UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

UPCommons

Portal del coneixement obert de la UPC

http://upcommons.upc.edu/e-print[s

This is a post-peer-review, pre-copyedit version of an article published in *Computational mechanics*. The final authenticated version is available online at: https://doi.org/10.1007/s00466-020-01860-y.

Computational Mechanics manuscript No.
(will be inserted by the editor)

A Scalable Framework for the Partitioned Solution of Fluid-Structure Interaction Problems

Alireza Naseri[1], Amin Totounferoush[2], Ignacio González[1], Miriam Mehl[2], Carlos David Pérez-Segarra[1]

[1] Heat and Mass Transfer Technological Center (CTTC)
Universitat Politécnica de Catalunya - BarcelonaTech (UPC)
ESEIAAT, C/ Colom 11, 08222 Terrassa (Barcelona), Spain
[2] Institute for Parallel and Distributed Systems (IPVS)
University of Stuttgart
Universitätsstraße 38, D-70569 Stuttgart, Germany

The date of receipt and acceptance will be inserted by the editor

Abstract In this work, we present a scalable and efficient parallel solver for the partitioned solution of fluid-structure interaction problems through multi–code coupling. Two instances of an in–house parallel software, *TermoFluids*, are used to solve the fluid and the structural sub–problems, coupled together on the interface via the *preCICE* coupling library. For fluid flow, the Arbitrary Lagrangian-Eulerian form of the Navier-Stokes equations is solved on an unstructured conforming grid using a second–order finite–volume discretization. A parallel dynamic mesh method for unstructured meshes is used to track the moving boundary. For the structural problem, the nonlinear elastodynamics equations are solved on an unstructured grid using a second–order finite–volume method. A semi–implicit FSI coupling method is used which segregates the fluid pressure term and couples it strongly to the structure, while the remaining fluid terms and the geometrical nonlinearities are only loosely coupled. A robust and advanced multi–vector quasi–Newton method is used for the coupling iterations between the solvers. Both the fluid and the structural solver use distributed–memory parallelism. The intra–solver communication required for data update in the solution process is carried out using non–blocking point–to–point communicators. The inter–code communication is fully parallel and point–to–point, avoiding any central communication unit. Inside each single–physics solver, the load is balanced by dividing the computational domain into fairly equal blocks for each process. Additionally, a load balancing model is used at the inter–code level to minimize the overall idle time of the processes. Two

practical test cases in the context of hemodynamics are studied, demonstrating the accuracy and computational efficiency of the coupled solver. Strong scalability test results show a parallel efficiency of 83% on 10,080 CPU cores.

Key words Fluid-Structure Interaction; Partitioned Method; Multi-Code Coupling; Scalability; High Performance Computing

1 Introduction

Numerical methods to solve fluid-structure interaction (FSI) problems can be broadly divided into two categories, monolithic and partitioned. In a monolithic approach, the fluid and the solid equations are discretized and solved as a single large system, inherently accounting for their mutual interaction (see e.g. [1,2]). In a partitioned approach, on the other hand, the FSI problem is divided into two domains, for fluid and solid. As a result, these methods use separate solvers for fluid and structural sub-problems and adopt a coupling technique to account for the interaction of the domains. One of the big advantages of the partitioned approach is the possibility to use the most adapted and well-validated numerical methods for each sub-problem. Moreover, it allows using previously developed and computationally optimized fluid and structural solver codes, thus saving excessive software development effort [3,4]. However, the partitioned approach introduces a new challenge to the problem which is the coupling between the separate solvers. This challenge has two different aspects. The first aspect concerns the methodology, i.e., coupling distinct sets of (discretized) partial differential equations and ensuring the physical equilibrium conditions on the interface. The second aspect is related to the implementation, i.e., coupling two parallel codes with different modules and structures, with the aim of achieving an efficient and scalable overall software.

Modern scientific and engineering problems are often very complex and require a huge computational effort. Therefore, any simulation software must be able to efficiently run on massively parallel computers. The parallel efficiency of a solver is crucial in order to be able to use the available resources adequately and perform a complex calculation. During the recent years, efficient parallel codes have been developed for many single-physics problems, particularly fluid and structure systems. While the monolithic approach to solve FSI problems requires developing a new solver and implementing a software, following a partitioned approach creates the opportunity to exploit the previously-developed efficient codes for FSI simulations. Nevertheless, using efficient single-physics simulation codes does not automatically guarantee achieving a good parallel efficiency for a coupled multi-physics simulation. Multi-code coupling introduces several new challenges. One particular difficulty is the data exchange between separate codes which often use different data structures and could even be written in different

languages. An efficiently parallel mechanism for data exchange between the codes is crucial for achieving parallel efficiency on the coupled framework. Moreover, by coupling two parallel codes, a new level of load balancing is introduced to the problem, as each code would be responsible for a different amount of calculations on a different number of CPUs.

Recent efforts have been made to develop efficient and scalable multi-physics solvers (particularly for FSI problems) using either monolithic [5–7] or partitioned [8–10] approaches. Monolithic FSI solvers are naturally more suitable for massive parallelization, since the whole coupled problem is solved as a single system of equations using a single solver. Therefore, this approach does not face two main computational challenges of the partitioned approach, i.e., parallel communication between separate solvers and the inter-solver load balancing. Nevertheless, achieving a high scalability for monolithic methods is also a very challenging task, as FSI problems are highly complex and nonlinear. In this work, we focus on the partitioned approach in order to create a scalable and efficient FSI solver using existing single–physics solvers. Loss of parallel efficiency in multi-code coupling is considered a main drawback for partitioned methods. This is due to the challenging issues on data structuring, domain decomposition, parallel data communication and inter-solver load balancing. Cajas et al. [9] presented a parallel partitioned solver for FSI problems, based on multi-code coupling. The single-physics solvers were two instances of an in-house code, communicating directly via MPI messages. The inter-code communication was parallel and point-to-point while each solver used a master-worker approach internally. An inter-code load balancing method was proposed based on overloading the available cores in order to minimize the idle time. The coupled framework was shown to scale well for 1280 MPI processes on 768 CPU cores [9]. Hewitt et al. [10] developed a multi-code coupled framework using open source single-physics solvers (OpenFOAM for fluid and ParaFEM for structure). The communication between the solvers was carried out sequentially using a master rank. The solver was shown to scale well on 1,536 cores for a coupled FSI problem. This appears to be the highest scalability reported for a partitioned FSI solver in the literature. Apart from sequential communication, a major problem of the framework in [10] is that it must use the same number of cores for the fluid and the solid solver. This problem arises from the complexity of the two-layered data communication structure (inside each solver and between them), which is essential in a partitioned multi-physics solver. Using the same number of cores for the solvers is a serious drawback, as it effectively rules out any load balancing between the solvers, greatly reducing the computational efficiency of the coupled framework. A computationally efficient partitioned FSI solver that could scale on several thousand processors is still missing in the literature. Towards that goal, we focus on the common issues of parallel partitioned solvers and propose effective solutions to overcome them.

In our multi-code coupled solver, the communication between the separate codes is managed by using a communication library. The communica-

tion library receives the data from each code via an *adapter* and contains functions to facilitate the exchange of data. It also accelerates the coupling iterations between the solvers in the strongly-coupled configuration. Adapters are used to connect the solvers and adjust the data structure while transferring data from one solver, through the communication library, to the other solver. This approach leads to a robust and powerful scheme to couple different codes for multi-physics simulations. An immediate advantage of this method is that one or both of the single-physics solvers could be replaced by other solvers with relatively small changes in the code, in the best case limited to the adapter. Examples of such communication libraries can be found in [11,8,12,13]. Examples of multi-physics simulation software using communication libraries can be found in [8,14,13,15]. There are other examples in the literature where the data exchange is handled directly and the communication functions are included in the single-physics codes themselves. This approach is shown to be efficient when two instances of the same code are coupled (e.g. [9]). However, it is not as robust and powerful as the first approach, especially in cases where two different codes are being coupled. Moreover, it does not allow to exchange one or both solvers. Even though in the current work we use two instances of the same solver for the single–physics sub-problems, we follow the first approach (using a coupling library) to create a robust and flexible framework.

At the coupling methodology level, partitioned methods are generally divided into explicit (or loosely-coupled) and implicit (or strongly-coupled) schemes. Explicit methods solve the fluid and structure equations only once per time step, using data from the previous solution of the partner solver. Therefore, explicit methods do not satisfy the exact equilibrium conditions at the interface, which causes instability issues in many FSI problems. The so-called *added-mass* instability is particularly strong in FSI problems with incompressible flow, a slender interface, and similar densities of fluid and solid [16,17]. Implicit methods, on the other hand, enforce the equilibrium condition at the interface through coupling iterations between fluid and structural solvers. These methods are stable for problems with strong added-mass effect. However, their computational cost is generally high due to the repetitive solution of the governing equations at each time step [3,4]. In the recently introduced semi-implicit coupling approach [18–20], the fluid pressure term is segregated and strongly coupled to the structure, while the remaining fluid terms are only loosely coupled. Strong coupling of the fluid pressure and structural deformation eliminates the added-mass instability issue, while loose coupling of the remaining fluid terms helps avoiding excessive computational cost [18,19].

In this work, two instances of a parallel in-house code, *TermoFluids* [21], are used to solve the fluid and the structure problem. TermoFluids is a robust general-purpose software for fluid and structure problems, using state-of-the-art methods for turbulent flow [22,23], multiphase flow [24,25], and complex thermal systems [26,27], with a high computational efficiency and parallel scalability [28–30]. It presents a conservative discretization of the

governing equations on unstructured grids based on a finite-volume method. It is also equipped with dynamic-mesh schemes to track the moving boundary. The coupling of the codes is carried out using the *preCICE* coupling library [12]. The preCICE library provides communication, data mapping and equation coupling for surface coupled multi-physics applications in a modular manner. It offers a fully parallel point-to-point communication, advanced quasi-Newton iterative coupling schemes and various advanced mapping methods (both consistent and conservative). A semi-implicit FSI coupling approach proposed in [19] is applied, which effectively segregates the fluid pressure term and couples it strongly to the structure. The remaining fluid terms and the geometrical non-linearities are treated explicitly, reducing the computational cost of the numerical solution. Practical test cases in the context of biological flow (flow inside deformable vessels) are studied and the scalability of the overall framework is evaluated.

The remainder of this article is organized as follows. In Section 2, the governing equations for each sub-problem and the coupling conditions are presented. Section 3 describes the proposed numerical methods. Section 4 presents the parallelization method for each single-physics solver, as well as the inter-code communications and load balancing. Numerical tests are presented in Section 5, while Section 6 summarizes and concludes the article.

2 Governing Equations

In this section, we present the fluid and structure governing equations and the coupling conditions on their common interface. The fluid and the structural domain are referred to as $\Omega_f(t) \subset \mathbb{R}^3 \times (0,T)$ and $\Omega_s(t) \subset \mathbb{R}^3 \times (0,T)$, respectively, where $t \in (0,T)$ denotes time. The fluid-structure interface is the common boundary of the domains, denoted by $\Gamma(t) = \partial\Omega_f(t) \cap \partial\Omega_s(t)$. An Arbitrary Lagrangian-Eulerian (ALE) formulation together with a conforming mesh technique is used to solve the fluid flow in a moving domain. A Lagrangian formulation is used for the structural equations.

The unsteady flow of an incompressible viscous fluid is governed by the Navier-Stokes equations. An ALE formulation of these equations in a moving domain is given by

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{c} \cdot \nabla \mathbf{u} = \frac{1}{\rho_f} \nabla \cdot \boldsymbol{\sigma}_f \tag{1}$$

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

where $\mathbf{u}$ is the fluid velocity and ρ_f the fluid density. Vector $\mathbf{c}$ is the ALE convective velocity $\mathbf{c} = \mathbf{u} - \mathbf{w}$, which is the fluid velocity relative to a domain moving with a velocity $\mathbf{w}$. The stress tensor $\boldsymbol{\sigma}_f$ for an incompressible Newtonian fluid is defined as

$$\boldsymbol{\sigma}_f = -p\mathbf{I} + \mu_f(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \tag{3}$$

where p is the fluid pressure, $\mathbf{I}$ the unit tensor and μ_f is the dynamic viscosity of the fluid.

The structural domain is governed by the conservation laws of mass and momentum, whose Lagrangian form is given by

$$\rho_s^0 = \rho_s J \tag{4}$$

$$\frac{\partial}{\partial t}\left(\rho_s \frac{\partial \mathbf{d}}{\partial t}\right) = \nabla \cdot \boldsymbol{\sigma}_s \tag{5}$$

where superscript 0 refers to the reference material (undeformed) configuration of the body, ρ_s is the structural density and $\mathbf{d}$ is the displacement from the reference configuration. The tensor $\boldsymbol{\sigma}_s$ is the Cauchy stress tensor, which can be related to the displacement field by the hyperelastic constitutive model of Saint Venant-Kirchhoff

$$\boldsymbol{\sigma}_s = \frac{\mathbf{B}}{2J}[2\mu_s(\mathbf{B} - \mathbf{I}) + \lambda_s \mathrm{tr}(\mathbf{B} - \mathbf{I})] \tag{6}$$

where $\mathbf{B}$ is the left Cauchy-Green deformation tensor $\mathbf{B} = \mathbf{F} \cdot \mathbf{F}^T$, and μ_s and λ_s are the Lamé's parameters. The material deformation tensor $\mathbf{F}$ is evaluated as $\mathbf{F} = \mathbf{I} + \nabla \mathbf{d}$ and its determinant is denoted by $J = \det(\mathbf{F})$.

The physical equilibrium on the fluid-solid common boundary (kinematic and dynamic equilibrium) constitutes the coupling conditions on the interface. For a non-slip type interface they read

$$\mathbf{u}_\Gamma = \frac{\partial \mathbf{d}_\Gamma}{\partial t} \tag{7}$$

$$\boldsymbol{\sigma}_s \mathbf{n}_\Gamma = \boldsymbol{\sigma}_f \mathbf{n}_\Gamma \tag{8}$$

at Γ, where $\mathbf{n}_\Gamma$ is the unit normal vector on the interface.

3 Numerical Methods

In this section we present the numerical methods for discretization and solution of the single-physics problems, as well as the coupling method. Throughout the development of our coupled framework and the single-physics solvers, an emphasis was made on making the software simple, modular and, as far as possible, matrix-free. Moreover, the fluid solver was mainly developed for turbulent flow simulations. Therefore, many aspects of the discretization and the numerical methods correspond to the particular considerations of turbulent flows.

3.1 Fluid Solver

For fluid flow, a fractional-step projection method along with an explicit time advancement is used to solve the velocity-pressure coupling of the momentum equation. This leads to a three step solution of the fluid governing equations from time step n to $n+1$, with a time increment of Δt

$$\mathbf{u}^p = \mathbf{u}^n - \Delta t[\frac{3}{2}(\mathbf{c}^n \cdot \nabla \mathbf{u}^n - \frac{\mu_f}{\rho_f}\Delta \mathbf{u}^n) - \frac{1}{2}(\mathbf{c}^{n-1} \cdot \nabla \mathbf{u}^{n-1} - \frac{\mu_f}{\rho_f}\Delta \mathbf{u}^{n-1})] \quad (9)$$

$$\frac{\Delta t}{\rho_{\mathrm{f}}}\Delta p^{\mathrm{n+1}} = \nabla \cdot \mathbf{u}^{\mathrm{p}} \quad (10)$$

$$\mathbf{u}^{\mathrm{n+1}} = \mathbf{u}^{\mathrm{p}} - \frac{\Delta t}{\rho_{\mathrm{f}}}\nabla p^{\mathrm{n+1}} \quad (11)$$

in Ω_f^{n+1}, where $\mathbf{u}^p$ is a predicted velocity field which does not satisfy the incompressibility condition (Eq. (2)). This intermediate velocity field is then projected onto a divergence-free field through the correction at Eq. (11). An explicit Adams-Bashforth method is used for the convective and diffusive terms in Eq. (9). Using an explicit method is particularly preferred in turbulent flow simulations where small time steps are indispensable. From a computational point of view, it avoids solving a nonlinear system for the fluid velocity field. It also offers advantages in parallelization as only the information from the previous time step is required (only one episode of data update between nodes is required at each time step).

A finite-volume method is used for the spatial discretization of the fluid equations on a collocated, unstructured mesh with second-order symmetry-preserving schemes. Symmetry-preserving schemes conserve the kinetic energy of the flow at the discrete level which is crucially important in turbulent flow simulations [31,32]. A Jacobi–preconditioned conjugate gradient solver is used to solve the Poisson equation for pressure. More details of the numerical methods for fluid flow equations can be found in [33,32].

We use a conforming mesh technique to track the moving boundary, thus the fluid mesh needs to move in order to adapt to the new location of the interface. The translated mesh and the evaluated domain velocity $\mathbf{w}$ must satisfy the *Geometric Conservation Law* (GCL) [34,35]. The GCL guarantees that no volume is lost while moving the grid, and a constant field is preserved by the ALE scheme. For any control volume (CV) in the fluid domain, the GCL is stated as

$$\frac{\partial v}{\partial t} - \int_s \mathbf{w} \cdot \mathrm{d}\mathbf{A} = 0 \quad (12)$$

where v and s stand for the volume and the boundary surface of the CV, respectively, and $\mathbf{A}$ is the area vector pointing outward.

A parallel moving mesh technique based on the radial basis function interpolation method [36] is used to move the fluid grid in accordance with

the new location of the interface and define the discretized fluid domain at the new time step Ω_f^{n+1}. The method uses the known displacement on the interface to evaluate an interpolated value for the interior vertices of the fluid grid. A great advantage of this method is that it does not need the connectivity of the mesh elements and can be applied to both structured and unstructured grids. Moreover, it only requires solving a linear system of equations whose size is limited to the number of vertices on the fluid-solid interface. A detailed description of the moving mesh method can be found in [36,19].

After the fluid mesh is moved, the domain velocity is evaluated at the surfaces of each control volume. We evaluate the surface velocities based on the GCL law in order to exactly satisfy it. The time rate of change of volume of a CV is equal to the sum of volumes swept by its faces. In this work, we evaluate the domain velocity $\mathbf{w}_{face}$ at each face based on the volume swept by that face. With a second-order backward discretization, it reads

$$\mathbf{w}_{face}^{n+1} = \frac{3}{2}(\frac{\delta v}{A\Delta t}\mathbf{n})^{n+1} - \frac{1}{2}(\frac{\delta v}{A\Delta t}\mathbf{n})^{n} \tag{13}$$

where A is the surface area, $\mathbf{n}$ the unit normal vector of the face, Δt the time step, and δv is the volume swept by the face at one time step. A more detailed description of the evaluation of the domain velocity field and the satisfaction of the geometric conservation law can be found in [20].

3.2 Solid Solver

Aiming at flow-induced deformations and oscillations of the structure, the solid equations are solved using an implicit time integration, where both the inertial and surface forces in Eq. (5) are evaluated at the current time instant t^{n+1}. A cell-centered finite-volume method with a total Lagrangian approach is used for the spatial discretization. The momentum equation is integrated on the undeformed configuration. After applying the Gauss theorem on the stress divergence and relating the undeformed and current area vectors with Nanson's formula, the momentum balance can be written as

$$\int_{v^0} \rho_s^0 \frac{\partial^2 \mathbf{d}}{\partial t^2}\, dv^0 = \int_{s^0} \boldsymbol{\sigma}_s (J\mathbf{F}^{-T}\mathbf{n}^0)\, ds^0 \tag{14}$$

The acceleration of the inertial term is computed according to the trapezoidal rule

$$\frac{\partial^2 \mathbf{d}}{\partial t^2} = \frac{4}{\Delta t^2}\mathbf{d}^{n+1} - \frac{4}{\Delta t^2}\mathbf{d}^{n} - \frac{4}{\Delta t}\left(\frac{\partial \mathbf{d}}{\partial t}\right)^{n} - \left(\frac{\partial^2 \mathbf{d}}{\partial t^2}\right)^{n} \tag{15}$$

The dependencies between different directions of the displacement and the geometrical and material non-linearities found in the right-hand side of Eq. (14) are deferred to the source term of the system. Therefore, outer

fixed-point iterations are needed to obtain a converged solution. This segregated algorithm is typically adopted within finite-volume procedures for structures due to its robustness and low memory requirements [37]. It divides the surface force term into an implicit diffusive component and a deferred correction

$$\int_{v^0} \rho_s^0 \frac{\partial^2 \mathbf{d}}{\partial t^2}\, dv^0 = \int_{s^0} \mathcal{K}_{\text{imp}}(\nabla \mathbf{d})\mathbf{n}^0\, ds^0 + \int_{s^0} \boldsymbol{\sigma}_s(J\mathbf{F}^{-T}\mathbf{n}^0)\, ds^0 - \int_{s^0} \mathcal{K}_{\text{imp}}(\nabla \mathbf{d})\mathbf{n}^0\, ds^0 \qquad (16)$$

The diffusive term is approximated by a central difference scheme with a non-orthogonal correction, presented as the over-relaxed scheme in [38]. The convergence of the iterative process is improved by selecting an optimal value of $\mathcal{K}_{\text{imp}} = (2\mu_s + \lambda_s)$, and using an Aitken Δ^2 acceleration technique for multidimensional problems [39].

Similar to the methods in [40,41], the gradient of the displacement (and hence, the strain and stress tensors) are evaluated directly on the cell face centers. To do so, the gradient is decomposed into the normal and the tangential derivatives and a specific numerical scheme is used for each term. The former uses the same central difference scheme dedicated to the implicit term whereas the latter follows the surface Gauss theorem to express the tangential derivative as a function of the displacement in the face vertices. Therefore, a method to interpolate the displacement from the cell centers to the vertices of the grid has to be defined. The second-order accurate interpolation technique described in [42] is used for this purpose. Additional details on the numerical method for solid mechanics are presented in [43].

3.3 Fluid-Structure Coupling

A Dirichlet-Neumann (DN) domain decomposition method is used to solve the coupled FSI problem. In the DN decomposition, the fluid equations are solved for a known location of the interface and the kinematic equilibrium condition (Eq. (7)) is used as a Dirichlet boundary condition for fluid flow. The structural equations are solved for a known traction on the interface, thus subject to a Neumann boundary condition derived from the dynamic equilibrium condition (Eq. (8)). A great advantage of the DN decomposition method is its consistency with most common solvers for fluid and structure equations.

A semi-implicit coupling method similar to the ones proposed in [19, 20] is followed in this work. The principal idea is to segregate the fluid pressure term and couple it strongly to the structure. It is argued that fluid pressure term is the main contributor to the added mass effect and coupling it loosely will cause numerical instabilities [16,18]. Segregation of the fluid pressure is naturally achieved by using a projection method to solve the

fluid equations. Therefore, only the Poisson equation for pressure in the discretized fluid equations (Eq. (10)) is strongly coupled to the structure. The remaining fluid equations (Eq. (9) and (11)) are solved only once per time step. The geometrical nonlinearities are also treated explicitly and the fluid mesh is only moved once per time step. The location of the interface for each time step is evaluated as an extrapolation of the location in the previous instants. During the coupling iterations, the mesh remains frozen and the coupling boundary condition is applied to the predicted velocity field $\mathbf{u}^p$, instead of the velocity $\mathbf{u}^{n+1}$. When the converged solution is achieved on the interface, the corrected velocity field $\mathbf{u}^{n+1}$ is calculated using the converged pressure solution, and the coupling boundary condition is applied to the final velocity field. This semi-implicit coupling strategy is concisely described in Algorithm 1. For a detailed description of the method see [19, 20]. Therefore, the coupled problem on the interface can be represented as

$$\mathcal{S} \circ \mathcal{P}(\mathbf{d}_\Gamma) = \mathbf{d}_\Gamma \tag{17}$$

where $\mathcal{S}$ represents the structure solver as a function of the traction on the interface $\mathcal{S} = \mathcal{S}(\boldsymbol{\sigma}_\Gamma)$ and $\mathcal{P}$ is the part of the fluid solver that is strongly coupled to the structure (the pressure equation). Both $\mathcal{S}$ and $\mathcal{P}$ are discretized nonlinear operators acting on the coupling interface. Also note that the operators $\mathcal{S}$ and $\mathcal{P}$ as well as the interface deformation field $\mathbf{d}_\Gamma$ are associated to the current time step t^{n+1}.

Algorithm 1 One time step in the semi-implicit FSI coupling method

start $t = t^{n+1}$

1: Predict the location of the interface by extrapolating from previous time steps.
2: Move the mesh and evaluate the surface velocities.
3: Solve the fluid ALE convective-diffusive equation for the predicted velocity field (Eq. (9)).

4: **while** coupling not converged **do** ▷ $\mathcal{S} \circ \mathcal{P}(\mathbf{d}_\Gamma) = \mathbf{d}_\Gamma$
5: Solve the Poisson's equation for pressure (Eq. (10)).
6: Solve the structure equations for the deformation.
7: Update boundary values for the predicted velocity using the new solid deformation.
8: **end while**

9: Evaluate the corrected velocity field using the converged pressure field (Eq. (11)).
10: Apply the boundary condition on the corrected velocity using the converged deformation.

The coupled interface problem in Eq. (17) needs to be solved at each time step similar to the interface problem in a fully implicit partitioned method. Fixed-point iterations with Aitken's relaxation (e.g. [3, 44, 9]) or

Newton-based methods (e.g. [45–47]) are commonly used to solve the interface problem. The execution of the fluid and the structure solver can be carried out sequentially (staggered solution) or simultaneously (parallel solution). In the staggered method, the fluid and the structure system are solved sequentially (one after another), corresponding to a Gauss-Seidel-type problem. On the other hand, in simultaneous or parallel solution, the systems are solved at the same time (using different processors), corresponding to a Jacobi-type approach. Staggered methods are commonly used, although they have a very limited parallel efficiency due to a severe load imbalance [48,49]. Although fixed-point iterations with Aitken's relaxation are seen to be efficient and robust in a staggered solution (see e.g. [44]), they show a poor performance in a parallel (simultaneous) solution method. On the other hand, parallel execution together with a quasi-Newton method results in a very efficient and robust method [48,49].

In this work, we take a parallel solution approach where the fluid and the structure systems are solved at the same time (on different processors) using inputs from the previous iteration of the partner solver. One iteration of this Jacobi-like fixed-point problem can be written in matrix-like notation as:

$$\begin{pmatrix} 0 & \mathcal{S} \\ \mathcal{P} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{d}_\Gamma^k \\ \boldsymbol{\sigma}_\Gamma^k \end{pmatrix} = \begin{pmatrix} \widetilde{\mathbf{d}}_\Gamma^{k+1} \\ \widetilde{\boldsymbol{\sigma}}_\Gamma^{k+1} \end{pmatrix} \tag{18}$$

where k indicates the iteration count in the current time step $t^n \rightarrow t^{n+1}$, and the tilde sign in $\widetilde{\mathbf{d}}_\Gamma^{k+1}$ and $\widetilde{\boldsymbol{\sigma}}_\Gamma^{k+1}$ means these new values are yet to be modified in a subsequent quasi-Newton step. In order to describe the quasi-Newton method used in this work, we write the vector form of the underlying fixed-point interface problem (Eq. (18)) shortly as

$$\mathcal{H}(\mathbf{x}) = \mathbf{x} \tag{19}$$

where $\mathbf{x} = \begin{pmatrix} \mathbf{d}_\Gamma \\ \boldsymbol{\sigma}_\Gamma \end{pmatrix}$. To solve Eq. (19), we accelerate the fixed-point iteration (Eq. (18)) by a subsequent Newton step

$$\mathbf{x}^{k+1} = \mathcal{H}(\mathbf{x}^k) - \mathcal{J}^{-1}\mathbf{R}(\mathbf{x}^k) \tag{20}$$

where k denotes the iteration count, $\mathbf{R}$ is the residual function $\mathbf{R}(\mathbf{x}^k) = \mathcal{H}(\mathbf{x}^k) - \mathbf{x}^k$, and $\mathcal{J}^{-1}$ is the inverse Jacobian of $\widetilde{\mathbf{R}}(\mathbf{x}^k) = \mathbf{x}^k - \mathcal{H}^{-1}(\mathbf{x}^k)$. Since the calculation of the inverse Jacobian is not feasible, it is approximated based on the secant equation

$$\hat{\mathcal{J}}_k^{-1}\mathcal{V}_k = \mathcal{W}_k \tag{21}$$

in which the hat sign indicates the approximation and $\mathcal{W}_k$ and $\mathcal{V}_k$ are two matrices which include increments of $\widetilde{\mathbf{x}} = \mathcal{H}(\mathbf{x})$ and the residual $\mathbf{R}$ in the previous iterations

$$\mathcal{W}_k = [\Delta\widetilde{\mathbf{x}}^k, \Delta\widetilde{\mathbf{x}}^{k-1}, ..., \Delta\widetilde{\mathbf{x}}^1] \tag{22}$$

$$\mathcal{V}_k = [\Delta\mathbf{R}^k, \Delta\mathbf{R}^{k-1}, ..., \Delta\mathbf{R}^1] \tag{23}$$

with $\Delta\widetilde{\mathbf{x}}^k = \mathcal{H}(\mathbf{x}^k) - \mathcal{H}(\mathbf{x}^{k-1})$ and $\Delta\mathbf{R}^k = \mathbf{R}^k - \mathbf{R}^{k-1}$. In this work, we use a multi-vector quasi-Newton method (MVJ) as proposed in [12,47]. In this approach, instead of minimizing the Forbenius norm of the Jacobian, the distance between the current and the previous time step Jacobian ($\hat{\mathcal{J}}_k^{-1}$ and $\hat{\mathcal{J}}^{-1,(n)}$, respectively) is minimized ($min(||\hat{\mathcal{J}}_k^{-1} - \hat{\mathcal{J}}^{-1,(n)}||)$). This results in the following approximation for the Jacobian inverse:

$$\hat{\mathcal{J}}_k^{-1} = \hat{\mathcal{J}}^{-1,(n)} + (\mathcal{W}_k - \hat{\mathcal{J}}^{-1,(n)}\mathcal{V}_k)(\mathcal{V}_k^T\mathcal{V}_k)^{-1}\mathcal{V}_k^T \tag{24}$$

where k indicates the iteration number at the current time step, while n refers to the previous time step t^n. It should be noted that we cannot use all the available data from previous iterations, since some of them might be linearly dependent which may result in an ill-conditioned problem. Therefore, linearly dependent data must be removed by using a proper filter. In this work we use a $QR2$ filter, which is described in detail in [47].

The estimated Jacobian is used to find the increment of $\mathbf{x}$, according to Eq. (20). This process must be repeated until the convergence criterion is met and we can move to the next time step. The convergence criterion is defined as

$$\frac{||\mathbf{R}^k||}{||\mathbf{R}^0||} < \epsilon_{FSI} \tag{25}$$

with a prescribed small value for ϵ_{FSI}. For the first iteration in the first time step, as there is no previous data available to use the quasi–Newton method, we use a fixed-point iteration with an under-relaxation factor of 0.1. In addition, at the beginning of each time step, a second-order extrapolation is used to create a better initial guess which helps to decrease the number of required iterations. This quasi–Newton method is part of the preCICE coupling library and can be consulted in details in [12].

4 Parallelization

The current framework is based on multi-code coupling of separate solvers for the fluid and the structure domain. Therefore, the parallelization aspects can be considered in two levels, namely the parallelization of each single-physics solver, and the inter-code communication level. Moreover, in this section we explain the load balancing methods applied to each single-physics solver, as well as the load balancing across the codes.

4.1 Single-Physics Solvers

Two instances of the same software are used for the single-physics subproblem domains in this work. Therefore, the fluid and the structure solver share the same parallelization method and the same programming model which are described in this section.

The overall computational efficiency of any code depends on both its single-core and its parallel performance. Computational fluid dynamics and solid mechanics are memory-bound applications, meaning that the single-core performance of the solver is limited by the cost of fetching data from the memory rather than the cost of computations on the CPU. This is a characteristic feature of these applications since their discretized systems of equations are represented by sparse matrices and the computations have low arithmetic intensity. Therefore, the single–core performance of the code in this work is optimized by minimizing memory transfers and using SIMD operations whenever possible, depending on the operation and the type of data.

For parallelization of the computations at each single-physics solver, a distributed-memory model is used and communication between processes is established using the Message Passing Interface (MPI) standard. A hybrid MPI–openMP arrangement is not followed in this work for the sake of simplicity and portability. Using the current configuration makes our code portable across all the distributed-memory systems, enabling us to efficiently use different clusters. As a future work, the authors are interested in extending the implementation of the framework to a hybrid MPI–openMP paradigm and study its effect on the computational efficiency. The code also has a multi-threading capability with CUDA or openCL to use GPUs as co–processors on a hybrid machine [50–52]. This configuration was not used in this work and we focused on CPU-only clusters. Extending the current framework to exploit hybrid CPU-GPU machines is a topic for future studies.

The distributed-memory parallelization is carried out based on a spatial domain decomposition. The computational grid Ω (either Ω_f or Ω_s) is divided into n non-overlapping subdomain blocks, $\Omega_0, ..., \Omega_{n-1}$, and each block is assigned to a different CPU core (MPI rank). Each block contains many cells (control volumes) that are *owned* by the assigned core, and several *halo* cells that represent the neighbours of the owned cells that are themselves owned by a different process. Neighbour cells are defined based on the notion of shared vertices. Figure 1 schematically shows a discretized domain and its decomposition into two subdomain blocks. The decomposition of the computational domain is carried out using the METIS library [53]. METIS divides the computational mesh into roughly equal partitions using a parallel multilevel *k-way* graph-partitioning method, minimizing the number of halo cells [53]. Roughly equal sizes of the blocks results in a roughly balanced load on different processes and a minimized number of halo cells means minimized required communications.

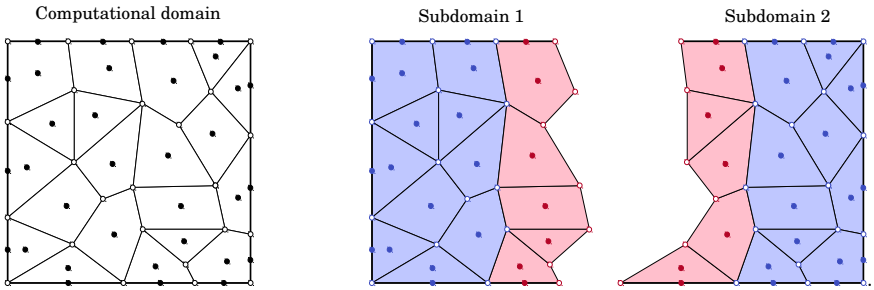

Fig. 1 Spatial domain decomposition for a single-physics solver: schematic illustration of a discretized domain (left) and its decomposition into two subdomain blocks (right). Cell and boundary nodes are represented as filled circles and vertices as empty circles. The owned elements (cells, nodes and vertices) of each process are shown in blue while the halo elements are shown in red.

At the parallel level, the efficiency is mainly limited by the inter-process communication. Two types of inter-process communication are present in our implementation:

(i) global reduction operations which are used in the evaluation of norms, dot products, and for obtaining global extrema (e.g., in time step evaluation),
(ii) point-to-point communication between the processes in order to update data in the halo cells.

The first type of communication is not required very frequently in the code and is carried out by simply calling the corresponding `MPI` collective operations. Communication for data updates in the halo cells is carried out via non–blocking functions `MPI_Isend` and `MPI_Irecv` in order to avoid unnecessary synchronization. The synchronization is deferred to a later time when the `MPI_Waitall` function is called. An efficient and sparse scheme is implemented where each process stores the list of other processes with which it needs to communicate. This is carried out only once at the initialization stage. In the communication episode, a loop is created over this small list to invoke the respective communication.

Another relevant aspect in computational performance of the code is checkpointing and input and output operations (IO). Checkpoints are saved instants of data which are used for post-processing or to restart the simulations from a specific point. The IO operations in this work are managed via the HDF5 library [54]. Our implementation of the IO operations for checkpointing is seen in our previous tests to have a good parallel performance and to generate a reasonably low overhead [30].

4.2 Inter-Code Communication

Efficient inter–code communication is a key element in parallel partitioned coupled simulations. For this purpose, we use the preCICE coupling li-

brary [12]. Inter–code communication via preCICE can be either based on MPI ports (MPI-2.0) or on lower level TCP/IP sockets. In the current work we have used TCP/IP sockets, because the MPI ports functionality is missing in the implemented MPI versions on the supercomputer that we used for our tests (SuperMUC supercomputer at the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences). To establish communication channels between the participants' ranks, the fluid and the solid mesh partitions are initially analyzed to find logical connections between them. Once the communication channels are established, data are exchanged in an asynchronous way to avoid unnecessary blocking. Establishing the communication channels is carried out only once at the initialization stage. For the rest of the run-time, the same channels are used for data exchange. Since using a central communication instance can degrade the scalability of our framework, we use a fully parallel point-to-point communication scheme, thus no central server–like unit is used. This way, data exchange happens locally between the connected ranks. It is obvious that, in case of having a big mesh or a high number of ranks and using a central communication instance, a single master rank or server would be a severe bottleneck.

Figure 2 schematically shows the parallel structure of the coupled framework and the different levels of communication.

4.3 Load Balancing

Load imbalance can be a serious source of inefficiency in parallel simulation codes. Inside each single–physics solver, the load is balanced by dividing the computational domain into fairly equal blocks for each process, as explained in Section 4.1. However, in a partitioned FSI simulation, there exists a new level of load balancing between the single–physics solvers. In this work with a simultaneous execution of the solvers (Jacobi-type problem described in Section 3.3), both solvers must finish an iteration and send the output to the partner solver before the next iteration can start. This means, in case the available CPUs are not distributed optimally among the solvers, one solver will be waiting for the partner to finish its own computations. In contrast to the load balancing within a single–physics solver, load balancing across the codes is more difficult since we do not know a priori the relation between the work per cell of different solvers.

In this work, we follow the approach proposed in [55] to address this issue. We first model the solver performance against the number of ranks for each domain, and then solve an integer optimization problem to find the appropriate pair of rank numbers for the domains that minimizes the waiting time. Since analytical modeling of the solvers' performance is very complex, we use an empirical regression-based approach instead, aiming to find an appropriate performance model.

We run m simulations with different number of cores, yielding a set of data points consisting of pairs (n, f_n) mapping the number of ranks n to the

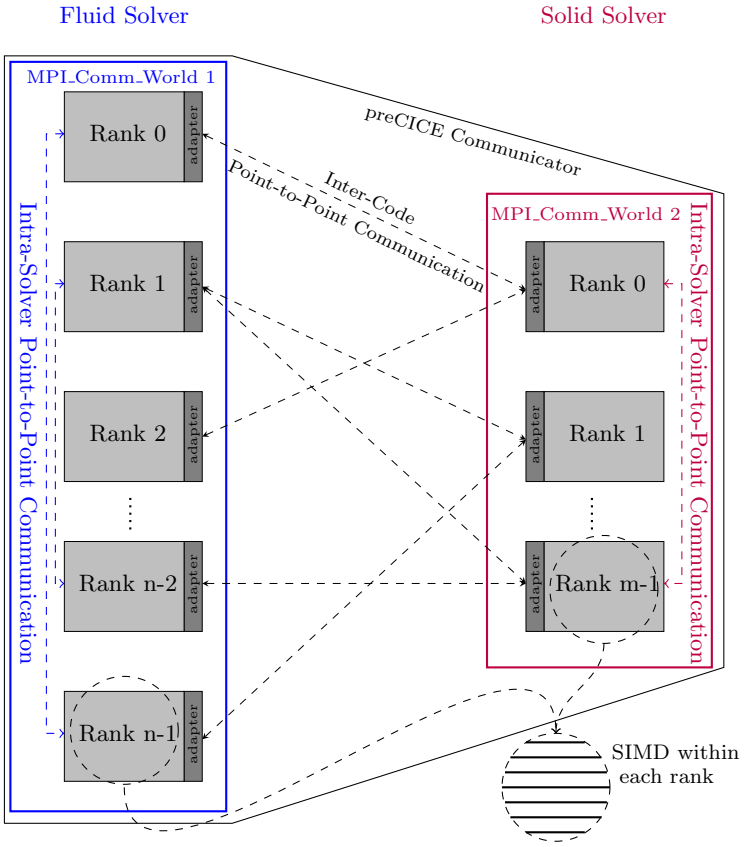

Fig. 2 Parallelization model of the coupled framework and the different levels of communication.

run-time f_n (in this work, m=5 simulations, each for only 3 time steps). In order to find a function $f(n)$ for each solver, which predicts the run-time for any n, we use the Performance Model Normal Form (PMNF) [56], defined in (Eq. (26)), as a basis for our prediction model:

$$f(n) = \sum_{k=1}^{q} c_k n^{i_k} \log_2^{j_k}(n), \tag{26}$$

where n is the number of ranks used by the solver, q is the number of terms used for the empirical run-time approximation, c_k is the weight of each term in the approximation function, and i_k and j_k are empirical coefficients.

Calotoiu et al. [56] suggest, that the search space given by $q = 2$, $i_k \in \{\pm\frac{0}{4}, \pm\frac{1}{4}, \ldots, \pm\frac{12}{4}\}$ and $j_k \in \{0, \pm 1, \pm 2\}$ is suitable for many applications. To find the optimal model, we simply check all the combinations within the search space, calculate a cross-validation-based loss for each combination (the accumulative error on the validation data set) and pick the combination with the smallest loss. By applying a PMNF regression, we are able to generate performance models $f(n)$ for each solver involved in the simulation.

As mentioned before, our goal is to find an optimal assignment of cores for each solver for a limited total number of available cores P, such that the overall run-time $F(n_f, n_s)$ is minimized (n_f and n_s are number of ranks used by the fluid and the solid solver respectively). This can be expressed by the following optimization problem:

$$\begin{aligned} &\underset{n_f, n_s}{\text{minimize}} \quad F(n_f, n_s) \qquad \text{with } F(n_f, n_s) = \max(f_s(n_s), f_f(n_f)) \\ &\text{subject to} \quad n_f + n_s \leq P. \end{aligned} \tag{27}$$

If the functions f_s and f_f are approximated by the PMNF regression, this optimization problem is a nonlinear, possibly non-convex integer optimization. We assume that f_s and f_f are both monotonically decreasing, i.e., assigning more cores to a solver never increases the run-time. With this, we can simplify the constraint to

$$P = n_f + n_s. \tag{28}$$

The optimization problem can then be solved by checking all possible values for n_s and n_f in order to choose the pair that minimizes the total run-time. This pair of rank counts is then used to divide the available processors between the fluid and the structural solver for the FSI simulation. For more details, please refer to [55].

5 Numerical Tests

Numerical tests are provided in this section to demonstrate the parallel scalability of the coupled framework in solving practically relevant FSI problems. The test cases are in the context of hemodynamics. The first test case is a benchmark problem representing the propagation of pressure waves inside a 3D deformable tube. The second test case is the blood flow inside a patient-specific aorta considering the elastic deformation of the aorta wall. The scalability tests are carried out on the SuperMUC supercomputer at the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities in Garching [57]. SuperMUC consists of 2.6GHz Intel Xeon E5-2697-v3 (Haswell) processors. Each computing node contains two processors with 14 cores per processor (28 cores per node) and 64GB of RAM. The nodes are connected via Mellanox Infiniband FDR14 interconnect. The GNU GCC compiler was used to compile both the TermoFluids solver and

the preCICE library. In addition, an Intel MPI implementation compatible with the GCC compiler was used for intra-solver parallelization.

5.1 Test case 1: 3D Flow Inside a Deformable Tube

This benchmark problem was proposed by [58] and studied, among others, by [59,44,60,61]. The problem is a 3D incompressible flow inside a straight tube with a deformable wall, motivated by the type of problems encountered in hemodynamics. The tube has a length of $l = 0.05$m, an inner radius of $R_0 = 0.005$m, and a wall thickness of $h = 0.001$m. The fluid density and viscosity are $\rho_f = 1000\text{kg/m}^3$ and $\mu_f = 0.003\text{Pa} \cdot \text{s}$, respectively. The structural density is $\rho_s = 1200\text{kg/m}^3$, the Young modulus $E = 3\times10^5\text{N/m}^2$, and the Poisson ratio $\nu = 0.3$.

The tube is clamped at both ends and the fluid is initially at rest. An overpressure of 1333.2Pa is applied at the tube inlet during a period of 0.003s and a constant pressure of 0Pa afterwards. The pressure at the outlet is 0Pa during the whole simulation. A Neumann boundary condition is used for the fluid velocity at both the inlet and the outlet boundaries. The outer surface of the tube wall is treated as a traction-free boundary.

An unstructured tetrahedral mesh is used for the fluid domain. The solid domain mesh is constructed by extruding the fluid mesh on the outer boundary (structured mesh). Three different mesh resolutions are used for the numerical tests. The resolution of the mesh for the fluid and the solid domain is provided in Table 1. Two meshes (M1 and M2 in Table 1) are used to solve the problem from $t = 0$ until $t = 0.02$s with constant time step sizes of $\Delta t = 10^{-4}$ and $\Delta t = 5 \times 10^{-5}$s, respectively (which means doubling the resolution in each spatial direction and time). Figure 3 presents the radial displacement at the mid-length of the tube during the simulation time. As seen in the figure, the results obtained by mesh M1 and M2 (and their corresponding time step) are fairly close. Small differences are visible near the peaks and the troughs. Results with the coarser mesh and the larger time step appear to be more damped, which is expected due to a larger numerical dissipation. Figure 3 also contains numerical results from Eken et al. [60] and Lozovskiy et al. [61]. Although the four sets of results follow the same trend, the results from [60,61] appear to be more damped, compared to the present solution. This could be due to the extra numerical dissipation in [60,61], as both works use a first-order Euler discretization in time with a time step $\Delta t = 10^{-4}$s (compared to a second-order time discretization in the current work). Figure 4 shows the velocity vectors inside the deformed tube at two instants $t = 0.005$ and $t = 0.01$s. The color contour in the solid domain is the von Mises equivalent stress. The deformation of the wall is magnified by a factor of ten to be better visible. The figure shows the propagation of the pressure wave with a finite velocity inside the tube.

For scalability tests, we have used a much finer grid (mesh M3 in Table 1). A complete simulation is not carried out for these tests, only the first

Table 1 Computational grids used for the deformable tube test case (test case 1).

Mesh name	No. of cells	
	Fluid	Structure
M1	15K	10K
M2	120K	80K
M3	9M	6M

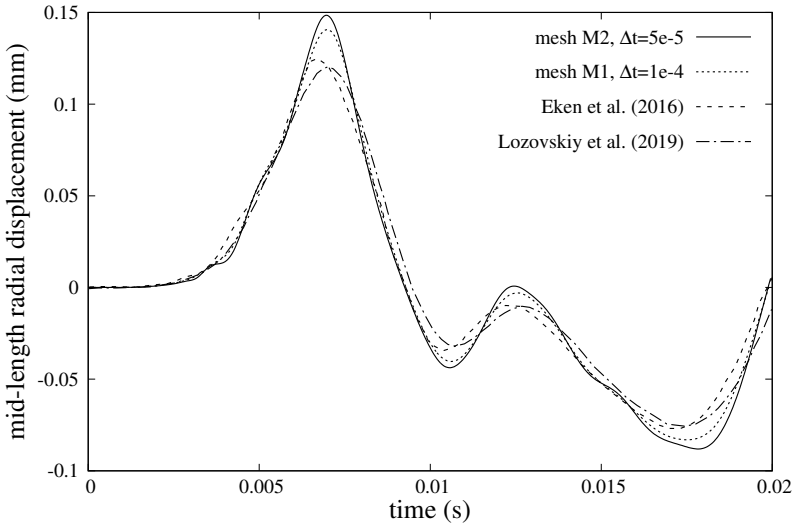

Fig. 3 Radial displacement at the mid-length of the tube (test case 1) for two different mesh resolutions and time step sizes, compared to numerical results from [60, 61].

ten time steps are solved. The solution results at the entire field (for both fluid and solid domains) are written into *h5* output files at every time step. Figure 5 shows the average run–time per time step for different numbers of CPU cores using mesh M3. This run–time includes the time for writing output data files, but does not include the initialization time of the solvers. The ideal (linear) reduction of the run-time by increasing the core count is also shown for comparison. The number of cores in the horizontal axis of the graph indicates the total sum of cores used for the fluid and the solid solver. The distribution of the available cores between the solvers is based on the load balancing model described in Section 4.3, and it is provided in Table 2. As seen in the figure, a very good reduction in the computational time is achieved by increasing the number of cores. Results show a very good scalability for up to 1400 cores, with an almost ideal speed-up for up to 1120 cores. Figure 6 shows the parallel efficiency at each point of the graph. The parallel efficiency for the strong scalability on m cores is evaluated as

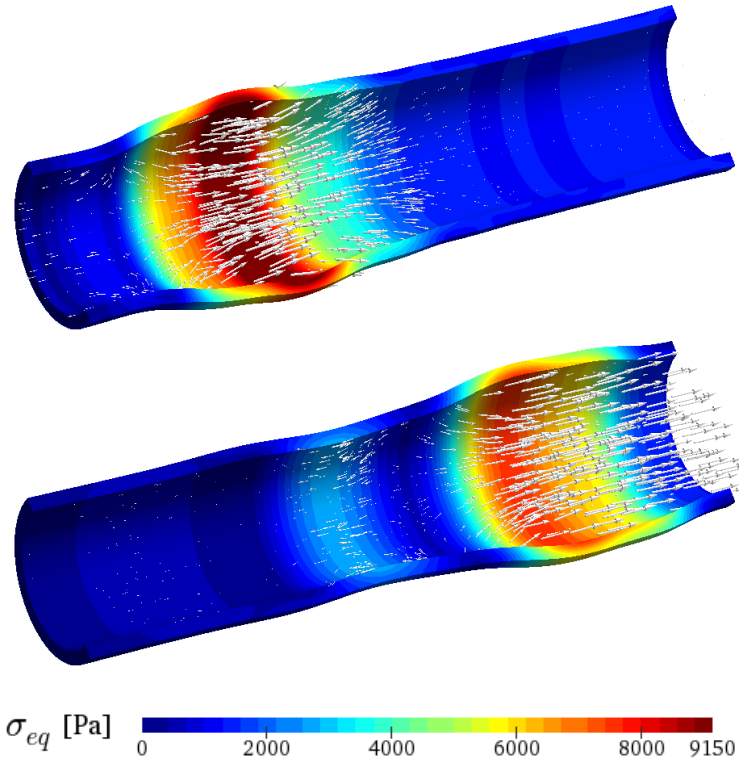

Fig. 4 Propagation of the pressure wave inside the deformable tube (test case 1). Shown in the figure are fluid velocity vectors inside the deformed domain and the von Mises equivalent stress at the wall. Top: $t = 0.005$s; Bottom: $t = 0.01$s. Deformations are magnified by a factor of ten.

$$\text{efficiency} = \frac{f_l \times l}{f_m \times m} \tag{29}$$

where f_m is the run-time on m cores, and f_l is the run-time on the smallest number of cores, indicated by l. If possible, the parallel efficiency is measured against the sequential run-time ($l = 1$). However, due to the limitation of memory on a single core, large problems cannot be solved sequentially. Therefore, the smallest number of cores which can be used to solve the problem is used as the basis to evaluate the efficiency ($l = 280$ for this test case). As seen in the figure, the parallel efficiency is 96% for 1120 cores which indicates an almost ideal parallel scalability. For 1400 cores the efficiency is still high, at 82%. For higher core numbers the parallel efficiency degrades. This limit appears to be mostly determined by the size of the mesh used for the tests. The size of the computational grid (total of 15M cells for mesh M3) is relatively small to be divided among over a thousand processes. Dividing mesh M3 among 1400 cores means each process owns roughly 10K

cells, which is too few calculations and corresponds to a very low arithmetic intensity (compared to communication). By repeating the scalability tests using finer grids, a greater scalability can be achieved on higher numbers of cores. Larger computational grids are used for the next test case. Moreover, the output data writing could limit the scalability as it essentially contains sequential steps. We have included the output result writing time in the scalability tests for the current test case, because it is a necessary part of any practical simulation.

Table 2 Distribution of the CPU cores between the fluid and the solid solver based on the inter-code load balancing model, for test case 1 (deformable tube) and mesh M3.

Total No. of cores	280	420	560	700	840	980	1120	1260	1400
Fluid solver	153	230	301	363	417	461	496	524	545
Solid solver	127	190	259	337	423	519	624	736	855

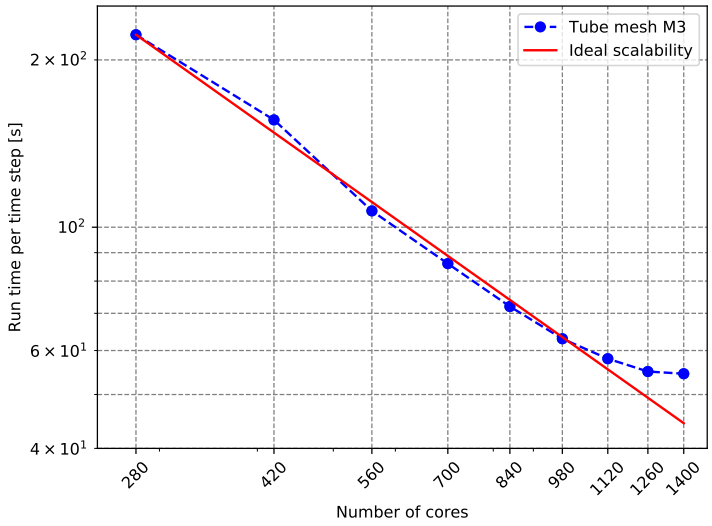

Fig. 5 Strong scalability test results for test case 1 (deformable tube) and mesh M3: average run–time per time step for different numbers of cores.

At every time step, an average of roughly 17 coupling iterations were required to achieve convergence on the coupled problem (using $\epsilon_{FSI} = 10^{-5}$). This number remained fairly constant by increasing the number of processes, as seen in Table 3. This shows that the overall methodology for the coupled FSI problem is mathematically scalable. Table 3 also contains the data for the initialization time of the coupled framework. The reported time

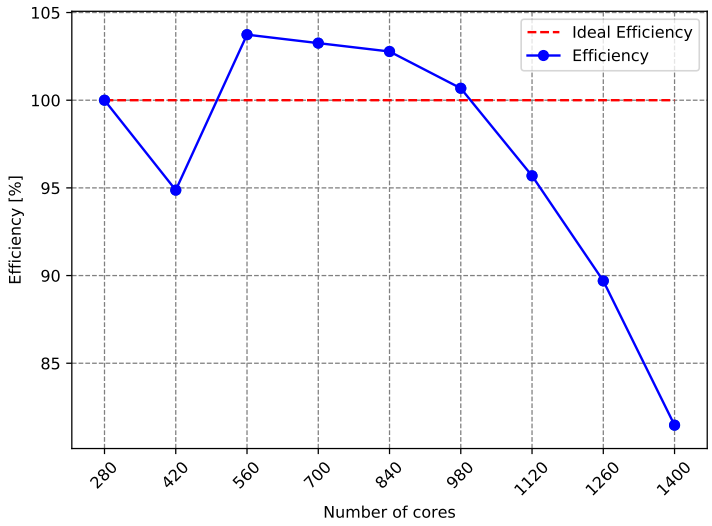

Fig. 6 Strong scalability test results for test case 1 (deformable tube) and mesh M3: parallel efficiency at different core counts.

includes the initialization times of the single–physics solvers, as well as the time for establishing the inter–code communication channels and initializing the coupling library modules. It does not include the pre-processing steps such as mesh generation or partitioning using the METIS library. The data in Table 3 show that the initialization time remains fairly constant and only slightly increases when increasing the number of processes.

Table 3 Strong scalability of test case 1 (deformable tube) and mesh M3. Results in the table are the average number of coupling iterations per time step, and the total initialization time of the coupled framework.

No. of cores	280	420	560	700	840	980	1120	1260	1400
coupling iterations (average)	17.4	17.6	16.8	17.4	16.8	16.8	17.2	17.4	17.2
initialization time [s]	329.6	335.8	339.2	345.6	351.7	353.9	359.2	365.4	370.6

5.2 Test case 2: Patient–Specific Aorta

The second test case is a simulation of blood flow inside a patient-specific aorta with a mild thoracic aortic coarctation. The 3D geometry of the aorta is obtained by contrast agent magnetic resonance angiography, provided by the 2nd CFD challenge of the STACOM 2013 conference [62]. Figure 7 shows the provided geometry and the location of the boundaries. The thickness

of the aorta wall and its mechanical properties were not provided in the challenge data. Therefore, we assume values in the typical physiological range. A uniform thickness of $h = 2\text{mm}$ is assumed for the wall along with density $\rho_s = 1200\text{kg/m}^3$, Young modulus $E = 3 \times 10^5\text{N/m}^2$, and Poisson ratio $\nu = 0.3$. The density and the viscosity of the blood are considered to be $\rho_f = 1000\text{kg/m}^3$ and $\mu_f = 0.004\text{Pa} \cdot \text{s}$. These values for the wall thickness and the properties of the wall and blood are similar to the values used in [63].

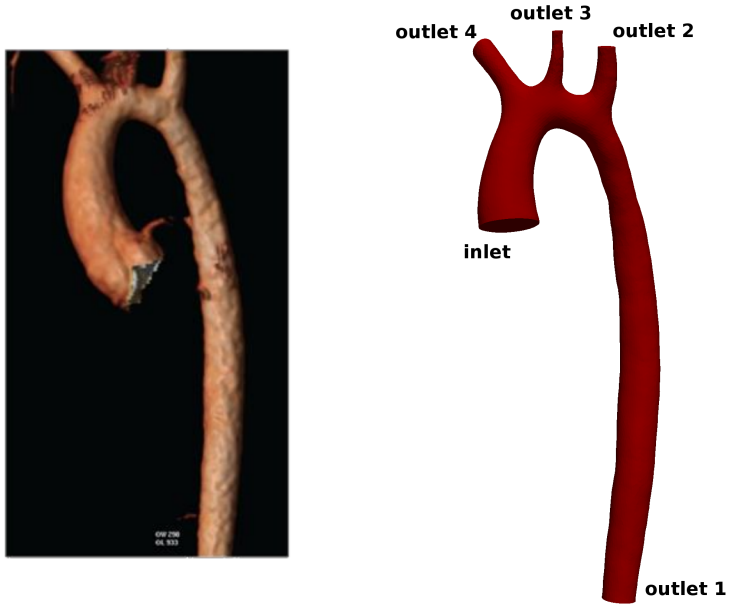

Fig. 7 Test case 2, 3D geometry of a patient-specific aorta provided in [62], and the location of the boundaries.

For the inlet boundary of the aorta, a Dirichlet boundary condition is used for the velocity, using measured physiological flow rate data provided in [62] for the rest condition. A Neumann boundary condition is used for the fluid pressure at the inlet. For the outlet boundaries, explicit RCR Windkessel boundary conditions [64] are used to model the effect of the rest of the vascular network. The Windkessel parameters are chosen as those reported in [65]. For the solid, a zero–displacement (clamped) boundary condition is set at the inlet and at the outlets, while a traction-free boundary condition is used on the outer surface of the wall.

The objective of the current work is not a deep study of the biophysical phenomena featured in this test case, but rather to show the capability of the developed framework to solve such a complex problem on a massively parallel configuration. Nevertheless, the obtained fluid velocity and pres-

sure fields, as well as the structural displacements were seen to be in the reasonable physiological range. Figure 8 shows the solution at two instants, $t = 0.05$s and $t = 0.1$s. The figure contains the velocity vector plot inside the deformed aortic wall. The color contours in the structural domain correspond to the von Mises equivalent stress.

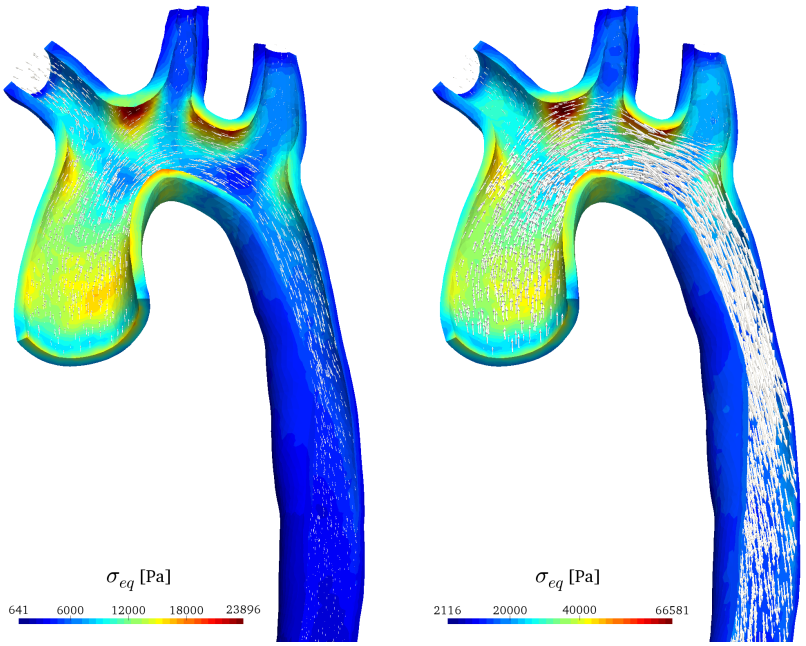

Fig. 8 Test case 2, fluid velocity vectors inside the deformed aortic wall and the von Mises equivalent stress at the wall. Left: at $t = 0.05$s; Right: at $t = 0.1$s.

The strong scalability of the developed framework is evaluated using the computational grids M1 and M2 (Table 4). Both fluid and solid grids are unstructured tetrahedral meshes. The first ten time steps are solved for the scalability tests. Unlike the previous test case, the output result writing step is disabled, thus the run-times only include the solution time of the coupled problem. Figure 9 shows the average run–time per time step for different numbers of cores for mesh M1, with and without inter-code load balancing. Similar to the previous test case, the number of cores indicates the total number of processes of the fluid and the solid solver together. For the case without load balancing, the available cores are divided between the solvers proportional to their respective mesh size. For the load balancing case, the division of the cores is based on the load balancing model in Section 4.3, and it is provided in Table 5. As seen in Figure 9, a very good reduction in computational time is achieved by increasing the number of cores, up

to 3920 cores. Moreover, applying our load balancing model reduces the run–time by an average of 15%, almost uniformly for different core numbers. The inter-code load balancing does not seem to affect the scalability limit of the overall framework. To evaluate the overall effect of the load balancing method on the computational efficiency, we must also consider the associated overhead. The main part of the load balancing overhead comes from the small simulations that are carried out to establish the performance model functions. For the current test case, we used five small simulations on mesh M1 with an accumulated total run-time of 3100 seconds. After the performance models are established, the extra cost to evaluate the core distribution for each simulation is insignificant (1-2 seconds to solve the optimization equation Eq. (27)). Therefore, this overhead is only paid once, and not repeated for each simulation. Moreover, we have used the same performance models for the case with a larger mesh (M2 in Table 4). Therefore, no extra overhead was paid for the large mesh case. This overhead for load balancing is relatively small, compared to the consequent reduction in the computational cost. For instance, the reduction in the run-time due to the load balancing method on 560 cores is 60 seconds per time step (Figure 9), which means the reduction for the first 52 time steps compensates the overhead of the load balancing method. Considering that real-world FSI simulations would normally be performed over thousands of time steps (depending on the simulation length and the time step size), we believe the overhead is well justified. The run-time reduction for larger meshes would be more significant. In practice, a 15% reduction of the run-time of a large scale FSI simulation would be a considerable amount of CPU-hours.

Figure 10 presents the corresponding parallel efficiency for the case with load balancing. Results in Figure 9 and 10 show a very good scalability for up to 3920 cores, with a high efficiency of 79%. For higher core numbers the parallel efficiency degrades. Similar to the previous test case, this limit appears to be mostly determined by the size of the mesh used for the scalability tests. The size of the computational grid (total of 29M cells in mesh M1) is not large enough to be divided among four thousand processes. To confirm this and to demonstrate the scalability of the framework on higher counts of CPUs, the scalability test is repeated using a larger mesh (M2 in Table 4).

Table 4 Computational grids used for the patient-specific aorta test case (test case 2).

Mesh name	No. of cells		No. of faces on the interface
	Fluid	Structure	
M1	20M	9M	400K
M2	95M	41M	1.5M

Figure 11 shows the average run–time per time step for mesh M2 and different numbers of cores. The number of cores indicate the total available

Table 5 Distribution of the CPU cores between the fluid and the solid solver based on the inter-code load balancing model, for test case 2 (patient–specific aorta) and mesh M1.

Total No. of cores	560	1120	1680	2240	2800	3360	3920
Fluid solver	366	730	1038	1346	1570	1766	1934
Solid solver	194	390	642	894	1230	1594	1986

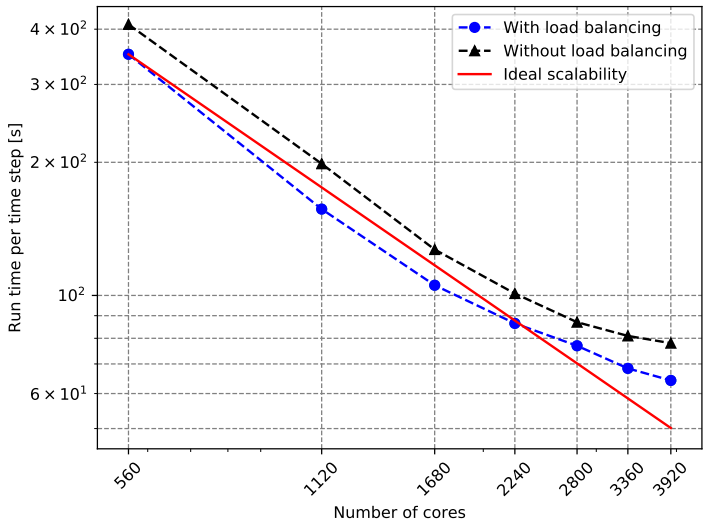

Fig. 9 Strong scalability test results for test case 2 (patient–specific aorta) and mesh M1: average run–time per time step for different core counts, with and without inter-code load balancing.

cores which are divided between the two solvers based on the load balancing model in Section 4.3, provided in Table 6. Again, a very good reduction in computational time is seen by increasing the number of cores. Figure 12 presents the corresponding parallel efficiency. Results in Figure 11 and 12 show a very good scalability for up to 10,080 cores with an efficiency of 83%.

Table 6 Distribution of the CPU cores between the fluid and the solid solver based on the inter-code load balancing model, for test case 2 (patient–specific aorta) and mesh M2.

Total No. of cores	2000	3360	4480	5600	6720	7840	8960	10080
Fluid solver	1280	2214	2970	3698	4286	4762	5098	5266
Solid solver	720	1146	1510	1902	2434	3078	3862	4814

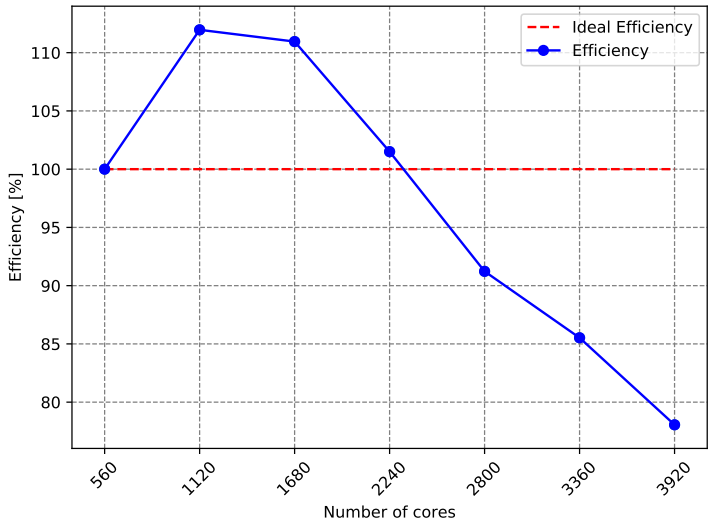

Fig. 10 Strong scalability test results for test case 2 (patient–specific aorta) and mesh M1: parallel efficiency at different core counts, with inter-code load balancing.

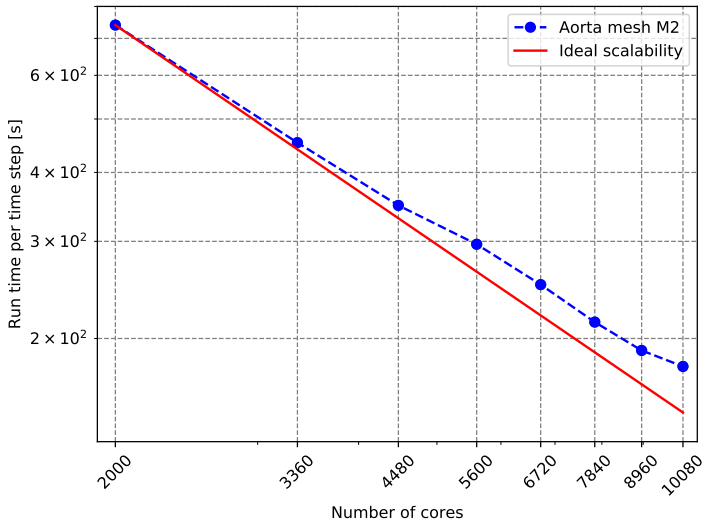

Fig. 11 Strong scalability test results for test case 2 (patient–specific aorta) and mesh M2: average run–time per time step for different core counts.

At every time step, an average of roughly 10 coupling iterations were required to achieve convergence of the coupled problem (using $\epsilon_{FSI} = 10^{-3}$). This number remained constant when increasing the number of processes,

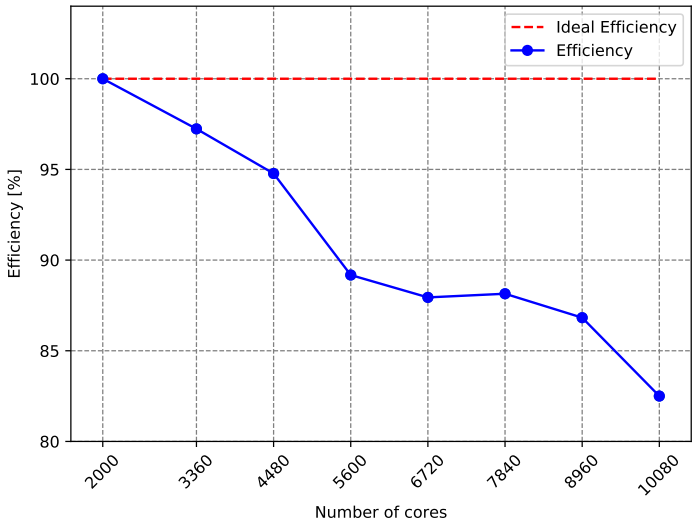

Fig. 12 Strong scalability test results for test case 2 (patient–specific aorta) and mesh M2: parallel efficiency at different core counts.

as seen in Table 7 for mesh M1 and Table 8 for mesh M2. This shows that the overall methodology for the coupled FSI problem is mathematically scalable. Tables 7 and 8 also contain the initialization time of the coupled framework. Similar to the previous test case, the reported initialization time includes the initialization times of the single–physics solvers, as well as the time for establishing the inter–code communication channels and initializing the coupling library modules. It does not include the pre-processing steps such as mesh generation or partitioning using the METIS library. The data in Table 7 show that the initialization time for mesh M1 only moderately increases by increasing the number of processes. Comparing the initialization time for 3920 and 560 cores in Table 7 (core count increased 7 times), we see a moderate 33% increase in the initialization time. On the other hand, data in Table 8 show a rather large increase in the initializing time for mesh M2 when increasing the number of cores. Increasing the core counts from 2000 to 10,080, the initialization time becomes more than double. The increase in the initialization time is moderate for up to 6720 cores, but rapidly grows for larger number of cores. Future work is required to optimize the initialization stage for large counts of CPUs. It is also worth noting that the initialization time increases linearly with the size of mesh. As seen in Table 4, the size of the mesh M2 is around 4.7 times larger than mesh M1. Comparing the initialization times in Tables 7 and 8 for similar numbers of cores, we see a similar ratio of around 4.7, which suggests a linear relation between the mesh size and the initialization time.

Finally, we compare the parallel scalability of the current framework to similar methods in the literature. Table 9 cites some of the most recently

Table 7 Average number of coupling iterations per time step, and the total initialization time of the coupled framework for test case 2 (patient–specific aorta) and mesh M1.

No. of cores	560	1120	1680	2240	2800	3360	3920
coupling iterations (average)	9.6	9.6	9.6	9.6	9.6	9.6	9.6
Initialization time [s]	263.9	261.3	274.8	308.8	309.7	328.8	352.5

Table 8 Average number of coupling iterations per time step, and the total initialization time of the coupled framework for test case 2 (patient–specific aorta) and mesh M2.

No. of cores	2000	3360	4480	5600	6720	7840	8960	10080
coupling iterations (average)	10.2	10.2	10.2	10.2	10.2	10.2	10.2	10.2
Initialization time [s]	1463.9	1680.7	1825.0	1836.0	1956.7	2146.8	2710.1	3145.4

published articles with similar methods to solve large-scale FSI problems and presents their scalability and parallel efficiency, as demonstrated by the test results in the referenced articles. The cited works in the table include both monolithic and partitioned methods. Moreover, they include works that have used publicly available software, as well as in-house codes. For this comparison, we have chosen works which included scalability test results. Therefore, other relevant works without such test results were not included. The cited works in Table 9 solve similar, but not identical, FSI problems for their scalability tests. Moreover, the tests were carried out on different supercomputers. These facts must be taken into account while comparing different results in Table 9. Nevertheless, it is still a helpful and indicative comparison that demonstrates the state-of-the-art in this field.

Table 9 Scalability of the current framework compared to other FSI solvers in the literature.

Source	FSI coupling method	Spatial grid type	Scalability (CPU cores)	Parallel efficiency on highest core count (%)
Present work	Partitioned	Unstructured	10,080	83
Hewitt et al. (2019) [10]	Partitioned	Structured	1,536	62
Tuković et al. (2019) [66]	Partitioned	Unstructured	16	62
Cajas et al. (2018) [9]	Partitioned	Unstructured	768	68
Kong et al. (2019) [7]	Monolithic	Unstructured	10,240	67
Seo et al. (2019) [67]	Monolithic	Unstructured	384	100
Jodlbauer et al. (2019) [68]	Monolithic	Structured	256	41
Forti et al. (2017) [69]	Monolithic	Unstructured	2,048	83
Kong et al. (2017) [5]	Monolithic	Unstructured	10,240	88
Deparis et al. (2016) [6]	Monolithic	Unstructured	8,192	75

As seen in Table 9, the scalability of the current framework is significantly better than other partitioned methods for FSI problems. The highest scalability for a partitioned FSI solver in the literature, to the best of our knowledge, is 1,536 cores [10]. However, the current framework is shown to scale well on over ten thousand cores with a very high parallel efficiency. The method presented by Hewitt et al. [10] has an important limitation as it cannot use distinct number of cores for fluid and solid solvers. This is a major issue since it effectively rules out any load balancing between the solvers, greatly reducing the computational efficiency. The method proposed in Cajas et al. [9] offers a high computational efficiency with an interesting load balancing method based on overloading the available cores in order to minimize the idle time. However, its scalability is limited to less than a thousand cores.

Comparing the scalability of the current work to monolithic methods in Table 9, it is seen that the scalability of the present framework is similar to the most scalable monolithic FSI solvers in the literature (e.g. [6,5,7]). Monolithic solvers offer an intrinsic advantage in terms of parallel scalability, since the whole coupled problem is solved as a single system of equations, using a single solver. Therefore, this approach does not face two of the most important issues of the partitioned approach, i.e., parallel communication between separate solvers and the inter-solver load balancing. Nevertheless, achieving a high scalability for monolithic methods is also a very challenging task, as FSI problems are highly complex and nonlinear. This is evident by examples of modern and advanced monolithic solvers which do not scale on more than a few hundred cores (e.g. [67,68]).

As partitioned methods offer certain advantages for FSI solution, one of their main drawbacks has been the loss of computational efficiency and parallel scalability in multi-code coupling. This is due to the challenging issues in terms of data structuring, domain decomposition, parallel data communication and inter-solver load balancing, as discussed throughout this article. Nevertheless, the present partitioned FSI solver scales as well as the most scalable monolithic solvers in the literature. This demonstrates the effectiveness of the methods presented in this work to overcome the common issues in multi-code coupled partitioned methods.

6 Conclusions

An efficient and scalable parallel solver is presented for the partitioned solution of fluid–structure interaction problems. The computational framework is developed through multi–code coupling, using separate fluid and structural solvers. Both single–physics solvers use distributed–memory parallelism and non–blocking point–to–point communicators. The inter–code communication is also fully parallel and point–to–point, avoiding any central communication unit. Both the intra–solver and the inter–code communication channels are established at the initialization stage. Two levels of

load balancing is considered, i.e., inside each solver and across the codes. The available processors are divided between the two solvers based on a load balancing model that minimizes the total idle time of the processes.

Both the fluid and the structural solver discretize and solve the corresponding governing equations on unstructured 3D meshes. A semi–implicit FSI coupling method is used, in which the fluid pressure term is segregated and strongly coupled to the structure, while the remaining terms are only loosely coupled. An efficient multi–vector quasi–Newton method is used to solve the coupled interface problem.

Two numerical test cases in the context of hemodynamics are considered and the strong scalability of the coupled framework is evaluated. The first test case is a benchmark FSI problem, solving an incompressible flow inside a deformable tube. The simulation results for this test case are compared to other numerical results from the literature in order to verify the accuracy of the solution methods and the overall framework. Scalability tests are carried out using a computational mesh consisting of 9 million unstructured tetrahedral cells for the fluid and 6 million structured hexahedral cells for the solid. Test results showed a parallel efficiency of 82% on 1400 CPU cores. This limit appears to be mostly determined by the size of mesh, as for higher number of cores the arithmetic intensity of the problem becomes too small. The number of coupling iterations at each time step remained nearly constant for different number of cores, which shows the developed methods are mathematically scalable. Moreover, the initialization time of the framework remained fairly constant and increased only slightly for different core counts.

The second test case solves the blood flow inside a patient–specific aorta. Two different meshes were used for the scalability tests, consisting of unstructured tetrahedral cells for both the fluid and the solid domain. Test results using a mesh with 20 million cells for the fluid and 9 million cells for the solid, showed a parallel efficiency of 79% on 3920 CPU cores. Similar to the previous test case, this limit appears to be determined by the mesh size and small arithmetic intensity for higher core numbers. Moreover, we have evaluated the effect of our inter-code load balancing model on the performance of the coupled solver. Applying the load balancing method reduced the run-time by an average of 15%, almost uniformly across the different total core counts. The scalability tests were repeated using a mesh with 95 million cells for the fluid and 41 million cells for the solid. Results demonstrated a parallel efficiency of 83% on 10,080 CPU cores. The number of FSI coupling iterations at each time step remained constant for different core counts, demonstrating the mathematical scalability of the methods. The initialization time of the framework increased only moderately for up to 6720 cores, however, it grew rapidly for higher number of cores. Furthermore, it was shown that the initialization time of the framework increases linearly by increasing the size of the mesh.

7 Acknowledgement

This work was financially supported by

- Ministerio de Economía y Competitividad, Secretaría de Estado de Investigación, Desarrollo e Innovación, Spain (ENE2017-88697-R),
- priority program 1648 - Software for Exascale Computing 214 (ExaFSA - Exascale Simulation of Fluid-Structure-Acoustics Interactions) of the German Research Foundation,
- and a FI PhD scholarship by the Agència de Gestió d'Ajuts Universitaris i de Recerca (AGAUR) of Generalitat de Catalunya (Spain).

The performance measurements were carried out on the SuperMUC supercomputer at Leibniz Rechenzentrum (LRZ) der Bayerischen Akademie der Wissenschaften. The authors wish to thank LRZ for the computing time and the technical support.

References

1. Y. Bazilevs, V. M. Calo, Y. Zhang, T. J. R. Hughes, Isogeometric fluid-structure interaction analysis with applications to arterial blood flow, Computational Mechanics 38 (4-5) (2006) 310–322. doi:10.1007/s00466-006-0084-3.
2. K. Takizawa, Y. Bazilevs, T. E. Tezduyar, Space–time and ALE–VMS techniques for patient-specific cardiovascular fluid–structure interaction modeling, Archives of Computational Methods in Engineering 19 (2) (2012) 171–225. doi:10.1007/s11831-012-9071-3.
3. J. Degroote, Partitioned simulation of fluid-structure interaction, Archives of Computational Methods in Engineering 20 (2013) 185–238. doi:10.1007/s11831-013-9085-5.
4. G. Hou, J. Wang, A. Layton, Numerical methods for fluid-structure interaction - A review, Communications in Computational Physics 12 (2) (2012) 337–377. doi:10.4208/cicp.291210.290411s.
5. F. Kong, X.-C. Cai, A scalable nonlinear fluid–structure interaction solver based on a schwarz preconditioner with isogeometric unstructured coarse spaces in 3d, Journal of Computational Physics 340 (2017) 498–518. doi:10.1016/j.jcp.2017.03.043.
6. S. Deparis, D. Forti, G. Grandperrin, A. Quarteroni, FaCSI: A block parallel preconditioner for fluid–structure interaction in hemodynamics, Journal of Computational Physics 327 (2016) 700–718. doi:10.1016/j.jcp.2016.10.005.
7. F. Kong, V. Kheyfets, E. Finol, X.-C. Cai, Simulation of unsteady blood flows in a patient-specific compliant pulmonary artery with a highly parallel monolithically coupled fluid-structure interaction algorithm, International journal for numerical methods in biomedical engineering 35 (7) (2019) e3208. doi:10.1002/cnm.3208.
8. S. Kataoka, S. Minami, H. Kawai, T. Yamada, S. Yoshimura, A parallel iterative partitioned coupling analysis system for large-scale acoustic fluid–structure interactions, Computational Mechanics 53 (6) (2014) 1299–1310. doi:10.1007/s00466-013-0973-1.

9. J. Cajas, G. Houzeaux, M. Vázquez, M. Garcia, E. Casoni, H. Calmet, A. Artigues, R. Borrell, O. Lehmkuhl, D. Pastrana, et al., Fluid-structure interaction based on hpc multicode coupling, SIAM Journal on Scientific Computing 40 (6) (2018) C677–C703. doi:10.1137/17M1138868.
10. S. Hewitt, L. Margetts, A. Revell, P. Pankaj, F. Levrero-Florencio, OpenFPCI: A parallel fluid–structure interaction framework, Computer Physics Communications 244 (2019) 469–482. doi:https://doi.org/10.1016/j.cpc.2019.05.016.
11. J. Larson, R. Jacob, E. Ong, The model coupling toolkit: a new fortran90 toolkit for building multiphysics parallel coupled models, The International Journal of High Performance Computing Applications 19 (3) (2005) 277–292. doi:10.1177/1094342005056115.
12. H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, preCICE–a fully parallel library for multi-physics surface coupling, Computers & Fluids 141 (2016) 250–258. doi:10.1016/j.compfluid.2016.04.003.
13. D. Thomas, M. L. Cerquaglia, R. Boman, T. D. Economon, J. J. Alonso, G. Dimitriadis, V. E. Terrapon, CUPyDO - An integrated Python environment for coupled fluid-structure simulations, Advances in Engineering Software 128 (2019) 69–85.
14. H.-J. Bungartz, F. Lindner, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, Partitioned fluid–structure–acoustics interaction on distributed data: Coupling via precice, in: H.-J. Bungartz, P. Neumann, W. E. Nagel (Eds.), Software for Exascale Computing - SPPEXA 2013-2015, Springer International Publishing, Cham, 2016, pp. 239–266. doi:10.1007/978-3-319-40528-5-11.
15. M. L. Cerquaglia, D. Thomas, R. Boman, V. Terrapon, J.-P. Ponthot, A fully partitioned lagrangian framework for fsi problems characterized by free surfaces, large solid deformations and displacements, and strong added-mass effects, Computer Methods in Applied Mechanics and Engineering 348 (2019) 409–442. doi:10.1016/j.cma.2019.01.021.
16. P. Causin, J. F. Gerbeau, F. Nobile, Added-mass effect in the design of partitioned algorithms for fluid-structure problems, Computer Methods in Applied Mechanics and Engineering 194 (2005) 4506–4527. doi:10.1016/j.cma.2004.12.005.
17. C. Förster, W. A. Wall, E. Ramm, Artificial added mass instabilities in sequential staggered coupling of nonlinear structures and incompressible viscous flows, Computer Methods in Applied Mechanics and Engineering 196 (2007) 1278–1293. doi:10.1016/j.cma.2006.09.002.
18. M. A. Fernández, J.-F. Gerbeau, C. Grandmont, A projection semi-implicit scheme for the coupling of an elastic structure with an incompressible fluid, International Journal for Numerical Methods in Engineering 69 (4) (2007) 794–821. doi:10.1002/nme.1792.
19. A. Naseri, O. Lehmkuhl, I. Gonzalez, E. Bartrons, C. D. Pérez-Segarra, A. Oliva, A semi-implicit coupling technique for fluid–structure interaction problems with strong added-mass effect, Journal of Fluids and Structures 80 (2018) 94–112. doi:10.1016/j.jfluidstructs.2018.03.012.
20. A. Naseri, I. Gonzalez, A. Amani, C. D. Pérez-Segarra, A. Oliva, A second–order time accurate semi–implicit method for fluid–structure interaction problems, Journal of Fluids and Structures 86 (2019) 135–155. doi:10.1016/j.jfluidstructs.2019.02.007.

21. Termo Fluids S.L., Termofluids, http://www.termofluids.com (2020).
22. I. Rodriguez, R. Borell, O. Lehmkuhl, C. D. Perez Segarra, A. Oliva, Direct numerical simulation of the flow over a sphere at Re = 3700, Journal of Fluid Mechanics 679 (2011) 263–287. doi:10.1017/jfm.2011.136.
23. I. Rodríguez, O. Lehmkuhl, J. Chiva, R. Borrell, A. Oliva, On the flow past a circular cylinder from critical to super-critical reynolds numbers: Wake topology and vortex shedding, International Journal of Heat and Fluid Flow 55 (2015) 91 – 103. doi:10.1016/j.ijheatfluidflow.2015.05.009.
24. N. Balcázar, L. Jofre, O. Lehmkuhl, J. Castro, J. Rigola, A finite-volume/level-set method for simulating two-phase flows on unstructured grids, International journal of multiphase flow 64 (2014) 55–72. doi:10.1016/j.ijmultiphaseflow.2014.04.008.
25. E. Gutiérrez, F. Favre, N. Balcazar, A. Amani, J. Rigola, Numerical approach to study bubbles and drops evolving through complex geometries by using a level set–moving mesh–immersed boundary method, Chemical Engineering Journal 349 (2018) 662–682. doi:10.1016/j.cej.2018.05.110.
26. P. Galione, O. Lehmkuhl, J. Rigola, A. Oliva, Fixed-grid numerical modeling of melting and solidification using variable thermo-physical properties–application to the melting of n-octadecane inside a spherical capsule, International Journal of Heat and Mass Transfer 86 (2015) 721–743. doi:10.1016/j.ijheatmasstransfer.2015.03.033.
27. E. Bartrons, C. Oliet, E. Gutierrez, A. Naseri, C. D. Pérez-Segarra, A finite volume method to solve the frost growth using dynamic meshes, International Journal of Heat and Mass Transfer 124 (2018) 615–628. doi:10.1016/j.ijheatmasstransfer.2018.03.104.
28. G. Colomer, R. Borrell, F. X. Trias, I. Rodríguez, Parallel algorithms for Sn transport sweeps on unstructured meshes, Journal of Computational Physics 232 (1) (2013) 118–135. doi:10.1016/j.jcp.2012.07.009.
29. L. Jofre, R. Borrell, O. Lehmkuhl, A. Oliva, Parallel load balancing strategy for volume-of-fluid methods on 3-d unstructured meshes, Journal of Computational Physics 282 (2015) 269–288. doi:10.1016/j.jcp.2014.11.009.
30. R. Borrell, J. Chiva, O. Lehmkuhl, G. Oyarzun, I. Rodríguez, A. Oliva, Optimising the Termofluids CFD code for petascale simulations, International Journal of Computational Fluid Dynamics 30 (6) (2016) 425–430. doi:10.1080/10618562.2016.1221503.
31. R. Verstappen, A. Veldman, Symmetry-preserving discretization of turbulent flow, Journal of Computational Physics 187 (1) (2003) 343–368. doi:10.1016/S0021-9991(03)00126-8.
32. F. X. Trias, O. Lehmkuhl, A. Oliva, C. D. Pérez-Segarra, R. Verstappen, Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, Journal of Computational Physics 258 (2014) 246–267. doi:10.1016/j.jcp.2013.10.031.
33. L. Jofre, O. Lehmkuhl, J. Ventosa, F. X. Trias, A. Oliva, Conservation properties of unstructured finite-volume mesh schemes for the navier-stokes equations, Numerical Heat Transfer, Part B: Fundamentals 65 (1) (2014) 53–79. doi:10.1080/10407790.2013.836335.
34. P. Thomas, C. Lombard, Geometric conservation law and its application to flow computations on moving grids, AIAA journal 17 (10) (1979) 1030–1037. doi:10.2514/3.61273.
35. M. Lesoinne, C. Farhat, Geometric conservation laws for flow problems with moving boundaries and deformable meshes, and their impact on aeroelas-

tic computations, Computer methods in applied mechanics and engineering 134 (1-2) (1996) 71–90. doi:10.1016/0045-7825(96)01028-6.
36. O. Estruch, O. Lehmkuhl, R. Borrell, C. D. P. Segarra, A. Oliva, A parallel radial basis function interpolation method for unstructured dynamic meshes, Computers and Fluids 80 (2013) 44–54. doi:10.1016/j.compfluid.2012.06.015.
37. P. Cardiff, I. Demirdžić, Thirty years of the finite volume method for solid mechanics, arXiv preprint arXiv:1810.02105 (2018).
38. H. Jasak, Error analysis and estimation for the finite volume method with applications to fluid flows, Ph.D. thesis, Imperial College London (University of London) (1996).
39. A. J. Macleod, Acceleration of vector sequences by multi-dimensional Δ^2 methods, Communications in Applied Numerical Methods 2 (4) (1986) 385–392.
40. Ž. Tuković, A. Ivanković, A. Karač, Finite-volume stress analysis in multi-material linear elastic body, International Journal for Numerical Methods in Engineering 93 (4) (2013) 400–419.
41. P. Cardiff, Tuković, H. Jasak, A. Ivanković, A block-coupled Finite Volume methodology for linear elasticity and unstructured meshes, Computers and Structures 175 (2016) 100–122. doi:10.1016/j.compstruc.2016.07.004.
42. P. Chandrashekar, A. Garg, Vertex-centroid finite volume scheme on tetrahedral grids for conservation laws, Computers & Mathematics with Applications 65 (1) (2013) 58–74.
43. I. González, A. Naseri, J. Chiva, J. Rigola, C. D. Pérez-Segarra, An enhanced finite volume based solver for thermoelastic materials in fluid-structure coupled problems, in: 6th European Conference on Computational Mechanics (ECCM-ECFD 2018), ECCOMAS, Glasgow, UK, June 2018.
44. U. Küttler, W. A. Wall, Fixed-point fluid-structure interaction solvers with dynamic relaxation, Computational Mechanics 43 (2008) 61–72. doi:10.1007/s00466-008-0255-5.
45. J. F. Gerbeau, M. Vidrascu, A quasi-Newton algorithm based on a reduced model for fluid-structure interaction problems in blood flows, ESAIM: Mathematical Modelling and Numerical Analysis 37 (2003) 631–647. doi:10.1051/m2an:2003049.
46. C. Michler, E. H. V. Brummelen, R. D. Borst, An interface Newton-Krylov solver for fluid-structure interaction, International Journal for Numerical Methods in Fluids 47 (10-11) (2005) 1189–1195. doi:10.1002/fld.850.
47. K. Scheufele, M. Mehl, Robust multisecant quasi-Newton variants for parallel fluid-structure simulations–and other multiphysics applications, SIAM Journal on Scientific Computing 39 (5) (2017) S404–S433. doi:10.1137/16M1082020.
48. H.-J. Bungartz, F. Lindner, M. Mehl, B. Uekermann, A plug-and-play coupling approach for parallel multi-field simulations, Computational Mechanics 55 (6) (2015) 1119–1129. doi:10.1007/s00466-014-1113-2.
49. M. Mehl, B. Uekermann, H. Bijl, D. Blom, B. Gatzhammer, A. Van Zuijlen, Parallel coupling numerics for partitioned fluid–structure interaction simulations, Computers & Mathematics with Applications 71 (4) (2016) 869–891. doi:10.1016/j.camwa.2015.12.025.
50. G. Oyarzun, R. Borrell, A. Gorobets, A. Oliva, MPI-CUDA sparse matrix–vector multiplication for the conjugate gradient method with an approximate inverse preconditioner, Computers & Fluids 92 (2014) 244–252. doi:10.1016/j.compfluid.2013.10.035.

51. G. Oyarzun, R. Borrell, A. Gorobets, A. Oliva, Portable implementation model for CFD simulations. application to hybrid CPU/GPU supercomputers, International Journal of Computational Fluid Dynamics 31 (9) (2017) 396–411. doi:10.1080/10618562.2017.1390084.
52. G. Oyarzun, R. Borrell, A. Gorobets, F. Mantovani, A. Oliva, Efficient CFD code implementation for the ARM-based Mont-Blanc architecture, Future generation computer systems 79 (2018) 786–796. doi:10.1016/j.future.2017.09.029.
53. G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392. doi:10.1137/S1064827595287997.
54. The HDF Group, Hierarchical Data Format, version 5, http://www.hdfgroup.org/HDF5/ (1997-2019).
55. A. Totounferoush, N. Ebrahimi Pour, J. Schroder, S. Roller, M. Mehl, A new load balancing approach for coupled multi-physics simulations, in: In Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Rio de Janeiro, Brazil, May 2019. doi:10.1109/IPDPSW.2019.00115.
56. A. Calotoiu, D. Beckinsale, C. W. Earl, T. Hoefler, I. Karlin, M. Schulz, F. Wolf, Fast multi-parameter performance modeling, in: 2016 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2016, pp. 172–181.
57. Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities , https://www.lrz.de.
58. L. Formaggia, J. F. Gerbeau, F. Nobile, A. Quarteroni, On the coupling of 3D and 1D Navier-Stokes equations for flow problems in compliant vessels, Computer Methods in Applied Mechanics and Engineering 191 (6-7) (2001) 561–582. doi:10.1016/S0045-7825(01)00302-4.
59. M. A. Fernández, M. Moubachir, A Newton method using exact Jacobians for solving fluid–structure coupling, Computers and Structures 83 (2005) 127–142. doi:10.1016/j.compstruc.2004.04.021.
60. A. Eken, M. Sahin, A parallel monolithic algorithm for the numerical simulation of large-scale fluid structure interaction problems, International Journal for Numerical Methods in Fluids 80 (12) (2016) 687–714. doi:10.1002/fld.4169.
61. A. Lozovskiy, M. A. Olshanskii, Y. V. Vassilevski, Analysis and assessment of a monolithic FSI finite element method, Computers & Fluids 179 (2019) 277–288. doi:10.1016/j.compfluid.2018.11.004.
62. 2nd CFD challenge predicting patient-specific hemodynamics at rest and stress through an aortic coarctation, http://www.vascularmodel.org/miccai2013/ (2013).
63. M. A. Fernández, M. Landajuela, M. Vidrascu, Fully decoupled time-marching schemes for incompressible fluid/thin-walled structure interaction, Journal of Computational Physics 297 (2015) 156–181. doi:10.1016/j.jcp.2015.05.009.
64. N. Westerhof, J.-W. Lankhaar, B. E. Westerhof, The arterial Windkessel, Medical & biological engineering & computing 47 (2) (2009) 131–141. doi:10.1007/s11517-008-0359-2.
65. S. Pant, B. Fabrèges, J.-F. Gerbeau, I. Vignon-Clementel, A methodological paradigm for patient-specific multi-scale cfd simulations: from clinical measurements to parameter estimates for individual analysis, International journal for numerical methods in biomedical engineering 30 (12) (2014) 1614–1648. doi:10.1002/cnm.2692.

66. Ž. Tuković, A. Karač, P. Cardiff, H. Jasak, A. Ivanković, OpenFOAM finite volume solver for fluid-solid interaction, Transactions of FAMENA 42 (3) (2018) 1–31. doi:10.21278/TOF.42301.
67. J. Seo, D. E. Schiavazzi, A. L. Marsden, Performance of preconditioned iterative linear solvers for cardiovascular simulations in rigid and deformable vessels, Computational Mechanics 64 (3) (2019) 717–739. doi:10.1007/s00466-019-01678-3.
68. D. Jodlbauer, U. Langer, T. Wick, Parallel block-preconditioned monolithic solvers for fluid-structure interaction problems, International Journal for Numerical Methods in Engineering 117 (6) (2019) 623–643. doi:10.1002/nme.5970.
69. D. Forti, A. Quarteroni, S. Deparis, et al., A parallel algorithm for the solution of large-scale nonconforming fluid-structure interaction problems in hemodynamics, Journal of Computational Mathematics 35 (3) (2017) 363–380. doi:10.4208/jcm.1702-m2016-0630.