INL/JOU-17-42412-Revision-1

An efficient parallel simulation of unsteady blood flows in patient-specific pulmonary artery

Fande Kong, Vitaly Kheyfets, Ender Finol, Xiao-Chuan Cai

February 2018

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/JOU-17-42412-Revision-1

An efficient parallel simulation of unsteady blood flows in patient-specific pulmonary artery

Fande Kong, Vitaly Kheyfets, Ender Finol, Xiao-Chuan Cai

February 2018

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN BIOMEDICAL ENGINEERING
Int. J. Numer. Meth. Biomed. Engng. 0000; **00**:1–20

An efficient parallel simulation of unsteady blood flows in patient-specific pulmonary artery

Fande Kong[†], Vitaly Kheyfets[‡], Ender Finol[§] and Xiao-Chuan Cai[¶*]

[†]*Modeling and Simulation, Idaho National Laboratory, P.O. Box 1625, Idaho Falls, ID 83415-3840, USA*

[‡]*School of Medicine, University of Colorado Denver, Aurora, CO 80045-7109, USA*

[§]*Department of Mechanical Engineering, University of Texas at San Antonio, San Antonio, TX 78249, USA*

[¶]*Department of Computer Science, University of Colorado Boulder, Boulder, CO 80309-0430, USA*

SUMMARY

Simulation of blood flows in the pulmonary artery provides some insight into certain diseases by examining the relationship between some continuum metrics, e.g. the wall shear stress acting on the vascular endothelium, which responds to flow-induced mechanical forces by releasing vasodilators/constrictors. In [1], V. Kheyfets studies numerically a patient-specific pulmonary circulation to show that decreasing wall shear stress is correlated with increasing pulmonary vascular impedance. In this paper, we develop a scalable parallel algorithm based on domain decomposition methods to investigate an unsteady model with patient specific pulsatile waveforms as the inlet boundary condition. The unsteady model offers tremendously more information about the dynamic behavior of the flow field, but computationally speaking, the simulation is a lot more expensive since a problem which is similar to the steady state problem has to be solved many times, and therefore, the traditional sequential approach is not suitable anymore. We show computationally that simulations using the proposed parallel approach with up to 10,000 processor cores can be obtained with much reduced compute time. This makes the technology potentially usable for the routine study of the dynamic behavior of blood flows in the pulmonary artery, in particular, the changes of the blood flows and the wall shear stress in the spatial and temporal dimensions.

Received . . .

KEY WORDS: unsteady blood flows; patient-specific pulmonary artery; finite element; domain decomposition; parallel processing

1. INTRODUCTION

Numerical simulation of blood flows in human arteries is a powerful tool for understanding certain cardiovascular diseases and for treatment planning [2, 3, 4]. In some situations, modeling and simulations provide essential metrics of blood flows that are otherwise immeasurable [1, 5]. For example, the dynamic value of wall shear stress is difficult to measure directly, but can be obtained via numerical computation using patient-specific arterial geometry and parameters. On the other hand, the computational approach faces its own challenges, and one of them is the high cost of solving a large system (hundreds of millions of degrees of freedom) of nonlinear equations arising from the discretization of the incompressible Navier-Stokes equations. One such system

*Correspondence to: Department of Computer Science, University of Colorado Boulder, Boulder, CO 80309-0430, USA

needs to be solved in the steady-state simulation, and many such systems need to be solved if an unsteady model is considered for the blood flow. With the proposed parallel approach, such a calculation can be accomplished in a few hours, instead of days when using a sequential approach. This makes the application of the technology clinically feasible. Over the past few years, several computationally inexpensive methods have been developed, for example, the three-dimensional Navier-Stokes equations was simplified to certain one-dimensional models in [6, 7, 8] to reduce the computational effort. A two-dimensional model was proposed in [9, 10] to fill the gap between the expensive three-dimensional model [11, 12, 13, 14] and the one-dimensional model. However, when certain features of the blood flow are desired, such as the wall shear stress, which is known to be highly correlated to some arterial diseases [1], the full three-dimensional model has to be considered. To resolve the full three-dimensional, time-dependent flow in a complete pulmonary artery, in this paper, we develop a parallel finite element method on large-scale supercomputers with tens of thousands of processor cores.

In the following, we briefly review some recent progresses of numerical simulation of blood flows in the pulmonary artery. In [15], the blood flow was computed by using a one-dimensional model discretized with finite element method, and the corresponding system of nonlinear equations is solved via a quasi Newton method. In [16], a 3D simulation of unsteady blood flows of a patient-specific pulmonary artery is obtained using a stabilized finite element method for the incompressible Navier-Stokes equations, and the sequential simulation takes a couple of days of computing time for a problem with a million elements. In [8], a multiscale mathematical and computational model of some 1D pulmonary networks are presented and used to analyze both arterial and venous pressure and flow. Some publications also considered the interaction of the blood flow with the elastic arterial wall, for example, in [2, 17], the CFD-ACE multiphysics package [18] is used for a 2D unsteady blood flow problem. In the simulation, a finite volume method is used for the fluid equations and a finite element method is used for the arterial wall, the resulting systems of equations are solved with an algebraic multigrid for the fluid and a direct method for the structural equations. A numerical method is developed to solve the steady fluid-structure interaction problems in three-dimensional pulmonary arterial bifurcation with collapsible tubes in [19], where a in-house FEM code is used to calculate the nonlinear deformation of the thin-walled structure and a commercial CFD solver, ANSYS [20], is used to resolve the fluid equations.

Most of the published works are based on commercial software which is easy to use but offers limited parallelism. For example, ANSYS [20] is scalable with only a few hundred processor cores. This is acceptable for the study of steady state problems or problems defined on a small portion of an artery tree, but not acceptable for the study of unsteady flows in a full pulmonary artery as each study will take days of run time. To take advantages of modern supercomputers with a large number of processor cores, in this paper, we focus on the development of new algorithms and software that have close to linear scalability in the sense that if the number of processor cores is doubled, the compute time is halved. In our previous works [12, 13, 14, 21], we successfully simulated a small portion of a pulmonary artery using the Newton-Krylov-Schwarz (NKS) methods. But the general NKS algorithm becomes much harder to use when the geometry of the computational domain is complex, such as the complete pulmonary artery, because the optimal choices of some of the parameters of the Schwarz preconditioner are geometry-dependent and are not well-understood. In this work, to overcome these difficulties, several important Schwarz parameters including the ordering of the unknowns in submeshs, the restriction/prolongation operators, and the level of fill-ins of submesh solver (incomplete LU factorization), are comprehensively studied and optimized. Precisely speaking, we introduce a fully implicit, parallel domain decomposition algorithm that partitions the pulmonary artery tree into a large number of sub-trees that are mapped onto different processor cores of the parallel computer. The algorithmic framework consists of several ingredients. First, an inexact Newton method [22] is used to solve the system of nonlinear equations that has high nonlinearities due to the complexity and irregularity of the computational domain. During each Newton iteration, the solution of the Jacobian system is obtained using a Krylov subspace method [23] together with a scalable Schwarz preconditioner. We show numerically that the proposed version of the parallel algorithm is scalable to thousands of processor cores for large-scale problems

with hundreds of millions of unknowns. We also mention that the NKS method has been successfully applied to several classes of problems including the Bidomain reaction-diffusion system in [24], some elasticity problems in [25], and fluid-structure interaction problems in [12, 13, 14, 21].

The remainder of this paper is organized as follows. In Section 2, we first introduce a partitioning strategy to subdivide the large artery tree into a large number of small artery trees that will be handled by different processor cores in parallel, and then the discretization of the incompressible Navier-Stokes equations in time and space. A fully implicit, scalable Newton-Krylov-Schwarz method is described in Section 3. In Section 4, some numerical experiments and observations are presented. We also show the parallel performance of the proposed approach. Lastly, some concluding remarks are given in Section 5.

2. PROBLEM DEFINITION

We first describe a patient-specific pulmonary artery, the mesh we use to discretize the artery, and a strategy we use to partition the mesh into a large number of submeshes for parallel processing. We then discuss the governing equations for the blood flow and the spatial and temporal discretization.

2.1. Complete pulmonary artery tree

Pulmonary circulation is the portion of the cardiovascular system which carries deoxygenated blood away from the heart, to the lungs, and returns oxygenated blood back to the heart. We consider a complete pulmonary artery tree here, denoted as Ω, consisting of one inlet and 274 outlets, as shown in Fig. 1. The pulmonary arterial geometry was derived in the previous study [1] based on the CT (computed tomography) image of a patient. Interested readers are referred to [1] for more details. For convenience, let Γ_I be the inlet, Γ_{O_i} the outlets, $i = 1, 2, .., M$ (M is the number of outlets, 274 here) and Γ_W the wall. We first cover Ω with an unstructured tetrahedral mesh, denoted as Ω_h, shown in Fig. 1. The mesh is generated using ANSYS [20]. In order to simulate the blood flow in parallel, we need to partition Ω_h into np submeshes $\Omega_{h,i}, i = 1, 2, \cdots, np$ (np is the number of processor cores to be used in the computation, around 10,000 in our simulations). The partitioning of the complex mesh is nontrivial, especially when the number of processor cores is large. At the first step, the mesh is converted into an undirected graph whose vertices correspond to the nodes of the mesh, and two vertices are connected if the nodes are connected by an edge. There are several algorithms for graph partitioning. We apply a hierarchal partitioning strategy developed in our previous work [12, 13, 25] which uses the standard method, ParMETIS/METIS [26, 27], twice as shown in Algorithm 1. This strategy offers better results than simply applying METIS or ParMETIS only once to obtain a large number of submeshes. Precisely speaking, we apply ParMETIS or METIS to divide Ω_h into np_1 submeshes (np_1 is the number of compute nodes), and then apply METIS to further partition each submesh into np_2 smaller submeshes (np_2 is the number of processor cores per compute node). The total number submeshes is then $np = np_1 \times np_2$. This idea is simple but very effective especially when the number of processor cores is large because the commutation cost among submeshes are significantly reduced. A sample partitioning is shown in Fig. 2, where the mesh is partitioned into 4 submeshes using ParMETIS, then each submesh is further cut into 2 smaller submeshes using METIS, and then we have 8 submeshes in total. The meshe is denser at the inlet, outlets, and sharp-curved regions, and is more or less uniform elsewhere. Since the partition is based on a pure graph algorithm, each partition may contain either/both fluid elements or/and solid elements and each subdomain may consist of disconnected smaller subdomains.

Remark 2.1
With this partitioning strategy, the quality of partition is quite good. The number of elements per processor is roughly the same. The interface between two adjacent submeshes is short, which is important for minimizing the communication cost.

Remark 2.2
Once a partition is obtained, we map each submesh $\Omega_{h,i}(i = 1, 2, ..., np)$ to a processor core. All

Algorithm 1 Hierarchical Partitioning of Ω_h

1: Partition Ω_h into np_1 submeshs $\tilde{\Omega}_{h,j}$ $(j = 1, 2, \cdots, np_1)$ using ParMETIS, METIS or other partitioners, where np_1 is usually the number of compute nodes
2: Partition $\tilde{\Omega}_{h,j}$ into np_2 smaller submeshs using METIS or other partitioners on the jth MPI rank, where np_2 is often the number of cores per compute node, and order the smaller submeshs appropriately
3: Output $\{\Omega_{h,i}\}$ $(i = 1, 2, \cdots, np$, where $np = np_1 \times np_2)$

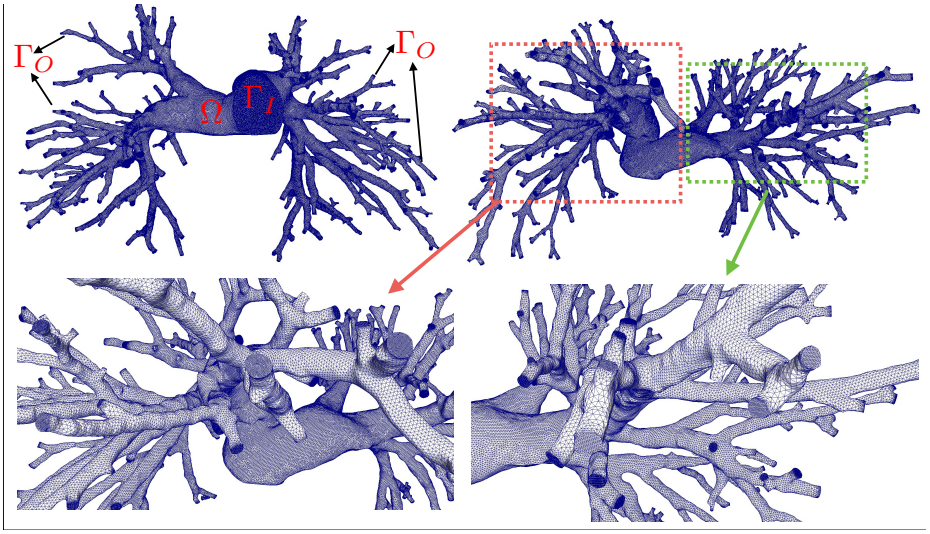


Figure 1. Pulmonary artery and its finite element mesh.

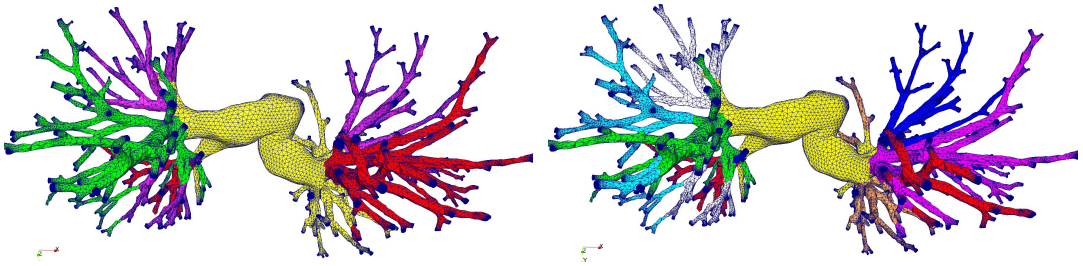


Figure 2. Hierarchical partitioning. Mesh is partitioned into 4 submeshes using ParMETIS, shown in the left figure, then each submesh is further cut into 2 smaller submeshes using METIS, and then we have 8 submeshes in total, shown in the right figure.

variables associated with this submesh are allocated on this processor core, and all computations related to $\Omega_{h,i}$ are carried out by this core.

2.2. *Governing equations and discretization*

It is reasonable to assume that the blood flow in large arteries is incompressible and Newtonian. Let $\boldsymbol{u}$ and p denote the velocity and pressure, respectively, and the incompressible Navier-Stokes

equations take the form

$$\begin{cases} \rho\dfrac{\partial \boldsymbol{u}}{\partial t} + \rho\left(\boldsymbol{u}\cdot\nabla\right)\boldsymbol{u} - \nabla\cdot\boldsymbol{\sigma} = \rho\boldsymbol{f} & \text{in } \Omega, \\ \nabla\cdot\boldsymbol{u} = 0 & \text{in } \Omega, \\ \boldsymbol{u} = \boldsymbol{v}_I & \text{on } \Gamma_I, \\ \boldsymbol{u} = 0 & \text{on } \Gamma_W, \\ \boldsymbol{\sigma}\boldsymbol{n} = 0 & \text{on } \Gamma_{O_i}, i = 1, 2, ..., M, \end{cases} \tag{1}$$

where ρ is the blood density, $\boldsymbol{f}$ is a given body force per unit mass, $\boldsymbol{v}_I$ is a velocity profile derived from a flow rate, $\boldsymbol{n}$ is the unit outward normal to the outlet surface, and $\boldsymbol{\sigma}$ is the Cauchy stress tensor defined as follows,

$$\boldsymbol{\sigma} = -p\boldsymbol{I} + 2\nu\boldsymbol{\varepsilon}(\boldsymbol{u}),\ \boldsymbol{\varepsilon}(\boldsymbol{u}) = 1/2\left(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^T\right).$$

Here ν is the viscosity coefficient and $\boldsymbol{I}$ is a 3×3 identity matrix. The traction-free boundary condition is applied to all outlets.

To discretize (1) in space, we consider a $P_1 - P_1$ finite element space for the velocity and the pressure. Because the $P_1 - P_1$ pair does not satisfy the LBB condition, additional stabilization terms are considered as described in [28, 29]. Following the standard notations, the weak form reads as: Find $\boldsymbol{u} \in V$ and $p \in P$ such that $\forall\boldsymbol{\phi} \in V_0$ and $\forall q \in P$,

$$\mathcal{B}_f(\{\boldsymbol{u}, p\}, \{\boldsymbol{\phi}, q\}) = 0, \tag{2}$$

with

$$\begin{aligned} \mathcal{B}_f(\{\boldsymbol{u}, p\}, \{\boldsymbol{\phi}, q\}) \equiv\ & \rho\int_\Omega \frac{\partial \boldsymbol{u}}{\partial t}\cdot\boldsymbol{\phi}\,\mathrm{d}\Omega - \int_\Omega p\left(\nabla\cdot\boldsymbol{\phi}\right)\,\mathrm{d}\Omega \\ & +\rho\int_\Omega \left(\boldsymbol{u}\cdot\nabla\right)\boldsymbol{u}\cdot\boldsymbol{\phi}\,\mathrm{d}\Omega \\ & +2\nu\int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{u}) : \boldsymbol{\varepsilon}(\boldsymbol{\phi})\,\mathrm{d}\Omega + \int_\Omega \left(\nabla\cdot\boldsymbol{u}\right) q\,\mathrm{d}\Omega \\ & -\rho\int_\Omega \boldsymbol{f}\cdot\boldsymbol{\phi}\,\mathrm{d}\Omega, \end{aligned}$$

and

$$\begin{aligned} & V = \left\{\boldsymbol{u} \in \left[H^1(\Omega)\right]^3 : \boldsymbol{u} = \boldsymbol{v}_I \text{ on } \Gamma_I \text{ and } \boldsymbol{u} = 0 \text{ on } \Gamma_W\right\}, \\ & V_0 = \left\{\boldsymbol{\phi} \in \left[H^1(\Omega)\right]^3 : \boldsymbol{\phi} = 0 \text{ on } \Gamma_I \cup \Gamma_W\right\}, \\ & P = L^2(\Omega). \end{aligned}$$

Here $H^1(\Omega)$ and $L^2(\Omega)$ are the standard Sobolev spaces. Denote the finite element subspaces V^h, V_0^h, and P^h as the counterparts of their infinite dimensional subspaces. The semi-discretized system of the Naver-Stokes equations with the stabilization is described as: Find $\boldsymbol{u}^h \in V^h$ and $p^h \in P^h$, such that $\forall\boldsymbol{\phi}^h \in V_0^h$ and $\forall q^h \in P^h$,

$$\mathcal{B}(\{\boldsymbol{u}^h, p^h\}, \{\boldsymbol{\phi}^h, q^h\}) = 0, \tag{3}$$

with

$$\begin{aligned} \mathcal{B}(\{\boldsymbol{u}^h, p^h\}, \{\boldsymbol{\phi}^h, q^h\}) \equiv\ & \mathcal{B}_f(\{\boldsymbol{u}^h, p^h\}, \{\boldsymbol{\phi}^h, q^h\}) \\ & + \sum_{K\in\Omega^h} \left(\nabla\cdot\boldsymbol{u}^h, \tau_c\nabla\cdot\boldsymbol{\phi}^h\right)_K \\ & + \sum_{K\in\Omega^h} \left(\mathcal{L}^h, \tau_m\left(\boldsymbol{u}^h\cdot\nabla\boldsymbol{\phi}^h + \nabla q^h\right)\right)_K \\ & + \sum_{K\in\Omega^h} \left(\bar{\boldsymbol{u}}^h\cdot\nabla\boldsymbol{u}^h, \boldsymbol{\phi}^h\right)_K \\ & + \sum_{K\in\Omega^h} \left(\bar{\boldsymbol{u}}^h\cdot\nabla\boldsymbol{u}^h, \tau_b\bar{\boldsymbol{u}}^h\cdot\nabla\boldsymbol{\phi}^h\right)_K, \end{aligned}$$

where $\Omega^h = \{K\}$ is the given unstructured tetrahedral mesh of the computational domain, $\mathcal{L}^h$ is computed as follows:

$$\mathcal{L}^h = \rho \frac{\partial \boldsymbol{u}^h}{\partial t} + \rho_f \boldsymbol{u}^h \cdot \nabla \boldsymbol{u}^h + \nabla p^h - \rho \boldsymbol{f}^h,$$

$\bar{\boldsymbol{u}}^h = -\tau_m \mathcal{L}^h$, and τ_m, τ_c and τ_b are the stabilization parameters to be defined later. $(\cdot,\cdot)_K$ represents an integral over element K. For more details of the spatial discretization scheme, we refer to [12, 30]. After the spatial discretization, (3) is rewritten as a time-dependent nonlinear system:

$$\frac{\partial y(t)}{\partial t} + \mathcal{N}(y(t)) = F, \tag{4}$$

where F is the right-hand side, $\mathcal{N}(y(t))$ is a nonlinear function, and $y(t)$ is a time-dependent vector of coefficients of the velocity and pressure values defined at the nodal points of the finite element mesh at time t. Equation (4) is further discretized in time using an implicit backward Euler formula as:

$$M_n y_n + \delta t \mathcal{N}(y_n) = \delta t F + M_n y_{n-1}. \tag{5}$$

where y_n is the approximation of $y(t)$ at the nth time step, M_n is the mass matrix at the nth time step, and δt is the time step size. With the time step size δt, the stabilization parameters τ_m, τ_c and τ_b are defined as follow:

$$\begin{aligned}
\tau_m &= \frac{1}{\sqrt{4/\delta t^2 + (\boldsymbol{u}_n^h \cdot G \boldsymbol{u}_n^h) + 36\nu^2/\rho^2 G : G}}, \\
\tau_c &= \frac{1}{8\tau_m \text{trace}\,(G)}, \\
\tau_b &= \frac{1}{\sqrt{\bar{\boldsymbol{u}}_n^h \cdot G \bar{\boldsymbol{u}}_n^h}}.
\end{aligned}$$

Here $(G)_{ij} = \sum_{l=1}^{3} \frac{\partial \xi_l}{\partial x_i} \frac{\partial \xi_l}{\partial x_j} (i,j = 1,2,3)$ is the covariant metric tensor, $\frac{\partial \xi}{\partial x}$ represents the Jacobian of the mapping between the reference and the physical elements, ξ_i are components of reference coordinates, x_i are components of current physical coordinates, and $\boldsymbol{u}_n^h$ and $\bar{\boldsymbol{u}}_n^h$ are the counterparts of $\boldsymbol{u}^h$ and $\bar{\boldsymbol{u}}^h$ at the nth time step. For convenience, let us rewrite (5) at the nth time step as a system of nonlinear equations:

$$\mathcal{F}(y) = 0, \tag{6}$$

where y (we drop the subscript n here for simplicity) is the vector of coefficients of the velocity and pressure values at the nth time step. Other time discretization schemes, such as Crank-Nicolson method, are also possible for (4). We use implicit backward Euler in this work because it is unconditionaly stable and does not suffer with numerical oscillations for large time step sizes and a fine spatial resolution. Solving (6) is very challenging, especially when the number of processor cores is large. For a patient-specific problem, we usually do not know the initial field of velocity and pressure. Therefore we often start with zero as the initial guess and carry out the simulation for some time steps and then use the simulation at that point as the true initial state of (1). Once we figure out the initial state, the main cost of the simulation is to solve (4) over and over again using the solution of the previous time step as the initial solution. In the next section, we describe a scalable and robust Newton-Krylov-Schwarz (NKS) method, which has been successfully studied for a transonic full potential equation in [31], elasticity problems [25], and fluid-structure interactions [12, 13, 14]. The standard setup of NKS used in [12, 13, 14, 25] doesn't work well for the case of pulmonary artery because of the complexity of the geometry. Several modifications will be discussed in the next section.

3. AN EFFICIENT PARALLEL SOLVER

We first describe the parallel algorithm framework consisting of an inexact Newton method for the system of nonlinear equations, a Krylov subspace method for the Jacobian equations, and a Schwarz

preconditioner for accelerating the convergence of the Jacobian solver. And then a reordering algorithm for the submesh solver is introduced which significantly improves the performance of the submesh solver.

3.1. Parallel algorithm framework

An inexact Newton method [22] is used to solve (6), where a Krylov subspace method is employed for approximately solving the Jacobian system. More precisely, the computation starts with an initial guess $y^{(0)}$, and updates the solution along the Newton direction $\delta y^{(k)}$, that is,

$$y^{(k+1)} = y^{(k)} + \alpha^{(k)} \delta y^{(k)}, k = 0, 1, \ldots,$$

where $\alpha^{(k)}$ is the step length calculated by the backtracking linesearch [32], and $y^{(k)}$ is the approximate solution at the kth Newton step. $\delta y^{(k)}$ is obtained by forming and solving the following linear system of equations using a Krylov subspace method; e.g., GMRES [23], together with a scalable Schwarz preconditioner,

$$B_k^{-1}\, \mathcal{J}(y^{(k)}) \delta y^{(k)} = -B_k^{-1}\, \mathcal{F}(y^{(k)}). \tag{7}$$

Here $\mathcal{J}(y^{(k)})$ is the Jacobian matrix at $y^{(k)}$, $\mathcal{F}(y^{(k)})$ is the nonlinear function residual evaluted at the kth Newton step, and B_k^{-1} is a scalable Schwarz preconditioner to be introduced shortly. $\mathcal{J}(y^{(k)})$ is computed analytically. Comparing with the approximate Jacobian calculated with a finite difference method, the analytic form reduces the overall computational cost in terms of the compute time and the number of Newton iterations.

To simplify the description of Schwarz preconditioner, we drop the subscript k in (7)

$$B^{-1}\, \mathcal{J} \delta y = -B^{-1}\, \mathcal{F}. \tag{8}$$

As discussed earlier, the mesh Ω_h is partitioned into np (np is the number of processor cores) submeshes $\Omega_{h,i}, i = 1, 2, \cdots, np$. For each submesh $\Omega_{h,i}$, we extend it by δ layers to overlap with its neighbors, and we denote these overlapping submeshes as $\Omega_{h,i}^{\delta}$. Note that extending the non-overlapping submeshs to the overlapping submeshs is accomplished according to the connectivity of the graph constructed based on the nonzero pattern of $\mathcal{J}$. Let r be a global vector defined on Ω_h. The submatrices and the subvectors associated with $\Omega_{h,i}$ and $\Omega_{h,i}^{\delta}$ are denoted as $\mathcal{J}_i$, r_i and $\mathcal{J}_i^{\delta}$, r_i^{δ}, respectively.

We define a restriction operator R_i^{δ} as

$$r_i^{\delta} = R_i^{\delta} r = \begin{pmatrix} I_i^{\delta} & 0 \end{pmatrix} \begin{pmatrix} r_i^{\delta} \\ r \backslash r_i^{\delta} \end{pmatrix},$$

where I_i^{δ} is an identity matrix which has the same dimension as r_i^{δ}. R_i^{δ} is used to restrict a global vector to the local overlapping submesh. $r \backslash r_i^{\delta}$ is a subvector consisting of all components that are in r but not in r_i^{δ}. The transpose of the restriction, $(R_i^{\delta})^T$, serves as a prolongation operator that maps a local vector defined on $\Omega_{h,i}$ to a vector defined on the whole domain Ω_h by padding zeros in appropriate locations. Similarly, R_i^0 extracts r_i from the global vector r to the non-overlapping submesh. The submesh Jacobian matrix $\mathcal{J}_i^{\delta}$ is formed using the restriction and prolongation operators, that is,

$$\mathcal{J}_i^{\delta} = R_i^{\delta}\, \mathcal{J} (R_i^{\delta})^T. \tag{9}$$

A basic additive Schwarz (AS) preconditioner [33, 34, 35] for Ω_h is defined as

$$B_{AS}^{-1} = \sum_{i=1}^{np} (R_i^{\delta})^T (\tilde{\mathcal{J}}_i^{\delta})^{-1} R_i^{\delta}. \tag{10}$$

Here $(\tilde{\mathcal{J}}_i^{\delta})^{-1}$ is an approximation of $(\mathcal{J}_i^{\delta})^{-1}$, usually, an incomplete LU (ILU) factorization [23]. In (10), different variants of preconditioner can be formed by replacing $(R_i^{\delta})^T$ or R_i^{δ} with $(R_i^0)^T$

or R_i^0. The replacements sometime significantly improve the performance. There are two important variants: restricted additive Schwarz (RAS) and additive Schwarz with harmonic extension (ASH) [36],

$$B_{RAS}^{-1} = \sum_{i=1}^{np} (R_i^0)^T (\tilde{\mathcal{J}}_i^{\delta})^{-1} R_i^{\delta}, \tag{11}$$

and

$$B_{ASH}^{-1} = \sum_{i=1}^{np} (R_i^{\delta})^T (\tilde{\mathcal{J}}_i^{\delta})^{-1} R_i^0. \tag{12}$$

Both B_{RAS}^{-1} and B_{ASH}^{-1} are not symmetric even when $\mathcal{J}$ is symmetric, but they can be symmetrized as

$$B_{RASH}^{-1} = \sum_{i=1}^{np} (R_i^0)^T (\tilde{\mathcal{J}}_i^{\delta})^{-1} R_i^0. \tag{13}$$

The performance of these four variants of preconditioner will be studied in the next section. As mentioned earlier, the system corresponding to $(\tilde{\mathcal{J}}_i^{\delta})^{-1}$ is obtained using ILU. To further improve the performance, the rows and columns of the sparse submesh matrix $\mathcal{J}_i^{\delta}$ is reordered using an appropriate reordering approach before factorization. We will discuss the reordering schemes in the next subsection. The overall algorithm is summarized in Algorithm 2.

Algorithm 2 Unsteady Newton-Krylov-Schwarz

Set the initial condition as $y_0 = 0$
Set convergence tolerance $stol$, $rtol$, $\max_{\text{time}}$ and $\max_{\text{Newton}}$
for $n = 1$ to $\max_{\text{time}}$ **do** ▷ Time stepper
  Use the soltuion at the previous time step as the initial guess for Newton, $y^{(0)} = y_{n-1}$
  for $k = 0$ to $\max_{\text{Newton}}$ **do** ▷ Newton solver
    Evaluate the Jacobian matrix $\mathcal{J}(y^{(k)})$
    Evaluate the nonlinear residual $\mathcal{F}(y^{(k)})$
    Construct the submesh preconditioner and compute the ILU factorizations
    Approximately solve $B_k^{-1}\,\mathcal{J}(y^{(k)})\delta y^{(k)} = -B_k^{-1}\,\mathcal{F}(y^{(k)})$ for a Newton direction $\delta y^{(k)}$
    Find a step size $\alpha^{(k)}$ using a cubic linesearch
    Update solution $y^{(k+1)} = y^{(k)} + \alpha^{(k)}\delta y^{(k)}$
    if $\|\delta y^{(k)}\| < stol\|y^{(k+1)}\|$ or $\|\,\mathcal{F}(y^{(k+1)})\| < rtol\|\,\mathcal{F}(y^{(1)})\|$ **then**
      break
    end if
  end for
  Store the solution at the last Newton step as the current time step solution, $y_n = y^{(k+1)}$
end for

Remark 3.1
In Algorithm 2, $stol$ is a pre-chosen small positive value that tells Newton to stop if $\|\delta y^{(k)}\|$ is small enough compared with $\|y^{(k+1)}\|$. $rtol$ is the relative stopping tolerance. B_k^{-1} is a Schwarz preconditioner which can be B_{AS}^{-1}, B_{RAS}^{-1}, B_{ASH}^{-1} or B_{RASH}^{-1}. Note that we can also apply a right preconditioner, that is,

$$\mathcal{J}(y^{(k)})B_k^{-1}B_k\delta y^{(k)} = -\,\mathcal{F}(y^{(k)}).$$

The Newton method is stopped when the linesearch iteration fails or the linear solver fails to converge. $\max_{\text{time}}$ is the maximum number of time steps, and $\max_{\text{Newton}}$ is the maximum number of Newton iterations.

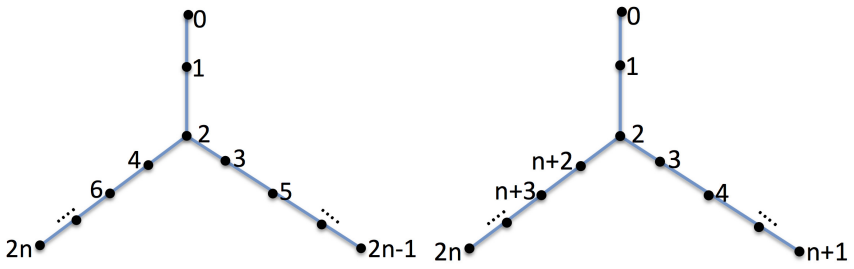
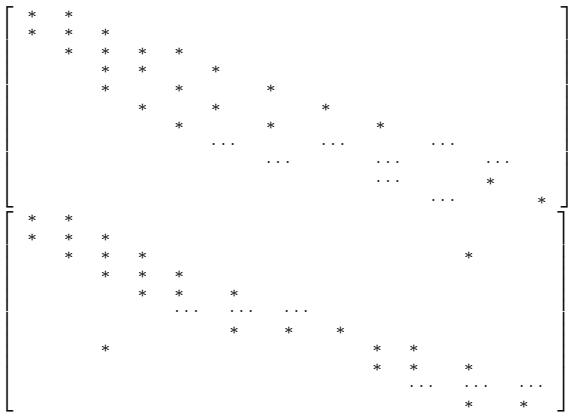


Figure 3. Two ordering approaches for one-dimensional two-branch artery. Left: alternating branch ordering. The right: one branch at a time ordering.

3.2. Matrix ordering for tree-like meshes

Computationally, the most expensive step of Algorithm 2 is the solving of the preconditioning system associated with B_k in which all the submesh matrices have to be constructed and solved. Because of the size of the problem, we don't solve these submesh problems exactly, instead, only an incomplete LU factorization of $\mathcal{J}_i^\delta$ is computed. Here "incomplete" refers to the fact that during the factorization process, lots of non-zeros are treated as zeros to save memory and arithmetic operations. Which non-zero entry is treated as a zero depends on how the matrix entries are ordered. A good ordering should result in a relatively small bandwidth. However, the initial ordering of the matrix, which comes from the mesh generation program, is often bad because of the tree-like structure of the pulmonary artery. Below we discuss the impact of different orderings on the nonzero pattern and bandwidth of the matrix, and then describe an effective ordering algorithm for the submesh matrices in (9).

The nonzero pattern and bandwidth of the matrix are very different if the matrix is ordered using different approaches. For example, in Fig 3, applying two ordering approaches, "alternating branch ordering" and "one branch at a time ordering", to a one-dimensional two-branch artery mesh leads to two matrix orderings, and the corresponding nonzero patterns and bandwidths are very different from each other. The nonzero patterns of the matrices for the alternating branch ordering and the one branch at a time ordering are shown in the top and the bottom matrices of (14), respectively.



(14)

The bandwidth of the top matrix of (14) is 5, and that for the bottom matrix of (14) is $n+2$ (n is the number of vertices on the right branch). When the number of branches increases, the bandwidth of the top matrix of (14) becomes larger, and the bandwidth of the bottom matrix of (14) is determined by the number of nodes on each individual branch. When a LU factorization is performed, the L and U matrices corresponding to the bottom matrix of (14) are much denser than that for the top matrix in (14). Similarly, ILU(k), an incomplete LU factorization with k levels of fill-ins, for the bottom matrix in (14) is less accurate than that of the above matrix in (14). Even with the same bandwidth, the ordering also plays a key role on the final nonzero pattern of the factored matrix.

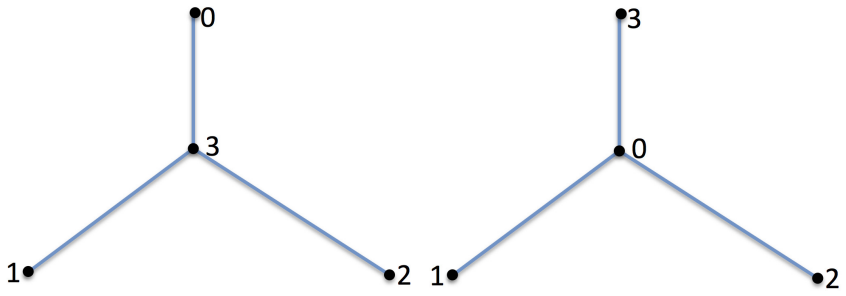


Figure 4. Different orderings for a small subtree.

$$\begin{bmatrix} * & & & * \\ & * & & * \\ & & * & * \\ * & * & * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * \\ * & * & & \\ * & & * & \\ * & & & * \end{bmatrix}. \tag{15}$$

For example, if the mesh points are ordered as in Fig. 4, the matrices would have the non-zero patterns as in (15), and the LU factorization of the left matrix in (15) does not change the sparsity pattern, and its L and U matrices have the same nonzero structure as the lower and upper triangular parts of the original matrix whose ILU(0) is the same as the LU factorization, on the other hand, the LU factorization of the right matrix in (15) involves extra fill-ins, and the corresponding ILU(0) is an approximation to LU. Therefore, ILU(0) of the left matrix is better than that of the right matrix. From the examples above, we see that a good ordering often leads to a more accurate factorization. Unfortunately, the original ordering, shown in Fig. 5, for a pulmonary artery mesh is created randomly, and the bad property is transferred to the submeshes when the global mesh is partitioned. A reordering is required to make the submesh solver efficient and save memory.

For a simple geometry such as a rectangular domain, the original (Natural) ordering is usually good, but for a more complex geometry, the ordering of the mesh generated using standard techniques based on a commercial package may be far from ideal. There are several reordering algorithms, namely, nested dissection (ND) [37], one-way dissection (1WD) [38], quotient minimum degree (QMD) [39], and reverse Cuthill-McKee (RCM) algorithm [40]. The basic idea of the reordering algorithm is to relabel the vertices of the underneath graph of a matrix in such a way that all entries in the matrix are aligned as close to the diagonal as possible and an LU factorization based on the permuted matrix costs as little memory as possible. After many numerical experiments, we find that RCM often works better than other choices. In the RCM ordering, a peripheral point (the point with the lowest degree) is chosen as the initial point and it is marked as a visited point, and then its immediate unvisited neighboring points are marked as candidates for the next visit. For each unvisited neighboring point, the same idea is recursively used. The neighboring points are visited from the lowest to the highest degree. The points are ordered according to the visiting sequence. Finally, this sequence is reversed. The detailed idea is shown in Algorithm 3, where nv is the number of vertices in the submesh, $N(v)$ represents the neighboring vertices of the current point v, and the stack S serves to reverse the ordering in the queue Q.

4. NUMERICAL EXPERIMENTS AND OBSERVATIONS

In this section, we report the results of some numerical experiments and also the parallel performance of the algorithm with respect to the number of processor cores when using different versions of Schwarz with various parameter settings, and the robustness with respect to the viscosity of the fluid flows. The algorithm is implemented based on PETSc [41], and all numerical simulations are performed on an IBM iDataPlex cluster consisting of Intel Sandy Bridge processor cores interconnected by an Infiniband network. For the rest of the section, "NI" denotes the average number of Newton iterations per time step, "LI" denotes the average number of GMRES iterations

Algorithm 3 Reverse Cuthill-McKee (RCM) [23]

Find a peripheral vertex as the initial point v_1
Initialize a queue $Q = \{v_1\}$, and $i = 1$
for $|Q| < nv$ **do**
    Retrieve the ith element of Q as v
    Find all neighboring points, $N(v)$, of v
    Increase i by 1, $i = i + 1$
    Order $N(v)$ from the lowest to the highest degree
    for each unmarked point $\tilde{v}$ in $N(v)$ **do**
        Append $\tilde{v}$ to Q
        Mark $\tilde{v}$ as visited
    end for
end for
Create an empty stack $S = \emptyset$
for $i = 1$ to nv **do**
    Retrieve the ith element of Q as v
    Push v into S
end for
Return S

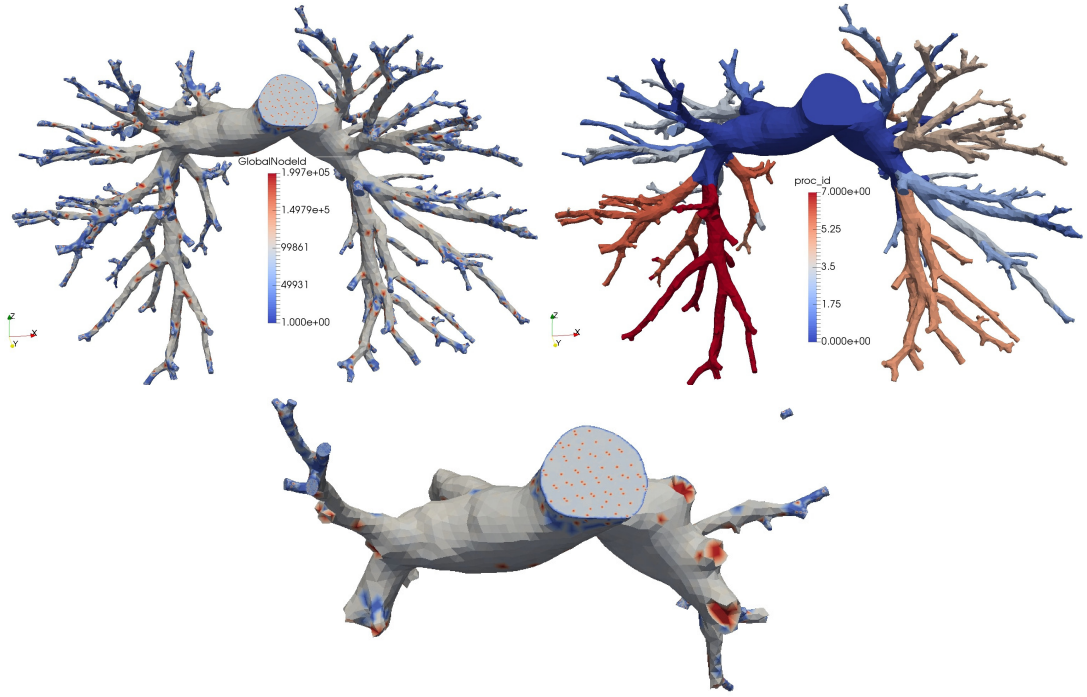


Figure 5. Top left: original (natural) ordering for the entire domain; top right: the entire domain is divided into 8 submeshs; bottom: the original ordering of the first submesh. This particular mesh has $199,721$ vertices, and we label them using integer $1, 2, ..., (199,721)$. Since there are too many numbers to be clearly shown in the figure, we represent them by colors as indicated by the legend in the top left figure. Similarly, in the top right figure, color is used to label the subdomain number. In the bottom figure, one can see that the color of some points is very different from its neighbors', which indicates that the corresponding matrix has a large bandwidth.

per Newton step, "EFF" is the parallel efficiency and "MEM" is the memory usage per processor core in megabytes, and "T" is the total compute time for 10 time steps in second. The restart value of GMRES is fixed at 500, the maximum number of GMRES iterations is set as 500, and the relative

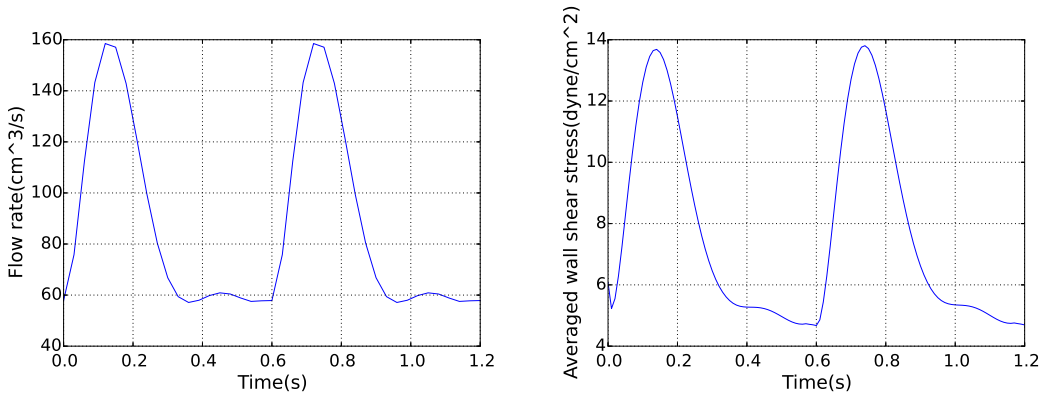


Figure 6. Left: the inlet blood flow rate for two heart beats. Right: Spatially averaged wall shear stress varying with time for two heart beats.

stopping condition for GMRES is set to be 10^{-4}. The relative stopping condition used for Newton is 10^{-6} and the maximum number of Newton iterations is fixed as 10. We set the overlapping size as 1, the submesh solver as ILU(1), the submatrix reordering as RCM by default, unless otherwise specified. B_{RAS}^{-1} is used as the default preconditioner. "np" denotes the number of processor cores.

The simulation is carried out for the patient-specific pulmonary, shown in Fig. 1, for two heart beats (1.2 seconds) with the time step size $\delta t = 10^{-3}$. A blood flow rate profile, shown in the left picture of Fig. 6, is applied as the inlet boundary condition and the traction-free boundary condition is applied to all outlets. The wall sheer stress (WSS) is an important metric, and it is calculated by the following formula

$$WSS = \boldsymbol{\sigma n} - (\boldsymbol{\sigma n} \cdot \boldsymbol{n})\boldsymbol{n},$$

and the spatially averaged WSS (SAWSS) is obtained by integrating the WSS on the entire domain surface and then normalized over the area, that is,

$$SAWSS = \frac{1}{A} \int_{\partial\Omega} WSS dA,$$

where A is total surface area of the pulmonary artery. In Fig. 6, we observe that the SAWSS is highly correlated with the input flow rate. The SAWSS increases when the input flow rate increases with time, and it decreases when the input flow rate decreases. In Fig. 8, the pressure and the velocity magnitude at three locations, L1, L2 and L3, as shown in Fig. 7, are plotted against time. It is easy to see that the pressure and the velocity magnitude at the proximal portion of the artery, such as L1, are highly correlated with the input flow rate, while those at the distal become flat over two heart beats. The velocity, the pressure, and the WSS at different time are shown in Fig 12, 13, and 14, respectively. Similarly, all values become larger when the input flow rate increases, and the WSS at non-smooth points are usually higher than other points where the geometry is smooth. From Fig. 13 and 14, it is easy to see that the pressure decreases in space from the inlet to the distal outlets, but the WSS is determined by the geometry; larger arteries have larger WSS values more often. In Fig. 12, in order to observe the detailed features of the velocity, the pictures are zoomed in at the main branch marked in the first picture of Fig. 12. The flow is turbulent at the right branch of the artery, and the speed of the flow becomes much larger at 0.75s. The pattern of the turbulent flow at the start and the end of the heart cycle are the same; indicating the flow is periodic in time.

Next, we investigate how different parameters impact the numerical performance of the algorithm. The nonlinear system is solved using Algorithm 2, and the performance of the algorithm in terms of the compute time and the number of iterations with respect to the number of processor cores are reported below using different Schwarz parameters.

(1) Impact of submatrix reordering. Algorithm 2 does not work well, and sometimes diverges, if the submesh matrices are ordered using the original ordering provided by the mesh generation

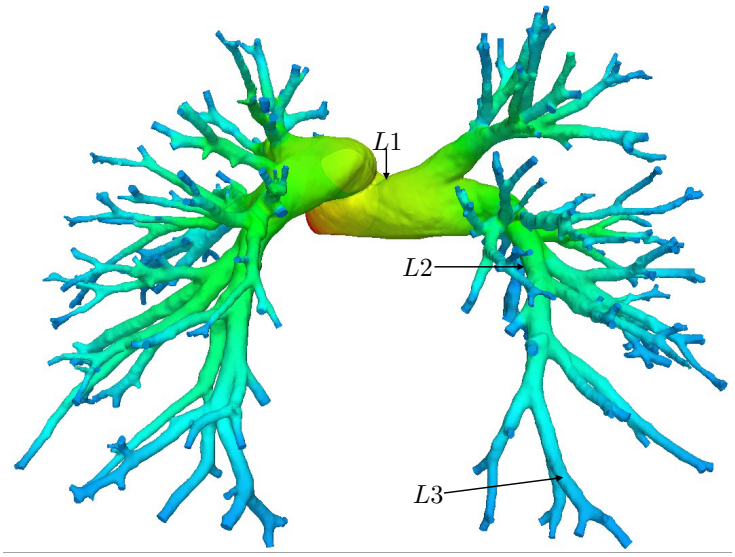


Figure 7. The locations of observation. We observe the velocity and the pressure varying with time at three different locations marked as $L1$, $L2$ and $L3$.

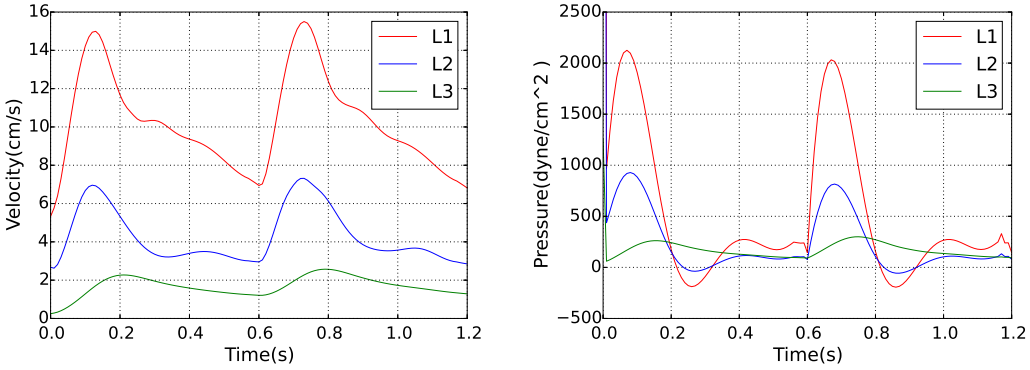


Figure 8. The velocity magnitude and the pressure at $L1$, $L2$ and $L3$.

program. To fix this issue, we need to effectively reorder the submatrices. In this test, several ordering schemes such as Natural, ND, 1WD, QMD and RCM are studied. "Natural" represents the original ordering. The mesh used in this test has 10,499,044 elements, 2,155,399 vertices and the corresponding system has 6,966,955 unknowns, and the same mesh will also be used in the following tests, unless otherwise specified. The simulation is carried out for 10 time steps using 128, 256, 512 and 1,024 processor cores, respectively. The results are summarized in Table I and the left of Fig. 9.

In Table I, the number of Newton iterations stays as a constant, 2.5, for all reordering schemes when we increase the number of processor cores. The number of GMRES iterations for different reordering schemes is different, sometime significantly different. For example, when using 128 processor cores, the number of GMRES iterations is 625 with Natural ordering, while it is 327 when we use RCM. The number of GMRES iterations is similar for 1WD and RCM, and the two schemes result in the similar performance in terms of the compute time and the memory usage. The compute time is 332 seconds for 1WD and 327 seconds for RCM when we use 128 processor cores, and the corresponding memory usage is 395 and 365 megabytes, respectively. It is easy to observe that the performance of 1WD is very similar to RCM, and RCM is a little better than 1WD. RCM is the best in most cases except when we use 512 processor cores, where RCM is slightly slower than

Table I. Impact of submatrix reordering with different number of processor cores. A nonlinear system with 6,966,955 unknowns is solved by NKS with different reordering schemes for submesh solvers. np is the number of processor cores, "NI" denotes the average number of Newton iterations per time step, "LI" denotes the average number of GMRES iterations per Newton step, "EFF" is the parallel efficiency and "MEM" is the memory usage per processor core in megabytes, and "T" is the total compute time in second for 10 time steps. "Ordering" represents a subdomain matrix reordering scheme.

np	Ordering	NI	LI	T	MEM	EFF
128	Natural	2.5	243	625	545	100%
128	ND	2.5	205	520	567	100%
128	1WD	2.5	131	332	395	100%
128	QMD	2.5	168	440	517	100%
128	RCM	2.5	128	327	365	100%
256	Natural	2.5	214	290	339	108%
256	ND	2.5	204	275	332	95%
256	1WD	2.5	133	177	221	94%
256	QMD	2.5	151	211	264	100%
256	RCM	2.5	129	172	234	95%
512	Natural	2.5	181	136	199	115%
512	ND	2.5	194	144	242	90%
512	1WD	2.5	136	96	190	87%
512	QMD	2.5	163	124	194	89%
512	RCM	2.5	107	102	229	80%
1,024	Natural	2.5	213	92	175	85%
1,024	ND	2.5	189	84	177	77%
1,024	1WD	2.5	138	56	136	74%
1,024	QMD	2.5	153	65	147	85%
1,024	RCM	2.5	137	55	115	74%

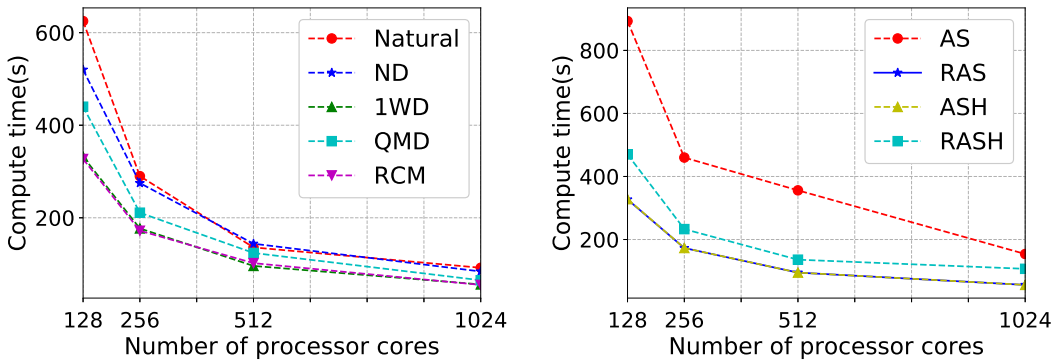


Figure 9. Compute time for different submatrix reordering schemes and different Schwarz variants. Left: submatrix reordering; right: variant of Schwarz preconditioner.

1WD in the compute time, and also a little more expensive in the memory usage. Natural ordering is the worst, ND is the second worst in the GMRES iteration and the compute time, and ND does not improve the performance much because it is originally designed for linear systems arising from regular meshes [37]. QMD is a popular approach for minimizing fill-ins in a factorization [23], but it does not perform well for the case of complex geometry, such as the pulmonary artery. The algorithm is scalable with respect to the number of processor cores in terms of the compute time and the number of iterations (GMRES iterations and Newton iterations) for all cases. Because RCM performs quite well in this test, we will use it as the submatrix reordering in the following tests by default.

Table II. Different variants of Schwarz preconditioner. Nonlinear equations with 6,966,955 unknowns is solved by an inexact Newton-Krylov method preconditioned by different variants of Schwarz preconditioner. np is the number of processor cores, "NI" denotes the average number of Newton iterations per time step, "LI" denotes the average number of GMRES iterations per Newton step, "EFF" is the parallel efficiency and "MEM" is the memory usage per processor core in megabytes, and "T" is the total compute time in second for 10 time steps. "type" represents a Schwarz variant.

np	type	NI	LI	T	MEM	EFF
128	AS	3	329	893	515	100%
128	RAS	2.5	128	327	403	100%
128	ASH	2.5	128	328	403	100%
128	RASH	2.5	205	470	508	100%
256	AS	2.8	345	460	296	97%
256	RAS	2.5	129	173	272	95%
256	ASH	2.5	129	173	270	95%
256	RASH	2.5	193	233	269	100%
512	AS	2.9	467	356	240	63%
512	RAS	2.5	132	95	171	86%
512	ASH	2.5	133	95	169	86%
512	RASH	2.5	210	136	177	86%
1,024	AS	2.8	360	154	170	72%
1,024	RAS	2.5	137	56	152	73%
1,024	ASH	2.5	137	56	179	73%
1,024	RASH	2.5	285	107	168	55%

(2) Different variants of Schwarz preconditioner. As discussed before, the overlap for either the prolongation or the restriction operator or both can be dropped to reduce the communication cost and improve the algorithm. Here we test the performance of these algorithms. The same test problem as before is reused here. We report the results in Table II and the right of Fig. 9.

It is easy to see that, in Table II, AS is the worst in terms of the compute time, the number of iterations and the memory usage. RAS and ASH have a similar performance in terms of the four metrics (the number of linear and nonlinear iterations, the compute time and the memory usage), except when the number of processor cores is 1,024, the memory usage per core for ASH is a little more than that for RAS. RASH is cheaper because its matrix is smaller, but unfortunately, it is not scalable when the number of processor cores is large. For example, the parallel efficiency drops to 55% at $np = 1,024$ while other three preconditioners still have more than 70% parallel efficiency. In all, both RAS and ASH are scalable for $np = 128, 256, 512,$ and $1,024$ in terms of all metrics. We use RAS as the default Schwarz preconditioner in the following tests.

(3) The level of fill-ins for ILU. ILU with l level of fill-ins is denoted as ILU(l), and ILU(0), ILU(1) and ILU(2) are tested. The results with $np = 128, 256, 512,$ and $1,024$ are summarized in Table III and the left of Fig. 10.

As shown in Table III, when the number of processor cores increases from 128 to 1,024, the number of Newton iterations stays as a constant for ILU(1) and ILU(2), but slightly varies for ILU(0). ILU(0) is worse than ILU(1) and ILU(2) for any core counts even through its compute time is superlinearly reduced. The number of GMRES stays near a constant for ILU(1) with different number of processor cores. ILU(2) further decreases the number of GMRES iterations about by 30, when compared with ILU(1). However, ILU(2) does not decrease the compute time, because its cost per iteration is higher than ILU(1) and the reduction of the number of GMRES iterations can not compensate for the extra cost. ILU(1) is the best in this case in terms of the compute time and the memory usage.

(4) The overlapping size. The overlap plays a critical role in a Schwarz-type preconditioner, and we study the impact of overalpping size in this test. A larger overlapping size often results in fewer number of GMRES iterations, but the communication and the computational cost per iteration increase.

Table III. Impact of submesh solver. Nonlinear system with 6,966,955 unknowns is solved an inexact NKS with an incomplete LU factorization with different levels of fill-ins as a submesh solver. np is the number of processor cores, "NI" denotes the average number of Newton iterations per time step, "LI" denotes the average number of GMRES iterations per Newton step, "EFF" is the parallel efficiency and "MEM" is the memory usage per processor core in megabytes, and "T" is the total compute time in second for 10 time steps. "subsolver" represents a subdomain solver.

np	subsolver	NI	LI	T	MEM	EFF
128	ILU(0)	2.8	450	1053	475	100%
128	ILU(1)	2.5	128	327	365	100%
128	ILU(2)	2.5	100	339	453	100%
256	ILU(0)	2.8	361	433	276	121%
256	ILU(1)	2.5	129	172	234	95%
256	ILU(2)	2.5	103	184	240	92%
512	ILU(0)	2.7	327	210	196	125%
512	ILU(1)	2.5	132	95	158	86%
512	ILU(2)	2.5	107	102	229	80%
1,024	ILU(0)	2.5	270	93	159	141%
1,024	ILU(1)	2.5	137	55	115	74%
1,024	ILU(2)	2.5	113	57	170	74%

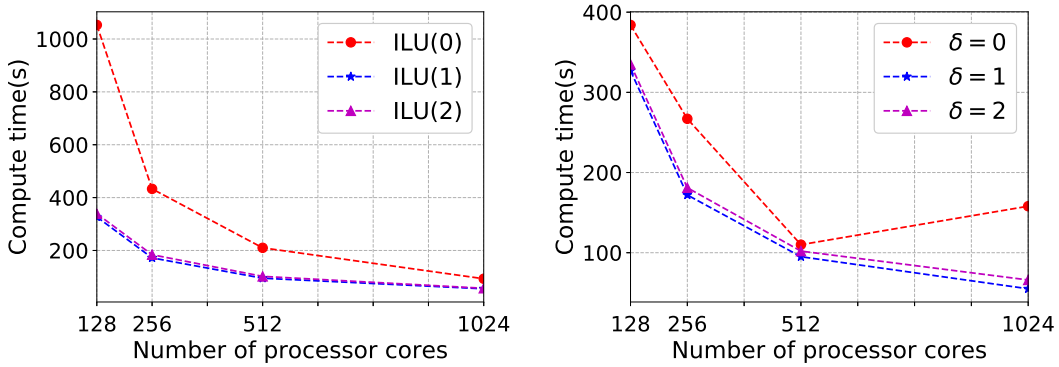


Figure 10. Compute time for different levels of fill-ins of ILU and different overlapping sizes. Left: level of fill-ins of ILU; right: overlapping size.

From the results in Table IV and the right of Fig. 10, we see that the number of Newton iterations stays as a constant for all cases when we increase the number of processor cores. When $\delta = 0$, the number of GMRES iterations increases with the increase of the number of processor cores, especially at $np = 1,024$, where the number of GMRES iterations is doubled. As a result, the parallel efficiency is deteriorated to 30%, which is an indication that a larger overlapping size is required. The performance improves a lot when we increase the overlapping size from 0 to 1. It is easy to see that the number of GMRES iterations and the compute time for $\delta = 1$ are reduced significantly, when compared with $\delta = 0$. Further increasing the overlapping size from 1 to 2 decreases the number of GMRES iterations slightly, but increases the compute time because the cost per iteration increases. It is interesting to note that $\delta = 2$ keeps the number of GMRES iteration as a constant for all processor counts, which implies that increasing the overlapping makes the algorithm arithmetically scalable, but quite scalable in terms of the compute time. We will use $\delta = 1$ in the following tests because it has the best performance in terms of the compute time.

(5) The robustness of algorithms with respect to different viscosity. The viscosity is an important physics parameter for incompressible flows, and smaller viscosity value often corresponds to a more complicated flow pattern, and at the same time, the corresponding nonlinear/linear systems become

Table IV. Impact of overlap. Nonlinear system with 6,966,955 unknowns is solved by a Schwarz preconditioned inexact Newton-Krylov method. np is the number of processor cores, "NI" denotes the average number of Newton iterations per time step, "LI" denotes the average number of GMRES iterations per Newton step, "EFF" is the parallel efficiency and "MEM" is the memory usage per processor core in megabytes, and "T" is the total compute time in second for 10 time steps. δ represents the subdomain overlapping size.

np	δ	NI	LI	T	MEM	EFF
128	0	2.5	167	384	456	100%
128	1	2.5	128	327	365	100%
128	2	2.5	125	335	419	100%
256	0	2.7	215	267	282	72%
256	1	2.5	129	172	234	95%
256	2	2.5	125	181	227	93%
512	0	2.5	183	110	168	87%
512	1	2.5	132	95	158	86%
512	2	2.5	125	102	213	82%
1,024	0	2.8	431	158	189	30%
1,024	1	2.5	137	55	115	74%
1,024	2	2.5	125	66	159	63%

Table V. Robustness of the algorithms with respect to different viscosity and different number of processor cores. A nonlinear system with 6,966,955 unknowns is solved by an inexact Newton-Krylov-Schwarz method. np is the number of processor cores, "NI" denotes the average number of Newton iterations per time step, "LI" denotes the average number of GMRES iterations per Newton step, "EFF" is the parallel efficiency and "MEM" is the memory usage per processor core in megabytes, and "T" is the total compute time in second for 10 time steps. ν represents the viscosity in $g/(cm\ s)$.

np	ν	NI	LI	T	MEM	EFF
128	0.05	2.4	101	330	453	100%
128	0.01	2.8	96	371	428	100%
128	0.001	3.6	89	463	458	100%
256	0.05	2.4	105	178	272	93%
256	0.01	2.8	101	201	242	92%
256	0.001	3.6	93	253	243	92%
512	0.05	2.4	109	100	231	83%
512	0.01	2.8	104	112	231	83%
512	0.001	3.6	95	137	202	84%
1,024	0.05	2.4	115	56	208	74%
1,024	0.01	2.8	110	64	203	72%
1,024	0.001	3.6	99	77	182	75%

harder to solve. We examine a few values of viscosity in this test to verify the robustness of the algorithm. The results are summarized in Table V and the left of FIg. 11.

From Table V, we observe that the number of Newton iterations increases for smaller viscosities because the problem becomes nonlinearly more difficult. The averaged number of GMRES iterations per Newton step decreases slightly when we reduce the viscosity. The compute time also increases because of the increase of Newton iterations since the Newton steps are expensive in the sense that the Jacobian and preconditioner have to be recomputed during each Newton step. The memory usage for $\nu = 0.05, 0.01,$ and 0.001 is similar. All cases have good scalability, which implies that the algorithm is robust with respect to different values of viscosity.

(6) Scalability with respect to a large number of processor cores. To test the parallel scalability of the proposed algorithm, a mesh with 83,992,352 elements, 15,531,731 nodes and 55,575,035 unknowns is used. The strong scalability with the number of processor cores ranging from 1,024 to 10,240 is studied. The problem is more difficult to solve when the mesh is larger. Therefore,

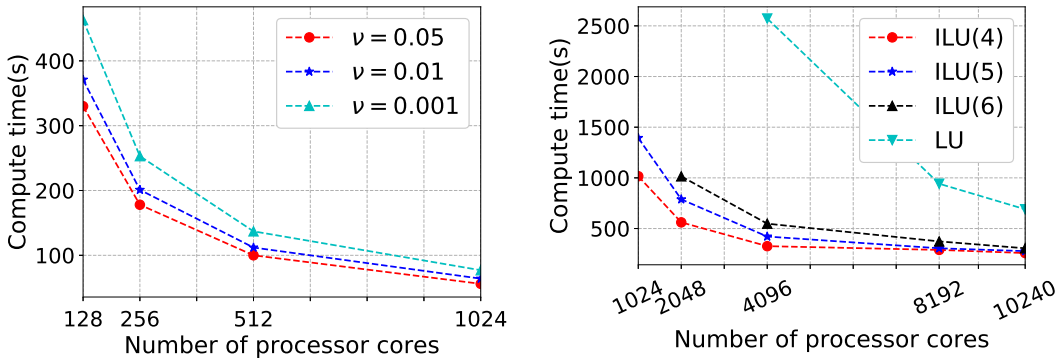


Figure 11. Robustness and parallel scalability. Left: robustness with respect to ν; right: parallel scalability with respect to the number of processor cores.

Table VI. Scalability with a large number of processor cores. A system with 55,575,035 unknowns is solved by a Schwarz preconditioned inexact Newton-Krylov method. np is the number of processor cores, "NI" denotes the average number of Newton iterations per time step, "LI" denotes the average number of GMRES iterations per Newton step, "EFF" is the parallel efficiency and "MEM" is the memory usage per processor core in megabytes, and "T" is the total compute time in second for 10 time steps. "subsolver" represents the subdomain solver.

np	subsolver	NI	LI	T	MEM	EFF
1,024	ILU(4)	2.5	163	1,016	1,120	100%
1,024	ILU(5)	2.5	154	1,394	1,383	100%
1,024	ILU(6)	–	–	–	–	–%
1,024	LU	–	–	–	–	—%
2,048	ILU(4)	2.5	183	561	579	91%
2,048	ILU(5)	2.5	176	788	664	88%
2,048	ILU(6)	2.5	172	1015	744	100%
2,048	lu	–	–	–	–	—%
4,096	ILU(4)	2.5	205	326	336	78%
4,096	ILU(5)	2.5	198	421	382	83%
4,096	ILU(6)	2.5	195	548	430	93%
4,096	LU	2.5	197	2572	866	100%
8,192	ILU(4)	2.5	366	289	272	44%
8,192	ILU(5)	2.5	298	305	332	57%
8,192	ILU(6)	2.5	291	373	328	68%
8,192	LU	2.5	264	942	459	137%
10,240	ILU(4)	2.5	404	258	251	39%
10,240	ILU(5)	2.5	345	277	238	50%
10,240	ILU(6)	2.5	305	305	256	67%
10,240	LU	2.5	308	690	359	149%

if ILU($k < 4$) were used as a submesh solver, the overall algorithm does not converge well. The smallest level of fill-ins is 4 in this particular case. Fortunately, the algorithm framework is flexible so that there are different choices of the individual component that make the algorithm work for a given problem. The numerical results are summarized in Table VI and the right of Fig. 11, from which we see that, the memory usage per core is halved when we double the number of processor cores, which indicates that the additive Schwarz method with the RCM reordering is scalable in terms of the memory. For example, the memory usage per core is 1,120 MB with ILU(4) on 1,024 cores, and it is reduced to 579 MB (that is almost half of 1,129 MB) when using 2,048 cores. The algorithm is scalable in terms of the nonlinear iterations because the number of Newton iterations

stays as a constant for all cases when we increase the number of processor cores. The number of linear iterations increases gradually with the number of processor cores, especially from 4,096 to 8,192, where the number of linear iterations is increased by more than 100. We hence see a parallel efficiency drop when go from 4,096 to 8,192, but we still have a reasonably good efficiency more than 40%. ILU(4) offers the best performance in terms of the compute time compared with other submesh solvers. We conclude that the overall algorithm is scalable in terms of Newton iterations and the memory usage for all processor counts, and it is also scalable in terms of the compute time and the number of linear iterations especially when the number of processor cores is less than or equal to 4,096. The parallel scalability using more than 8,192 processor cores will be further improved in the future by exploring a way to use a multilevel version of the proposed approach.

5. CONCLUDING REMARKS

A highly scalable parallel unsteady fluid solver was proposed for patient-specific pulmonary artery trees with up to the 7th generation. A hierarchal partitioning is used to divide the pulmonary artery mesh to a large number of submeshs, each submesh is assigned to a processor core, and the computation is equally distributed over a large number of processor cores. The system of incompressible Naiver-Stokes equations is employed to model the blood flow, and it is discretized using a stabilized finite element method in space and an implicit scheme in time. The resulting system of nonlinear equations is solved using an inexact Newton method, where the Jacobian system is solved using a Krylov subspace method together with a Schwarz preconditioner with reordered submesh matrices. The traditional Schwarz precondtioner without a careful reordering does not work well, and the RCM reordering method significantly improves the overall algorithm performance. We numerically show that the reorganized NKS approach is scalable on a machine with more than 10,000 processor cores for a complex unsteady flow problem in an extremely complex geometry, with hundreds of outlets and secondary flows, such as the pulmonary arterial tree.

ACKNOWLEDGMENTS

This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 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 nonexclusive, 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.

REFERENCES

1. Kheyfets VO, Rios L, Smith T, Schroeder T, Mueller J, Murali S, Lasorda D, Zikos A, Spotti J, Reilly JJ, *et al.*. Patient-specific computational modeling of blood flow in the pulmonary arterial circulation. *Comput. Methods Programs Biomed.* 2015; **120**(2):88–101.
2. Su Z, Hunter KS, Shandas R. Impact of pulmonary vascular stiffness and vasodilator treatment in pediatric pulmonary hypertension: 21 patient-specific fluid–structure interaction studies. *Comput. Methods Programs Biomed.* 2012; **108**(2):617–628.
3. De Leval M, Dubini G, Jalali H, Pietrabissa R, *et al.*. Use of computational fluid dynamics in the design of surgical procedures: application to the study of competitive flows in cavopulmonary connections. *J. Thorac. Cardiovasc. Surg.* 1996; **111**(3):502–513.
4. Taylor CA, Steinman DA. Image-based modeling of blood flow and vessel wall dynamics: applications, methods and future directions. *Ann. Biomed. Eng.* 2010; **38**(3):1188–1203.
5. Taylor CA, Figueroa C. Patient-specific modeling of cardiovascular mechanics. *Annu. Rev. Biomed. Eng.* 2009; **11**:109.
6. Müller LO, Parés C, Toro EF. Well-balanced high-order numerical schemes for one-dimensional blood flow in vessels with varying mechanical properties. *J. Comput. Phys.* 2013; **242**:53–85.

7. Sherwin S, Formaggia L, Peiro J, Franke V. Computational modelling of 1D blood flow with variable mechanical properties and its application to the simulation of wave propagation in the human arterial system. *Int. J. Numer. Methods Fluids* 2003; **43**(6-7):673–700.
8. Qureshi MU, Vaughan GD, Sainsbury C, Johnson M, Peskin CS, Olufsen MS, Hill N. Numerical simulation of blood flow and pressure drop in the pulmonary arterial and venous circulation. *Biomech. Model. Mechanobiol.* 2014; **13**(5):1137–1154.
9. Čanić S, Mikelić A, Tambača J. A two-dimensional effective model describing fluid–structure interaction in blood flow: analysis, simulation and experimental validation. *Comptes Rendus Mécanique* 2005; **333**(12):867–883.
10. Casulli V, Dumbser M, Toro EF. Semi-implicit numerical modeling of axially symmetric flows in compliant arterial systems. *Int. J. Numer. Methods Biomed. Eng.* 2012; **28**(2):257–272.
11. Badia S, Quaini A, Quarteroni A. Splitting methods based on algebraic factorization for fluid-structure interaction. *SIAM J. Sci. Comput.* 2008; **30**(4):1778–1805.
12. Kong F. A Parallel Implicit Fluid-structure Interaction Solver with Isogeometric Coarse Spaces for 3D Unstructured Mesh Problems with Complex Geometry. PhD Thesis, UNIVERSITY OF COLORADO BOULDER 2016.
13. Kong F, Cai XC. Scalability study of an implicit solver for coupled fluid-structure interaction problems on unstructured meshes in 3D. *Int. J. High Perform. Comput. Appl.* 2016; :DOI: 10.1177/1094342016646 437.
14. Kong F, Cai XC. A scalable nonlinear fluid–structure interaction solver based on a Schwarz preconditioner with isogeometric unstructured coarse spaces in 3D. *J. Comput. Phys.* 2017; **340**:498–518.
15. Spilker RL, Feinstein JA, Parker DW, Reddy VM, Taylor CA. Morphometry-based impedance boundary conditions for patient-specific modeling of blood flow in pulmonary arteries. *Ann. Biomed. Eng.* 2007; **35**(4):546–559.
16. Tang BT, Fonte TA, Chan FP, Tsao PS, Feinstein JA, Taylor CA. Three-dimensional hemodynamics in the human pulmonary arteries under resting and exercise conditions. *Ann. Biomed. Eng.* 2011; **39**(1):347–358.
17. Hunter KS, Lanning CJ, Chen SYJ, Zhang Y, Garg R, Ivy DD, Shandas R. Simulations of congenital septal defect closure and reactivity testing in patient-specific models of the pediatric pulmonary vasculature: a 3d numerical study with fluid-structure interaction. *J. Biomech. Eng.* 2006; **128**(4):564–572.
18. CFD-ACE+. *https://www.esi-group.com* 2017; .
19. Yang X, Liu Y, Yang J. Fluid-structure interaction in a pulmonary arterial bifurcation. *J. Biomech.* 2007; **40**(12):2694–2699.
20. ANSYS. *http://www.ansys.com* 2017; .
21. Wu Y, Cai XC. A parallel two-level method for simulating blood flows in branching arteries with the resistive boundary condition. *Comput. Fluids* 2011; **45**(1):92–102.
22. Eisenstat SC, Walker HF. Globally convergent inexact Newton methods. *SIAM J. Optim.* 1994; **4**(2):393–422.
23. Saad Y. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.
24. Munteanu M, Pavarino LF, Scacchi S. A scalable Newton-Krylov-Schwarz method for the Bidomain reaction-diffusion system. *SIAM J. Sci. Comput.* 2009; **31**(5):3861–3883.
25. Kong F, Cai XC. A highly scalable multilevel schwarz method with boundary geometry preserving coarse spaces for 3D elasticity problems on domains with complex geometry. *SIAM J. Sci. Comput.* 2016; **38**(2):C73–C95.
26. Karypis G, Schloegel K. ParMETIS - Parallel Graph Partitioning and Sparse Matrix Ordering Library Version 4.0. *University of Minnesota* 2013; .
27. Karypis G. METIS - A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 5.1.0. *University of Minnesota* 2013; .
28. Taylor CA, Hughes TJ, Zarins CK. Finite element modelling of blood flow in arteries. *Comput. Methods Appl. Mech. Engrg.* 1998; **158**(1):155–196.
29. Whiting CH, Jansen KE. A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. *Int. J. Numer. Meth. Fluids* 2001; **35**(1):93–116.
30. Wu Y, Cai XC. A fully implicit domain decomposition based ALE framework for three-dimensional fluid-structure interaction with application in blood flow computation. *J. Comput. Phys.* 2014; **258**:524–537.
31. Cai XC, Gropp WD, Keyes DE, Melvin RG, Young DP. Parallel Newton-Krylov-Schwarz algorithms for the transonic full potential equation. *SIAM J. Sci. Comput.* 1998; **19**(1):246–265.
32. Nocedal J, Wright S. *Numerical Optimization*. Springer Science & Business Media, 2006.
33. Quarteroni A, Valli A. *Domain Decomposition Methods for Partial Differential Equations*. CMCS-BOOK-2009-019, Oxford University Press, 1999.
34. Smith B, Bjorstad P, Gropp W. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 2004.
35. Toselli A, Widlund O. *Domain Decomposition Methods: Algorithms and Theory*. Springer: Berlin, 2005.
36. Cai XC, Sarkis M. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM J. Sci. Comput.* 1999; **21**(2):792–797.
37. George A. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.* 1973; **10**(2):345–363.
38. GEoRGE A. An automatic one-way dissection algorithm for irregular finite element problems. *Numerical Analysis*. Springer, 1978; 76–89.
39. George A, Liu JW. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Trans. Math. Software* 1980; **6**(3):337–358.
40. George A, Liu JW. Computer solution of large sparse positive definite 1981; .
41. Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, Dalcin L, Eijkhout V, Gropp WD, Kaushik D, *et al.*. PETSc users manual. *Technical Report ANL-95/11 - Revision 3.7*, Argonne National Laboratory 2016. URL `http://www.mcs.anl.gov/petsc`.

APPENDIX

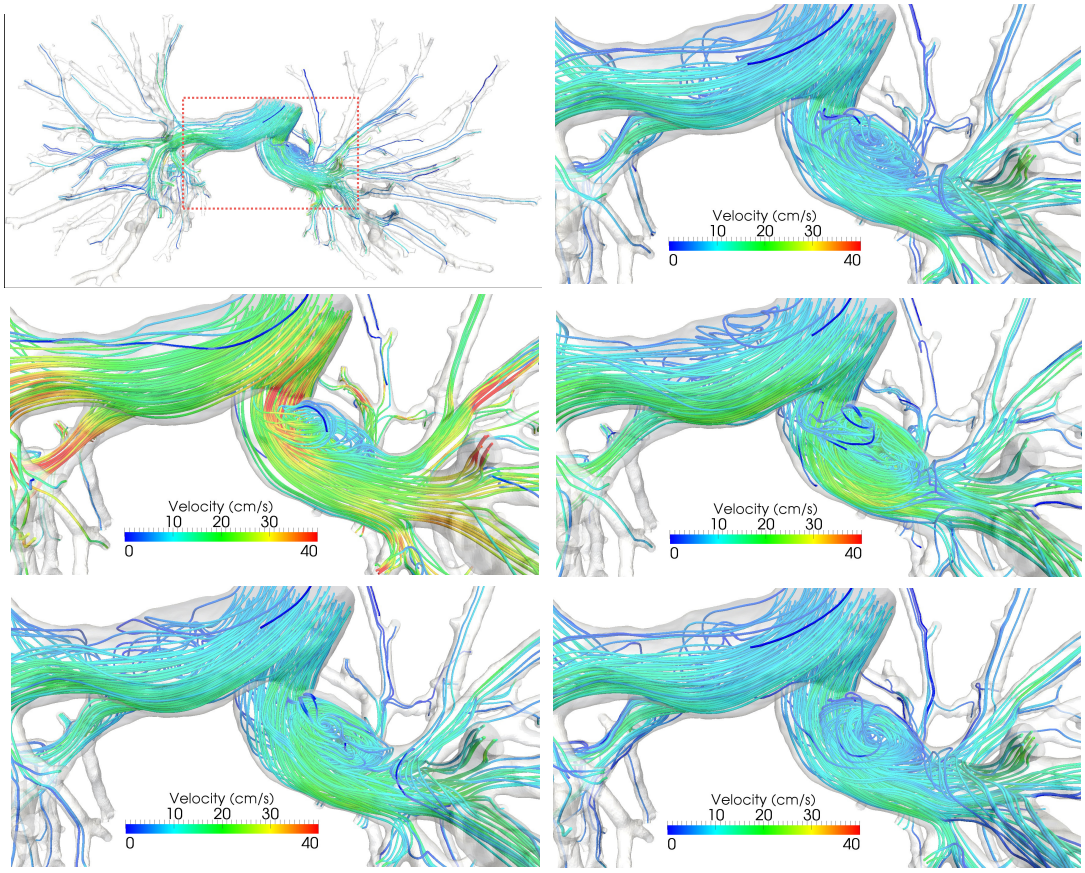


Figure 12. Streamlines of the velocity field at $t = 0.6s, 0.75s, 0.9s, 1.05s$ and $1.2s$. The pictures are ordered from left to right and from top to bottom.

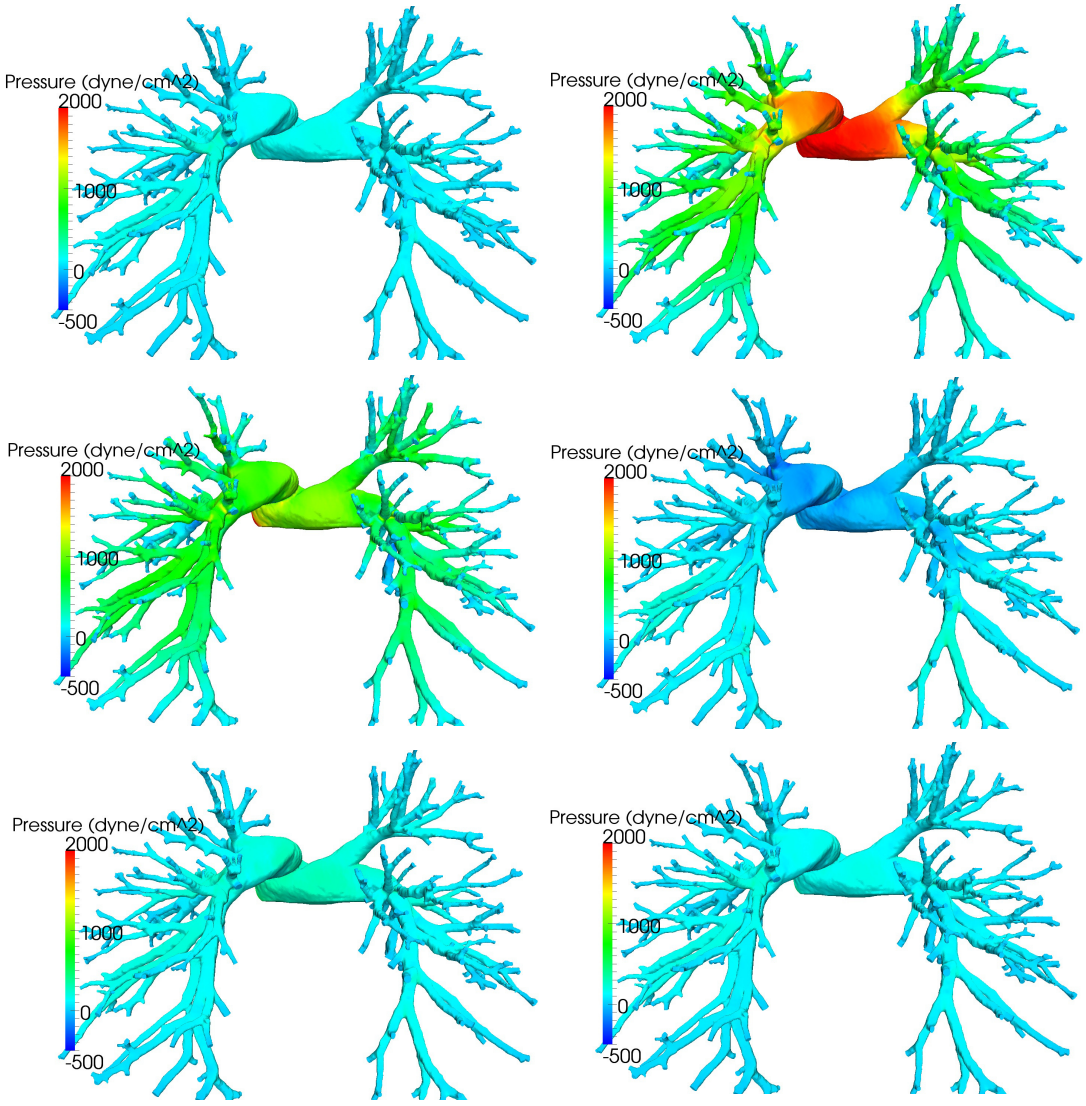


Figure 13. Pressure distribution at $t = 0.6s, 0.7s, 0.75s, 0.9s, 1.05s$ and $1.2s$. The pictures are ordered from left to right and from top to bottom.

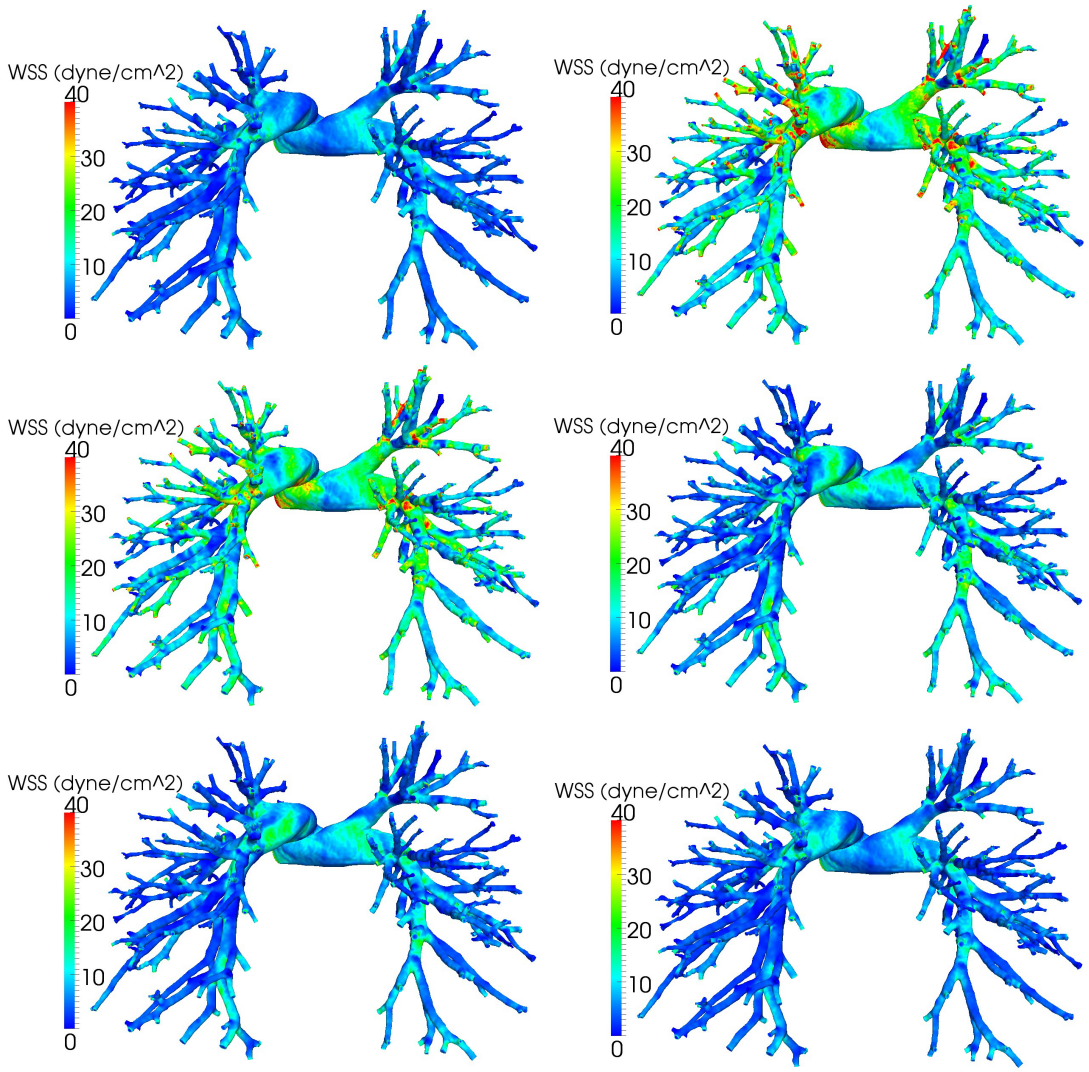


Figure 14. Wall shear stress distribution at $t = 0.6s, 0.75s, 0.8s, 0.9s, 1.05s$ and $1.2s$. The pictures are ordered from left to right and from top to bottom.