluid dynamics (e.g. [5]) or may coarsen the spatial domain into a single dimension (e.g. [6]). Other models consider continuum (e.g. [7]) or stochastic (e.g. [8]) closure relationships to determine the effect of cells. Yet other models consider coarse grained particle models for the cells themselves (e.g. [9]); in these systems a significantly smaller region of the arterial system is considered. Some models account for the red blood cells in detail however are computationally limited to resolve even smaller domains [10].

Linking disparate length scales has been a continual challenge that has either required coarsening and parameter fitting, or the inability to scale to large systems. Multi-scale models have shown a promising alternative to alleviate these issues of scale. For example in [11] a fine scale domain using dissipative particle dynamics was coupled to a coarse scale domain using Navier-Stokes to model cerebrovasculature aneurysms. To date, however, multi-scale models have not yet accounted for moving domains in which the fine scale region of interest is transported throughout the domain.

Although both works pushed computational scales to unprecedented levels, neither was capable of accurately tracking cell nor particle trajectories over long length scales. Other groups were able to simulate hundreds of millions of cells in the left coronary arterial tree [9], [12]; however, these works each pushed the computational storage limits of the time.

Cell trajectories are fundamentally statistical in nature, which means that any viable algorithm to understand such trajectories must (i) capture multiple computational realizations with reasonable computational expense, and (ii) be ready to be validated with more detailed and higher cost computational experiments. Furthermore, cells travel long distances relative to their size, meaning that any viable computational algorithm must have an appropriately fast time to solution.

In this work, we develop an multi-scale framework to accurately track cell or particle trajectories with in large arterial regions. The framework is an adaptive domain (or moving window), multi-physics algorithm set in the context of the lattice Boltzmann and immersed boundary methods. The moving window domain explicitly resolves cells; outside of the window blood is modeled as a Newtonian fluid. The window

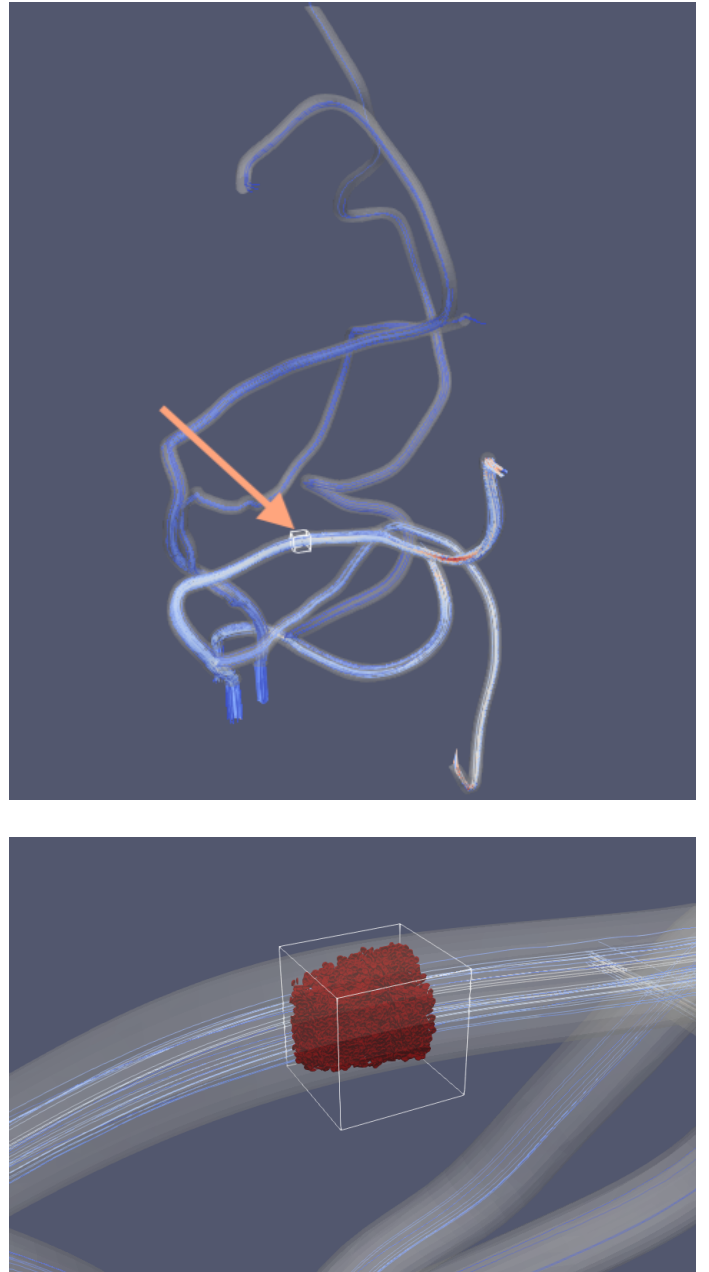


Fig. 1. We show a section of the cerebral vasculature with an embedded window pointed out by an orange arrow (top) shown with stream lines. We zoom in on a window domain with resolved red blood cells (bottom); a tracked cell (not shown) will lie within the window interior.

may either remain fixed so that one may study averaged cell trajectories in a specified point of the arterial network, or may move to track a particular cell. With this framework, we capture local cell dynamics, nearby flow complexity, and account for the long-range effects of the arterial tree. We also avoid the computational complexity of distant red blood cells which significantly reduces memory consumption, allowing for both larger domains to be resolved and for available computational resources to be dedicated toward fast time stepping rather than large system time steps (see Figure 1).

Computationally, we adopt a heterogeneous framework in order to both take advantage of more system resources and to account for the benefits of different architectures. For the large fluid domain we utilize GPUs, on which the lattice Boltzmann method is well suited [13]–[16]. In the smaller window domain, we utilize CPUs to handle both the lattice Boltzmann computation and the immersed boundaries. GPU performance of the immersed boundary method has been found to suffer due to the atomic adds required at each lattice point. In [17], [18], it was shown that allowing immersed boundary points to be updated on the CPUs and the bulk of the fluid to be updated on GPUs gave the best overall performance. In addition, the aggregate compute power of the CPUs in modern supercomputers is typically a small fraction of the total theoretical peak performance. Despite the higher computational cost of the explicit cells, the relatively small volume of the window compared with the full arterial geometry presents an opportunity to gain substantial scientific fidelity at little to no additional cost by using the computational power of these traditionally underutilized resources.

The moving window framework allows us to capture larger regions of the arterial system while simultaneously tracking how red blood cells will affect given cell or particle trajectories. When establishing the appropriate size of the window we may examine convergence studies by incrementally increasing the window size; then smaller window domain sizes may be used to gather a larger number of statistical trajectories. By decreasing the number of resolved cells throughout the system, more resources are available for faster system time integration, thus allowing us to resolve longer particle trajectories.

## II. CURRENT STATE OF THE ART

There has been a significant amount of work in high performance computing to simulate red blood cells and flow through the circulatory system. In 2010, red blood cells were resolved at petascale under direct numerical simulation of blood flow in a rectangular domain [10]. These simulations resolved up to 200 million deformable red blood cells run on roughly 200,000 cores with good weak scaling results. The first cardiac cycle was also simulated in the left coronary arterial tree including 300 million red blood cells [9].

In 2011, there were two major performance contributions in modeling cells in arterial flows: There was a multi-scale in which a platelets and red blood cells were modeled with dissipative particle dynamics in a  $4\text{mm}^3$  micro-domain; surrounding the domain was coupled to a Navier-Stokes solve exterior to the micro-domain and was shown to run at 90% parallel efficiency on 122,800 cores of an IBM Blue Gene/P supercomputer [11]. Additionally, hundreds of millions of coarsely resolved red blood cells were modeled in coronary flow using on 4000 GPUs [12].

In 2013, over a trillion of cells were resolved on 1.8 million threads on 147,456 cores [19]. The code showed excellent scaling results on complex geometries at sub-cellular resolutions. However, at the largest scale, the cells were updated less than twice per second with a corresponding physical time step

on the order of  $10^{-14}$ s or less. In short, even with excellent scaling, long time simulations at this scale remain untenable.

In 2015, arterial flows through-out a complete human arterial system (with vessels down to 1mm in diameter) were resolved with good strong and weak scaling results on over 1.5 million Blue Gene/Q cores at super-cellular resolutions [5]. Additionally, a significant advance was made in resolving massive scales of red blood cells, in which in 200,000 red blood cells were simulated in a micro-fluidic device at realistic experimental scales [20].

The majority of the above papers were either Gordon Bell finalists or Gordon Bell winners.

Despite these significant advances, simulating cell trajectories throughout arterial regions over time scales of interest has remained out of reach. One potential solution is to turn toward multi-scale simulations. A major advance in high performance computing of resolving cells in complex blood flow in a multi-scale context came in 2011 [11]; in this work a fine scale domain using dissipative particle dynamics was coupled to a coarse scale domain using Navier-Stokes to model cerebrovasculature aneurysms. The simulation ran on a cerebrovasculature; the micro-scale comprised a volume of  $12\text{mm}^3$  within which 3 billion unknowns were resolved with dissipative particle dynamics.

Dynamically initializing cells which are realistically deformed according to the local fluid dynamics remains a significant challenge. Most efforts have focused on handling initialization within special inflow and outflow regions, which allow sufficient time and space for artificially inserted cells to adjust to the flow [21], [22]. However, addressing this problem in an arbitrary three-dimensional setting requires a more general solution and, to the best of our knowledge, this problem remains unsolved.

In the current work we develop a multi-scale algorithm to track a cell of interest through the body. In developing the multi-scale approach we must develop techniques to (i) insert new red blood cells into the fine scale window domain which already holds a dense collection of red blood cells, and (ii) move the fine-scale domain in a way that accurately tracks the cell of interest while preserving relevant physical quantities across scales. We develop each of these techniques in the following section, and validate them in Section V. We then demonstrate scaling results and discuss computational implications of the adaptive system.

## III. A MULTI-PHYSICS MOVING WINDOW ALGORITHM

In this work, we introduce a moving window algorithm to address the aforementioned challenges. The primary challenges are inserting red blood cells into the window and moving the window when appropriate. We discuss our solutions to each of these hurdles in the current section, but begin by describing the algorithms used within each domain.

Exterior to the window, we must capture the aggregate blood flow; inside the window we must capture the flow of the plasma. To simulate these flows, we solve the Navier-Stokes equations with the lattice Boltzmann method. In the current

work, we use the D3Q19 lattice Boltzmann stencil with the the Bhatnagar, Gross and Krook (BGK) collision operator [23]. For boundary conditions at arterial walls we use half-way bounce back [24]; for inlet and outlet boundary conditions we use Zou-He velocity/pressure driven boundary conditions [25]. For more details on LB methods, see [26].

In the window, we also resolve cells. Cells are modeled with a finite element model which is coupled to the fluid via the immersed boundary method [27]. In this model cells are represented by a fluid-filled triangulated mesh, derived with successive refinements of an icosahedron. Using a finite element method, we model the cell surface as a hyperelastic membrane using a Skalak constitutive law in which elastic energy  $W$  is computed with the relation

$$W = \frac{G}{4} (I_1^2 + 2I_1 - 2I_2) + \frac{C}{4} I_2^2 \quad (1)$$

in which  $(I_1, I_2)$  are the strain invariants and  $G$  and  $C$  are the shear and dilational elastic moduli, respectively [28].

For red blood cells, we use the parameters  $G = 6 \times 10^{-6} \text{J}$ , and  $C = 6 \times 10^{-4} \text{J}$ . We track a spherical cell with a diameter of  $10 \mu\text{m}$ , and  $G = 10^{-5} \text{J}$  and  $C = 10^{-4} \text{J}$ . Both the red blood cells and the tracked cell are refined twice from an icosahedron giving 320 finite elements (or 162 vertices) per cell. The fluid and cells are coupled by a 2 point smoothing kernel taken from [29]. Force is imposed on the fluid domain by spreading a vertex point force on the finite element Lagrangian domain, and spreading it to the Eulerian fluid domain with the smoothing kernel; conversely, the Lagrangian velocities are interpolated from the Eulerian grid through a similar method [30]. For more details on the immersed boundary method and how it is coupled to the lattice Boltzmann method, see [31].

We now turn to our primary algorithmic contributions in coupling the dynamics across the multi-physics domains and in advancing the window with the tracked cell.

**Coupling scales and particle insertion.** To couple the window to the bulk flow, we must conserve relevant physical quantities across the domains to ensure the physics are appropriately coupled. The two quantities of interest we consider are momentum and particle flux, and ensure that they are preserved across scales. To account for the flux of momentum, the bulk and window domains are coupled via a shared fluid boundary (see Figure 3).

When resolving cells in the window at physiologically relevant levels of hematocrit, red blood cells are densely packed within the fluid domain. It is not trivial to introduce cells to a domain in which they are already prevalent as finding locations with which to introduce new cells may require that the introduced cells undergo extreme deformation in order to avoid overlapping with the existing cells. In order to overcome the issue of packing new cells in the window, we parametrize an ‘‘insertion’’ region with a length scale  $\ell_I$  (see Figure 2). Cells may enter this region from the window interior, and may leave this region by either traveling to the window interior, or exiting to the bulk domain; a cell will be deleted in the insertion region if it is within a certain distance,  $\delta_d$ , away from

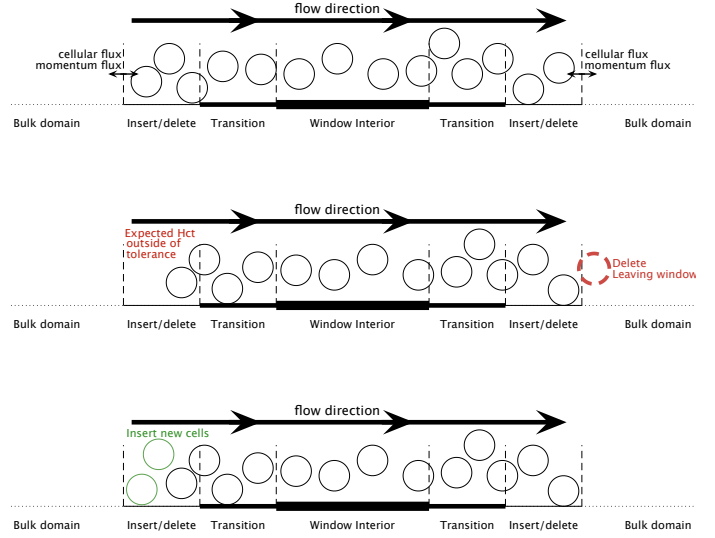


Fig. 2. A schematic of the window accounting for momentum and particle flux along with the coupled bulk, insertion/deletion, buffer, and interior regions (top). As cells flow through the domain, the concentration of cells in the insertion deletion decays; other cells are deleted (middle). Cells are replenished when the hematocrit falls below a given threshold (bottom).

the bulk/window boundary. Mean bulk cellular flux across the two domains may be tracked explicitly, as may be the flux from the window interior regions to both the bulk and the window interior. To enforce consistent particle flux, we balance the flux of cells in the bulk domain with that of the window domain. In the bulk, we assume constant hematocrit, which means that tracking the cellular flux may be determined directly by tracking the bulk material flux.

Different domains of the window interior, however, may experience different fluxes. To account for such differences, we break the window into multiple (overlapping) sub-domains and explicitly track the flux in each. We then set a tolerance in each of these sub-domains ( $\tau_i$  for the  $i$ th sub-domain). If the remaining number of particles in the sub-domain is roughly equal to the expected number of particles, within the tolerance  $\tau_i$ , we allow the simulation to progress. If/when the number of remaining particles is outside of the allowable tolerance, we insert pre-packed cells in the insertion region, and continue this insertion procedure until a minimal concentration is attained within the insertion region (see Figure 2). The pre-packing algorithm is based on ellipsoid packing [32] and the details of our implementation are provided in our previous work [33].

The sudden deletion and re-instantiation of cells within a transition region may generate some numerical artifacts. Because (i) our system is diffusive and (ii) we do not re-establish cells under extreme stresses, we hypothesize that such artifacts will decay over some length scale. We therefore introduce several length scales for the window: An interior length scale,  $\ell_{\text{int}}$  sets the sub-domain size of the window in which cell trajectories are assumed to be accurately resolved; a transition

region, with length scale  $\ell_t$ , separates the insertion/deletion region from the window interior and acts a buffer for numerical artifacts introduced within the insertion/deletion region. The transition region length scale for the entire window domain is a weighted sum of the insertion, transition, and interior length scales –  $\ell = 2\ell_I + 2\ell_t + \ell_{\text{int}}$ . The factors appear due to the regions appearing on both ‘sides’ of the window (see Figure 2). We save a careful study of these artifacts for future work, however we perform several validations of these techniques below.

**Moving the window.** When tracking a cell, the window must move and adapt to retain the cell within the interior. Conceptually, the window motion is separated from the time dynamics of the system – at any point in time the system treats the window as a fixed object. Between time steps, however, we may alter the window domain by re-centering at the position of the tracked cell. This window move occurs if any part of the tracked cell exits the interior of the window. Cells that have any part of them exterior to the window after the move are deleted; we assume that the aggregate momentum of the fluid with the cells is equal to the momentum of the fluid with the immersed boundaries, and thus do not alter the system momentum upon cell deletion (in the current work, cytoplasm is assumed to have the same density as blood plasma). With the cells gone, the complex flow structures now exterior to the window will gradually decay to the bulk dynamics. Conversely, cells introduced into the new regions of the window will begin with the bulk momentum, but will then begin forcing the fluid to retain their shape and introduce the more complex dynamics (see Figure 3).

In this work, we assume that bulk fluid always has uniform hematocrit. Due to this assumption, cells are established in these new regions with the same insertion method used to enforce flux (see above). Depending on the size of the interior and buffer domains, the updated interior of the window may be instantiated from the bulk/generated cells. The buffer domain combined with the insertion/deletion region should provide a large enough length scale for the new interior domain to relax into the correct dynamics.

To minimize window movements, it may be possible to estimate the cell trajectory with the bulk domain and then re-establish the window domain in such a way as to maximize the time a tracked cell will remain in the window interior. We omit this feature in the current work and simply re-establish the window center at the centroid of the tracked cell.

#### IV. IMPLEMENTATION

Our implementation builds off of the HARVEY code base, which has been shown to scale efficiently on millions of CPU cores [34] and thousands of GPUs [16]. Recently, the immersed boundary framework has been integrated in to HARVEY [31], and we utilize this feature in the window interior.

In choosing the available architectures for each aspect of the algorithm, we first note that it has been shown that implementing the immersed boundary method purely on GPUs

can throttle performance due to the prevalence of atomic adds in the spreading and interpolating communications between the Lagrangian and Eulerian grids [17], [18]. Furthermore, these works demonstrated that a heterogeneous framework may restore good scaling results on GPUs. For the current work, we assume that the length scale of the window needed to resolve the effects of surrounding cellular dynamics will be small compared to the overall domain. With these observations and assumptions, we adopt a heterogeneous computational framework in which GPUs accelerate lattice Boltzmann simulations in the bulk, and CPU cores are used to update both the cells and the fluid in the window domain.

In the window domain, CPU cores employ the push method in which a particle distribution in the lattice Boltzmann scheme first collides, and is then written to the advected locations via the streaming step. The particle distribution is stored in the Array of Structure layout. Each cell is assigned to a specific MPI task; this task tracks all of this cells face and vertex positions such as the vertex velocities and forces [31].

Fluid and cell data must be communicated across MPI processes. For the fluid data, halo points in the fluid domain are also communicated. The halo is a distance greater than the support of the smoothing immersed boundary kernel to ensure that vertex points inside a processors bounding box may appropriately interpolate their velocity and extrapolate their force from/to the Eulerian mesh. For the cellular data, any cell that interacts with the fluid dynamics on a nearby MPI task is communicated with the nearby task (see Figure 4). To reduce the whole cell communications across a large number of MPI tasks, we reduce the overall number of tasks by using OpenMP and assigning the multiple cores on a node to OpenMP threads on the task associated with the node.

The bulk flow model on the GPUs employs the pull method in the lattice Boltzmann framework, in which the streaming step first makes uncoalesced reads to gather the resulting streamed particle distribution; after the distribution is gathered, the collide step relaxes the distribution toward equilibrium within the same kernel. Pull methods have been demonstrated to be more efficient than push methods on GPUs [16], [35], [36]. The particle distribution for the bulk fluid is stored in the Structure of Array layout. Communicating particle distribution data is read to and from a buffer from an indirect layout. Lattice points that share data with neighboring processors are launched on high priority streams; non-communicating lattice points (or interior lattice points) are launched low priority streams. When the high priority streams finish, the GPUs may communicate asynchronous with the continuing update of the interior lattice points.

To load balance and set the domains of the bulk and window cores, each processor is assigned a bounding box making up some subsection of the domain. We establish and fix the bulk fluid domains (bounding boxes), using the bisection load balancer presented in [5]. The window will occasionally overlap with parts of the bulk domain, meaning that bulk processors containing the window will perform less work than other bulk domains; we find this an acceptable loss of

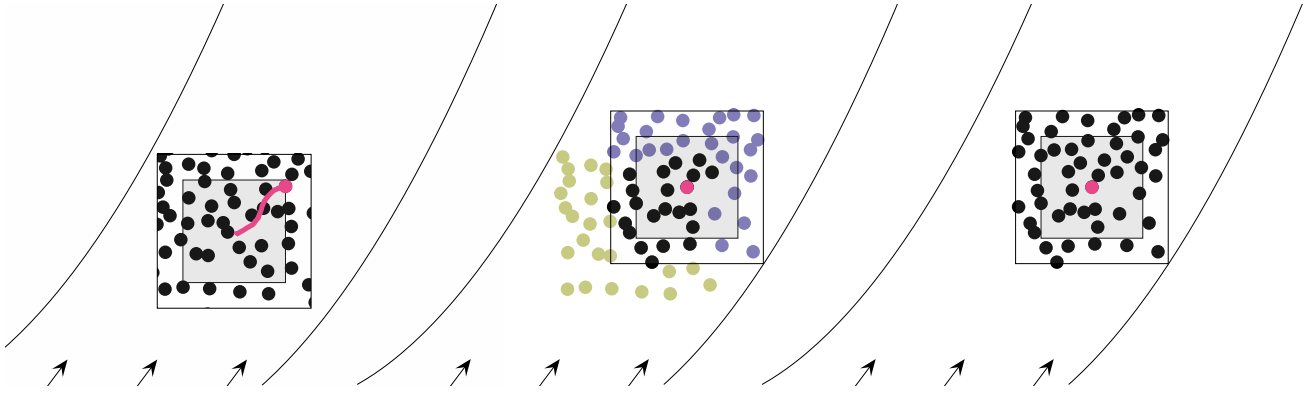


Fig. 3. A schematic of how the window moves. A tracked cell (magenta) advects to the boundary of the window interior (left). The window is re-established and centered around the tracked cell (middle). New cells are introduced (purple) and cells that are now in the exterior of the moved window are deleted (yellow). After re-establishing the window, the simulation continues with the window in a new, temporarily fixed, location to continue to track the cell (right).

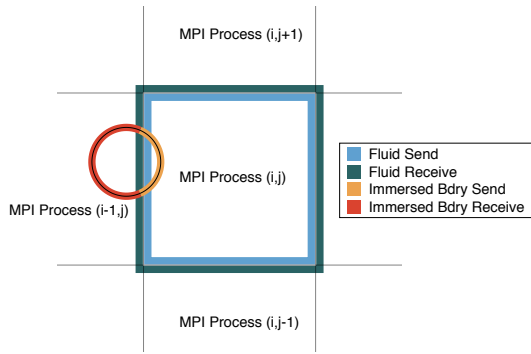


Fig. 4. We display the communication scheme on each window process on a 2D domain. MPI process  $(i, j)$  with displayed bounding box communicates relevant fluid data on halo points to neighboring processors; the halo distance is set so that immersed boundary points within a processes may accurately interpolate/extrapolate from/to the Eulerian mesh. Any cell that interacts with the fluid domain must have all its vertices communicated to/from the bounding box so that forces on the cell may be accurately computed. See [31] for more details.

efficiency as (i) load balancing the entire bulk domain is expensive, and (ii) fixing the load balance once provides a consistent unit of time with which to expect the bulk system update that is independent of the window location.

The window is assumed to be a cube, with the understanding that the length scale of this cube represents the physical length scale needed to accurately resolve the effects of the surrounding cells. In an arterial geometry, this cube will either lie entirely within the fluid domain, or partially intersect it. In either case, the cube will model the surrounding cellular interactions that affect the tracked cell. Because the window will utilize the same computational resources in either case, the amount of work per processor may change as a function of this overlap. In the present work, we have fixed the load balance within the window *a priori*; by fixing the load balance, a window processor will have a fixed upper bound on the number of fluid points it will update as well as a fixed maximal

number of fluid points to be communicated. The upper bound on the fluid points ensures that there is an *a priori* known computational update time, whereas the latter point ensures there is an *a priori* known upper bound for the amount of communication time. If we allowed these processors bounding boxes to vary, we may find certain geometries within the window in which total communication time may increase due an increase in the communicated boundary points and red blood cells. We omit optimizing this process by simply fixing the load balance.

We run with MPI, and the MPI tasks are split into bulk and window sub-communicators. Each bulk GPU is assigned a unique process that handles kernel launches and communication. The remaining cores on a node are assigned to OpenMP threads to minimize the overall communication across MPI tasks. When communicating between the bulk and window domains, global GPU memory is transferred to the host and read into an Array of Structure buffer so that the communication buffers are in the same format. Upon receiving either GPU or CPU data, the GPUs then have a single framework to interpret the received data.

**Implementing window motion** When the window is stationary, fluid data is communicated via halo points and cells are communicated to neighbor tasks when they affect their neighbors fluid domain. In moving the window, we developed a secondary communication framework in which all processors were given the new bounding box data for all window processors. We then transferred fluid particle distributions along with the lattice coordinates to the new processor owner. Cells were similarly transferred, however were deleted if any part fell outside the window. Our fluid halo communication scheme requires a specific ordering of the fluid domain; upon receiving a new set of lattice points with altered communicative properties, each processor kept track of the reordering of the lattice to correctly reorder and store the incoming fluid particle distribution, while the communication structures were being reset.

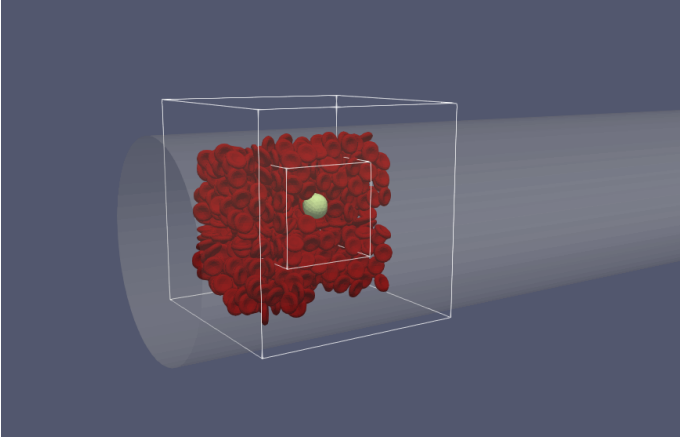


Fig. 5. The test set up used for the validations. Here, the entire window is displayed along with the interior as outer and inner cubes, respectively. The tracked cell (not present in the insertion validation) is shown in light green whereas the red blood cells are shown in red. Only half of the red blood cells are displayed in order to reveal the tracked cell.

## V. VALIDATIONS

To validate the insertion method, we examine cellular dynamics in a cylindrical pipe with length  $400\mu\text{m}$  and a radius of  $40\mu\text{m}$ . The physical setup is chosen so that we may compare a simulation with cells throughout the domain with a multi-scale simulation with the window. We then place a window with side lengths  $\ell = 100\mu\text{m}$ , an insertion length  $\ell_I = 25\mu\text{m}$ , and an interior length of  $\ell_{\text{int}} = 46\mu\text{m}$ . The insertion regions are set to be the six sides of the box up to the end of the window interior (with overlap), and  $\tau_i = 0.6$ . We then fix the window location and initialize it with a hematocrit (red blood cell volume fraction) of 30%, which is a low, but physiological relevant value [37]. A comparable setup is shown in Figure 5. We initialize a Poiseuille flow of  $3\text{cm/s}$  and run the simulation for roughly 0.02 seconds, which means that the window will completely refresh the cells within it roughly 6 times. We find that the red blood cell volume fraction within the window fluctuates, but remains close to the original value (see Figure 6). On average, the observed hematocrit is 29.2% (a relative error of less than 3%), meaning that the insertion algorithm causes only a modest decrease in the imposed red blood cell volume fraction. Given that the time averaged fluctuations are significantly larger than the measured error, we do not expect this deviation to have a significant effect on the dynamics, however careful validations of this are left for future study.

To validate the window motion, we use the same physical setup as above, but add a spherical cell that is to be tracked by the window. In cylindrical coordinates, the cell is initialized with  $r = 10\mu\text{m}$ ; varying  $\theta$  changes to the initial conditions. We then compare ten runs of the moving window dynamics to ten runs in which we resolve cells throughout the domain. Hematocrit is set to be 30%. We also examine ten runs without cells, again rotating the initial  $\theta$  to account for any anisotropic grid effects.

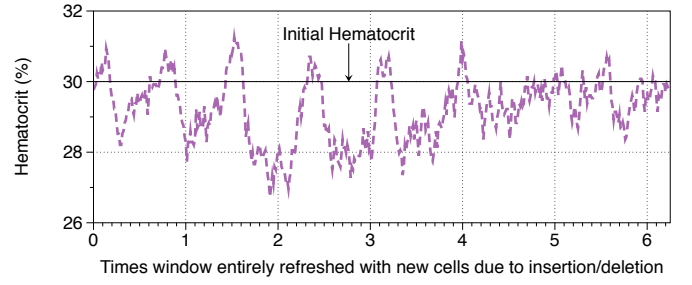


Fig. 6. We validate the insertion algorithm by observing the hematocrit as a function of how many times the window has refreshed the cells within it. The physical set up is cylindrical flow with a Reynolds number of 0.4.

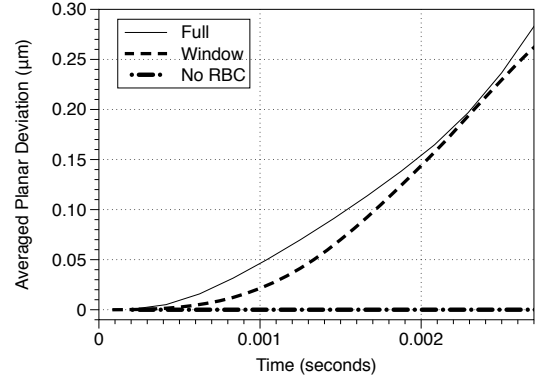


Fig. 7. We validate the moving window algorithm by examining the effective non-axial deviation as a function of time when considering a fully resolved domain and the window. We contrast these results when there are no red blood cells present.

We rotate each output to begin at a common value of  $\theta$  and then examine the path deviation in the plan perpendicular to the axial direction (i.e. the  $(r, \theta)$ , or  $(x, y)$  plan), which is given by

$$\sqrt{\langle ((x(t), y(t)) - (\langle x \rangle(t), \langle y \rangle(t)))^2 \rangle}, \quad (2)$$

where  $\langle \cdot \rangle(t)$  is a spatial average at time  $t$ , and  $\| \cdot \|_2$  is the two norm. In essence, we examine the variation in the cell trajectory introduced by the presence of red blood cells. We plot the deviation of the tracked cell when using a window with a total width of  $100\mu\text{m}$  (and the same parameters as the previous test) to the a system in which cells are tracked through the entire domain. We run the system for just over 0.0025 seconds. During this time, the cell moves approximately  $60\mu\text{m}$  in the axial direction and the window is re-established about 6 times for each run.

We display the planar deviation in Figure 7 showing good agreement between the simulation with cells throughout the entire domain and window simulations when accounting for red blood cells. In contrast, the simulation without red blood cells leads to nearly no path variation in the non-axial plane.

## VI. PERFORMANCE RESULTS

All performance measurements presented here were run on the Summit supercomputer [4]. Summit has 4,608 nodes, with

two IBM POWER9 processors and six NVIDIA Volta V100 GPUs per node. Each GPU has 16 GB of HBM2 memory, for a total of 96 GB per node, in addition to 512 GB of DDR4 memory. To take full advantage of all 22 cores of the POWER9 processor, we use a custom Explicit Resource File (ERF) to bind three MPI tasks per socket each to a single GPU, with the remaining cores assigned to a single MPI task per socket. OpenMP threading was used for the window parallelism across cores, with SMT4 mode found to produce the best performance. The theoretical peak performance of Summit is 200.8 PFLOP/s, over 95% of which comes from the GPUs.

The algorithm and associated timers are shown in Figure 8. Performance analysis was focused on the time spent updating the fluid and cells on each window processor, the time it takes for the bulk processors to update the fluid domain, the time spent communicating processor boundary data to neighboring processors, and the total time per step (both update and communication). The interior bulk lattice updates on the low priority streams, run asynchronously to both the border lattice updates and their communication to both the other bulk tasks and window tasks. Because of the asynchronous launches, we cannot directly measure the time spent on the low priority streams of the GPU. Instead, we indirectly assess this time by comparing the time in the window/border update and communication on the MPI tasks mapped to GPUs against the total time to see if the GPU interior updates are either faster, or dominate the timing. When probing the performance of timing within these steps, we also include timers for stream/boundary condition update, the collide update, and both the immersed boundary computation and communication time; these timers are not explicitly shown in the figure and are not used for the primary results.

To examine scaling, we separate performance results between updating the fluid and cells and the relative cost of moving the window.

**Computational cost of updating the fluid and cells** To evaluate the performance of updating the fluid and cells within the window, we examine a section of the cerebral vasculature shown in Figure 1. We assume an appropriate window length scale of  $150\mu\text{m}$ ; this length scale provides a buffer region for the tracked cell of 7.5 times its length in each direction and holds roughly 4,500 red blood cells along with a tracked cell in the current domain. The complete geometry has a total volume of  $0.7\text{mm}^3$ ; at 30% hematocrit it would hold 6.5 million red blood cells.

The grid spacing of the fluid domain is constrained by (i) the requirement to accurately capture the finite elements comprising the red blood cells and tracked cell, and (ii) system limitations. Using the same cell refinement as the validation study, we find that an appropriate fluid grid spacing would range between  $0.1\mu\text{m}$  and  $1\mu\text{m}$ . For the current study we examine fluid grid spacings of  $0.5\mu\text{m}$  and  $0.8\mu\text{m}$ . Figure 9 shows the strong scaling results for each of these resolutions. At a grid resolution of  $0.8\mu\text{m}$ , we see a scaling efficiency of roughly 50% between 64 and 512 nodes. At  $0.5\mu\text{m}$ , the

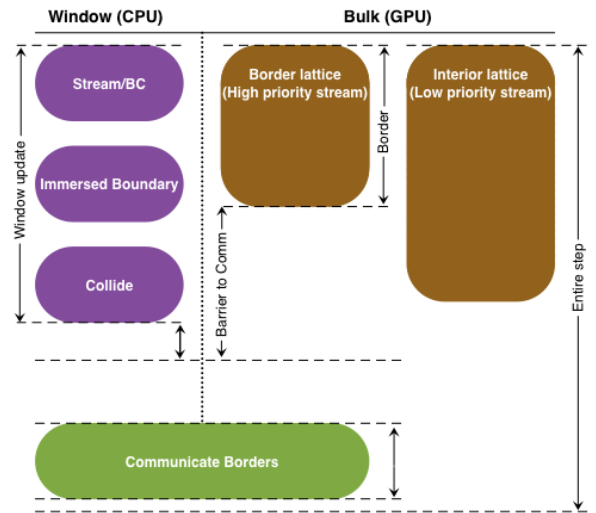


Fig. 8. An overview of the timers used to measure performance. The Columns are run asynchronously. Horizontal dashed lines display locations where the computational elements sync.

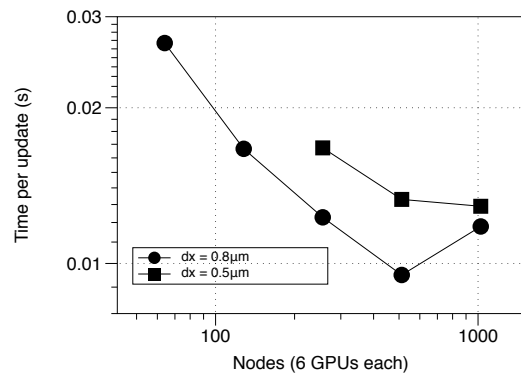


Fig. 9. We show strong scaling results for the cerebral arterial system which is updated for 100 times steps. We display the time taken to update only the GPUs (with no window) and compare to the time taken when updating only the window, and with the fully coupled multi-scale simulation. We find that the CPU processors are the primary bottle neck.

strong scaling efficiency drops to just 20% between 256 and 1024 nodes. Given the excellent scaling of our fluid-only lattice Boltzmann GPU implementation [16], we surmise that this poor scaling is due to limited scalability of the window on the CPUs. This is confirmed by the time spent in post-communication barriers by the MPI tasks bound to the GPUs.

To explicitly confirm that the simulation is CPU-bound, we ran fluid-only simulations of the cerebral geometry shown in Figure 1, with all fluid on the GPUs. We compare the performance of the GPU-only runs with those of the total system and find that a grid resolution of  $0.8\mu\text{m}$  the GPU-only part of the code runs between 5 and 7 times faster than the heterogeneous simulation; at a grid resolution of  $0.5\mu\text{m}$ , the GPU-only part of the code runs between 2 and 3 times faster

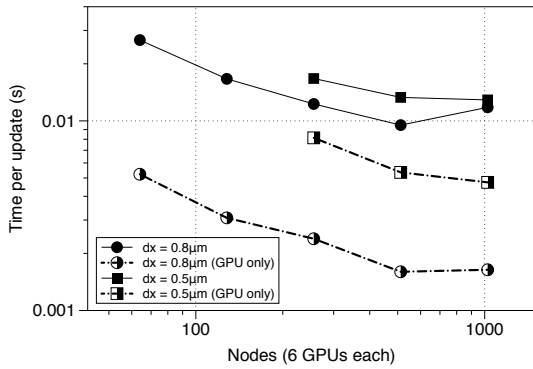


Fig. 10. We compare the full simulation when running only on the GPUs. At a grid resolution of  $0.8\mu\text{m}$  the GPU-only part of the code runs between 5 and 7 times faster than the heterogeneous simulation; at a grid resolution of  $0.5\mu\text{m}$ , the GPU-only part of the code runs between 2 and 3 times faster than the heterogeneous simulation.

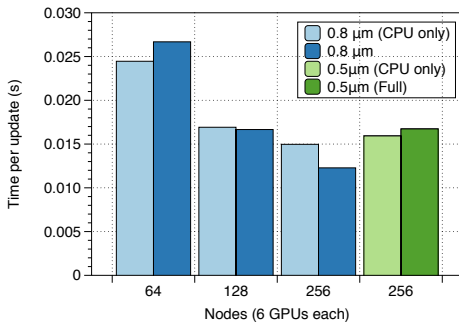


Fig. 11. We compare the full window simulation to a cubic geometry run on CPUs; the cubic geometry is the same size as the window and has cells throughout its domain.

than the heterogeneous simulation (see Figure 10). The loss of ideal scaling on the GPUs is likely due to a combination of load imbalance and sparsity of threads per GPU, but we do not investigate this in the current study as the GPUs are not impeding the computation.

Because we are not bound by the GPUs, we investigate the CPU timing both independent of the bulk, and then break down the CPU timers in the heterogeneous framework. Without the bulk, we do not have appropriate boundary conditions, and therefore examine a non-sparse cubic geometry and examine a bulk simulation with fluid everywhere. We repeat the scaling results for a selection of the node counts and find that the CPU-only, full cube simulation leads to timing results that are nearly identical to the heterogeneous system (see top of Figure 11).

We use the sub-timers to break down the performance of the CPU (see Figure 12). Several challenges to performance may be observed, stemming from the small problem size per MPI task. First, for the lattice Boltzmann method, we find that runtime for the LBM update does not substantially decrease over the range of node counts considered. We hypothesize that this is the consequence of our memory layout for the LBM distribution data, which has been optimized to facilitate data transfer during LBM communication, but

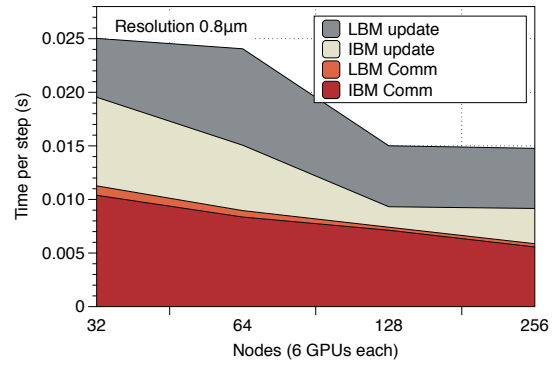


Fig. 12. We then break down the timing results between fluid (LBM) communication and computation, and immersed boundary (IBM) communication and computation at grid resolutions of  $0.8\mu\text{m}$  (left) and  $0.5\mu\text{m}$  (right).

may lead to decreased and variable performance for collision and streaming on the halo. As the problem size per rank is small, percentage of halo points is high and the resulting diminished performance for collision and streaming erodes performance as node counts become higher. Related but distinct challenges occur with respect to the immersed boundary method and related communication. As a typical task has 25-50 red blood cells, virtually all cells are located adjacent to task boundaries and may need to be communicated (see Figure 4). Consequently, immersed boundary communication dominates the corresponding update time. Because of this, the entirety of the cellular data will be communicated to more processes as the relative size of the processor bounding boxes decreases, which will increase the overall burden on communication. In summary, the algorithm's performance is limited with respect to both the lattice Boltzmann method and cell based communication due to the small problem size per task.

**Computational cost of moving the window** The relative computational cost between moving the window and updating the domain will primarily depend on the frequency with which the window is moved. This frequency will depend on the length scales of the window and on the speed of the tracked cell within the window. We consider the set up of the validation study in Section V above, and consider three different window side lengths of  $80\mu\text{m}$ ,  $100\mu\text{m}$ , and  $120\mu\text{m}$ , with interior lengths of  $26\mu\text{m}$ ,  $46\mu\text{m}$ , and  $66\mu\text{m}$  and buffer lengths of  $2\mu\text{m}$ . We update the system for  $5.4 \times 10^4$  lattice Boltzmann time steps, which corresponds to a physical run time of 2.2ms at an inlet velocity of 30mm/s. We then repeat these runs for velocities of 20mm/s and 10mm/s over 3.3ms and 6.6ms, respectively, so that the tracked cell travels roughly the same distance in all cases. We run these systems on 2 nodes of a local cluster with 2 NVIDIA Pascal P100 GPUs and 2 Intel Xeon Processor E5-2699 v4 per node. We display the fractional time of updating the window compared to the total run time in Table I.

The fraction of time spent moving the window decreases with the size of the window and the speed of the domain. In

	80 $\mu\text{m}$	100 $\mu\text{m}$	120 $\mu\text{m}$
30mm/s	1.10%	0.37%	0.09%
20mm/s	0.75%	0.26%	0.07%
10mm/s	0.37%	0.25%	0.05%

TABLE I

WE DISPLAY THE FRACTION OF THE RUN TIME SPENT MOVING THE WINDOW AT A VARIETY OF INLET VELOCITIES (ROWS) AND WINDOW SIZES (COLUMNS).

all of these cases, moving the window is a small fraction of the overall run time, making up less than 1.2% of the total run time in all cases and less than 0.1% of the total run time at the larger window size.

## VII. IMPLICATIONS

Despite the limited strong scaling of the current study, these results are in fact quite promising. Our initial analysis shows that it should be possible to significantly improve the performance of the threaded immersed boundary code, removing it as the primary bottleneck to scalability. This will allow us to continue to increase the overall system size, as the window size will remain fixed. The absolute time to solution is also quite good. Our algorithm has shown the capability of updating a fairly large system roughly 100 times per second; we can contrast this with previous work in which system updates occurred roughly twice per second, albeit with many more cells [10]. Assuming a grid resolution of 0.8 $\mu\text{m}$  and a cell traveling through a vascular region at a rate of 0.008 $\mu\text{m}$  per system update, our moving window algorithm could track a cell for roughly 7cm given 24 hours and 512 nodes on Summit. In contrast, if we were to resolve cells throughout the cerebral vascular domain, we can estimate the number of resources that would be required to obtain a similar wall time by assuming the cell update per node would remain constant. With this assumption, we estimate that one would need over 700,000 nodes to obtain similar time updates per cell, or two orders of magnitude more nodes than exist on Summit.

## VIII. DISCUSSION

We have implemented a multi-physics moving window algorithm using a heterogeneous computational framework. The algorithm is capable of tracking cell trajectories through arterial systems with a significant reduction in memory when compared to a system that accounts for cells throughout the computational domain. Our algorithm scales with roughly 50% efficiency under high ratios of nodes to cells, but has the promise to increase significantly once the window code is further optimized for the CPU. We note that moving from a whole cell-based communication scheme to a vertex- or element-based communication scheme should alleviate both the amount of communication and the amount of computation needed in updating the immersed boundary elements. In addition, we have shown that our algorithm is already capable of resolving cell trajectories at decimeter scales.

Our algorithm was validated on two tests. First we have confirmed that the hematocrit within the window interior remains stable over long time scales. Next, we have shown

that the path deviation in the non-axial plane is consistent between fully resolved systems and the window algorithm. More validations and tests are the subject of future studies. Such studies include determining minimal appropriate length scales for the window. An in depth understanding of how this algorithms breaks down under various changes to the parameters is the target of future work.

In the current implementation, we have assumed the same resolution mesh over the window and the bulk. In general, this will not always be necessary and will allow for much greater regions of the arterial region to be resolved. Implementing such a scheme may either be done via an adaptive mesh scheme, or simply coupling two meshes at disparate scales via interpolation.

In tracking cell trajectories there are three fundamental issues at play. First, computing cellular trajectories are computational expensive both in terms of memory consumption and computational time; any reduction to this expense will provide us with a greater ability to probe these dynamics. Second, cellular trajectories are statistical in nature; the only way to appropriately determine these trajectories is to run many simulations with variations in initial conditions; therefore, reductions on the resource demand per run will be crucial. Third, cellular trajectories depend on physical length scales spanning several orders of magnitude. High performance computing in the context of multi-physics simulations is a crucial component in accounting for these disparate issues.

## ACKNOWLEDGMENT

The authors thank Jeff Ames and Jianfeng Lu for helpful discussions and feedback. We are also grateful for the Duke University OIT support of the Duke Compute Cluster.

## REFERENCES

- [1] D. Wirtz, K. Konstantopoulos, and P. C. Searson, "The physics of cancer: the role of physical interactions and mechanical forces in metastasis," *Nature Reviews Cancer*, vol. 11, no. 7, p. 512, 2011.
- [2] A. Sen Gupta, "Role of particle size, shape, and stiffness in design of intravascular drug delivery systems: insights from computations, experiments, and nature," *Wiley Interdisciplinary Reviews: Nanomedicine and Nanobiotechnology*, vol. 8, no. 2, pp. 255–270, 2016.
- [3] P. Balogh and P. Bagchi, "A computational approach to modeling cellular-scale blood flow in complex geometry," *Journal of Computational Physics*, vol. 334, pp. 280–307, 2017.
- [4] [Online]. Available: <https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/>
- [5] A. Randles, E. Draeger, T. Ooppelstrup, L. Krauss, and J. Gunnels, "Massively parallel models of the human circulatory system," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 1.
- [6] P. Reymond, F. Merenda, F. Perren, D. Rufenacht, and N. Stergiopoulos, "Validation of a one-dimensional model of the systemic arterial tree," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 297, no. 1, pp. H208–H222, 2009.
- [7] M. Mehrabadi, D. N. Ku, and C. K. Aidun, "A continuum model for platelet transport in flowing blood based on direct numerical simulations of cellular blood flow," *Annals of biomedical engineering*, vol. 43, no. 6, pp. 1410–1421, 2015.
- [8] V. Kislukhin and E. Kislukhina, "Math model of the passage of a diffusible indicator throughout microcirculation based on a stochastic description of diffusion and flow," in *Trends in Biomathematics: Modeling, Optimization and Computational Problems*. Springer, 2018, pp. 365–373.

- [9] A. Peters, S. Melchionna, E. Kaxiras, J. Lätt, J. Sircar, M. Bernaschi, M. Bison, and S. Succi, "Multiscale simulation of cardiovascular flows on the IBM BlueGene/P: Full heart-circulation system at red-blood cell resolution," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–10.
- [10] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc *et al.*, "Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–11.
- [11] L. Grinberg, J. A. Insley, V. Morozov, M. E. Papka, G. E. Karniadakis, D. Fedosov, and K. Kumaran, "A new computational paradigm in multiscale simulations: Application to brain blood flow," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 5.
- [12] M. Bernaschi, M. Bisson, T. Endo, S. Matsuoka, M. Fatica, and S. Melchionna, "Petaflop biofluidics simulations on a two million-core system." IEEE, 2011, pp. 1–12.
- [13] W. Xian and A. Takayuki, "Multi-gpu performance of incompressible flow computation by lattice boltzmann method on gpu cluster," *Parallel Computing*, no. 37, pp. 521–535, 2011.
- [14] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux, "Multi-gpu implementation of the lattice boltzmann method," *Computers and Mathematics with Applications*, no. 65, pp. 252–261, 2013.
- [15] M. Januszewski and M. Kostur, "Sailfish: A flexible multi-gpu implementation of the lattice boltzmann method," *Computer Physics Communications*, no. 185, pp. 2350–2368, 2014.
- [16] G. Herschlag, S. Lee, J. S. Vetter, and A. Randles, "Gpu data access on complex geometries for d3q19 lattice boltzmann method," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 825–834.
- [17] P. Valero-Lara, F. D. Igual, M. Prieto-Matías, A. Pinelli, and J. Favier, "Accelerating fluid–solid simulations (lattice-boltzmann & immersed-boundary) on heterogeneous architectures," *Journal of Computational Science*, vol. 10, pp. 249–261, 2015.
- [18] P. Valero-Lara and J. Jansson, "Heterogeneous cpu+ gpu approaches for mesh refinement over lattice-boltzmann simulations," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 7, p. e3919, 2017.
- [19] C. Godenschwager, F. Schornbaum, M. Bauer, H. Köstler, and U. Rüde, "A framework for hybrid parallel flow simulations with a trillion cells in complex geometries," in *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2013, pp. 1–12.
- [20] D. Rossinelli, Y.-H. Tang, K. Lykov, D. Alexeev, M. Bernaschi, P. Hadjidoukas, M. Bisson, W. Joubert, C. Conti, G. Karniadakis *et al.*, "The in-silico lab-on-a-chip: petascale and high-throughput simulations of microfluidics at cell resolution," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 2.
- [21] K. Lykov, X. Li, H. Lei, I. V. Pivkin, and G. E. Karniadakis, "Inflow/outflow boundary conditions for particle-based blood flow simulations: application to arterial bifurcations and trees," *PLoS computational biology*, vol. 11, no. 8, p. e1004410, 2015.
- [22] V. W. A. Tarkaloooyeh, G. Závodszky, B. J. van Rooij, and A. G. Hoekstra, "Inflow and outflow boundary conditions for 2D suspension simulations with the immersed boundary lattice Boltzmann method," *Computers & Fluids*, vol. 172, pp. 312–317, 2018.
- [23] S. Chen and G. Doolen, "Lattice boltzmann method for fluid flows," *Annual review of fluid mechanics*, vol. 30, no. 1, pp. 329–364, 1998.
- [24] D. Ziegler, "Boundary conditions for lattice boltzmann simulations," *Journal of Statistical Physics*, vol. 71, no. 5, pp. 1171–1177, 1993.
- [25] Q. Zou and X. He, "On pressure and velocity boundary conditions for the lattice boltzmann bgk model," *Physics of fluids*, vol. 9, no. 6, pp. 1591–1598, 1997.
- [26] S. Succi, *The lattice Boltzmann equation: for fluid dynamics and beyond*. Oxford university press, 2001.
- [27] C. S. Peskin, "The immersed boundary method," *Acta numerica*, vol. 11, pp. 479–517, 2002.
- [28] D. Barthes-Biesel, A. Diaz, and E. Dhenin, "Effect of constitutive laws for two-dimensional membranes on flow-induced capsule deformation," *Journal of Fluid Mechanics*, vol. 460, pp. 211–222, 2002.
- [29] T. Krüger, F. Varnik, and D. Raabe, "Efficient and accurate simulations of deformable particles immersed in a fluid using a combined immersed boundary lattice Boltzmann finite element method," *Comput Math Appl*, vol. 61, no. 12, pp. 3485–3505, 2011.
- [30] Z. Guo, C. Zheng, and B. Shi, "Discrete lattice effects on the forcing term in the lattice Boltzmann method," *Phys Rev E*, vol. 65, no. 4, p. 046308, 2002.
- [31] J. Gounley, E. W. Draeger, and A. Randles, "Numerical simulation of a compound capsule in a constricted microchannel," *Procedia computer science*, vol. 108, pp. 175–184, 2017.
- [32] E. Birgin, R. Lobato, and J. Martínez, "A nonlinear programming model with implicit variables for packing ellipsoids," *J Global Optim*, vol. 68, no. 3, pp. 467–499, 2017.
- [33] J. Gounley, E. W. Draeger, and A. Randles, "Immersed boundary method halo exchange in a hemodynamics application," *Lecture Notes in Computer Science*, (to appear).
- [34] A. Randles, E. Draeger, and P. Bailey, "Massively parallel simulations of hemodynamics in the primary large arteries of the human vasculature," *J of Comp. Sci.*, vol. 9, pp. 70–75, 2015.
- [35] P. Rinaldi, E. Dari, M. Vénere, and A. Clause, "A lattice-boltzmann solver for 3d fluid simulation on gpu," *Simulation Modelling Practice and Theory*, no. 25, pp. 163–171, 2012.
- [36] J. Habich, C. Feichtinger, H. Köstler, G. Hager, and G. Wellein, "Performance engineering for the lattice boltzmann method on gpgpus: Architectural requirements and performance results," *Computers & Fluids*, vol. 80, pp. 276–282, 2013.
- [37] A. Besarab, W. K. Bolton, J. K. Browne, J. C. Egrie, A. R. Nissenson, D. M. Okamoto, S. J. Schwab, and D. A. Goodkin, "The effects of normal as compared with low hematocrit values in patients with cardiac disease who are receiving hemodialysis and epoetin," *New England Journal of Medicine*, vol. 339, no. 9, pp. 584–590, 1998.